Learning Similarity Preserving Binary Codes for Recommender Systems

Yang Shi
yang.shi@rakuten.com
Rakuten Group, Inc.
San Mateo, CA, USA

Young-joo Chung
youngjoo.chung@rakuten.com
Rakuten Group, Inc.
San Mateo, CA, USA

ABSTRACT

Hashing-based Recommender Systems (RSs) are widely studied to provide scalable services. The existing methods for the systems combine three modules to achieve efficiency: feature extraction, interaction modeling, and binarization. In this paper, we study an unexplored module combination for the hashing-based recommender systems, namely Compact Cross-Similarity Recommender (CCSR). Inspired by cross-modal retrieval, CCSR utilizes Maximum a Posteriori similarity instead of matrix factorization and rating reconstruction to model interactions between users and items. We conducted experiments on both public and real-world E-commerce datasets and confirmed CCSR outperformed the existing matrix factorization-based methods. On the Movielens1M dataset, the absolute performance improvements are up to 15.69% in NDCG. In addition, we extensively studied three binarization modules: *sign*, scaled *tanh*, and sign-scaled *tanh*. The result demonstrated that although differentiable scaled *tanh* is popular in recent discrete feature learning literature, a huge performance drop occurs when outputs of scaled *tanh* are forced to be binary.

CCS CONCEPTS

• **Information systems** → *Top-k retrieval in databases*; **Collaborative filtering**.

KEYWORDS

recommender systems, binary hashing, cross-modal retrieval

ACM Reference Format:
Yang Shi and Young-joo Chung. 2022. Learning Similarity Preserving Binary Codes for Recommender Systems. In *Proceedings of August 15, 2022 (AdKDD '22).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural recommender systems have dramatically improved the recommendation performance in recent years. However, it is difficult to scale them up due to their high computational cost [13]. To overcome this issue, hashing-based recommender systems have been widely studied. The binarized user and item representations not only reduce the memory requirement but also accelerate the recommendation speed. For example, if items are represented by 256-dim double-precision float vectors, 10 million items' representations will take over 20 GB of storage space. With Hashing-based recommendation, it will only need 0.3GB[1].

The existing methods for the hashing-based RSs can be divided into three modules: feature extraction, interaction modeling, and binarization. The feature extraction module takes user's implicit or explicit feedback on items as inputs and learns representations by modeling their interactions (i.e., user/item preferences). After learning these real-valued representations, binarization methods are applied to convert them into binary codes. These binarizations can be done after learning real-valued representation [26, 27] (the two-step approach) or jointly done with feature learning [14, 24] (the direct approach). The final recommendation will be conducted by measuring the distances of the codes in Hamming space. Figure 1

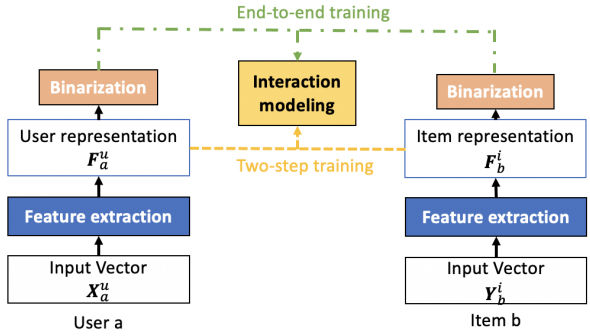

Figure 1: Hashing-based recommendation systems. Prediction is made by computing Hamming distances between binary codes.

Table 1: Comparison of different hashing-based recommender systems in terms of feature extraction, interaction modeling, and binarization

Paper	Loss function						Binarization
	Feature extraction			User-item interaction			
	MF	AE	Other NN	Dot product	CE	MAP	
CFCodeReg[27]	✓			✓			*Sign*
DCF[23]	✓			✓			*Sign*
NBR[25]	✓	✓		✓			LPR
NeuHash[8]	✓	✓		✓			STE
HashGNN[19]			✓		✓		*Sign*, STE
CCSR (ours)		✓				✓	*Sign*, Scaled *tanh*

[1]If we use double-precision float vectors for representations, we need 2,048 bytes (64bits * 256dims / 8) for one representation. If we use binary values, we need 32 bytes (1bit * 256dims / 8).

AdKDD '22, Washington, D.C., USA,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

shows the general framework for the hashing-based recommender system.

With the success of Collaborative Filtering (CF) that creates the user/item representations based on their interactions, earlier works focused on combining hashing and CF [12, 27]. These approaches obtain real-valued user/item features using Matrix Factorization (MF) and then convert them into binary codes using Linear Programming Relaxation (LPR).Recently, several methods have been proposed to integrate neural networks and hashing for recommender systems [15, 17]. These methods utilize neural networks (e.g., autoencoders, graph neural networks) to obtain real-valued user/item representations and binarize them by a hard threshold (*sign*) operator or an approximation (scaled *tanh*) function or straight-through-estimator (STE). Users' preferences on items are usually modeled with dot-product similarity, cross-entropy (CE) loss or rank loss.

Table 1 shows that previous works have different design choices on each module and certain combinations have not been explored. In this paper, we study an unexplored combination; we learn real-valued user/item representation through autoencoders and MAP-based similarity, and obtain binary codes for these user/item representations. We call our method **C**ompact **C**ross-**S**imilarity **R**ecommender (CCSR). Modeling entities with MAP-based similarity has shown its effectiveness in cross-modal retrieval (CR), where the goal of the system is retrieving similar entities from different modalities (e.g., image/text, audio/text)[3]. The recommendation task can be considered as cross-modal retrieval where the user space is one modality and the item space is the other. From this viewpoint, user/item representations are created by mapping them to the shared latent space while preserving the original similarities (i.e., preference). In the recommendation time, we retrieve the k-most similar items to users by comparing user/item similarity in this shared space.

Our contributions are as follows:

- We categorize the previous hash-based RS methods based on their design choices and explore a new design, CCSR, which is inspired by cross-modal retrieval literature.
- We demonstrate that MAP-based similarity loss is better than MF-based rating reconstruction loss in the top-k recommendation task. This is because the former loss emphasizes more on similarity learning which is important for the recommendation task. The latter focuses on rating reconstruction which is an indirect approach to recommendation.
- CCSR model outperforms hashing-based models with Matrix-factorization by up to 15.69%, 1.53% and 1.69% NDCG absolute improvements on Movielens1M, Amazon and Ichiba datasets respectively.
- We show that different models prefer different binarization methods. The simple *sign* function still performs well compared to other more complicated methods. Even though differentiable scaled *tanh* is popular in recent discrete feature learning literature, a huge performance drop occurred when scaled *tanh* outputs are forced to be binary.

2 RELATED WORK

Hashing-based CF Collaborative filtering utilizes observed user-item interaction (e.g., ratings, clicks, and purchases) to estimate unobserved interactions. Earlier works on hashing-based CF utilized MF and a two-stage binarization [26, 27]. They first learned real-valued representations through MF and then converted them to binary codes. Compositional Coding for Collaborative Filtering (CCCF)[14] and Discrete Collaborative Filtering (DCF) [23] proposed learning binary codes directly using discrete bit-by-bit optimization.

Hashing-based neural CF Recent hashing-based CF methods utilize neural networks to extract user/item features (representation). Autoencoders (AEs) [22], Variational Autoencoders (VAEs)[1, 8], and Graph neural networks were used for feature extraction[19, 21]. In terms of interaction modeling, cosine similarity loss[22] and rank loss[15] were studied in addition to traditional rating reconstruction loss. Several binarization techniques were also introduced. HashNet [4] used a scaled *tanh*function which forces real-valued features close to +1 and −1 during training. Others used straight-through-estimator (STE) to solve the discrete optimization problem directly[7, 16, 19].

Hashing-based similarity-preserving cross-modal retrieval Creating compact codes for cross-modal retrieval has been intensively explored to satisfy the needs from the massive growth of multi-modal data[2]. The state-of-the-art compact cross-modal retrieval methods utilize neural networks to extract real-valued features and convert them to binary codes use binarizations such as *sign* and scaled *tanh* functions [3, 5, 11, 18]. The main focus of these methods is defining better objective functions that can express similarities between entities from different modalities.

3 NOTATIONS

Assume we have m users, n items, and an implicit feedback matrix $\mathbf{S} \in \mathbb{R}^{m \times n}$ which contains binary values 0 or 1. $\mathbf{S}_{ab} = 1$ means User a likes Item b, $\mathbf{S}_{ab} = 0$ means User a dislikes Item b or has an unknown preference about Item b. Note that an explicit feedback matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ can be converted to $\mathbf{S}$ by setting a threshold (i.e., if $\mathbf{R}_{ab} > 3$, $\mathbf{S}_{ab} = 1$, 0 otherwise). Given $\mathbf{S}$, we want to learn real-valued feature matrices $\mathbf{F}^u \in \mathbb{R}^{m \times r}$, $\mathbf{F}^i \in \mathbb{R}^{n \times r}$, and binary feature matrices $\mathbf{H}^u \in \{-1, +1\}^{m \times r}$, $\mathbf{H}^i \in \{-1, +1\}^{n \times r}$ for users and items. $\mathbf{F}^u_a$ and $\mathbf{F}^i_b$ are User a's and Item b's continuous features (i.e., real-valued representations) resepectively. All users and items are represented by r-dimensional vectors.

4 COMPACT CROSS-SIMILARITY RECOMMENDATION

In this section, we detail the modules and the final objective function of CCSR.

4.1 Feature Extraction

We use autoencoders to learn continuous features and convert them into binary codes, following [8, 22]. We do not use multilayer perceptron as previous literature showed that autoencoders converge faster than multilayer perceptrons [22]. We construct two autoencoders, one for users and the other for items. The inputs are user and item low-level features. It can be purchase history, ratings, and side information such as user's demographics or item titles. Here we use a rating matrix as an input: $\mathbf{X} = \mathbf{R}$ for user, and $\mathbf{Y} = \mathbf{R}^\top$ for items. The autoencoders convert the original input $\mathbf{X}_a$

and $\mathbf{Y}_b$ into low-dimensional representations $\mathbf{F}_a^u$ and $\mathbf{F}_b^i$, and produce reconstructed input $\hat{\mathbf{X}}_a$ and $\hat{\mathbf{Y}}_b$ from these low-dimensional representations. The loss function for the autoencoders is:

$$\mathcal{L}_{ae} = \sum_{a,b}(||\mathbf{X}_a - \hat{\mathbf{X}}_a||_F^2 + ||\mathbf{Y}_b - \hat{\mathbf{Y}}_b||_F^2), \tag{1}$$

4.2 User-item Interaction

The user-item interaction loss function of CCSR consists of two parts: 1) similarity loss and 2) balance loss. Let's assume that we have two data points a, b that come from different modalities (i.e., users and items). They have continuous features $\mathbf{F}_a^u$, $\mathbf{F}_b^i$, as shown in Figure 1. We define similarity label $\mathbf{S}_{ab}$. $\mathbf{S}_{ab} = 1$ implies a, b are similar, whereas $\mathbf{S}_{ab} = 0$ implies they are dissimilar.

Similarity loss To measure the similarity between entities from different modalities, we adopt the Maximum a Posteriori (MAP) estimation. The logarithm MAP is:

$$\log p(\mathbf{F}_a^u, \mathbf{F}_b^i|\mathbf{S}_{ab}) \propto \log p(\mathbf{S}_{ab}|\mathbf{F}_a^u, \mathbf{F}_b^i)p(\mathbf{F}_a^u)p(\mathbf{F}_b^i) \tag{2}$$

The conditional likelihood for similarity label S_{ab} in Equation 2 is: $p(\mathbf{S}_{ab}|\mathbf{F}_a^u, \mathbf{F}_b^i) = \sigma(\langle \mathbf{F}_a^u, \mathbf{F}_b^i \rangle)^{\mathbf{S}_{ab}}(1 - \sigma(\langle \mathbf{F}_a^u, \mathbf{F}_b^i \rangle))^{1-\mathbf{S}_{ab}}$, where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, $\langle\rangle$ is the inner product operator. Assuming the priors for $\mathbf{F}_a^u$ and $\mathbf{F}_b^i$ are known and follow Gaussian distribution, By maximizing Equation 2 , we obtain the cross-entropy loss for MAP-based similarity:

$$\mathcal{L}_{sim} = \sum_{a,b}(\log(1 + e^{\langle \mathbf{F}_a^u, \mathbf{F}_b^i \rangle}) - \mathbf{S}_{ab}\langle \mathbf{F}_a^u, \mathbf{F}_b^i \rangle). \tag{3}$$

Note that previous recommender system models estimate the ratings using dot-product similarity between users and items. Therefore, their loss function is as follows: $\mathcal{L}_{sim}^{'} = \sum_{a,b}(\mathbf{R}_{ab} - \langle \mathbf{F}_a^u, \mathbf{F}_b^i \rangle)^2$.

Balance loss To ensure we use the bit information maximally, we utilize a balance loss to balances the number of +1 and −1 in the binary code, which is a widely used technique in binary code learning [6, 20]. Since it is difficult to directly optimize on binary codes, we apply the balance loss to the continuous features: $\mathcal{L}_b = \sum_{a,b}(||\mathbf{F}_a^{u\top}\mathbf{1}||_F^2 + ||\mathbf{F}_b^{i\top}\mathbf{1}||_F^2)$ where $\mathbf{1}$ is a vector of 1s.

As a result, the loss function of CCSR is defined as:

$$\mathcal{L} = \mathcal{L}_{sim} + \lambda_b\mathcal{L}_b + \lambda_{ae}\mathcal{L}_{ae} \tag{4}$$

where λ_{ae} and λ_b are the hyper-parameters to balance between the losses.

4.3 Binarization

In the previous section, we optimize the model over continuous features. The binarization can be done by applying a *sign* function to the continuous feature:

$$h = sgn(f) = \begin{cases} 1 & f \geq 0 \\ -1 & f < 0, \end{cases} \tag{5}$$

which is a two-stage binarization. In [4, 18], authors replaced Eq. 5 with a scaled $tanh$ function: $h = tanh(\alpha f)$. Starting with $\alpha = 1$, the method increases α exponentially per training epoch so that eventually $tanh(\alpha f) \approx sgn(f)$. This replacement aim to learnining binary codes by directly optimizing the neural networks (i.e., end-to-end binarization). However, this approximation still outputs *continuous* values. We apply a *sign* function to output of $tanh(\alpha f)$ to get true binary codes. We call this method sign-scaled $tanh$.

Table 2: Details of the datasets

Dataset	#User	#Item	#Ratings	Density
Movielens1M	6,040	3,952	1,000,209	4.19%
Amazon	35,736	38,121	1,960,674	0.14%
Ichiba	36,314	8,514	1,267,296	0.41%

5 EXPERIMENTS

In this section, we perform experiments to answer the following questions: **Q1** What is the performance difference when using MF-based and Cross-modal retrieval-based (CR-based) similarity measurements? **Q2** What is the performance difference between different binarization methods: sign (S), scaled $tanh$ (ST), and sign-scaled $tanh$ (SST)? **Q3** When similarity loss is more effective than rating reconstruction-based loss?

5.1 Dataset

We use three datasets for the experiments: Movielens1M [9], Amazon (books) dataset [10] and a real-word e-commerce (Ichiba[2]) purchase dataset. Movelens1M and Amazon datasets contain explicit feedback such as movie and book ratings from users. Ichiba dataset contains purchase histories which are implicit feedback.

Pre-processing We utilize Movielens1M without any user/item filtering. Following previous work [8], for Amazon and Ichiba datasets, we remove users and items with insufficient interactions.To generate the similarity matrix, ratings greater than 3 are converted to 1, and the rest to 0.

Training/test split For Movielens1M, we followed previous work [27]. *For each user*, randomly select 80% ratings for training and 20% for the test. For Amazon and Ichiba, since they are extremely sparse, *for all ratings*, we randomly select 80% for training and 20% for the test. A detailed datasets summary is listed in Table 2.

5.2 Models

We compared the recommendation performance of the following models:

(1) **Random** We randomly select items from all items and recommend them to each test user.
(2) **Top** We select the most frequently rated items from training data and recommend them to all test users.
(3) **CF-S** Matrix Factorization method based on Stochastic Gradient Descend with loss function: $\sum_{a,b}(\mathbf{R}_{ab} - < \mathbf{F}_a^u, \mathbf{F}_b^i >)^2 + \lambda||\mathbf{F}^u||_F^2 + \lambda||\mathbf{F}^i||_F^2$. Continuous features are converted to binary codes using *sign* function.
(4) **CFcodeReg** [27] This method first solves a relaxed optimization problem where features can be continuous and between -1 and 1. The loss function is: $\sum_{a,b}(\mathbf{R}_{ab} - \frac{1}{2} - \frac{1}{2r} < \mathbf{F}_a^u, \mathbf{F}_b^i >)^2 + \lambda||\mathbf{F}^u||_F^2 + \lambda||\mathbf{F}^i||_F^2$,. All features are rounded to the closest binary code with median threshold.

[2]https://www.rakuten.co.jp

Table 3: NDCG@k of different models on Movielens1M. Binary code lengths are set to 5,10,20, and 40. We highlighted the best results in each binarization method (ST, SST, and S).

Models	@2				@6				@10			
	5	10	20	40	5	10	20	40	5	10	20	40
Random	0.0094				0.0091				0.0083			
Top	0.0				0.3101				0.3196			
AECF-C	0.6496	0.7093	0.6750	0.6694	0.7072	0.7358	0.7129	0.7100	0.7420	0.7603	0.7424	0.7393
CCSR-C	0.6861	0.7010	0.6901	0.7246	0.7296	0.7344	0.7255	0.7605	0.7538	0.7605	0.7516	0.7885
AECF-ST	0.6201	0.6862	**0.6769**	**0.6918**	0.6762	0.7220	**0.7180**	**0.7246**	0.7095	0.7495	**0.7458**	**0.7518**
DJSRH-ST	0.5661	0.6020	0.5974	0.5879	0.6274	0.6533	0.6481	0.6409	0.6673	0.6886	0.6845	0.6787
CCSR-ST	**0.6820**	**0.6913**	0.6441	0.6610	**0.7226**	**0.7295**	0.6841	0.7016	**0.7470**	**0.7542**	0.7115	0.7291
AECF-SST	0.5637	**0.6798**	**0.6883**	0.5404	0.6233	**0.7201**	**0.7265**	0.6032	0.6663	**0.7478**	**0.7526**	0.6445
DJSRH-SST	0.5676	0.5825	0.5895	0.5859	0.6319	0.6388	0.6458	0.6446	0.6703	0.6776	0.6835	0.6814
CCSR-SST	**0.6492**	0.6668	0.6587	**0.6609**	**0.6930**	0.7077	0.7108	**0.7138**	**0.7238**	0.7384	0.7438	**0.7486**
CF- S	0.5492	0.5599	0.5672	0.5833	0.6172	0.6198	0.6297	0.6450	0.6593	0.6613	0.6698	0.6840
CFCodeReg	0.5692	0.5690	0.5738	0.5728	0.6314	0.6303	0.6323	0.6317	0.6712	0.6701	0.6724	0.6711
AECF-S	0.4983	0.5555	0.5310	0.4874	0.5817	0.6154	0.5989	0.5689	0.6311	0.6534	0.6423	0.6175
CCSR-S	**0.6332**	**0.6523**	**0.6912**	**0.7402**	**0.6997**	**0.7128**	**0.7277**	**0.7677**	**0.7371**	**0.7475**	**0.7529**	**0.7897**

(5) **AECF** We use autoencoders introduced in Section 4 with a rating reconstruction loss.The loss function is: $\sum_{a,b}((\mathbf{R}_{ab} - < \mathbf{F}_a^u, \mathbf{F}_b^i >)^2 + \lambda_{ae}\mathcal{L}_{ae}$. The number of layers and the size of layers are set after cross-validation. The hidden-layer sizes are 512, 256 and 128 for the three-layer encoder and 128, 256, and 512 for the decoder. Depending on the binarization methods, we have **AECF-S** with a *sign* function, **AECF-ST** with a scaled *tanh* function, and **AECF-SST** with a sign-scaled *tanh* function.

(6) **DJSRH [18]** Deep joint-semantics reconstructing hashing (DJSRH) is the state-of-the-art compact cross-modal retrieval model. It uses semantic loss in both single-modality and cross-modality for better performance. The features come from a MLP with two hidden layers whose sizes are 512 and 256. It utilizes an unsupervised similarity loss: $||\mu\mathbf{M} - \mathbf{F}^u\mathbf{F}^{i\top}||_{\mathbf{F}}^2$, where M is a joint semantics affinity matrix that includes both single and cross modality similarities ($\mathbf{M} = (1-\eta)\mathbf{N} + \eta\mathbf{N}\mathbf{N}^\top, \mathbf{N} = \beta\mathbf{X}\mathbf{X}^\top + (1-\beta)\mathbf{Y}\mathbf{Y}^\top$). η and β are parameters to balance between different terms. We have **DJSRH-ST** and **DJSRH-SST**.

(7) **CCSR** Our proposed method. The loss function is Equation 4. We choose a one-layer encoder/decoder using cross-validation. The size of the hidden-layer is 128 for the encoder/decoder. We also investigate different binarization methods: **CCSR-S**, **CCSR-ST**, and **CCSR-SST**.

The first two models are rule-based. The following four methods are MF-based and the last two are CR-based. Besides binarized AECF and CCSR, we also performed the AECF and CCSR with continuous-values, noted as **AECF-C** and **CCSR-C**, for reference.

We trained and tuned DJSRH using the code provided by the authors[3]. We carefully implemented and tuned the rest of the models. In CF and CFcodeReg, we set $\lambda = 0.4$. In AECF and CCSR models, we cross-validated the hyper parameters and finally set $\lambda_{ae} = 0.1$, $\lambda_b = 0.0001$ for Movielens1M and $\lambda_{ae} = 10$, $\lambda_b = 0.0001$ for Amazon and Ichiba. We added one dropout layer before last encoder layer in both AECF and CCSR. We set the dimension of feature vectors (i.e., the binary code length) to 5, 10, 20, and 40. Dropout rate was set at 0.6 for code length 5, 10 and 20 experiments, and 0.8 for code length 40 through cross-validation.

[3]https://github.com/zzs1994/DJSRH

5.3 Evaluation Metrics

We use Normalized Discounted Cumulative Gain (NDCG)@k as the evaluation matrices. We recommend the top-k items to a query user. The top-k items are ranked based on the Hamming distance between binary codes of a query user and the items in the database.

5.4 CR-based Recommender v.s. MF-based Recommender (Q1)

In Table 3, 4, and 5 we show NDCG results on Movielens1M, Amazon, and Ichiba. CCSR-S models achieve the best performance across different binary code length in all models with *sign* binarization on Movielens1M. It has an average 15.21% NDCG relative improvement compared to the best MF-based models. On Amazon dataset, CCSR-S performed as good as the best MF-based model and CCSR-ST outperforms all MF-based models. On the Ichiba purchase dataset, the CR-based model DJSRH performs best, followed by the CCSR model. We conclude that CR-based similarity loss is better than MF-based loss in the top-k recommendation task. This is because the former loss emphasizes more on similarity learning which is more important for the recommendation task. The latter focuses on rating reconstruction which is an indirect approach to the task and may cause over-fitting. While performance still increases when we increase feature size from 20 to 40 in CCSR models, AECF models suffer from performance drop. We present a further analysis in Section 5.6.

Among MF-based models, AECF achieves the best performance because it uses neural feature extractors. *Among CR-based models*, CCSR performs better than DJSRH on Movielens1M. This is reasonable since unsupervised similarity measurements in DJSRH are less

Table 4: NDCG@k of different models on Amazon

Models	@2				@6				@10			
	5	10	20	40	5	10	20	40	5	10	20	40
Random	0.0003				0.0003				0.0001			
Top	0.0132				0.0327				0.0485			
AECF-C	0.7919	0.7915	0.7944	0.7930	0.8563	0.8558	0.8578	0.8566	0.8822	0.8816	0.8834	0.8824
CCSR-C	0.7875	0.7883	0.7880	0.7895	0.8526	0.8522	0.8526	0.8539	0.8791	0.8791	0.8792	0.8801
AECF-ST	0.7800	0.7716	0.7810	0.7835	0.8471	0.8419	0.8478	0.8489	0.8748	0.8705	0.8752	0.8765
DJSRH-ST	0.7460	0.7653	0.7707	0.7792	0.8263	0.8381	0.8412	0.8469	0.8579	0.8673	0.8701	0.8744
CCSR-ST	**0.7833**	**0.7869**	**0.7837**	**0.7870**	**0.8490**	**0.8521**	**0.8503**	**0.8530**	**0.8762**	**0.8786**	**0.8770**	**0.8792**
AECF-SST	**0.7657**	0.7500	**0.7817**	**0.7733**	**0.8377**	0.8288	**0.8485**	**0.8432**	**0.8670**	0.8599	**0.8759**	**0.8714**
DJSRH-SST	0.7610	0.7617	0.7627	0.7698	0.8356	**0.8372**	0.8379	0.8421	0.8654	**0.8668**	0.8676	0.8709
CCSR-SST	0.7632	**0.7633**	0.7518	0.7530	0.8355	0.8362	0.8285	0.8306	0.8657	0.8658	0.8598	0.8614
CF-S	0.7661	0.7673	0.7680	0.7691	0.8399	0.8409	0.8422	0.8428	0.8692	0.8700	0.8711	0.8716
CFCodeReg	0.7657	0.7650	0.7653	0.7663	0.8398	0.8400	0.8405	0.8413	0.8692	0.8693	0.8698	0.8704
AECF-S	0.7703	**0.7755**	**0.7818**	**0.7793**	0.8411	**0.8447**	**0.8481**	**0.8477**	0.8697	**0.8726**	**0.8756**	**0.8750**
CCSR-S	**0.7707**	0.7685	0.7648	0.7752	**0.8420**	0.8408	0.8381	0.8446	**0.8705**	0.8697	0.8676	0.8725

Table 5: NDCG@k of different models on Ichiba

Models	@2				@6				@10			
	5	10	20	40	5	10	20	40	5	10	20	40
Random	0.0010				0.0013				0.0009			
Top	0.0558				0.1113				0.1550			
AECF-C	0.9180	0.9233	0.9199	0.9168	0.9508	0.9537	0.9506	0.9479	0.9606	0.9628	0.9604	0.9581
CCSR-C	0.9216	0.9236	0.9196	0.9293	0.9482	0.9567	0.9561	0.9531	0.9593	0.9607	0.9677	0.9625
AECF-ST	0.9062	0.9141	0.8955	0.9108	0.9386	0.9473	0.9387	0.9474	0.9512	0.9574	0.9513	0.9578
DJSRH-ST	0.8730	**0.9202**	**0.9406**	**0.9656**	0.9196	**0.9491**	**0.9593**	**0.9730**	0.9385	**0.9597**	**0.9674**	**0.9784**
CCSR-ST	**0.9095**	0.9166	0.9124	0.9227	**0.9437**	0.9487	0.9463	0.9526	**0.9551**	0.9588	0.9572	0.9617
AECF-SST	0.9020	0.8951	0.9016	0.9072	0.9377	0.9389	0.9365	0.9398	0.9510	0.9511	0.9500	0.9523
DJSRH-SST	0.8954	**0.9119**	**0.9188**	**0.9434**	0.9365	**0.9464**	**0.9497**	**0.9629**	0.9500	**0.9577**	**0.9602**	**0.9705**
CCSR-SST	**0.9145**	0.9066	0.8965	0.8695	**0.9449**	0.9384	0.9317	0.9187	**0.9562**	0.9512	0.9464	0.9369
CF-S	0.8915	0.8921	0.8927	0.8956	0.9329	0.9339	0.9338	0.9352	0.9475	0.9482	0.9483	0.9494
CFCodeReg	0.8913	0.8911	0.8875	0.8856	0.9327	0.9322	0.9307	0.9300	0.9475	0.9470	0.9457	0.9452
AECF-S	0.9150	0.9102	**0.9123**	0.9123	0.9454	0.9416	**0.9430**	0.9426	0.9566	0.9532	**0.9547**	0.9542
CCSR-S	**0.9169**	**0.9104**	0.9026	**0.9158**	**0.9467**	**0.9432**	0.9384	**0.9478**	**0.9576**	**0.9548**	0.9513	**0.9583**

powerful than supervised measurements when data density is relatively high. However, on Ichiba dataset, DJSRH achieves better performances. Besides the effects of low data density, the low diversity in input features (implicit) in Ichiba also makes autoencoder-based CCSR less competitive.

5.5 Different Binarization Methods (Q2)

Recent papers prefer the scaled *tanh* (ST) function to *sign* (S) function because ST can be computed in the back-propagation step and learn binary codes in the end-to-end fashion. However, the output of ST is not binary code, so it is unfair to compare the performance of ST with S or SST whose output is binary. *We find the performance drops drastically when we switch from ST to SST.* In Table 3, for each method, NDCG value decreases more than 1% when the output of ST was forced to be binary (SST). We explain the possible reasons in Figure 2. In summary, the performance change is because of the limited representation power of binary codes.

The drop by SST in AECF and CCSR is higher than the drop in DJSRH. This might be due to the different focuses in the similarity loss; DJSRH compares the similarities in a batch (rank loss between different users and items), so it learns robust codes. AECF and CCSR only consider pair-wise loss (only between one user and one item). Thus, AECF and CCSR are more likely to be affected by the precision drop of feature values.

5.6 Similarity loss v.s. Rating reconstruction loss (Q3)

In this section, we investigate when similarity loss works better for recommendation tasks. To remove the effect of binary loss, we investigate the result of AECF-C and CCSR-C models. We separate test users into two groups: the first group that obtained better results

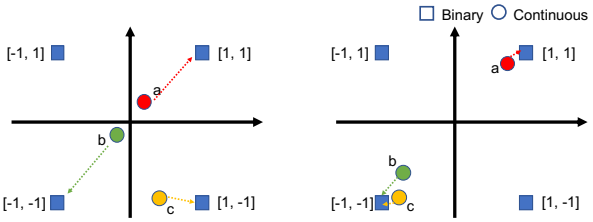

Figure 2: A toy example of converting continuous features to various binary features (ST and SST). Left: general limitations of binary features. Even though a is more similar to b than to c in continuous space, after being converted to binary codes, a is more similar to c than to b when comparing Hamming distance. Right: limitations of SST: In ST models, continuous features are close to +1 and -1, a is more similar to b than to c. But b and c are equally similar to a after converting use SST.

Table 6: Chi-squared statistics of user's characteristics in Movielens1M.* indicates the value is statistically significant ($p \leq 0.05$)

code length	# of ratings	avg. ratings	std. ratings
5	47.41*	0.26	0.02
10	5.93*	0.05	0.08
20	2.37	0.33	0.53
40	8.96*	0.03	0.11

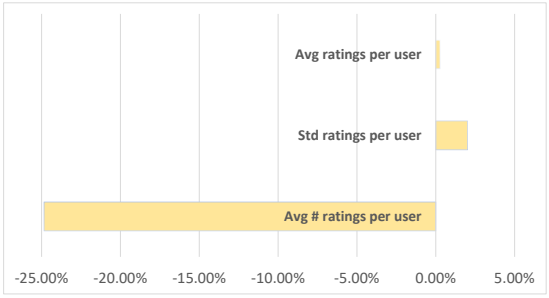

Figure 3: Relative differences of the three characteristics between two user groups with code length 40 on Movielens1M.

in NDCG@10 with CCSR-C, and the second group with the better results with AECF-C. Then, we computed Chi-squared statistics of user characteristics for classifying these user groups and listed in Table 6. We can see the number of ratings is the most important feature for classification. We further compared the ratio of the raw values of features in both groups in Figure 3 and conclude that *CCSR is helpful for users who rated less items and with higher rating variances*. This makes sense since: (1) AECF benefits more from more ratings as it tries to reconstruct original ratings to learn the representations. (2) with higher rating variance, CR-based models learn better representations using similar and dissimilar pairs.

6 CONCLUSION

In this paper, we proposed a new hashing-based RS, Compact Cross-Similarity Recommender. To the best of our knowledge, this is the first work that builds efficient recommender systems from the viewpoint of compact neural cross-modal retrieval. From extensive studies of several large-scale datasets, we observed the performance changes on different datasets while using the same model. It suggested us using different models based on data sparsity and data types. We also studied different binarization methods and discovered that scaled *tanh* suffered from the performance drop when its codes were converted to the binary.

REFERENCES

[1] Jan Van Balen and Mark Levy. 2019. PQ-VAE: Efficient Recommendation Using Quantized Embeddings. In *ACM RecSys 2019 Late-breaking Results*.
[2] Michael M. Bronstein, Alexander M. Bronstein, Fabrice Michel, and Nikos Paragios. 2010. Data fusion through cross-modality metric learning using similarity-sensitive hashing. In *CVPR'10*. 3594–3601.
[3] Yue Cao, Bin Liu, Mingsheng Long, and Jianmin Wang. 2018. Cross-Modal Hamming Hashing. In *The European Conference on Computer Vision (ECCV)*.
[4] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. 2017. HashNet: Deep learning to hash by continuation.. In *ICCV*. 5608–5617.
[5] Yong Chen, Yuqing Hou, Shu Leng, Qing Zhang, Zhouchen Lin, and Dell Zhang. 2021. Long-Tail Hashing. In *SIGIR '21*. 1328–1338.
[6] Guiguang Ding, Yuchen Guo, and Jile Zhou. 2014. Collective Matrix Factorization Hashing for Multimodal Data. In *CVPR'14*. 2083–2090.
[7] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised Neural Generative Semantic Hashing. In *SIGIR'19*.
[8] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Content-aware Neural Hashing for Cold-start Recommendation. In *SIGIR'20*. 971–980.
[9] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. In *ACM Transactions on Interactive Intelligent Systems (TiiS)*.
[10] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *ACM Conference on World Wide Web (WWW'16)*.
[11] Qing-Yuan Jiang and Wu-Jun Li. 2017. Deep Cross-Modal Hashing. In *CVPR'17*. 3232–3240.
[12] Alexandros Karatzoglou, Alex Smola, and Markus Weimer. 2010. Collaborative Filtering on a Budget. In *AISTATS*. 389–396.
[13] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. LightRec: A Memory and Search-Efficient Recommender System.
[14] Chenghao Liu, Tao Lu, Xin Wang, Zhiyong Cheng, Jianling Sun, and Steven C.H. Hoi. 2019. Compositional Coding for Collaborative Filtering. In *SIGIR'19*.
[15] Zhi Lu, Yang Hu, Yunchao Jiang, Yan Chen, and Bing Zeng. 2019. Learning Binary Code for Personalized Fashion Recommendation. In *CVPR*.
[16] Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Lawrence Carin, and Ricardo Henao. 2018. NASH: Toward End-to-End Neural Architecture for Generative Semantic Hashing. In *ACL*.
[17] Shaoyun Shi, Weizhi Ma, Min Zhang, Yongfeng Zhang, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2020. Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation. In *SIGIR'20*. 319–328.
[18] Shupeng Su, Zhisheng Zhong, and Chao Zhang. 2019. Deep Joint-Semantics Reconstructing Hashing for Large-Scale Unsupervised Cross-Modal Retrieval. In *ICCV'19*.
[19] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In *WWW'20*. 1988–1998.
[20] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral Hashing. In *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), Vol. 21. 1753–1760.
[21] Wei Wu, Bin Li, Chuan Luo, and Wolfgang Nejdl. 2021. Hashing-Accelerated Graph Neural Networks for Link Prediction. In *WWW '21*. 2910–2920.
[22] Zhenghua Xu, Thomas Lukasiewicz, Cheng Chen, Yishu Miao, and Xiangwu Meng. 2017. Tag-Aware Personalized Recommendation Using a Hybrid Deep Model. In *IJCAI*.
[23] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete Collaborative Filtering *(SIGIR'16)*. 325–334.
[24] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete Personalized Ranking for Fast Collaborative Filtering from Implicit Feedback. In *AAAI*. 1669–1675.
[25] Y. Zhang, J. Wu, and H. Wang. 2019. Neural Binary Representation Learning for Large-Scale Collaborative Filtering. *IEEE Access* 7 (2019), 60752–60763.
[26] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference Preserving Hashing for Efficient Recommendation *(SIGIR'14)*. 183–192.
[27] Ke Zhou and Hongyuan Zha. 2012. Learning Binary Codes for Collaborative Filtering *(KDD'12)*. 498–506.