2-2020

Stochastically robust personalized ranking for LSH recommendation retrieval

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

Stochastically Robust Personalized Ranking for LSH Recommendation Retrieval

Dung D. Le, Hady W. Lauw
School of Information Systems,
Singapore Management University
ddle@smu.edu.sg, hadywlauw@smu.edu.sg

Abstract

Locality Sensitive Hashing (LSH) has become one of the most commonly used *approximate nearest neighbor search* techniques to avoid the prohibitive cost of scanning through all data points. For recommender systems, LSH achieves efficient recommendation retrieval by encoding user and item vectors into binary hash codes, reducing the cost of exhaustively examining all the item vectors to identify the top-k items. However, conventional matrix factorization models may suffer from performance degeneration caused by randomly-drawn LSH hash functions, directly affecting the ultimate quality of the recommendations. In this paper, we propose a framework named SRPR, which factors in the stochasticity of LSH hash functions when learning real-valued user and item latent vectors, eventually improving the recommendation accuracy after LSH indexing. Experiments on publicly available datasets show that the proposed framework not only effectively learns user's preferences for prediction, but also achieves high compatibility with LSH stochasticity, producing superior post-LSH indexing performances as compared to state-of-the-art baselines.

1 Introduction

Matrix factorization (MF) gives rise to a widespread class of techniques for recommender systems (Koren, Bell, and Volinsky 2009). Deployment of MF-based recommender system typically involves two phases: *learning* and *retrieval*.

- The *learning phase* analyzes historical feedback to learn users' preferences (e.g., ratings, clicks). Fundamentally, a MF-based method 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 ${x_u}^T y_i$.
- The *retrieval phase* seeks to arrive at a personalized recommendation list for a target user u using the vectors learnt from the learning phase. To do so, we identify the top-k items according to preference scores ${x_u}^T y_i$.

In both phases, we are concerned with the measures of recommendation accuracy and computational efficiency. The learning phase is concerned with efficiency in terms of learning time. In most cases, learning algorithms can be run offline. Meanwhile there have been works on algorithms that learn from large datasets with millions of users and items.

Just as essential is the retrieval phase, which has received relatively less attention in the literature. This latter phase is just as sensitive (if not more) to efficiency, but of a different sort, namely latency of retrieval. A target user would expect retrieval of her personalized recommendation list in real-time. A naive approach that scans all items to select the top-k most relevant ones would incur per-query cost of $\mathcal{O}(n \times d)$, which scales linearly with the number of items n and the number of factors d. Given an enormous catalogue of candidate items, such exhaustive approach may not achieve truly real-time performance. On the other hand, precomputing the top-k for all users would be impractical given the storage costs involved for all the precomputed lists, as well as the rigidity built into those lists that may affect the system's adaptability to drifting user preferences. Having a faster alternative for top-k recommendation retrieval is desirable.

LSH Recommendation Retrieval. In face of these challenges, recent approaches turn to indexing schemes to overcome the prohibitive cost of performing exhaustive top-k recommendation search for each user. In particular, one of the most popular such schemes is Locality-Sensitive Hashing (LSH) (Shrivastava and Li 2015; Bachrach et al. 2014; Fraccaro, Paquet, and Winther 2016; Le and Lauw 2017; Hsieh et al. 2017; Liu and Wu 2016; Koenigstein and Koren 2013; Qi et al. 2017; Smirnov and Ponomarev 2014). In the prevalent binary variant, LSH approximates relative distances between data points, by computing the Hamming distance between the corresponding codes, and hashing similar data points to similar codes with high probability. Section 2 gives an overview of LSH-based recommendation retrieval.

There are significant advantages to using LSH as an indexing scheme for recommendation retrieval. By using hashcodes, we can quickly retrieve a small subset of items that likely contain the k "most relevant" items to the query (user), because computing Hamming distance involves mainly bitwise operations. Hashing-based indexing provides provably sub-linear algorithms for search which is beneficial for large scale systems where even linear search algorithms are impractical due to latency. Unlike spatial

partitioning techniques, both the running time and the accuracy guarantee of LSH are independent of the dimensionality of the data. Hashing-based indexing schemes are also massively parallelizable. This makes LSH suitable for large-scale distributed processing system dealing with high-dimensional datasets now common in modern applications.

Problem. As LSH is inherently stochastic, there is inevitable loss of information due to randomized hash functions. There are two reasons. For one, binary LSH may suffer from large variances in its estimation of the similarity between data points (Ji et al. 2012). For another, data points may be distributed non-uniformly over the space. Some hashcodes may be shared by disproportionately many items.

For recommender system in particular, due to these variances, the output vectors of the learning algorithm might not be well-aligned with the structural property of LSH, i.e., the hashcodes may not preserve the ranking of original vectors, causing significant degeneration of recommendation accuracy post-indexing. To achieve better similarity estimation of point-wise similarity and data space approximation, longer codes (more hash functions) are needed, which is in conflict with the original objective of speeding up retrieval.

Approach. There are several possible directions to get around the potential lack of compatibility between a recommendation algorithm and LSH hash functions. One is to bypass the hash functions altogether and to learn binary codes directly, but this transfers the efficiency issue to the learning phase as such methods may require solving NP-hard discrete optimization problems (see Section 4). Another direction, which we follow here, is to learn a "better" set of user/item vectors that achieve greater compatibility with LSH.

In particular, we trace the compatibility issue to the stochastic nature of the randomized hash functions. We will explore this in greater depth in Section 3.1. In essence, the variance due to the probabilistic hash functions may end up violating the preferences expressed by the user and item vectors from the learning phase. To alleviate the resulting performance degeneration, we explore the idea of deriving user and item representations that are robust to the stochasticity of LSH, such that the post-LSH hashcodes would still preserve the rankings by the original vectors with high probability. We achieve this *"LSH-friendly"* property by incorporating the stochasticity of LSH in the learning phase.

Contributions. *Firstly,* to our best knowledge, we are the first to factor the stochasticity of LSH in learning user/item latent vectors. This is based on an analysis of the effect of LSH stochasticity on recommendation retrieval in Section 3.1. *Secondly,* we propose a probabilistic framework SRPR that optimizes for the recommendation accuracy post-LSH indexing in Section 3.2, accommodating two widely-adopted LSH families, i.e., sign random projection and L2-LSH. *Thirdly,* we conduct comprehensive experiments on public datasets to validate the effectiveness of our proposed framework against baselines in Section 5.

2 LSH Recommendation Retrieval

Figure 1 illustrates a recommendation retrieval process based on LSH, consisting of two distinct phases. One phase, which could be offline, is the *learning phase*. In this phase,

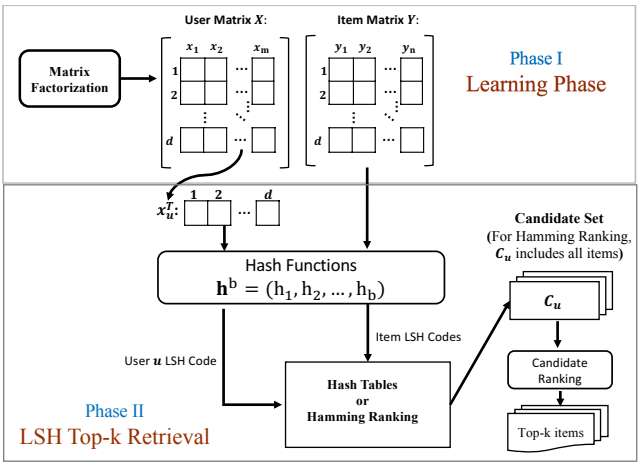

Figure 1: LSH Recommendation Retrieval

we presume that an MF-based method, e.g., (Rendle et al. 2009; Salakhutdinov and Mnih 2007), 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. The other phase, commonly online, is the *top-k recommendation retrieval*. In this phase, before observing any recommendation request, a set of bit functions $\mathbf{h}^b = (h_1, h_2, \ldots, h_b)$ are drawn from a LSH family and used to map each item vector y_i into a binary code $\mathbf{h}^b(y_i) \in \{0,1\}^b$.

With the hashcodes in place, we could pursue one of two possible search strategies as outlined below:

- *Hamming Ranking*: Given a user u, we hash x_u to get the hashcode $\mathbf{h}^b(x_u)$. We then measure the Hamming distances between $\mathbf{h}^b(x_u)$ and the hashcode of every item $\mathbf{h}^b(y_i), \forall 1 \leq i \leq n$, and keep the k items with the smallest Hamming distances. Technically, this still requires scanning all items. In practice, Hamming ranking with simple XOR operations is much more efficient than exhaustive search with real-valued vectors.
- *LSH Hash Tables*: In this alternative, we organize items of the same hashcode into a bucket. Upon the appearance of a target user u, we hash x_u into $\mathbf{h}^b(x_u)$. The candidate set C_u consists of items from the hash table of the same hashcode. A ranking process is performed on C_u to arrive at the top-k based on the inner product computation on the original real-valued vectors. The speedup comes from the fact that C_u is expected to be a small subset of all items.

In Section 5, we will experiment with both search strategies.

3 Stochastically Robust Personalized Ranking

For high quality top-k recommendations with LSH, the hashcodes should preserve the ranking induced by vectors from the *learning phase*. However, since the hash functions are randomly drawn, the LSH codes may not reflect the order of the real-valued vectors. In Section 3.1, we analyze the stochastic properties of LSH hash functions in detail, which underpins the proposal to factor in the structural property of LSH in the learning phase as described in Section 3.2.

3.1 Analysis of LSH Stochasticity

One important notion in the following analysis is a quantity p_{ui}, which is the probability that a hash function results in different hash values for x_u and y_i, i.e., $p_{ui} = \Pr(h(x_u) \neq h(y_i))$. For *locality-sensitive* hash functions, this probability is inversely proportional to the similarity between data vectors, i.e., $p_{ui} \propto \frac{1}{\text{sim}(x_u, y_i)}$ (Shrivastava and Li 2015). For x_u, y_i, y_j in $\mathbb{R}^d$, we can estimate the probability that their LSH codes would preserve the correct ordering as follows.

The Hamming distance $\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H$ is the number of positions $1 \leq l \leq b$ that $h_l(x_u) \neq h_l(y_i)$, i.e.,

$$\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H = \sum_{1 \leq l \leq b} z^l_{ui},$$

in which $z^l_{ui} = 1$ if $h_l(x_u) \neq h_l(y_i)$ and 0 otherwise.

It is straightforward to see that $z^l_{ui}, 1 \leq l \leq b$ are binary random variables following the Bernoulli distribution parameterized by success probability p_{ui}. Since the hash functions $h_1, h_2, \ldots, h_b$ are independent of one another, $\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H = \sum_{1 \leq l \leq b} z^l_{ui}$ therefore follows the binomial distribution with mean bp_{ui} and variance $bp_{ui}(1 - p_{ui})$, which can be approximated by a normal distribution with same mean and variance, i.e.,

$$\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H \sim \text{Normal}\left(bp_{ui}, bp_{ui}(1 - p_{ui})\right) \quad (1)$$

As the difference between two normal distributions is also a normal distribution, we thus have:

$$\begin{aligned}\lambda^b_{uij} &= \left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_j)\right|_H - \left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H \quad (2)\\ &\sim \mathcal{N}\left(bp_{uj} - bp_{ui}, bp_{uj}(1 - p_{uj}) + bp_{ui}(1 - p_{ui})\right)\end{aligned}$$

i.e., λ^b_{uij} follows the normal distribution with mean $b(p_{uj} - p_{ui})$ and variance $b(p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui}))$. From Eq. 2, we have the following lemma.

Lemma 1 *Without loss of generality, assume that x_u is closer to y_i than to y_j, i.e,* $\text{sim}(x_u, y_i) > \text{sim}(x_u, y_j)$. *The probability that their hashcodes preserve the correct ranking order is:*

$$\begin{aligned}&\Pr\left(\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_j)\right|_H > \left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H\right)\\ &= \Phi\left(\frac{\sqrt{b}(p_{uj} - p_{ui})}{\sqrt{p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui})}}\right), \quad (3)\end{aligned}$$

in which, Φ is the cummulative distribution function of the standard normal distribution.

Proof: From Equation 2, we have the following analysis:

$$\begin{aligned}&\Pr\left(\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_j)\right|_H > \left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H\right)\\ &= \Pr\left(\tilde{\lambda}^{\mathbf{b}}_{uij} > \frac{-b(p_{uj} - p_{ui})}{\sqrt{b(p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui}))}}\right)\\ &\left(\tilde{\lambda}^{\mathbf{b}}_{uij} = \frac{\lambda^{\mathbf{b}}_{uij} - b(p_{uj} - p_{ui})}{\sqrt{b(p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui}))}} \sim \mathcal{N}(0, 1)\right)\\ &= \Phi\left(\frac{b(p_{uj} - p_{ui})}{\sqrt{b(p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui}))}}\right) \quad (4)\end{aligned}$$

Insights. Lemma 1 reveals several insights. *Firstly*, a large denominator $p_{uj}(1-p_{uj})+p_{ui}(1-p_{ui})$ would likely lead to erroneous ranking between x_u, y_i, y_j using their hash codes. That binary LSH usually suffers from large variances (Ji et al. 2012) is a concern. One deceptively simple solution, also the *second* insight, is to substantially increase code length b, as $\sqrt{b}$ increases the probability in Lemma 1 to preserve the original ranking. However, the resulting code may contain redundant bits. It also consumes more computations to the contrary of the speedup objective. *Thirdly*, if the quantity

$$\gamma_{uij} = \frac{p_{uj} - p_{ui}}{\sqrt{p_{uj}(1 - p_{uj}) + p_{ui}(1 - p_{ui})}} \quad (5)$$

is large enough, it is more likely that the hashcodes will preserve the ranking (lessening the need for longer codes). Since γ_{uij} is defined over trainable vectors x_u, y_i, y_j, we can learn these vectors to maximize γ_{uij} so that their LSH codes would in turn preserve the ranking better. Importantly, this third insight is the basis for our proposed framework to be elaborated in the next section.

3.2 Factoring LSH Stochasticity in Learning

To achieve concise yet informative binary codes, we take into account the stochasticity of LSH functions when learning user and item vectors. The objective is to derive a representation that is robust to the stochasticity of LSH functions so that the hashcodes would preserve the ranking of original items with high probability.

Input. To build such representation, we learn from ordinal preference observations. The input that we consider is a set of triples $\mathcal{T} \subset \mathcal{U} \times \mathcal{I} \times \mathcal{I}$, where $\mathcal{U}$ and $\mathcal{I}$ are sets of users and items respectively. 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. From ratings, we can induce an ordinal triple for each instance when user u rates item i higher than she rates item j (Le and Lauw 2017). Triples can also model the implicit feedback (Rendle et al. 2009).

Output. The goal is to derive a d-dimensional latent vector $x_u \in \mathbb{R}^d$ for each user $u \in \mathcal{U}$ (denoted X collectively for $\mathcal{U}$), and a latent vector $y_i \in \mathbb{R}^d$ for each item $i \in \mathcal{I}$ (denoted Y collectively for $\mathcal{I}$), that not only capture the user preference accurately pre-hashing, but also ensure that their post-hashing codes would preserve the ranking established by the real-valued latent representation with high probability. In other words, for a triple t_{uij}, $\mathbf{h}^b$ are expected to produce hash codes with a smaller Hamming distance between $\mathbf{h}^b(x_u)$ and $\mathbf{h}^b(y_i)$ than the Hamming distance between $\mathbf{h}^b(x_u)$ and $\mathbf{h}^b(y_j)$, i.e., $\left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_j)\right|_H > \left|\mathbf{h}^b(x_u) - \mathbf{h}^b(y_i)\right|_H$.

Triplet Likelihood. A crucial component is how the latent vectors of users and items would generate the ordinal triplet comparisons in $\mathcal{T}$. The principle relating a triple t_{uij} is: if u prefers i to j, the probability of the event that *the Hamming distance between the hashcode of x_u and y_i is smaller than that between x_u and y_j* is as high as possible. From Lemma 1, that is equivalent to having the quantity γ_{uij} (Eq. 5) to be as positive as possible. We therefore propose to formulate

the probablity of triples t_{uij} in terms of γ_{uij} as in Eq. 6.

$$\mathrm{P}(t_{uij}|x_u, y_i, y_j) = \Phi\left(\sqrt{b}\gamma_{uij}\right), \tag{6}$$

For each observed triple t_{uij}, we want to maximize the likelihood probability $\Phi\left(\sqrt{b}\gamma_{uij}\right)$, which is equivalent to maximizing γ_{uij}. Interestingly, as p_{uj} moves towards 1 and p_{ui} moves towards 0, the value of γ_{uij} keeps increasing. Hence as we learn X and Y to maximize γ_{uij} for each $t_{uij} \in \mathcal{T}$, a user would be located closer to her more preferred items (p_{ui} is low) as compared to less preferred items (p_{uj} is high).

Objective Function. Given the set of triples $\mathcal{T}$ and the number of intended LSH functions b, find the set of user coordinates X and item coordinates Y, such that the following sum of the log-likelihood-probabilities is maximized:

$$\mathcal{L} = \max_{X,Y} \sum_{t_{uij} \in \mathcal{T}} \ln\left(\Phi(\sqrt{b}\gamma_{uij})\right) \tag{7}$$

There are two widely-adopted LSH schemes, Sign Random Projection and L2-LSH. We describe how SRPR could accommodate them, creating two variants: SRPR-SRP and SRPR-L2.

Sign Random Projection. SRP-LSH (Charikar 2002) is a LSH scheme for angular similarity, i.e., $\mathrm{sim}(x_u, y_i) = 1 - \mathring{cos}^{-1}\left(\frac{x_u^T y_i}{||x_u||||y_i||}\right)$. The SRP-LSH hash function is:

$$h_a^{\mathrm{srp}}(x) = \mathrm{sign}(a^T x); \tag{8}$$

in which, the parameter a is random vector chosen with each component from i.i.d normal. The corresponding probability of x_u, y_i having different hash values is:

$$p_{ui}^{\mathrm{srp}} = \Pr(h_a^{\mathrm{srp}}(x_u) \neq h_a^{\mathrm{srp}}(y_i)) = \frac{1}{\pi}.\cos^{-1}\left(\frac{x_u^T y_i}{||x_u||.||y_i||}\right), \tag{9}$$

L2-LSH. L2-LSH is the LSH scheme for L2 distance (Datar et al. 2004), whose hash function is defined as:

$$h_{a,b}^{\mathrm{srp}}(x) = \lfloor \frac{a^T x + b}{r} \rfloor; \tag{10}$$

where, r is the window size, a is the random vector with each component from i.i.d normal, i.e, $a_i \sim \mathrm{Normal}(0, 1)$, and a scalar $b \sim \mathrm{Uni}(0, r)$. The probability of two points x, y having different hash values under L2-LSH function is:

$$p_{ui}^{l_2} = \Pr\left(h_{a,b}^{l_2}(x_u) \neq h_{a,b}^{l_2}(y_i)\right) = \tag{11}$$
$$2\Phi\left(-\frac{r}{d_{ui}})\right) + \frac{1}{\sqrt{(2\pi)}(r/d_{ui})}\left(1 - \exp\left(-(\frac{r}{d_{ui}})^2/2\right)\right)$$

where, $\Phi(x)$ is cumulative probability function of normal distribution and $d_{ui} = ||x_u - y_i||$ is the L2 distance between x_u, y_i. In this study, we use $r = 2.5$ as in (Shrivastava and Li 2014).

Parameter Learning. The learning algorithm for SRPR framework is based on gradient ascent. It first initializes the user and item latent vectors with each coordinate drawn from normal standard distribution. In each iteration, the model parameters are updated based on the gradients, with a decaying learning rate ϵ over time. The time complexity of the algorithm is linear to the number of triples in $\mathcal{T}$, i.e., $\mathcal{O}(|\mathcal{T}|)$, which is similar to that of triple-based methods (Rendle et al. 2009), (Le and Lauw 2017).

4 Related Work

LSH for Maximum Inner Product Search (MIPS). (Shrivastava and Li 2014) claims the infeasible existence of LSH scheme for MIPS. A workaround is to transform MIPS into *Nearest Neighbor Search (NNS)* (Shrivastava and Li 2014) or *Maximum Cosine Similarity Search (MCSS)* (Shrivastava and Li 2015) by augmenting user and item vectors into higher-dimensional space. Later, (Neyshabur and Srebro 2015) achieves better performance by extending the output latent vectors by one dimension to equalize the magnitude of item vectors. However, (Huang et al. 2018) points out that even the vector transformation in (Neyshabur and Srebro 2015) suffers from large distortion error in reducing MIPS to NNS, and proposes Query Normalized First (QNF) transformation to reduce the distortion and achieve better performance. In the experiment we compare to the composition of ranking-based MF method BPR followed by QNF transformation, code-named BPR-QNF.

Indexable Representation. Another approach is to obviate the need for transformation by modelling the user-item interaction with a proper metric distance. For instance, CFEE (Khoshneshin and Street 2010) fits a rating $\hat{r}_{ui}$ in terms of the squared Euclidean distance between x_u and y_i, i.e., $\hat{r}_{ui} \propto ||x_u - y_i||^2$. COE (Le and Lauw 2016) seeks to ensure that an item i liked by a user u would be placed closer to the user on the Euclidean representation than a less preferred item j. For COE and CFEE, the retrieval of top recommendations becomes nearest neighbor search (NNS). IPMF (Fraccaro, Paquet, and Winther 2016) keeps the classic formulation of matrix factorization, but incorporates additional constraint that all item vectors have the same magnitude. IBPR (Le and Lauw 2017) proposes the use of angular distance kernel, evaluated as the `arccos` of the inner product between the normalized vectors, to model pairwise ordinal preferences. These methods however does not take into account the LSH stochasticity. In Section 5, we compare SRPR to IPMF, IBPR, CFEE, and COE to validate our stochasticity-robust representations.

Discrete Representation. Discrete representation methods (Zhou and Zha 2012), (Zhang et al. 2016), (Zhang, Lian, and Yang 2017) attempt to directly represent each user/item by a binary code. Quantization-based discretization such as (Zhang et al. 2014) adopts a two-stage process: constraining the learning process and applying a binary-quantization algorithm to generate the code from the learned real-valued vectors. Optimization-based discretization such as DCF (Zhang et al. 2016), DPR (Zhang, Lian, and Yang 2017) adopt classic matrix factorization formulations, while imposing further constraints on balance and decorrelation for the binary codes. Learning binary codes may require solving a NP-hard discrete optimization problem. Different

	#users	#items	#ratings	#training ordinal triples
Movielens 20M	138493	27278	20000263	5.46×10^8
Netflix	480189	17770	100480507	2.29×10^{10}

Table 1: Dataset Statistics

from stochastic LSH, disrete representation learning is deterministic and data-dependent. We compare to PPH, DCF, and DPR in Section 5.

Data-Independent Hashing. (Jin et al. 2014), (Park, Cafarella, and Mozafari 2015), (Ji et al. 2012) propose different designs for the hash functions that are sensitive to the density or the neighborhood structure of the data, effectively reducing the variance of the similarity estimation of the hashcodes. These developments are orthogonal to us, as we are interested in deriving the vector representation from preference data that are robust to LSH stochasticity.

5 Experiments

5.1 Experiment Setting

Datasets. We experiment on two large public datasets: *MovieLens 20M*[1] and *Netflix*[2] (Table 1). By default, *MovieLens 20M* includes users with at least 20 ratings. For consistency, we apply the same to *Netflix*. We randomly keep 60% of the ratings for training and hide 40% for testing in a stratified manner. We report the average results over five such random training/testing splits. We generate a triple t_{uij} if user u has higher rating on item i than on item j with i, j, and these triples are formed within the training set only.

Comparative Methods. For SRPR-SRP and SRPR-L2, after learning the representations, we produce the hashcodes using SRP-LSH and L2-LSH respectively. We experiment with 10 different sets of LSH functions, and the average results are reported in Tables 2 and 3. For the LSH itself, we use the implementation in (Aly, Munich, and Perona 2011). We consider the following categories of baselines.

LSH-Indexed Baselines:

- BPR-QNF: BPR (Rendle et al. 2009), followed by transformation in (Huang et al. 2018).
- CFEE, COE: Euclidean embedding methods that fits ratings (Khoshneshin and Street 2010) and ordinal triples (Le and Lauw 2016) respectively.
- IPMF: matrix factorization that learns fixed-length item vectors (Fraccaro, Paquet, and Winther 2016).
- IBPR: a development of BPR with the use of arccos kernel for indexable representation (Le and Lauw 2017).

Comparing to these LSH-indexed methods allows us to prove the effectiveness of factoring in the stochasticity of LSH when learning user and item vectors.

Discrete Representation:

- PPH (Zhang et al. 2014): a two-step quantization-based discretization method.
- DCF , DPR : optimization-based discretization methods that fits ratings (Zhang et al. 2016) and ordinal triples (Zhang, Lian, and Yang 2017) respectively.

Comparing to PPH, DCF, and DPR would validate the benefit of learning real-valued vectors that are robust to LSH stochasticity as compared to learning the binary vectors.

We tune the hyper-parameters of all models for the best performances. For IPMF, we adopt the setting used for *Netflix* dataset. For the ranking-based methods (BPR, COE, IBPR and SRPR), the learning rate and the regularization are 0.05 and 0.001 respectively. For CFEE, these values are 0.1 and 0.001 respectively. All LSH-indexed models use $d = 20$ dimensionalities in their latent representations. Similar trends are observed with other dimensionalities.

5.2 Comparing to LSH-Indexed Baselines

Hamming Ranking. To investigate how well the ranking induced by the hashcodes preserve the ordering of observed test ratings, we measure *nDCG* (Normalized Discounted Cummulative Gain) with *Hamming Ranking* search strategy. For LSH-indexed models, we measure *nDCG* on LSH codes (*Absolute LSH nDCG*) and its relative ratio to *nDCG* of original vectors (*Relative LSH nDCG*).

Table 2 shows the performances of all models with $k = 10$ for *MovieLens 20M* (top panel) and *Netflix* (bottom panel). The second column, *Brute Force nDCG@10*, presents the *nDCG@10* with linear scanning for all models with $d = 20$. SRPR-based models consistently show better *Absolute LSH nDCG@10* values compared to the baselines across different number of bits b for both datasets. This implies that the LSH codes of SRPR models are more compact and informative as compared to those of the baselines. Also, the *Relative LSH nDCG@10*s of SRPR-based models are higher than other methods and are closest to 1, showing that SRPR is more robust to the stochasticity of LSH, resulting in LSH codes that preserve the ranking of the original vectors with greater fidelity.

Hash Tables. When using hash tables for top-k retrieval, one specifies the number of tables T and the code length b. We experiment with various values of T, and $T = 10$ returns the best performances for all models. We also vary the number of hash functions b and larger b is expected to lead to fewer items in each bucket.

The hash table approach processes only a subset of items to produce the top-k, which in the test set may include both rated and unrated items. In practice, we are interested in whether a user's most preferred items could be found in the top-k. We define preferred items as those with maximum rating. For each user u with at least one highest rating item in the test set, we compute the percentage of these that are returned in the top-k. The higher the percentage, the better is the model at identifying the items a user prefers the most. Eq. 12 presents the formula for *Recall@k*:

$$\text{Recall@}k = \frac{1}{|\mathcal{U}_{\max}|} \sum_{u \in \mathcal{U}_{\max}} \frac{|\{i \in \psi_k^u : r_{ui} = \text{max rating}\}|}{|\{i \in \mathcal{I} : r_{ui} = \text{max rating}\}|}, \tag{12}$$

in which $\mathcal{U}_{\max}$ is the set of users with at least one item with max rating and ψ_k^u is the returned top-k. We exclude train-

[1] http://grouplens.org/datasets/movielens/20m/

[2] http://academictorrents.com/details /9b13183dc4d60676b773c9e2cd6de5e5542cee9a

	Brute Force nDCG@10 ($d = 20$)	Absolute LSH nDCG@10					Relative LSH nDCG@10				
b	-	**8**	**16**	**32**	**64**	**128**	**8**	**16**	**32**	**64**	**128**
MovieLens 20M											
CFEE	0.703	0.582	0.582	0.585	0.586	0.587	0.828	0.829	0.832	0.834	0.836
COE	0.698	0.605	0.609	0.608	0.610	0.611	0.867	0.872	0.870	0.873	0.875
BPR-QNF	0.715	0.646	0.661	0.678	0.698	0.718	0.895	0.911	0.905	0.896	0.913
IPMF	0.751	0.640	0.651	0.647	0.641	0.653	0.861	0.880	0.904	0.930	0.957
IBPR	0.753	0.679	0.703	0.721	0.734	0.742	0.901	0.934	0.957	0.975	0.985
SRPR-L2	0.750	**0.684**	**0.710**	**0.721**	**0.736**	**0.741**	**0.912**	**0.947**	**0.961**	**0.981**	**0.988**
SRPR-SRP	0.752	**0.697**	**0.715**	**0.729**	**0.739**	**0.745**	**0.927**	**0.950**	**0.970**	**0.983**	**0.991**
Netflix											
CFEE	0.625	0.541	0.561	0.562	0.564	0.565	0.867	0.897	0.899	0.901	0.904
COE	0.654	0.570	0.565	0.575	0.575	0.578	0.872	0.865	0.880	0.880	0.884
BPR-QNF	0.644	0.519	0.514	0.511	0.513	0.514	0.806	0.798	0.794	0.796	0.798
IPMF	0.772	0.600	0.621	0.649	0.684	0.712	0.776	0.803	0.840	0.886	0.921
IBPR	0.767	0.650	0.695	0.714	0.738	0.744	0.848	0.906	0.932	0.963	0.971
SRPR-L2	0.759	**0.650**	**0.702**	**0.722**	**0.740**	**0.749**	**0.856**	**0.925**	**0.951**	**0.975**	**0.987**
SRPR-SRP	0.762	**0.675**	**0.708**	**0.730**	**0.744**	**0.752**	**0.886**	**0.929**	**0.958**	**0.976**	**0.987**

Table 2: *Absolute LSH nDCG@10* and *Relative LSH nDCG@10* (as b varies) - Hamming Ranking

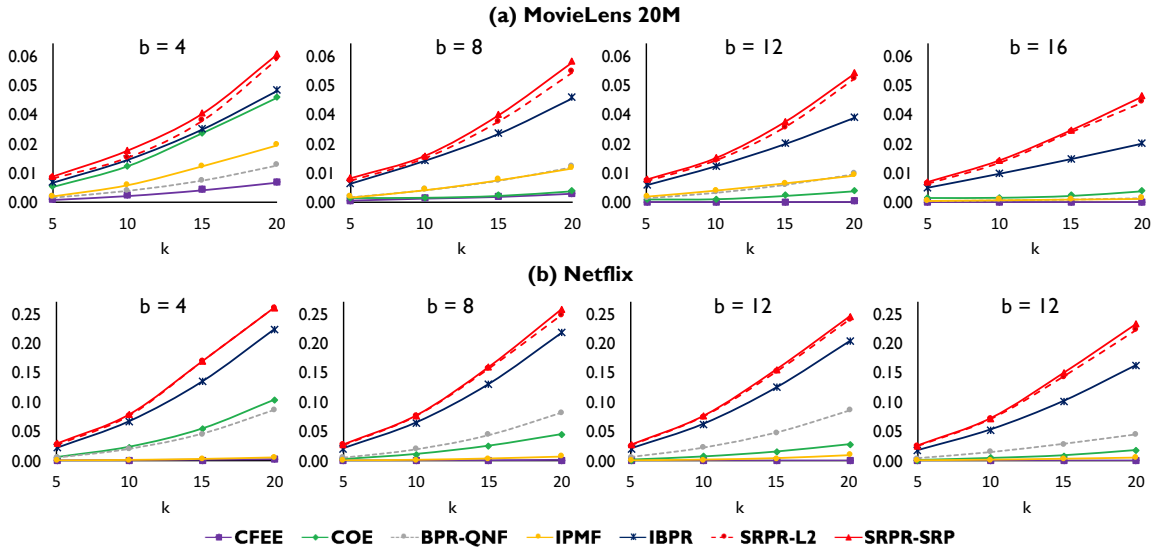

Figure 2: nRecall@k (k = 5, 10, 15, 20) with Locality Sensitive Hashing (T=10 Hash Tables)

ing items for u from both numerator and denominator. We normalize *Recall@k* with the ideal *Recall@k* value that an algorithm can achieve, and denote the metric as *nRecall@k*.

Figure 2(a) shows the *nRecall@k* using hash table lookup with $T = 10$ tables and different values of code length $b = 4, 8, 12, 16$ for *MovieLens 20M*. Across the b's, the trends are similar. SRPR has the highest *nRecall@k* values across all k. It outperforms BPR-QNF that conducts vector transformation as post-processing, which indicates that learning inherently stochastically robust vectors is helpful. Interestingly, SRPR and IBPR perform better than other matrix factorization models that fit ratings IPMF and CFEE, suggesting that learning from relative comparisons may be more suitable for top-k recommendation. Figure 2(b) shows the results for *Netflix*. Similarly, SRPR also shows higher *nRecall@k* values across all k, except that IBPR is more competitive when $b = 4$, though still lower than SRPR.

Accuracy-Efficiency Tradeoff. Another metric *speedup* is to investigate the efficiency of using LSH hash tables.

$$\text{Speedup} = \frac{\text{Retrieval time taken by exhaustive search}}{\text{Retrieval time taken by the index}}. \quad (13)$$

We investigate the tradeoff between the speedup achieved and the accuracy of the top-k returned by the hash tables. Figure 3 shows the *nRecall@10* values and the speedup with different number of hash functions b. When given the same desired speedup, SRPR achieves higher performance compared to the baselines. Also, given the same desired accuracy level, SRPR returns the highest speedup rate. Generally, for all models, the accuracy descreases and the speedup increases as b increases. This is expected, as the longer the codes, the smaller the set of candidates returned by the hash tables on which we need to compute the similarity scores.

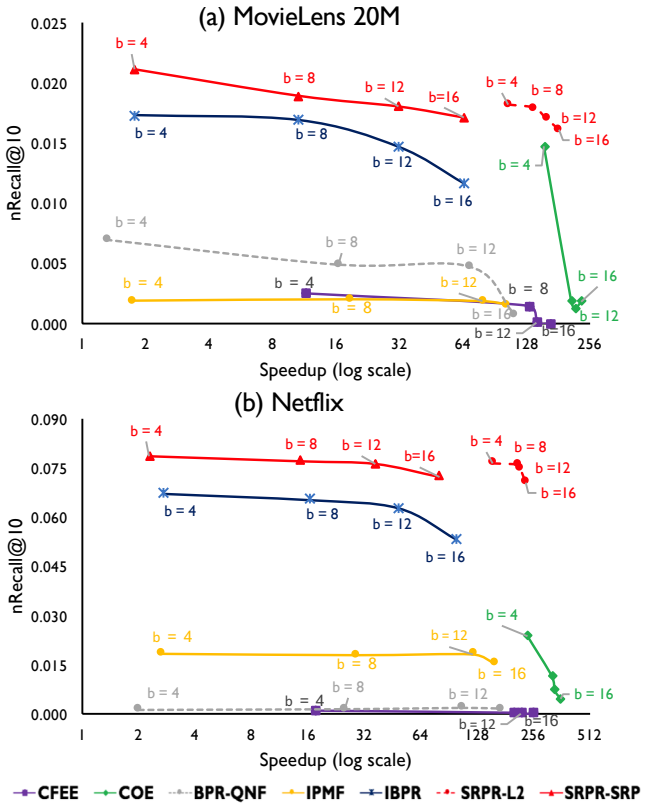

Figure 3: nRecall@10 vs. Speedup (T =10 LSH Tables)

5.3 Comparing to Discrete Representation

To validate the effectiveness of learning LSH-friendly vectors over learning the binary codes directly, we compare our SRPR models to the three discrete learning baselines: PPH, DCF, and DPR on Hamming ranking. Table 3 shows the nDCG@10s. For both *MovieLens 20M* and *Netflix*, SRPR-SRP's performances are significantly better than the three compared baselines. The values in the parentheses indicate the percentages of improvement of SRPR as compared to the best discrete model DPR. We intuit two possible reasons. For one, LSH-indexed methods have richer representation space than discrete representation methods. For another, as SRPR's vectors are compatible with LSH, the loss of recommendation accuracy caused by LSH is alleviated.

5.4 Top-k Recommendations without LSH

As the purpose of LSH is to approximate the proximity between the real-valued vectors learnt from the underlying preference models, the quality of LSH-based top-k recommendations may be inherited from and bounded by that of original vectors. In this section, we measure the *nDCG* and *nRecall* values of all models with linear scanning to better understand the quality of the learnt vectors.

Figure 4 shows the *nDCG@10* and *nRecall@10* values for the two datasets at various d. We observe that SRPR consistently outperforms the baselines in terms of *nRecall@10* across dimensions. SRPR is also among the best,

b	8	16	32	64	128
MovieLens 20M					
PPH	0.627	0.635	0.640	0.642	0.650
DCF	0.641	0.655	0.660	0.662	0.667
DPR	0.652	0.663	0.674	0.682	0.694
SRPR-L2 (d = 20) + LSH	0.684 (+4.9%)	0.710 (+7.1%)	0.721 (+7.0%)	0.736 (+7.9%)	0.741 (+6.7%)
SRPR-SRP (d = 20) + LSH	0.697 (+6.9%)	0.715 (+7.8%)	0.729 (+8.2%)	0.739 (+8.3%)	0.745 (+7.3%)
Netflix					
PPH	0.581	0.544	0.556	0.560	0.578
DCF	0.635	0.660	0.681	0.709	0.726
DPR	0.644	0.673	0.679	0.713	0.732
SRPR-L2 (d = 20) + LSH	0.650 (+0.9%)	0.702 (+4.3%)	0.722 (+6.3%)	0.740 (+3.8%)	0.749 (+2.3%)
SRPR-SRP (d = 20) + LSH	0.675 (+4.9%)	0.708 (+5.2%)	0.730 (+7.4%)	0.744 (+4.3%)	0.752 (+2.7%)

Table 3: SRPR versus Discrete Representation (nDCG@10)

with the most competitive baselines in terms of *nDCG@10* being IBPR and IPMF, which do not take into account LSH stochasticity. This shows that incorporating LSH stochasticity has not hurt the ranking performance in the original vector space, while improving the post-LSH performance.

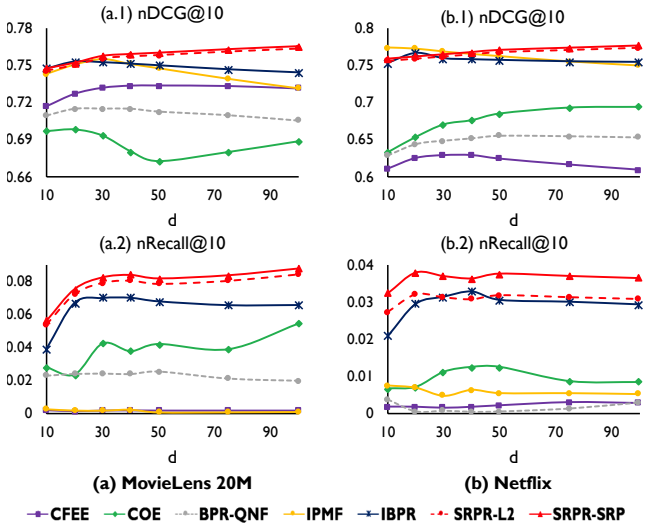

Figure 4: Top-10 Recommendations with Linear Scanning

6 Conclusion

In this paper, we deal with efficient recommendation retrieval with LSH. Our framework, namely SRPR, factors in the stochasticity of LSH when learning vector representations of users/items from ordinal triples. SRPR produces a representation that is robust to the stochasticity of LSH, resulting in significant gain in retrieval efficiency, while still maintaining high accuracy for top-k recommendations.

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

Aly, M.; Munich, M.; and Perona, P. 2011. Indexing in large scale image collections: Scaling properties and benchmark. In *WACV*, 418–425. IEEE.

Bachrach, Y.; Finkelstein, Y.; Gilad-Bachrach, R.; Katzir, L.; Koenigstein, N.; Nice, N.; and Paquet, U. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys*, 257–264. ACM.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, 380–388. New York, NY, USA: ACM.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, 253–262. New York, NY, USA: ACM.

Fraccaro, M.; Paquet, U.; and Winther, O. 2016. Indexable probabilistic matrix factorization for maximum inner product search. In *AAAI*, 1554–1560.

Hsieh, C.-K.; Yang, L.; Cui, Y.; Lin, T.-Y.; Belongie, S.; and Estrin, D. 2017. Collaborative metric learning. In *WWW*, 193–201.

Huang, Q.; Ma, G.; Feng, J.; Fang, Q.; and Tung, A. K. H. 2018. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *KDD*, 1561–1570.

Ji, J.; Li, J.; Yan, S.; Zhang, B.; and Tian, Q. 2012. Super-bit locality-sensitive hashing. In *NIPS*, 108–116.

Jin, Z.; Li, C.; Lin, Y.; and Cai, D. 2014. Density sensitive hashing. *IEEE transactions on cybernetics* 44(8):1362–1371.

Khoshneshin, M., and Street, W. N. 2010. Collaborative filtering via euclidean embedding. In *RecSys*, 87–94. ACM.

Koenigstein, N., and Koren, Y. 2013. Towards scalable and accurate item-oriented recommendations. In *RecSys*, RecSys '13, 419–422.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37.

Le, D. D., and Lauw, H. W. 2016. Euclidean co-embedding of ordinal data for multi-type visualization. In *SDM*, 396–404. SIAM.

Le, D. D., and Lauw, H. W. 2017. Indexable bayesian personalized ranking for efficient top-k recommendation. In *CIKM*, 1389–1398. ACM.

Liu, C.-L., and Wu, X.-W. 2016. Fast recommendation on latent collaborative relations. *Knowledge-Based Systems* 109:25 – 34.

Neyshabur, B., and Srebro, N. 2015. On symmetric and asymmetric lshs for inner product search. In *ICML*, 1926–1934. JMLR.org.

Park, Y.; Cafarella, M.; and Mozafari, B. 2015. Neighbor-sensitive hashing. *Proc. VLDB Endow.* 9(3):144–155.

Qi, L.; Xiang, H.; Dou, W.; Yang, C.; Qin, Y.; and Zhang, X. 2017. Privacy-preserving distributed service recommendation based on locality-sensitive hashing. In *2017 IEEE International Conference on Web Services (ICWS)*, 49–56. Los Alamitos, CA, USA: IEEE Computer Society.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2009. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461. AUAI Press.

Salakhutdinov, R., and Mnih, A. 2007. Probabilistic matrix factorization. In *NIPS*, NIPS'07, 1257–1264. USA: Curran Associates Inc.

Shrivastava, A., and Li, P. 2014. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, 2321–2329.

Shrivastava, A., and Li, P. 2015. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *UAI*, 812–821. Arlington, Virginia, United States: AUAI Press.

Smirnov, A., and Ponomarev, A. 2014. A hybrid peer-to-peer recommendation system architecture based on locality-sensitive hashing. In *Proceedings of 15th Conference of Open Innovations Association FRUCT*, 119–125. IEEE.

Zhang, Z.; Wang, Q.; Ruan, L.; and Si, L. 2014. Preference preserving hashing for efficient recommendation. In *SIGIR*.

Zhang, H.; Shen, F.; Liu, W.; He, X.; Luan, H.; and Chua, T.-S. 2016. Discrete collaborative filtering. In *SIGIR*, 325–334. ACM.

Zhang, Y.; Lian, D.; and Yang, G. 2017. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In *AAAI*.

Zhou, K., and Zha, H. 2012. Learning binary codes for collaborative filtering. In *KDD*, 498–506.