DaisyRec 2.0: Benchmarking Recommendation for Rigorous Evaluation

Zhu Sun, Hui Fang, Jie Yang, Xinghua Qu, Hongyang Liu, Di Yu,
Yew-Soon Ong, *Fellow, IEEE,* and Jie Zhang

Abstract—Recently, one critical issue looms large in the field of recommender systems – there are no effective benchmarks for rigorous evaluation – which consequently leads to unreproducible evaluation and unfair comparison. We, therefore, conduct studies from the perspectives of practical theory and experiments, aiming at benchmarking recommendation for rigorous evaluation. Regarding the theoretical study, a series of hyper-factors affecting recommendation performance throughout the whole evaluation chain are systematically summarized and analyzed via an exhaustive review on 141 papers published at eight top-tier conferences within 2017-2020. We then classify them into model-independent and model-dependent hyper-factors, and different modes of rigorous evaluation are defined and discussed in-depth accordingly. For the experimental study, we release DaisyRec 2.0 library by integrating these hyper-factors to perform rigorous evaluation, whereby a holistic empirical study is conducted to unveil the impacts of different hyper-factors on recommendation performance. Supported by the theoretical and experimental studies, we finally create benchmarks for rigorous evaluation by proposing standardized procedures and providing performance of ten state-of-the-arts across six evaluation metrics on six datasets as a reference for later study. Overall, our work sheds light on the issues in recommendation evaluation, provides potential solutions for rigorous evaluation, and lays foundation for further investigation.

Index Terms—Recommender Systems, Reproducible Evaluation, Fair Comparison, Benchmarks, Standardized Procedures

1 Introduction

With the advent of the big data era, we are flooded by the exponentially increased information on the Internet. To ease the severe information overload problem [1], recommender systems have been extensively studied in academia and widely applied in industry across different domains, such as e-commerce (e.g., Amazon, Tmall), location-based social networks (e.g., Foursquare, Yelp), multi-media (e.g., Netflix, Spotify), and so forth. With a massive amount of recommendation approaches being proposed, one critical issue has attracted much attention from researchers in the field of recommender systems: there are few effective benchmarks for evaluation [2], [3], [4], which, consequently, leads to unreproducible evaluation and unfair comparison. As indicated by the recent study [5], results for baselines that have been used in numerous publications over the past five years are suboptimal; with a careful setup, the baselines even outperform the reported results of any newly proposed method. This is in alignment with another latest study [6], which discovers that the recent proposed deep learning models (DLMs) can be defeated by comparably simple baselines, such as MostPop and ItemKNN [7] with fine-tuned parameters. These findings initiate an extremely heated discussion on the evaluation of recommendation methods and inspire us to deeply consider the underlying barriers that hinder the rigorous evaluation in recommendation.

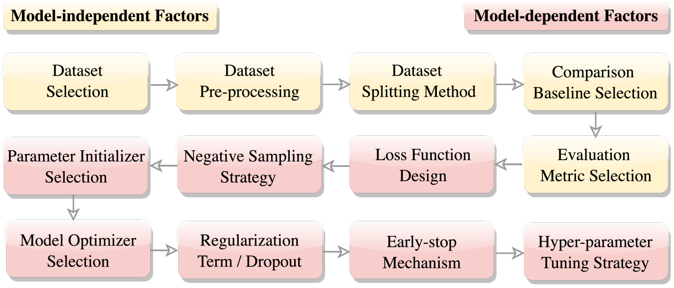

Model-independent Factors
Model-dependent Factors
Dataset Selection
Dataset Pre-processing
Dataset Splitting Method
Comparison Baseline Selection
Evaluation Metric Selection
Loss Function Design
Negative Sampling Strategy
Parameter Initializer Selection
Model Optimizer Selection
Regularization Term / Dropout
Early-stop Mechanism
Hyper-parameter Tuning Strategy

Fig. 1. Hyper-factors within the whole recommendation evaluation chain.

As a matter of fact, there are a number of *hyper-factors* which may affect the recommendation performance throughout the whole evaluation chain, and their best settings are unknown. They can be broadly classified into two types as depicted in Figure 1, namely model-independent and model-dependent hyper-factors. The former refers to the hyper-factors that are isloated from the model design and optimization process (e.g., dataset and comparison baseline selection); whilst the latter indicates the ones involved in the model development and parameter optimization procedure (e.g., loss function design and regularization

- *Zhu Sun is with Institute of High Performance Computing and Centre for Frontier AI Research, A*STAR, Singapore. E-mail:sunzhuntu@gmail.com*
- *Hui Fang (corresponding author) is with Shanghai University of Finance and Economics, China. E-mail: fang.hui@mail.shufe.edu.cn*
- *Jie Yang is with Delft University of Technology, the Netherlands. E-mail: j.yang-3@tudelft.nl*
- *Xinghua Qu is with ByteDance AI Lab, Singapore. E-mail:quxinghua17@gmail.com*
- *Hongyang Liu is with Yanshan University, China. E-mail: hyliu767289@gmail.com*
- *Di Yu is with Singapore Management University, Singapore. E-mail: yudi201909@gmail.com*
- *Yew-Soon Ong is with Nanyang Technological University, Singapore. E-mail: asysong@ntu.edu.sg. He is also with A*STAR Centre for Frontier AI Research, Singapore. E-mail: Ong_Yew_Soon@hq.a-star.edu.sg*
- *Jie Zhang is with Nanyang Technological University, Singapore. E-mail: zhangj@ntu.edu.sg*

TABLE 1
Summary of the collected papers.

Venue	No.	Reference			
		2017	2018	2019	2020
AAAI	22	[8], [9]	[10], [11], [12]	[13], [14], [15], [16], [17], [18], [19], [20]	[21], [22], [23], [24], [25], [26], [27] [28], [29]
CIKM	19	[30], [31], [32]	[33], [34], [35]	[36], [37], [38]	[39], [40], [41], [42], [43], [44] [45], [46], [47], [48]
IJCAI	20	[49], [50], [51]	[52], [53], [54], [55], [56], [57]	[58], [59], [60], [61], [62], [63], [64]	[65], [66], [67], [68]
KDD	12	[69]	[70], [71], [72]	[73], [74], [75], [76], [77]	[78], [79], [80]
RecSys	14	[81], [82]	[83], [84], [85]	[86], [87], [88], [89], [90], [91]	[92], [93], [94]
SIGIR	22	[95]	[96], [97], [98], [99], [100]	[101], [102], [103], [104]	[105], [106], [107], [108], [109], [110] [111], [112], [113], [114], [115], [116]
WSDM	16	[117]	[118], [119], [120]	[121], [122], [123], [124]	[125], [126], [127], [128], [129], [130] [131], [132]
WWW	16	[133], [134]	[135], [136]	[137], [138], [139], [140], [141], [142]	[143], [144], [145], [146], [147], [148]
Total	141	15	28	43	55

terms). According to this categorization, three main aspects may inherently lead to such non-rigorous evaluation.

- **Diverse settings on model-independent hyper-factors**. With being prominent in different platforms, there are diverse recommendation datasets (i.e., dataset selection) in various application domains shown in Table 6 (Appendix). Taking the movie domain as an example, the datasets vary from MovieLens (ML), Netflix to Amazon-Movie, etc. Even for the same dataset, it may have different versions with different sizes covering different durations, such as, ML-100K/1M/10M/20M/25M/Lastest. Different researchers may choose different datasets across different domains based on their requirements, and only report results on their selected datasets; meanwhile different settings on other model-independent hyper-factors (e.g., dataset pre-processing and splitting strategies) may generate entirely different recommendation performance.
- **Diverse settings on model-dependent hyper-factors**. There are different choices for the model-dependent hyper-factors. For instance, the loss function could be either point-wise (square error loss [149] and cross-entropy loss [134]) or pair-wise (log loss [150], hinge loss [54] and top-1 loss [151]); different types of model optimizers are also available, ranging from stochastic gradient descent (SGD) to adaptive moment estimation (Adam). The recommendation results may vary a lot with different settings on these model-dependent factors even with fixed settings on model-independent ones.
- **Missing setting details**. Most importantly, a majority of papers do not report details on the settings of either model-independent and model-dependent hyper-factors, such as data processing and parameter settings, which increases the difficulty on evaluation and leads to inconsistent results in reproduction by different researchers, thus heavily aggravating the unreproducible evaluation and unfair comparison issues.

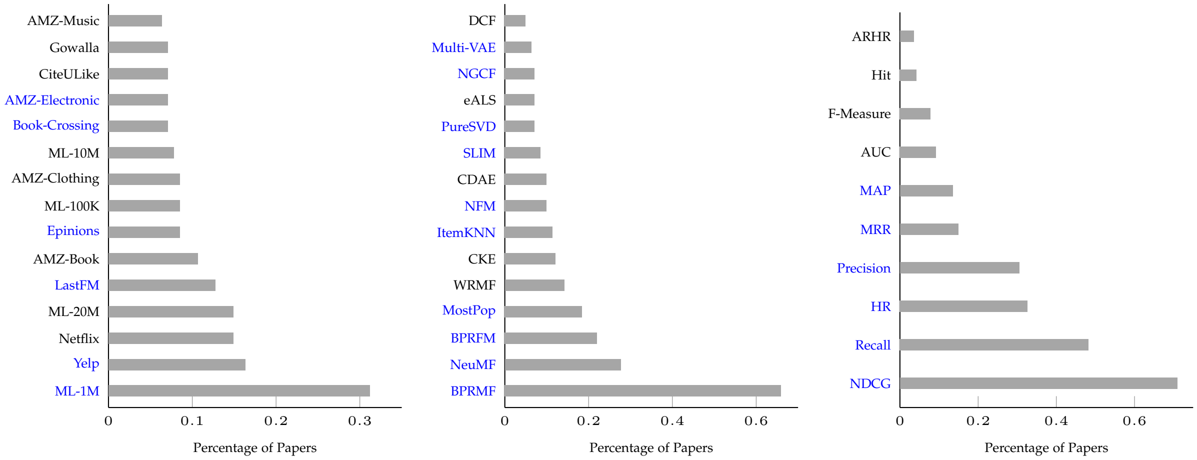
With the above issues in mind, we are seeking at benchmarking recommendation for rigorous (i.e., reproducible and fair) evaluation, thus helping achieve a healthy and sustainable growth of research in this area. Considering the diverse recommendation tasks (e.g., temporal, sequential, diversity, explanation, location, group and cross-domain aware recommendation), we first mainly focus on the **general top-N** recommendation task, which is one of the hottest and most prominent topics in recommendation. To this end, we conduct extensive studies from the perspectives of both practical theory and experiments.

- Regarding the theoretical study, we conduct an exhaustive review on 141 papers related to top-N recommendation published in the recent four years (2017-2020) on eight prestigious conferences as representatives, including KDD, SIGIR, WWW, IJCAI, AAAI, RecSys, WSDM and CIKM[1]. In doing so, we systematically extract and summarize hyper-factors throughout the whole evaluation chain, and classify them into model-independent and model-dependent factors in Figure 1. Accordingly, different modes (e.g., relax, strict and mixed) of rigorous evaluation are defined and discussed in-depth, acting as valuable guidance for later study.
- For the experimental study, we release a Python-based recommendation testbed – DaisyRec 2.0 to integrate the hyper-factors throughout the evaluation chain[2]. Our testbed advances existing libraries (e.g., LibRec [152], DeepRec [153] and RecBole [154]), which mainly aim to implement various state-of-the-art recommenders, in the light of performing rigorous evaluation in recommendation. Based on DaisyRec 2.0, a holistic empirical study has been performed to comprehensively analyze the impact of different hyper-factors on recommendation performance.

Supported by both theoretical and experimental study, we finally create benchmarks by proposing the standardized procedures to help enhance the reproducibility and fairness of evaluation. Meanwhile, the performance of ten well-tuned state-of-the-arts on six widely-used datasets across six metrics is provided as a reference for fair evaluation. Additionally, a number of interesting findings are noted throughout our study, for example, **(1)** the recommendation performance does not necessarily improve with denser datasets; **(2)** some non-deep learning based baselines, e.g., PureSVD [155] can achieve a better balance between recommendation accuracy and complexity; **(3)** the best hyper-parameter settings for one specific metric does not necessarily guarantee optimums w.r.t. other metrics; **(4)** although the objective function with pair-wise log loss generally outperforms others, different methods may have their best fit objective functions; **(5)** uniform sampler, though simple, performs better than the popularity based sampler; and **(6)** different parameter initializers and model optimizers can

[1] They are most important venues (full names see Table 7 in Appendix) to accept high-quality recommendation papers and other related conferences and journals will be considered in our future study.

[2] https://github.com/recsys-benchmark/DaisyRec-v2.0



(a) Distribution of Datasets (b) Distribution of Baselines (c) Distribution of Metrics

Fig. 2. (a) popularity of the top-15 datasets, where 'ML, AMZ' denote MovieLens and Amazon, respectively; (b) popularity of the top-15 baselines; and (c) popularity of the top-10 evaluation metrics. Note that the selected datasets, baselines and metrics in our study are highlighted in blue.

extensively affect the final recommendation accuracy. To sum up, our work sheds lights on the issues in evaluation for recommendation, provides potential solutions for rigorous evaluation, and paves the way for further investigation[3].

2 Practical Theory Study

2.1 Hyper-factor Extraction

As we seek to benchmark recommendation for rigorous evaluation, we first conduct study from the angle of practical theory by an exhaustive literature review, so as to extract and summarize hyper-factors affecting recommendation performance throughout the whole evaluation chain. In particular, we review papers published in the recent four years (2017-2020) on eight top tier conferences, namely, AAAI, CIKM, IJCAI, KDD, RecSys, SIGIR WSDM and WWW. As a starting point, we mainly focus on recommendation methods for implicit feedback based top-N recommendation, which is one of the hottest topics in recommendation. Other tasks (e.g., sequential recommendation) are remained for future exploration. Specifically, we first search the accepted full paper lists ($8 * 4 = 32$) for the eight conferences in the four years. Given our interest and the 32 lists, we only consider papers with titles containing keywords 'recommend$*$' or 'collaborative filtering'. After that, we manually select papers towards top-N recommendation adopting ranking metrics (e.g., Precision, Recall) to evaluate the *accuracy* of recommendation. In the end, we obtain a collection of 141 relevant papers as listed in Table 1.

By delicately reviewing the collected papers in Table 1, we find that there are a bunch of hyper-factors that may affect the recommendation performance along with the entire evaluation chain. Typically, they can be classified into two types: (1) **model-independent** factors that are isolated from the model design and optimization process (e.g., dataset and baseline selection); and (2) **model-dependent** ones involved in the model development and parameter optimization process (e.g., loss function and regularization terms). Figure 1 illustrates the two types of hyper-factors along with the whole evaluation chain, starting with the dataset selection and ending with hyper-parameter tuning strategy. Next, we will analyze these hyper-factors one by one.

2.2 Analysis on Model-independent Hyper-factors

2.2.1 *Dataset Selection*

We find two major issues on the utilized datasets by analyzing the collected papers: (1) domain diversity, i.e., there are massive different datasets within and across various domains, as shown in Table 6; and (2) version diversity, i.e., many datasets, though with the same names, may have different versions. For example, we find more than three versions for Yelp, as it has been updated for different rounds of the challenge. By treating different versions as a same dataset, there are 84 different datasets used in the 141 papers. Figure 2(a) shows the dataset popularity, i.e., percentage of papers for the top-15 used datasets. Around 90% of the 141 papers adopt at least one of the 15 datasets.

For a practical study, we further delicately select six among them by considering popularity and domain coverage, thus resolving the domain diversity issue. They are **ML-1M** (Movie), **Yelp** (LBSNs), **LastFM** (Music), **Epinions** (SNs), **Book-X** (Book) and **AMZe** (Consumable), covering 62% papers of the collection. To ease the version diversity issue, we conduct a careful selection by considering the authority and information richness of data sources, which could benefit the study on diverse recommendation models. Specifically, we use MovieLens-1M (ML-1M) released by

[3] A preliminary report of our work was published at RecSys'20 as a reproducibility paper [4]. In this study, we have extended it from two aspects: (1) with regards to theoretical study, we conduct a more in-depth analysis on the hyper-factors throughout the whole evaluation chain by reviewing more latest literature in 2020, whereby several new hyper-factors (e.g., regularization terms, parameter initializers and model optimizers) are further considered; we systematically classify these hyper-factors into model-independent and -dependent ones, whereby different modes of rigorous evaluation are well defined and discussed in-depth; and (2) for the experimental study, we release DaisyRec 2.0 by further fusing these new hyper-factors and extending existing ones (e.g., more types of loss function designs, negative sampling strategies, data splitting methods and deep learning based baselines). To be more user-friendly, we design a user interface tool for automatic command generation. Thereby, more holistic experiments are conducted based on DaisyRec 2.0 to unveil the impacts of different hyper-factors on recommendation performance, where more interesting and insightful observations are gained.

TABLE 2
Statistics of the six selected datasets.

Dataset		ML-1M	Yelp	LastFM	Epinions	Book-X	AMZe
origin	#User	6,038	1,326,101	1,892	22,164	105,283	4,201,696
	#Item	3,533	174,567	17,632	296,277	340,556	476,002
	#Record	575,281	5,261,669	92,834	922,267	1,149,780	7,824,482
	Density	2.697e-2	2.273e-5	2.783e-3	1.404e-4	3.207e-5	3.912e-6
5-filter	#User	6,034	227,109	1,874	21,995	22,072	253,994
	#Item	3,125	123,985	2,828	31,678	43,748	145,199
	#Record	574,376	3,419,587	71,411	550,117	623,405	2,109,869
	Density	3.046e-2	1.214e-4	1.348e-2	7.895e-4	6.456e-4	5.721e-5
10-filter	#User	5,950	96,168	1,867	21,111	12,720	63,161
	#Item	2,811	80,351	1,530	14,030	18,318	85,930
	#Record	571,549	2,458,153	62,984	434,162	443,196	949,416
	Density	3.412e-2	3.181e-4	2.205e-2	1.466e-3	1.902e-3	1.749e-4
Timestamp		√	√	×	√	×	√

GroupLens[4]; Yelp was created by Kaggle in 2018[5]; LastFM was released by the 2nd international workshop HetRec 2011 [156]; Epinions was crawled by [157] containing timestamp and item category information; Book-Crossing (Book-X) [158] was collected by Cai-Nicolas Ziegler from the Book-Crossing community[6]; Amazon Electronic (AMZe) was released by Julian McAuley[7]. The statistics of all datasets are listed in Table 2 and all links for the datasets are available at the homepage of DaisyRec 2.0.

2.2.2 Dataset Pre-Processing

There are typically two core steps for the dataset pre-processing, namely binarization and filtering.

Binarization. As our current study focuses on the implicit feedback, all the datasets with explicit feedback (e.g., ratings or counts) should be binarized into implicit data. Let $u \in \mathcal{U}, i \in \mathcal{I}$ denote user u and item i; $\mathcal{U}, \mathcal{I}$ are user and item sets; and $r_{ui} \in \mathcal{R}$ is the binary feedback of user u over item i. For each user u, we transform all her explicit feedback with no less than a threshold (denoted by r) into positive feedback ($r_{ui} = 1$); otherwise, negative feedback ($r_{ui} = 0$). Different papers may have different settings for r (e.g., $r = 1/2/3/4$). By following the majority studies [74], [76], [103], we recommend to set $r = 4$ for ML-1M, and $r = 1$ for the rest datasets.

Filtering. The original datasets are generally quite sparse, where some users (items) only interact with few items (users), e.g., less than 5. To ensure the quality, the filtering strategy is usually adopted to help remove the inactive users and items. By analyzing the paper collection, we have found out that around 57% of papers adopt filtering strategies; while 22% of papers utilize the original datasets; and 21% of papers do not report details on data filtering. Among the papers adopting pre-processing strategies, more than 58% of them utilize 5- or 10-filter/core setting [38], [73], [102] on the datasets, which filter out users and items with less than 5 or 10 interactions, respectively. While, others adopt, such as 1-, 2-, 3-, 4-, 20- or 30-filter/core settings. Therefore, to check the performance and robustness of different methods w.r.t. various data sparsity levels, besides original datasets, we also take the two most common settings (i.e., **5-** and **10-filter**) on all selected datasets, the statistics of which are summarized in Table 2. Note that F-fiter is different from F-core: the former means that users and items are only filtered with less than F interactions in one pass; by contrast, the latter indicates a recursive filtering until all users and items have at least F interactions.

TABLE 3
Average size of training & test sets for each user.

Type	Setting	ML-1M	Yelp	LastFM	Epinions	Book-X	AMZe
Train	origin	86	4	39	35	10	2
	5-filter	86	13	30	23	23	7
	10-filter	86	21	27	20	28	12
Test	origin	64	2	10	30	5	2
	5-filter	64	5	8	13	7	3
	10-filter	64	8	7	10	8	5

2.2.3 Dataset Splitting Methods

Three types of data splitting methods are mainly used in the collected papers, including **split-by-ratio** (69% of the papers), **leave-one-out** (21% of the papers) and **split-by-time** (6% of the papers). There are also 4% of papers not reporting their data splitting methods. In particular, split-by-ratio means that a proportion ρ (e.g., $\rho = 80\%$) of the dataset (i.e., user-item interaction records) is treated as training set, and the rest ($1 - \rho = 20\%$) as test set; leave-one-out refers to that for each user, only one record is kept as test set and the remaining are for training; and split-by-time indicates directly dividing training and test sets by a fixed timestamp, that is, the data before the fixed timestamp is used as training set, and the rest as test set.

Although 69% of papers adopt split-by-ratio, they are quite different from each other due to: (1) different proportion settings, e.g., $\rho = 50\%, 60\%, 70\%, 80\%, 90\%$; (2) global- or user-level split. That is, some globally split the entire records into training and test sets regardless of different users; whilst others split training and test sets on the user basis; and (3) random- or time-aware split. Among papers exploiting split-by-ratio, 87% of papers merely randomly split the data, whereas 13% of papers split the data based on the timestamp, i.e., the earlier (e.g., $\rho = 80\%$) records as training and the later ones as test. In terms of leave-one-out, the split is generally on the user basis and timestamp is taken into account by 60% of papers. To validate the impact of different data splitting methods, in our study, we compare the recommendation performance with random-/time-aware **split-by-ratio** at **global-level** with $\rho = 80\%$ and random-/time-aware **leave-one-out** at **user-level**, and leave split-by-time as our future exploration.

Besides, to improve the test efficiency, they usually randomly sample a number of negative items (e.g., $neg\_test = 99, 100, 999, 1,000$) that are not interacted by each user, and then rank each test item among the $(neg\_test + 1)$ items for recommendation [44], [104], [110], [138]. To speed up the test process, we randomly sample negative items for each user to ensure her test candidates to be $1,000$, and then rank all test items among the $1,000$ items w.r.t. both split-by-ratio and leave-one-out. Table 3 depicts the average number of test items for each user on the six datasets across origin, 5- and 10-filter settings w.r.t. split-by-ratio, where all values are smaller than 100, indicating that $1,000$ test candidates is sufficient to examine the performance of recommenders.

2.2.4 Comparison Baseline Selection

As observed, the compared baselines vary a lot in different collected papers. We show the top-15 widely-compared

[4] grouplens.org/datasets/movielens/
[5] www.kaggle.com/yelp-dataset/yelp-dataset
[6] grouplens.org/datasets/book-crossing/
[7] jmcauley.ucsd.edu/data/amazon/links.html

TABLE 4
Objective functions of different baselines.

Method	Origin	To explore		
BPRMF	$\mathcal{L}_{pai} + f_{ll}$	$\mathcal{L}_{poi} + f_{cl}$	$\mathcal{L}_{pai} + f_{tl}$	$\mathcal{L}_{pai} + f_{hl}$
BPRFM	$\mathcal{L}_{pai} + f_{ll}$	$\mathcal{L}_{poi} + f_{cl}$	$\mathcal{L}_{pai} + f_{tl}$	$\mathcal{L}_{pai} + f_{hl}$
SLIM	$\mathcal{L}_{poi} + f_{sl}$	–	–	–
NeuMF	$\mathcal{L}_{poi} + f_{cl}$	$\mathcal{L}_{pai} + f_{ll}$	$\mathcal{L}_{pai} + f_{tl}$	$\mathcal{L}_{pai} + f_{hl}$
NFM	$\mathcal{L}_{pai} + f_{ll}$	$\mathcal{L}_{poi} + f_{cl}$	$\mathcal{L}_{pai} + f_{tl}$	$\mathcal{L}_{pai} + f_{hl}$
NGCF	$\mathcal{L}_{pai} + f_{ll}$	$\mathcal{L}_{poi} + f_{cl}$	$\mathcal{L}_{pai} + f_{tl}$	$\mathcal{L}_{pai} + f_{hl}$
Multi-VAE	$\mathcal{L}_{poi} + f_{cl}$	–	–	–

baselines in these papers in Figure 2(b), covering 98% of papers in total, that is, 98% of the papers consider at least one of the 15 baselines. They can be classified into three types, (1) memory-based methods (MMs): MostPop, ItemKNN [7]; (2) latent factor methods (LFMs): BPRMF [150], FM [159], WRMF [160], SLIM [161], PureSVD [155], eALS [162] and DCF [163]; and (3) deep learning methods (DLMs): NeuMF [134], CKE [164], NeuFM [134], CDAE [165], NGCF [102] and Multi-VAE [136].

For a practical study, we ultimately take 10 baselines into account, as highlighted in blue in Figure 2(b). Specifically, two MMs are considered. **MostPop** is a non-personalized method and recommends most popular items to all users; and **ItemKNN** is a K-nearest neighborhood based method recommending items based on item similarity. We adapt it for implicit feedback data by following [160], and adopt cosine similarity. In terms of LFMs, **BPRMF** is selected as the representative of matrix factorization method (WRMF, eALS and DCF are remained for future exploration); **BPRFM** (factorization machine) considers the second-order feature interactions between inputs and we train it by optimizing the BPR loss [150]; **SLIM** [161] learns a sparse item similarity matrix by minimizing a constrained reconstruction square loss; and **PureSVD** directly performs conventional singular value decomposition on the user-item implicit interaction matrix, where all the unobserved entries are set as 0. Regarding DLMs, **NeuMF** [134] is a state-of-the-art neural method taking advantage of both generalized matrix factorization and multi-layer perceptron (MLP); **NFM** [134] seamlessly combines the linearity of FM in modelling second-order feature interactions and the non-linearity of neural network in modelling higher-order feature interactions, and we train it by optimizing the BPR loss; **NGCF** [102] leverages graph neural networks to capture the high-order connectivity in the user-item graph; and **Multi-VAE** [136] is a generative model with multinomial likelihood and extends variational autoencoders to collaborative filtering. CKE and CDAE are remained for future study, as CKE involves textual and visual information besides user-item interaction data; both CDAE and Multi-VAE are in the family of autoencoders, while Multi-VAE has proven to be a stronger baseline [6].

2.2.5 Evaluation Metric Selection

The evaluation metrics vary a lot in different papers in the collection. Figure 2(c) depicts the popularity of the used evaluation metrics. We thus adopt the top-6 evaluation metrics covering 99% of the collected papers. That is to say, 99% of these papers adopt at least one of the six metrics. They are **Precision**, **Recall**, Mean Average Precision (**MAP**), Hit Ratio (**HR**), Mean Reciprocal Rank (**MRR**) and Normalized Discounted Cumulative Gain (**NDCG**). In particular, the first four metrics intuitively measure whether a test item is present in the top-N recommendation list, whilst the latter two accounts for the ranking positions of test items. Detailed formulas are given by Table 8 (Appendix), where $R(u), T(u)$ represent the recommendation set and the test set for user u, respectively; $rel_j = 1/0$ indicates whether the item at rank j is in the intersection of $R(u)$ and $T(u)$, i.e., $(R(u) \cap T(u))$; $\delta(x) = 1$ if x is true, otherwise 0; and IDCG means the maximum possible DCG through ideal ranking.

2.3 Analysis on Model-dependent Hyper-factors

2.3.1 Loss Function Design

Two types of objective functions are widely utilized by the collected papers: **point-wise** (55% of the collected papers) and **pair-wise** (40% of the collected papers). The former only relies on the accuracy of the prediction of individual preferences; whilst the latter approximates ranking loss by considering the relative order of the predictions for pairs of items. Regardless of which one is deployed, it is critical to properly exploit unobserved feedback within the model, as merely considering the observed feedback fails to account for the fact that feedback is not missing at random, thus being not suitable for top-N recommenders [165]. Let $\mathcal{L}$ denote the objective function of recommendation task. The point- and pair-wise objectives are thus given by:

$$\begin{aligned} \mathcal{L}_{poi} &= \sum\nolimits_{(u,i) \in \widetilde{\mathcal{O}}} f(r_{ui}, \hat{r}_{ui}) + \lambda \Omega(\mathbf{\Theta}); \\ \mathcal{L}_{pai} &= \sum\nolimits_{(u,i,j) \in \widetilde{\mathcal{O}}} f(r_{uij}, \hat{r}_{uij}) + \lambda \Omega(\mathbf{\Theta}), \end{aligned} \tag{1}$$

where $\widetilde{\mathcal{O}} = \{\mathcal{O}^+ \cup \mathcal{O}^-\}$ is the augmented dataset with the unobserved user-item set $\mathcal{O}^- = \{(u,j)|r_{uj} = 0\}$ in addition to the observed user-item set $\mathcal{O}^+ = \{(u,i)|r_{ui} = 1\}$; $f(\cdot)$ is the loss function; $r_{ui}, \hat{r}_{ui}$ are the observed and estimated feedback of user u on item i, respectively; (u,i,j) is the triple meaning that u prefers positive item i to negative item j; $r_{uij} = r_{ui} - r_{uj}, \hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$; $\Omega(\mathbf{\Theta})$ is the regularization term, whose impact is illustrated in Section 2.3.5; and $\mathbf{\Theta}$ is the set of model parameters.

W.r.t. the loss function $f(\cdot)$, point-wise objective usually adopts square loss and cross-entropy (CE) loss, whereas pair-wise objective generally employs log loss, top-1 loss and hinge loss:

$$\begin{aligned} \mathcal{L}_{poi} &= \begin{cases} \text{Square Loss} & f_{sl} = \frac{1}{2}(r_{ui} - \hat{r}_{ui})^2 \\ \text{CE Loss} & f_{cl} = -r_{ui} log(\hat{r}_{ui}) - (1 - r_{ui}) log(1 - \hat{r}_{ui}) \end{cases} \\ \mathcal{L}_{pai} &= \begin{cases} \text{Log Loss} & f_{ll} = -log(\sigma(\hat{r}_{uij})) \\ \text{Top-1 Loss} & f_{tl} = \sigma(-\hat{r}_{uij}) + \sigma(\hat{r}^2_{uj}) \\ \text{Hinge Loss} & f_{hl} = max(0, 1 - \hat{r}_{uij}). \end{cases} \end{aligned} \tag{2}$$

Table 4 shows the original objectives used by BPRMF, BPRFM, SLIM, NeuMF, NFM, NGCF and Multi-VAE. Besides, we vary different objectives on these baselines to further examine their respective impacts. Note that MostPop, ItemKNN and PureSVD do not have objective functions; we did not consider the square loss, as it is more suitable for rating prediction task instead of ranking problem [150]; Multi-VAE cannot be easily adapted with different objective functions; and we did not study the impacts of different objectives on SLIM due to its high complexity and low scalability, which will be discussed in Section 3.2.3.

2.3.2 Negative Sampling Strategies

As pointed out in Section 2.3.1, properly exploiting the unobserved feedback (i.e., negative samples) helps learn users' relative preferences and benefits more accurate top-N recommendation. This can be further supported by the fact that around 65% of the collected papers consider the unobserved feedback when designing objective functions regardless of point- and pair-wise ones. However, it is not practical to leverage all unobserved feedback in large volume, as most users only provide feedback for a small number of items. Negative sampling is, therefore, adopted to balance the efficiency and effectiveness. It is noteworthy that we follow majority studies [73], [84], [134], [137] and directly treat the unobserved feedback as negative feedback. There may be different explanations behind the unobserved feedback [166], but we leave it for further exploration.

There are various kinds of negative sampling strategies. Specifically, **uniform sampler** [134], where all unobserved items of each user are sampled with an equal probability, has been adopted by almost all papers in the collection. To better study the impact of negative sampling, we additionally consider item popularity-based samplers, which have also been adopted in recommendation [38], [162]. **Low-popularity sampler** refers to that for each user, her unobserved items with a lower popularity are sampled with a higher probability. This is based on the assumption that a user is less likely to prefer less popular items. **High-popularity sampler** is opposite to the low-popularity sampler, where the unobserved items of each user with a higher popularity are more likely to be sampled. The rationale behind is that if a user provides no feedback for a quite popular item favored by a large number of users, it indicates that she may be not into this item. Moreover, we also compare two types of hybrid samplers by leveraging both uniform and popularity samplers, namely **uniform+low-popularity sampler** and **uniform+high-popularity sampler**. In these cases, half of the unobserved items are sampled via the uniform sampler, while the rest half are sampled via popularity samplers.

2.3.3 Parameter Initializer Selection

There are normally a set of learnable parameters (e.g. user/item representation matrix and the network weights) for the recommendation models, ranging from early LFMs to recently emerged DLMs. A proper parameter initializer will assist in a faster model convergence and better model performance. Specifically, the core of LFMs is to learn accurate user and item representations, which are generally initialized based on either a **Uniform distribution** in the range of $(0, a)$ or a **Normal/Gaussian distribution** with zero mean and a variance of σ^2 [28], [92], given by,

$$\boldsymbol{v}_{uf}/\boldsymbol{v}_{vf} \sim \boldsymbol{U}(0, a); \quad \boldsymbol{v}_{uf}/\boldsymbol{v}_{vf} \sim \mathcal{N}(0, \sigma^2), \tag{3}$$

where the common settings are $a = 1$; and $\sigma = 0.01$.

Regarding DLMs, besides user and item representations, initializing with proper weights helps ensure the network to converge in a reasonable amount of time; otherwise the network loss function does not go anywhere even after hundreds of thousands of iterations. Given too small weights, the variance of the input diminishes as it passes through each layer in the network, and eventually drops to a really low value thus failing to work. Contrarily, a too-large weights leads to exploding gradients. Xavier initialization [167] has been proven as an effective fashion, and widely adopted by DLMs in recommendation [105], [114], [116], [130]. Typically, the weights are also initialized based on either a Uniform or Normal distribution, defined as[8],

$$\boldsymbol{W}_{ij} \sim \boldsymbol{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right); \ \boldsymbol{W}_{ij} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right), \tag{4}$$

where $\boldsymbol{W}_{ij}$ is the network weight; $n_{\text{in}}, n_{\text{out}}$ are the number of input and output neurons, respectively.

By analyzing the collected papers, we find that around 59% of them do not report the parameter initializers. Among the papers mentioning parameter initializers, 36% of them are based on a Normal distribution; 10% of them use a Uniform distribution; 18% of them adopt the Xavier Initialization; 13% of them employ the pre-trained embeddings (e.g., via BPRMF) for initialization; and the rest 23% ultilize other methods. The impacts of different parameter initializers are investigated in Section 3.3.3.

2.3.4 Model Optimizer Selection

Opitimizer is used to update model parameters, thus minimizing the loss function; meanwhile loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, i.e., global minimum. Different optimizers also affect the recommendation performance. By looking into the collected papers, we find that 23% of them do not report their optimizers. Among the papers mentioning used optimizers, 50% of them adopt Adam [134], [136], [165]; 23% of them use SGD [150], [159]; and the rest 27% employ other optimizers (e.g., ALS [35], AdaGrad [104] and RMSProp [85]).

Here, we discuss six commonly-selected optimizers as shown in Table 9 (Appendix). (1) Gradient Descent (**GD**) is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. The parameters are then updated in the negative gradient direction to minimize the loss; (2) Stochastic Gradient Descent (**SGD**) is a variant of GD to update the model's parameters after computation of loss on each training example, whilst parameters are changed after calculating gradient on the whole dataset by GD; (3) Mini Batch Stochastic Gradient Descent (**MB-SGD**) is an improvement on both SGD and standard GD, where the dataset is divided into various batches and after every batch, the parameters are updated; (4) Adaptive Gradient (**AdaGrad**) is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for frequently parameters, whilst larger updates (i.e. high learning rates) for infrequent parameters; (5) Root Mean Square Propagation (**RMSProp**) is devised to resolve AdaGrad's radically diminishing learning rates, and divides the learning rate by the average of the exponential decay of squared gradients; and (6) Adaptive Moment Estimation (**Adam**) calculates the adaptive learning rate for each parameter from estimates of first and second moments of the gradients. In addition to the decaying average of past squared gradients like RMSprop, it also keeps a decaying average of past gradients.

[8] https://pytorch.org/docs/stable/nn.init.html

2.3.5 Strategies to Avoid Over-fitting

In machine learning, different strategies are exploited to combat the issue of over-fitting, which refers to the model over-fits the training data, thus achieving poor performance on the validation or test data. As a matter of fact, the most widely used regularization techniques include regularization terms, dropout and early-stop mechanism.

Regularization. It is generally integrated into the loss function, so as to help avoid over-fitting while training a recommendation model. Two types of terms are mainly adopted, namely $L1$ and $L2$ regularization (i.e. norm). $L1$ norm is also known as Manhattan Distance, which is the most natural way of measure distance between vectors. It is the sum of the magnitudes of the vectors in a space, where all the components of the vector are weighted equally. $L2$ norm is the most popular norm, also known as the Euclidean norm, which is the shortest distance between two points. Different from $L1$ norm, each component of the vector is squared for $L2$ norm, indicating that the outliers have more weighting, so it can skew results. The main difference between the $L1$ and $L2$ regularization lies in (1) $L1$ regularization attempts to estimate the median of the data, whereas $L2$ regularization tries to estimate the mean of the data to avoid over-fitting; and (2) $L1$ regularization helps in feature selection by eliminating less important features, which is helpful given a large number of feature points.

Dropout. It has been widely adopted in DLMs to help avoid over-fitting [168]. The key idea is to randomly drop units (along with their connections) from the neural network during training, which prevents units from co-adapting too much. Hence, an extra hyper-parameter, i.e., the probability of retaining a unit p, is introduced, controlling the intensity of dropout. For instance, $p = 1$ implies no dropout and low values of p mean more dropout. Smaller p could lead to under-fitting; whereas large p may not produce enough dropout to prevent over-fitting. Typical values of p for hidden units are in the range 0.5 to 0.8 [168].

Early-stop Mechanism. Early-stop is also a form of regularization used to avoid over-fitting. A major issue with training recommenders (e.g., LFMs and DLMs) is in the choice of the number of training epochs to use. Too many epochs can lead to over-fitting of the training dataset, whereas too few may result in an under-fit model. Early-stop is a method that allows us to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. To be more specific, if the validation loss stops decreasing for several epochs in a row, the training stops. Through analyzing on the collected papers, only 11% of them point out early-stop strategy is adopted in their papers. In our study, we also investigate the impacts of ealry-stop mechanism on recommendation evaluation.

2.3.6 Hyper-parameter Tuning Strategies

Hyper-parameter tuning, including validation and searching strategies, plays a vital role in the training process of recommendation approaches, and greatly influences the final recommendation performance.

Validation Strategy. Through the paper analysis, we notice that more than 33% of papers directly tune hyper-parameters according to the performance on the test set. That is to say, they use the same data to tune model parameters and evaluate the model performance. Information may thus leak into the model and overfit the historical data. As a matter of fact, besides the training and test sets, an extra validation set should be retained to help tune the hyper-parameters, which is called *nested validation*[9]. With nested validation, the optimal hyper-parameter settings are obtained when the model achieves the best performance on the validation set. By doing so, the information leak issue is well avoided in the model training and evaluation process. Therefore, in our study, we adopt the nested validation strategy. To be more specific, we further select **10%** of records from the training set as the validation set for split-by-ratio; and for leave-one-out, we retain one record from the training set as the validation set to tune hyper-parameters. Finally, the performance on the test set is reported. Due to the computational requirements of certain of baselines, we were unable to search the hyper-parameter space for cross-validation in a reasonable amount of time.

Searching Strategy. From our observation, almost all collected papers employ the most straightforward and simple method – **grid search** [73], [134] to find out the optimal parameter settings. In particular, each hyper-parameter is provided with a set of possible values (i.e., search space) based on the prior-knowledge, and the optimal setting is then obtained by traversing the entire search space. Suppose a model has m parameters, where each parameter has an average of n possible values, the model needs to be executed for n^m times to find out optimal settings for all parameters. Hence, grid search is more suitable for models with less hyper-parameters; otherwise, it may suffer from the combination explosion issue. To improve the tuning efficiency, other strategies have been introduced. Given the search space of each parameter, **random search** [169] randomly chooses trials for a pre-defined times (e.g., 30) instead of traversing the entire search space. It is able to find models that are as good or slightly worse but within a smaller fraction of the computation time. In contrast, **Bayesian HyperOpt** [170] is not a brute force but more intelligent technique compared to grid and random search. It makes use of information from past trials to inform the next set of hyper-parameters to explore, while not compromising the quality of the results [6]. Therefore, for each baseline, we adopt Bayesian HyperOpt to perform hyper-parameter optimization on **NDCG**, which is the most popular metrics as shown in Figure 2(c); and other metrics are expected to be simultaneously optimized with the optimal results on NDCG.

2.4 Categorization on Evaluation Modes

Based on the model-independent and model-dependent hyper-factors introduced in Sections 2.2-2.3, we define four modes of rigorous evaluation as below.

- **Relax Mode** keeps exactly the same settings for all model-independent hyper-factors and follows the original settings as per individual approach for model-dependent hyper-factors.

[9] https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

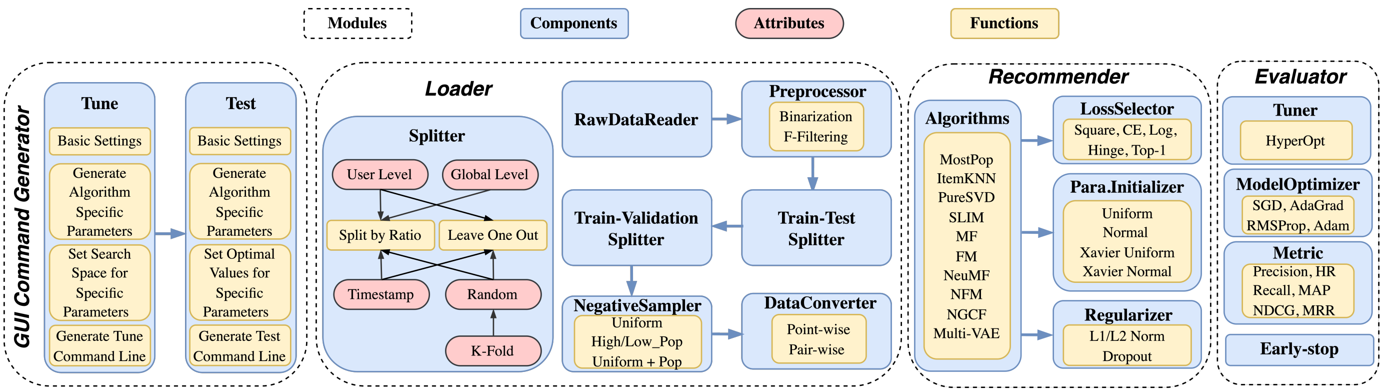


Fig. 3. The overall structure of DaisyRec 2.0, composed of four components, i.e., GUI Command Generator, Loader, Recommender, and Evaluator.

- **Hard-strict Mode** keeps exactly the same settings for both model-independent and model-dependent hyper-factors for all approaches.
- **Soft-strict Mode** keeps exactly the same settings for all model-independent hyper-factors and empirically finds out the optimal settings for per individual approach for model-dependent hyper-factors.
- **Mixed Mode** keeps exactly the same settings for all model-independent hyper-factors; while applies hard-strict mode on some model-dependent hyper-factors (e.g., the same initializer/optimizer), and relax or soft-strict modes on the others (e.g., empirically searching the optimal hyper-parameter settings for different baselines).

Through analysis, we find that most of the collected papers adopt the mixed mode for evaluation [42], [46], [116], for example, using the same model optimizer and parameter initializer for all approaches; adopting different loss functions as indicated in the original papers; and empirically finding out best parameter settings for all approaches. Regardless of different modes, it is essential to keep exactly the same settings for all model-independent hyper-factors for a rigorous evaluation. W.r.t. the model-dependent hyper-factors, different modes have their own pros and cons.

- Relax mode can extremely reduce the cost on exploring the optimal performance. However, it relies on the original settings indicated by the individual approach, which are not always available and may lead to a unfair comparison, e.g., one model defeats the others merely because it adopts a different loss function.
- Hard-strict mode ensures a fair comparison among different baselines, while it may not always be reasonable to, e.g., have the same settings for all shared hyper-parameters for all baselines, as the optimal hyper-parameter settings for different baselines may vary a lot across different datasets.
- Soft-strict mode could help find out the optimal performance for per individual approach, which, however, may be quite expensive due to the large amount of combinations of model-dependent hyper-factors.
- Mixed mode would be a better balance among complexity, performance and fairness for per individual approach. For instance, soft-strict mode can be applied regarding, e.g., optimal hyper-parameter settings, to maintain model performance; hard-strict mode can be used for, e.g., parameter initializer and model optimizer, to ensure fair comparison and less exploration complexity.

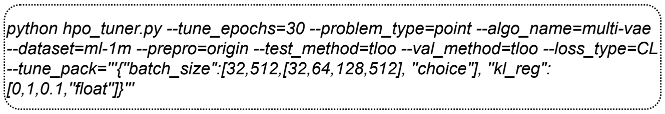
In summary, **mixed mode** could be the most practical way for achieving rigorous evaluation in recommendation.

3 Experimental Study

3.1 Introduction to DaisyRec 2.0

To support the empirical study, we release a user-friendly Python toolkit named as **DaisyRec 2.0**, where Daisy is short for 'Multi-**D**imension f**AI**rly comp**A**r**I**son for recommender **SY**stem'. Different from existing open-source libraries (e.g., LibRec [152], OpenRec [171] and DeepRec [153]), which mainly aim to reproduce various state-of-the-art recommenders, DaisyRec 2.0 is designed with the goal of performing rigorous evaluation in recommendation by seamlessly accommodating the extracted hyper-factors in Section 2. It is built upon the widely-used deep learning framework Pytorch (pytorch.org), and Figure 3 depicts its overall structure consisting of four modules: GUI Command Generator, Loader, Recommender and Evaluator.

In particular, GUI Command Generator[10] is used to help generate tune and test commands in a more user-friendly fashion. Taking the tune command generator as an example, users first need to select values for the basic settings (e.g., algorithm name and dataset) from a drop-down menu. Based on the selected algorithm, it then automatically displays the algorithm-specific parameters (e.g., KL regularization for Multi-VAE). Accordingly, users can select and set the search space for the algorithm-specific parameters. Lastly, it generates the corresponding tune command (shown in Figure 4) based on all selected settings, which can be directly copied and pasted into the terminal to run.

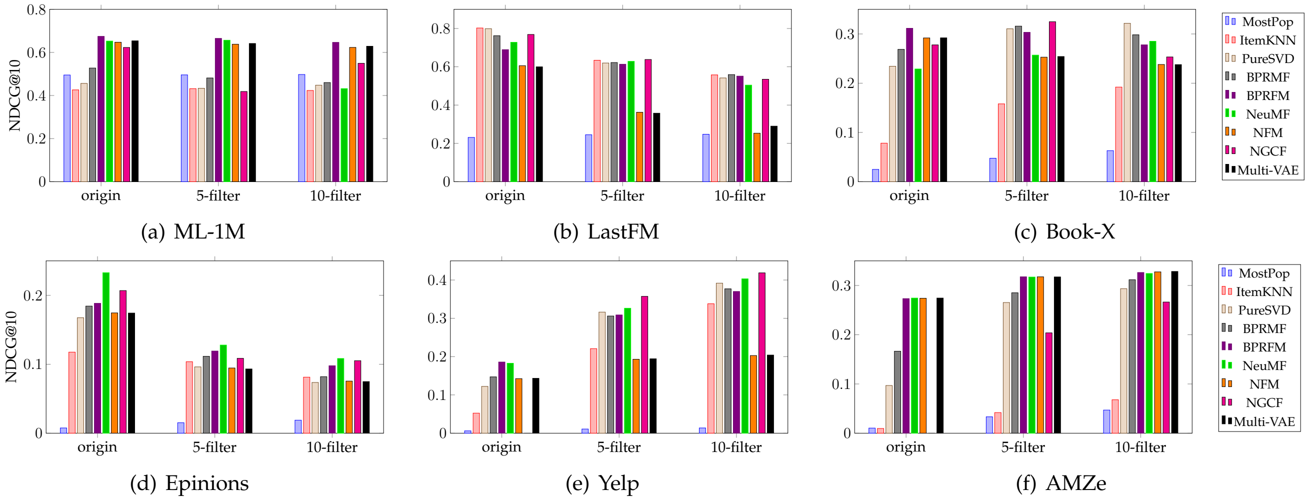


```
python hpo_tuner.py --tune_epochs=30 --problem_type=point --algo_name=multi-vae
--dataset=ml-1m --prepro=origin --test_method=tloo --val_method=tloo --loss_type=CL
--tune_pack="'{"batch_size":[32,512,[32,64,128,512], "choice"], "kl_reg":
[0,1,0.1,"float"]}'"
```

Fig. 4. An example of the generated tune command for Multi-VAE.

Loader mainly aims to: (1) load and pre-process the dataset; (2) split it into training and test sets based on the selected Splitter; (3) divide validation set from training set by choosing proper Splitter according to Step 2; (4) sample negative items for training by choosing different samplers; and (5) convert the data into the specific format to fit the Recommender. Four components are included in Recommender, where 'Algorithms' implements the ten selected

[10] http://DaisyRecGuiCommandGenerator.pythonanywhere.com



(a) ML-1M (b) LastFM (c) Book-X

(d) Epinions (e) Yelp (f) AMZe

Fig. 5. Performance of baselines w.r.t. time-aware split-by-ratio on the six datasets across origin, 5- and 10-filter settings.

state-of-the-arts in Section 2.2.4 (more recommenders will be implemented); 'LossSelector' makes it flexible to change different objective functions for the algorithms; 'ParameterInitializer' allows to select different initialization methods (e.g., Xavier uniform distribution); and 'Regularizer' provides options for different regularization terms to avoid overfitting (e.g., $L1$ and $L2$). Evaluator is equipped with 'Tuner', 'ModelOptimizer', 'Metric', and 'Early-stop', where 'Tuner' helps accomplish hyper-parameter optimization; 'ModelOptimizer' provides options for different optimizers; 'Metric' implements the classic ranking metrics, e.g., Precision; and 'Early-stop' helps further avoid over-fitting.

To sum up, all modules in DaisyRec 2.0 are wrapped friendly for users to deploy, and new algorithms can be easily added into this extensible and adaptable framework. We keep DaisyRec 2.0 updated by adding more features.

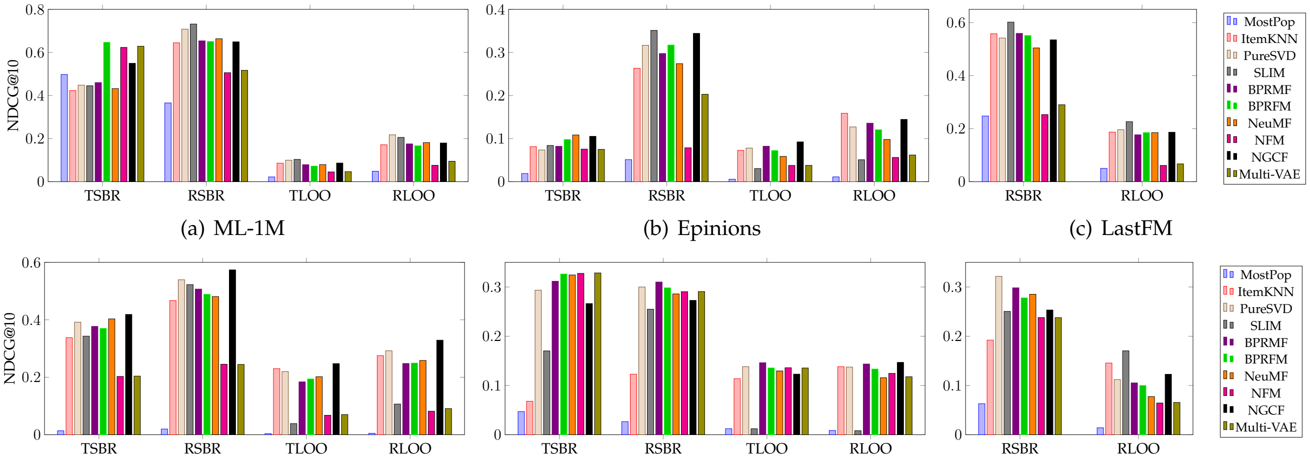
3.2 Analysis on Model-independent Hyper-Factors

3.2.1 *Impacts of Dataset Pre-processing*

To study the impacts of pre-processing strategies (origin, 5- and 10-filter), we adopt Bayesian HyperOpt to perform hyper-parameter optimization w.r.t. NDCG@10 for each baseline under each view on each dataset for 30 trails [6]. We keep original objective functions for each baseline (see Table 4), employ the uniform sampler, and adopt time-aware split-by-ratio at global level ($\rho = 80\%$) as the data splitting method. Besides, 10% of the latest training set is held out as the validation set to tune the hyper-parameters. Once the optimal hyper-parameters are decided, we feed the whole training set to train the final model and report the performance on the test set. Figure 5 depicts the final results, where SLIM is omitted due to its extremely high computational complexity on large-scale datasets, which is unable to complete in a reasonable amount of time; and NGCF on Yelp and AMZe under origin view is also omitted because of the same reason (see Section 3.2.3). Due to the space limitation, we only report the results on NDCG@10.

Overall, three different trends can be observed from the results: (1) the performance of different baselines keeps relatively stable on ML-1M with varied settings; (2) on Book-X, Yelp and AMZe, the performance of all baselines generally climbs up; and (3) an obvious performance drop is observed on Lastfm and Epinions. The most probable explanation is that although the density of the datasets increases (origin → 5-filter → 10-filter) as shown in Table 2, the average length of the training sets for each user keeps stable on ML-1M (86); increases on Book-X, Yelp and AMZe; and decreases on Lastfm (39 → 30 → 27) and Epinions (35 → 23 → 20), as depicted by Table 3. The more training data per user, the better a model can be trained, meaning that the more accurate performance can be achieved, and *vice versa*.

Regarding the performance of different baselines, several major findings can be noted as below. (1) Regarding MMs, MostPop performs the worst in most cases, showing the importance of personalization in recommendation; and ItemKNN is defeated by LFMs and DLMs, indicating the superiority of LFMs and DLMs on effective recommendation. However, on ML-1M, the performance of MostPop exceeds that of ItemKNN, PureSVD and even BPRMF, demonstrating the potential of popularity in effective recommendation; and on Lastfm, ItemKNN achieves the best performance compared with LFMs and DLMs. This implies that, the neighborhood-based idea, though simple, could be absorbed by LFMs and DLMs to further improve the recommendation accuracy [172]. (2) W.r.t. the three LFMs, BPRMF generally performs better than PureSVD but worse than BPRFM. Although PureSVD is simple – directly applying conventional sigular value decomposition on the user-item interaction matrix, it sometimes achieves comparable and even better performance in comparison with BPRMF and BPRFM (see LastFM, Book-X and Yelp with 5/10-filter views). (3) For the four DLMs (i.e., NeuMF, NFM, NGCF and Multi-VAE), their performance varies across different datasets. For instance, NeuMF obtains the highest accuracy on Epinions; NFM is the winner on AMZe; NGCF defeats the others on both LastFM and Yelp; and Multi-VAE achieves extraordinary results on ML-1M. However, they generally perform comparably to BPRFM across all datasets, and sometimes even worse than BPRFM (e.g., ML-1M and Book-X). Besides, on LastFM, they even underperform ItemKNN. This is consistent with the previous findings [6] that DLMs are not always better than traditional methods with well-tuned parameters, but mostly cost much more in training as verified by Table 5.



(a) ML-1M (b) Epinions (c) LastFM

(d) Yelp (e) AMZe (f) Book-X

Fig. 6. Performance of baselines w.r.t. 10-filter on the six datasets with different data splitting methods.

TABLE 5
Baseline comparisons on training time w.r.t. time-aware split-by-ratio on the 10-filter view (seconds).

	MMs		**LFMs**				**DLMs**			
	MostPop	ItemKNN	PureSVD	BPRMF	BPRFM	SLIM	NeuMF	NFM	NGCF	Multi-VAE
ML-1M	0.0048	22.882	0.6808	2286.0	1847.4	62.924	9627.6	1682.6	2216.2	58.197
Lastfm	0.0010	2.4913	0.4728	246.35	103.35	5.1933	95.156	1178.6	538.06	31.807
Book-X	0.0055	29.013	3.1499	506.89	2129.0	860.71	5250.0	895.42	6251.6	101.00
Epinions	0.0070	25.847	2.8042	1497.0	3525.4	3283.7	4265.6	3387.0	9453.9	177.75
Yelp	0.0314	244.80	16.284	1566.4	6882.0	63990	5931.2	2459.4	38351	1388.6
AMZe	0.0167	121.96	1.7262	491.56	1258.4	18273	8625.5	2290.9	10099	1497.3

3.2.2 Impacts of Dataset Splitting Methods

We now test the impacts of different data splitting methods on the recommendation performance. To this end, we compare *random-* and *time-aware split-by-ratio* (i.e., **RSBR** vs. **TSBR**) at global level with $\rho = 80\%$ as well as *random-* and *time-aware leave-one-out* (i.e., **RLOO** vs. **TLOO**) on the 10-filter view. Note that we adopt Bayesian HyperOpt to perform hyper-parameter optimization w.r.t. NDCG@10 for each baseline under each data splitting method on each dataset fro 30 trails. Meanwhile, LastFM and Book-X do not have the timestamp information, so their results on TSBR and TLOO for each baseline are omitted.

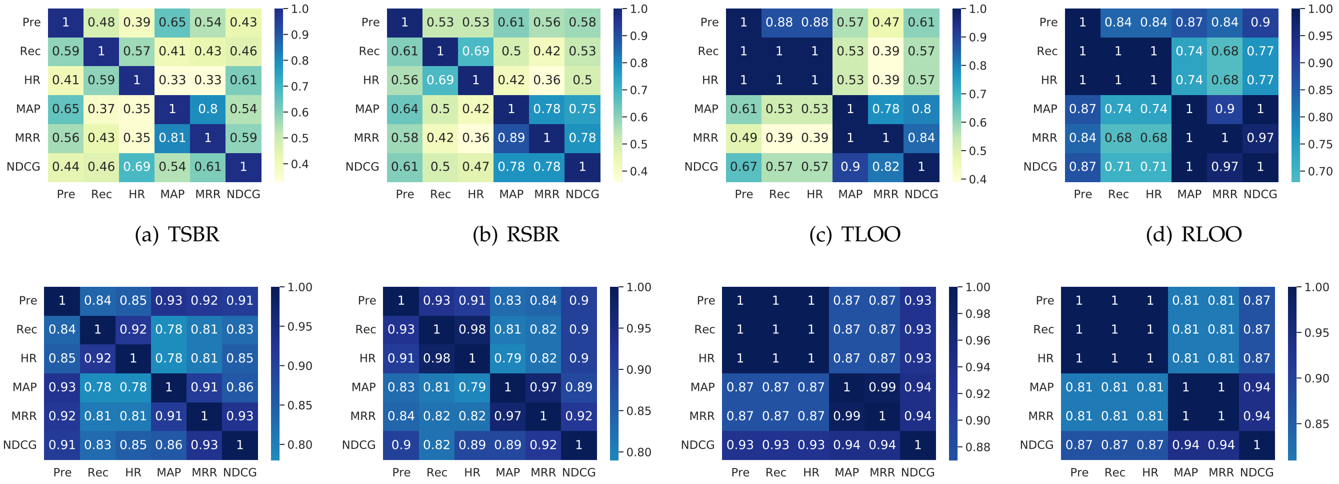
Figure 6 displays the results of ten baselines on the six datasets. First of all, we can clearly observe that the performance of TSBR/SBR (split-by-ratio) is generally better than that of TLOO/LOO (leave-one-out). This could be largely affected by the different settings on the test procedure. To be specific, as introduced in Section 2.2.3, to improve the test efficiency, we randomly sample negative items for each user to ensure her test candidates to be $1,000$, and then rank all test items among the $1,000$ items w.r.t. both SBR and LOO. However, the number of positive items in the test set of SBR (> 1) is normally larger than that of LOO ($= 1$), thus leading to a higher accuracy of SBR. Second, baselines with RSBR/RLOO outperform those with TSBR/TLOO, especially on Epinions. The reason behind is that compared with random-aware split, time-aware split poses a stronger constraint on the pattern of training and test data, thus increasing the training difficulty. However, this is more close to the real recommendation scenario, which strives to infer future by history. Our study also implies that the empirical results disclosed in previous studies using RSBR might be overestimated compared to those for real-world scenarios.

3.2.3 Complexity of Comparison Baselines

Table 5 shows the training time for the ten baselines on the six datasets with optimal hyper-parameters found by Bayesian HyperOpt on 10-filter view via split-by-ratio. Note that, the optimal batch size may differ for different baselines, which may also affect the training time. All the experiments are executed on an Nvidia V100 GPU with 32 GiB memory, each running is paired with 11 Intel(R) Xeon(R) Platinum 8260 CPU (2.4GHz) sharing 40 GiB memory.

According to Table 5, we can note that *MostPop* is the fastest one in training, as it merely ranks all the items by the calculated popularity. *PureSVD* is the runner-up with time complexity $\mathcal{O}(\min\{m^2 f, n^2 f\})$, where m, n, f are the number of users, items and singular values, respectively. Compared with other LFMs and DLMs, it achieves a better balance between time complexity and ranking accuracy. Particularly, it performs comparably and sometimes even better than, e.g., BPRMF and NeuMF on LastFM, as depicted by Figure 5, while its training time is hundreds or thousands times less than that of BPRMF and NeuMF as shown in Table 5. Although the training efficiency of ItemKNN ranks third among all baselines with 10-filter setting, the time cost quadratically increases with origin setting due to its time complexity $\mathcal{O}(mn^2)$. Besides, the similarity matrix also takes up huge memory, for example, on the original AMZe ($n \approx 10^6$), it will cost $(64 \text{ bit} * 10^6 * 10^6)/10^{12} = 64T$ to save the similarity matrix. To ease this issue, we only keep the top-100 similar items for each target item in the memory.

The training time of BPRMF and BPRFM is comparable, where the time complexity for both methods is $\mathcal{O}(|\mathcal{R}|d)$, where $\mathcal{R}$ is the total number of observed feedback and d is the dimension of latent factors. Similar to ItemKNN, the time cost of SLIM with 10-filter setting is acceptable, while



(a) TSBR (b) RSBR (c) TLOO (d) RLOO

(e) TSBR (f) RSBR (g) TLOO (h) RLOO

Fig. 7. The correlations of evaluation metrics w.r.t. different data splitting methods on 10-filter. ‘Pre’ and ‘Rec’ are Precision and Recall, respectively.

it tremendously increases with origin setting due to its time complexity $\mathcal{O}(|R|n)$. Even with the 10-filter view, it takes the longest training time among all baselines on the two large datasets (i.e., Yelp and AMZe). Meanwhile, it also suffers from the huge memory cost issue because of the learned item similarity matrix. Hence, both ItemKNN and SLIM are not scalable for large-scale datasets. Regarding the four DLMs (i.e., NeuMF, NFM, NGCF and Multi-VAE), NFM and Multi-VAE are usually more efficient than NeuMF and NGCF regarding time complexity. Although DLMs yield comparable performance with LFMs, they generally cost much more training time, especially on larger datasets. For example, on AMZe, the training time of NeuMF and NGCF is around 20 times larger than that of BPRMF.

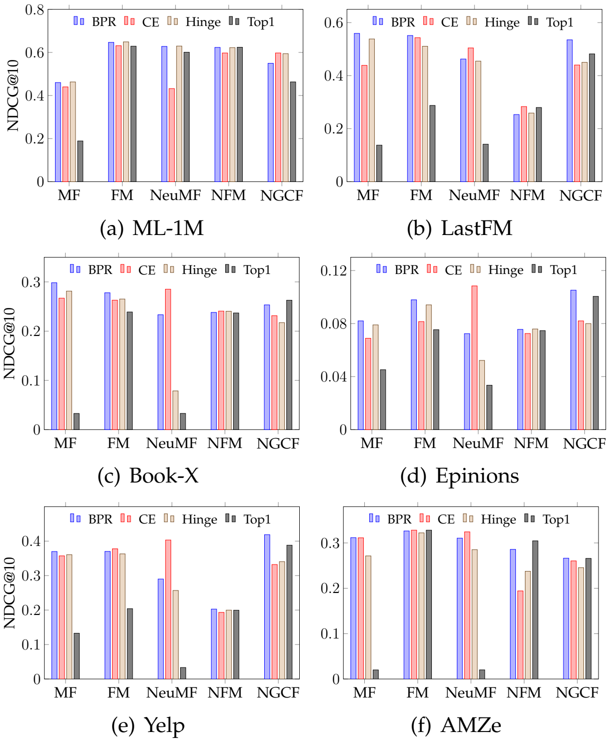
3.2.4 Correlations of Evaluation Metrics

As discussed in Section 3.2.1, we adopt Bayesian HyperOpt to perform hyper-parameter optimization for 30 trials via optimizing NDCG@10. However, six metrics are used in our study, namely Precision, Recall, HR, MAP, MRR and NDCG. The best hyper-parameter settings for optimal NDCG does not necessarily guarantee optimums w.r.t. the other five metrics. Hence, we study the correlation of different metrics when their respective optimums are achieved. In particular, for each baseline on each dataset with 10-filter view, the Bayesian HyperOpt executes 30 trails; we thus have 30 entries for the validation performance of the baseline correspondingly, where each entry includes the results on the six metrics, e.g., [Precision: 0.24; Recall: 0.07; HR: 0.57; MAP: 0.17; MRR: 0.76; NDCG: 0.42]. Due to the optimal results for the six metrics may not achieve simultaneously, we select the optimal one among the 30 entries for each metric, and ultimately obtain six entries, where each entry records the best result on the corresponding metric.

Based on the six selected entries of each method per dataset, we pair-wisely calculate and record the times that any two of them (e.g., NDCG and HR) can achieve their best results simultaneously entry by entry. For example, given the optimal entry for NDCG, we will check whether the rest five metrics (e.g., HR) in this entry are optimal or not. If yes, we will add one at the corresponding position (NDCG, HR) of the correlation matrix; otherwise 0. The same rule is applied to the optimal entries for the other five metrics. Except MostPop, as it does not have any hyper-parameters, we accumulate the results of nine baselines across the six datasets (9*6=54), and ultimately obtain their correlation matrices regarding time-/random-aware split-by-ratio and leave-one-out as illustrated by Figures 7(a-d), where all values are normalized into the range of $[0, 1]$ (divided by 54), and a darker color indicates a stronger correlation, that is, a higher probability of two metrics achieving their best results in the meanwhile.

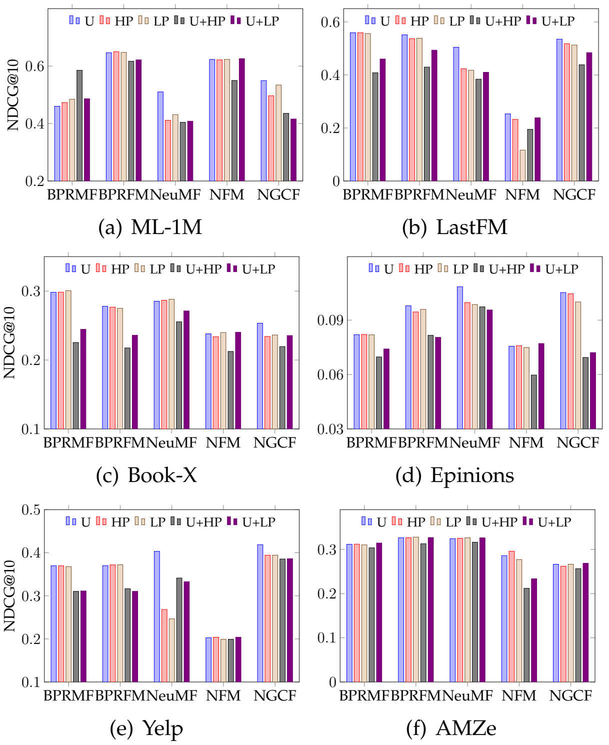
The results help verify our argument that best hyper-parameter settings for optimal NDCG cannot ensure optimal results for all the other five metrics. Several detailed findings can be noted. **(1)** The correlation matrix is asymmetrical, for instance, the correlation for (NDCG, HR, 0.69) is higher than (HR, NDCG, 0.61) as shown in Figure 7(a). That is to say, the probability of a model with best NDCG to achieve the best HR is higher than that of a model with best HR to reap the optimal NDCG. **(2)** Similar trends can be noticed within a same base data splitting method (i.e., TSBR and RSBR; TLOO and RLOO) no matter whether the timestamp information is considered or not; whilst the patterns are different across different base splitting fashions (i.e., TSBR/RSBR and TLOO/RLOO). **(3)** Regarding TSBR/RSBR, the best MRR and MAP are more easily to be guaranteed concurrently, e.g., (MRR, MAP, 0.81) and (MAP, MRR, 0.80) in Figure 7(a); and (MRR, MAP, 0.89) and (MAP, MRR, 0.78) in Figure 7(b). **(4)** For TLOO/RLOO, Precision, Recall and HR are more likely to be optimized simultaneously; and MAP, MRR and NDCG have a higher probability to reach their peaks together. This is mainly due to only one positive item inside the test set for each user; consequently, Recall and HR are equivalent, which is positively correlated with Precision; meanwhile, MAP, MRR and NDCG are also positively correlated with each other.

Additionally, we examine the Kendall’s correlation [83] among metrics in terms of indicating recommendation performance on the ten baselines across the six datasets under 10-filter view with different data splitting methods. The results are depicted in Figures 7(e-h), where a darker color (a



(a) ML-1M (b) LastFM (c) Book-X (d) Epinions (e) Yelp (f) AMZe

Fig. 8. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter view across the six datasets with different loss functions.



(a) ML-1M (b) LastFM (c) Book-X (d) Epinions (e) Yelp (f) AMZe

Fig. 9. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter view across the six datasets with different sampling strategies.

stronger correlation) implies that the metrics produce more identical ranking. We find that **(1)** different from Figures 7(a-d), the Kendall's correlation matrix is symmetrical; **(2)** similarly, the trends are consistent within a same base data splitting method, e.g., Figures 7(g-h), while vary slightly across different base splitting ways, e.g., Figures 7(e) and 7(g); and **(3)** a common observation across Figures 7(e-h) is that MAP, MRR and NDCG are more likely to generate consistent ranking. Besides, for TSBR/RSBR, (Precision, NDCG) and (Recall, HR) show a fairly strong correlation, while w.r.t. TLOO/RLOO, (Precision, Recall, HR) exhibits obvious correlation, which is also caused by the single positive item inside the test set for each user as explained previously. In summary, a convincing and solid evaluation should be performed w.r.t. more diverse metrics.

3.3 Analysis on Model-dependent Hyper-Factors

3.3.1 Impacts of Loss Functions

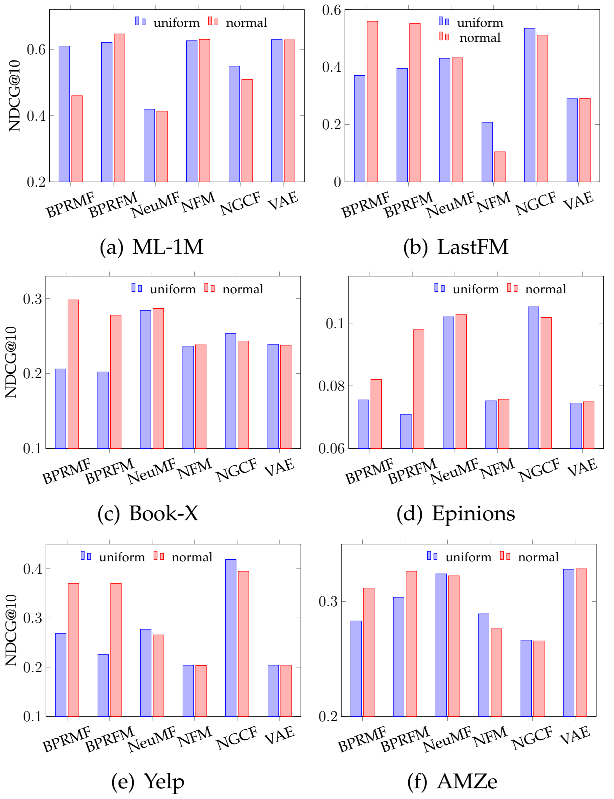
To examine the impacts of different objective functions, we adopt the optimal parameters for the baselines found on 10-filter view with time-aware split-by-ratio in Section 3.2.1, and only vary objective functions for MF, FM, NeuMF, NFM and NGCF. The results are depicted in Figure 8, where BPR (pair-wise log loss), CE (point-wise cross entropy loss), Hinge (pair-wise hinge loss) and Top1 (pair-wise top1 loss) correspond to $\mathcal{L}_{pai}+f_{ll}$, $\mathcal{L}_{poi}+f_{cl}$, $\mathcal{L}_{pai}+f_{hl}$ and $\mathcal{L}_{pai}+f_{tl}$ in Table 4, respectively. Several conclusions can be drawn. As a whole, for different baselines on the six datasets, (1) BPR loss generally achieves the best performance; (2) CE loss and Hinge loss perform comparably; and (3) Top1 loss possesses the largest performance variation. From the perspective of different baselines, (1) MF and FM usually achieve the best performance with BPR loss; (2) NeuMF performs better with CE loss in most cases; (3) NFM is relatively less sensitive to different losses; and (4) NGCF generally obtains better accuracy with either BPR or Top1 loss.

3.3.2 Impacts of Negative Sampling Strategies

We now explore the impact of different negative samplers, i.e., uniform (U), high-popularity (HP), low-popularity (LP), uniform+high-popularity (U+HP) and uniform+low-popularity (U+LP) on BPRMF, BPRFM, NeuMF, NFM and NGCF across the six datasets under 10-filter view with time-aware split-by-ratio. To this end, we only vary negative samplers for the baselines while keeping other parameters fixed. First, the uniform sampler, though simple, achieves comparable performance in comparison with popularity samplers (HP and LP) as illustrated in Figure 9. Intuitively, users may not tend to buy the less popular items, that is, the items with low popularity are more likely to be the negative items for users. However, it is overturned by the empirical results. Second, U+HP and U+LP samplers are generally defeated by U/HP/LP samplers. However, there are some exceptions, e.g., BPRMF on ML-1M and NeuMF on Yelp. Lastly, U+LP exceeds U+HP in most cases, indicating that generally the popular items have a lower probability to be negative items than the less popular ones.

3.3.3 Impacts of Parameter Initializers

To study the impact of different parameter initializers, we compare the results of six baselines (BPRMF, BPRFM, NeuMF, NFM, NGCF and Multi-VAE) across the six datasets under 10-filter view with time-aware split-by-ratio. Specifically, for BPRMF and BPRFM, we adopt uniform ($a = 1$) and normal distribution ($\sigma = 0.01$) for initialization; while for the rest four deep learning baselines, we utilize Xavier



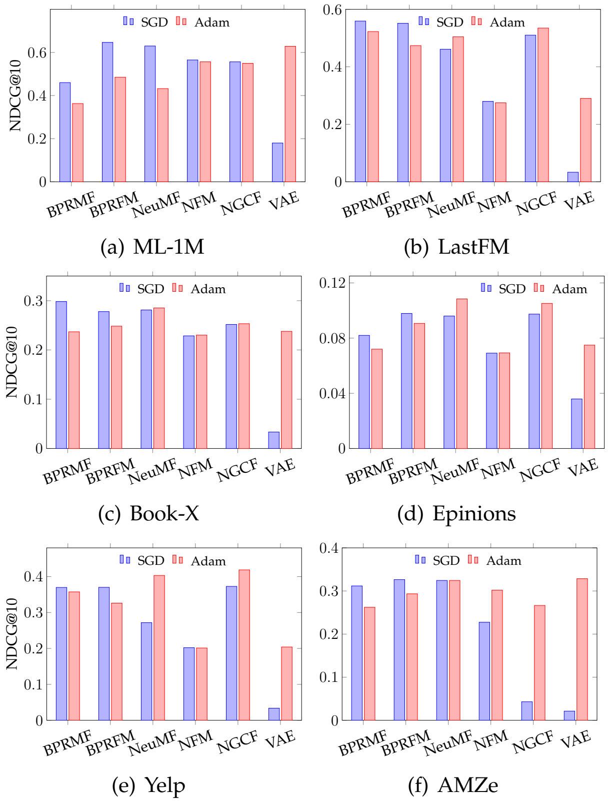
(a) ML-1M
(b) LastFM
(c) Book-X
(d) Epinions
(e) Yelp
(f) AMZe

Fig. 10. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter across the six datasets with different initializers.

uniform and normal distribution for initialization. As depicted in Figure 10, we can note that (1) for the two LFMs, i.e., BPRMF and BPRFM, initializer with normal distribution dramatically beats that with uniform distribution; and (2) for the four DLMs, some baselines (e.g., NGCF) gains better accuracy with uniform distribution than normal distribution on the six datasets; while some perform comparably with the two types of initializers, e.g., Multi-VAE, across the six datasets. In a nutshell, different parameter initializers produce different recommendation performance. With a proper initializer, the LFMs may easily beat DLMs, for example, BPRMF defeats DLMs on LastFM and Book-X.

3.3.4 Impacts of Model Optimizers

We further investigate the impacts of different optimizers on the final recommendation performance. In particular, we vary optimizers (i.e., SGD, and Adam) for the six baselines (i.e., BPRMF, BPRFM, NeuMF, NFM, NGCF and Multi-VAE) on the six datasets under 10-filter view with time-aware split-by-ratio. The results are presented in Figure 11, where we observe that a better performance is achieved via SGD in comparison with Adam for LFMs (i.e., BPRMF and BPRFM); whereas Adam generally outperforms SGD regarding DLMs (i.e., NeuMF, NFM, NGCF and Multi-VAE).



(a) ML-1M
(b) LastFM
(c) Book-X
(d) Epinions
(e) Yelp
(f) AMZe

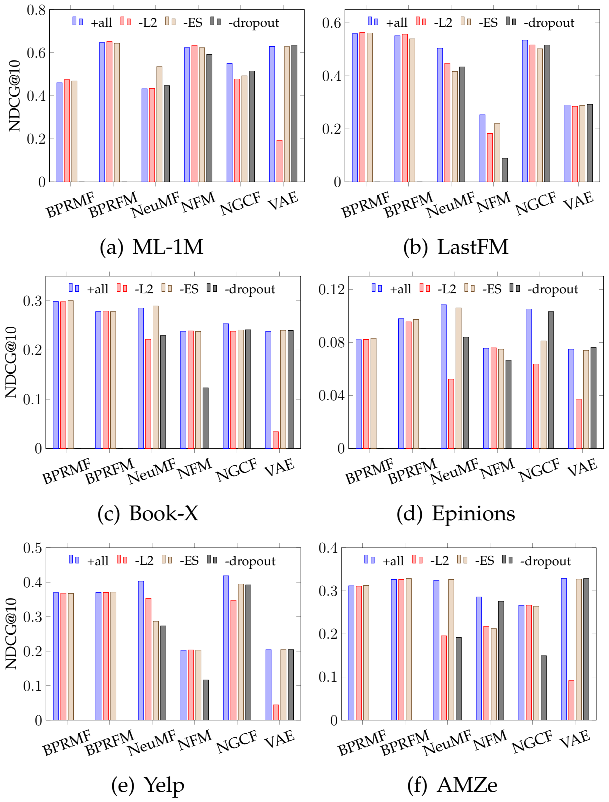
Fig. 11. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter across the six datasets with different optimizers.

3.3.5 Impacts of Strategies to Avoid Over-fitting

As illustrated in Section 2.3.5, regularization term, dropout and early-stop mechanism are widely adopted to avoid over-fitting. To verify their impacts, we compare the results of six baselines (BPRMF, BPRFM, NeuMF, NFM, NGCF and Multi-VAE) across the six datasets under 10-fiter view with time-aware split-by-ratio by removing these strategies. In particular, +all, -L2, -dropout and -ES respectively indicate the baseline with all over-fitting prevention strategies, variant without L2 regularization term, variant without dropout (only for deep learning baseline), and variant without early-stop. The results are displayed in Figure 12, where several observations can be noted. First, the overfitting prevention strategies generally facilitate to enhance the recommendation accuracy to some extent for all baselines across the six datasets. However, there are a few exceptions, e.g., NeuMF on ML-1M, indicating some of these strategies may also lead to the underfitting issue occasionally. Second, the impact of these strategies is more significant on DLMs (e.g., NeuMF) than LFMs (e.g., BPRMF). Lastly, the performance of DLMs may be remarkably affected by a certain strategy, e.g., NFM is heavily affected by dropout, whilst a major impact of L2 regularization term on Multi-VAE can be observed.

3.3.6 Impacts of Hyper-parameter Tuning Strategies

As illustrated in Section 2.3.6, a validation set should be held out for hyper-parameter tuning to avoid data leakage. To investigate its impact, we compare with the results by directly tuning hyper-parameters on the test set under 10-filter view with time-aware split-by-ratio. For simplicity, we only select four representative baselines on four datasets as displayed in Figure 13. Accordingly, we can easily find that in most cases directly tuning hyper-parameters on the test set indeed guarantees a better performance compared with tuning hyper-parameters on the validation set. This implies that the empirical results disclosed in previous studies without validation set might be overestimated.



(a) ML-1M

(b) LastFM

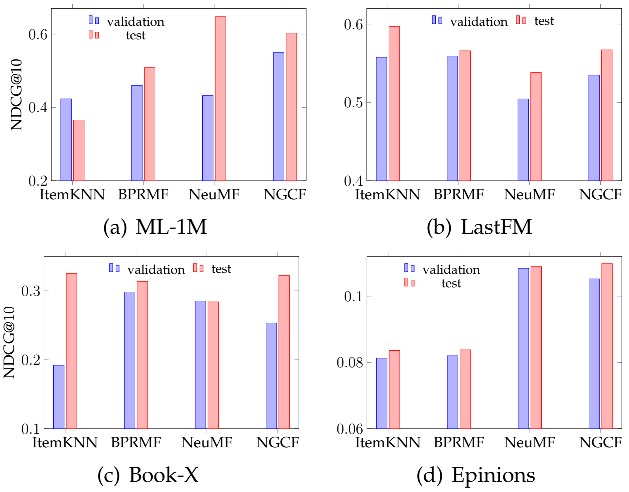
(c) Book-X

(d) Epinions

(e) Yelp

(f) AMZe

Fig. 12. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter across the six datasets with different strategies to avoid over-fitting.



(a) ML-1M

(b) LastFM

(c) Book-X

(d) Epinions

Fig. 13. Performance of baselines w.r.t. time-aware split-by-ratio on 10-filter across the six datasets by tuning on validation and test sets.

4 Benchmarking Recommendation

4.1 Standardized Procedures

Section 2 shows the hyper-factors in recommendation evaluation, and their impacts are empirically analyzed in Section 3. To achieve a rigorous evaluation, the **mixed mode** discussed in Section 2.4 is encouraged to be adopted. Accordingly, we propose a series of standardized procedures and correspondingly call for endeavors of all researchers, aiming to effectively enhance the standardization of recommendation evaluation. Regarding model-independent hyper-factors, five procedures are recommended.

- It is impossible to evaluate recommenders on all public datasets covering each domain. However, at least one widely-used dataset discussed in Section 2.2.1 should be considered, especially for the papers evaluated on the private datasets (e.g., confidential data from commercial companies). Otherwise, the results could not be easily reproduced by the subsequent studies.
- Section 3.2.1 verifies that different data pre-processing strategies impact the performance. Besides origin view, 5- and 10-filter views are recommended to ease the data sparsity issue, and a clear description on data pre-processing details is indispensable.
- For data splitting methods, both time-aware split-by-ratio and time-aware leave-one-out are recommended. With timestamp, the real recommendation scenario will be better simulated. W.r.t. split-by-ratio, both global- and user-level work well and $\rho = 80\%$ is recommended for a more feasible and convenient comparison.
- The representative baselines with different types (MMs, LFMs and DLMs) in Section 2.2.4 are recommended to be selected and compared. As shown in Section 3.2.1, the performance of different types of baselines vary a lot in different scenarios, that is, the MMs (e.g., MostPop) and simple LFMs (e.g., PureSVD) sometimes even perform better than DLMs (e.g., NeuMF). The more diverse baselines are compared, the more comprehensive and reliable the evaluation is.
- At least two of the six discussed metrics in Section 2.2.5 should be adopted, where one (e.g., Precision) measures whether a test item is present on the top-N recommendation list, and the other (e.g., NDCG) measures the ranking positions of the recommended items.

With respect to model-dependent hyper-factors, there are also five procedures recommended as below.

- For a fair comparison, it is better to evaluate all methods with a same type of objective functions and thus better positioning a proposed method's contributions.
- All the compared methods should adopt the same negative sampler, except the papers with the goal of proposing or studying different negative sampling strategies.
- The parameter initializer and model optimizer should be consistent across all compared methods as demonstrated in Section 3.3.3 and Section 3.3.4.
- The same basic overfitting prevention strategies should be applied to all compared methods, except the methods with specially-designed strategies, e.g., the message dropout in NGCF [102].
- With regards to the hyper-parameter tuning, a nested validation is mandatory, that is, retaining partial (e.g., 10%) training data as validation set. Bayesian HyperOpt, as a more intelligent parameter searching strategy, is recommended, and the search space should be kept the same for the shared parameters of different baselines. The number of trails (we set 30 in this study by following [6]) may be increased for further performance improvements. Most importantly, the optimal parameter settings should be well reported for reproduction.

Meanwhile, the source codes and datasets for each publication should be available for reproduction [173]. The conference venues could make them as necessities, measure the quality, and even require a short code demonstration along with each accepted paper during the conference.

4.2 Performance of Baselines

With the goal of providing a better reference for fair comparison, Tables 10-15 (Appendix) show the performance of ten baselines across six metrics on the six datasets under three different views (i.e., origin, 5-filter and 10-filter) with time-aware split-by-ratio ($N = 10$). Due to space limitation, other results (e.g., $N = 1, 5, 20, 30, 50$ and other data splitting methods) are on our GitHub. All optimal hyper-parameters are found by Bayesian HyperOpt to optimize NDCG@10 for 30 trials (see Section 3.2.1), and the corresponding detailed parameter settings are shown in Tables 16-19.

Based on the results, several major observations can be noted. **(1)** BPRFM achieves the best performance on ML-1M across all views. **(2)** Regarding LastFM, ItemKNN/NGCF performs the best on origin and 5-filter views, while SLIM achieves the best performance on 10-filter view. **(3)** For Book-X, BPRFM/NGCF and PureSVD/NGCF respectively beat other baselines on origin and 5-filter views, and PureSVD is the winner on 10-filter view. **(4)** W.r.t. Epinions, NeuMF obtains the highest accuracy on origin view; and NeuMF/NGCF helps reach the best performance on 5- and 10-filter views. **(5)** On Yelp, the best performance on the origin, 5- and 10-filter views are respectively gained by BPRFM, NGCF and NGCF. **(6)** With regards to AMZe, NeuMF/Multi-VAE defeat other baselines on origin view; BPRFM/NFM obtains the optimal results on 5-filter view; and Multi-VAE is the top method on 10-filter view.

5 Related Work

While long been recognized as a key feature of scientific discoveries, reproducibility has been increasingly characterized as a crisis recently [174], [175], [176]. It is becoming a primary concern in computer and information science, evidenced by the recently developed ACM policy on Artifact Review and Badging[11] and emerging efforts including seminars [178], workshops [179], reproducibility checklist[12] [180], and focused tracks at major conferences, such as ECIR [181], ACM MM [182], SIGIR[13], and ISWC [183]. Specific to recommender systems research, besides the reproducibility track starting from 2020 on the premier conference for recommender systems – RecSys [184], the discussions have been concentrated on the fairness of comparison between newly proposed and baseline methods [5], [6]. In very recent work, Dacrema et al. [6] find neural models hardly outperform fine-tuned memory- and latent factor-based methods, a similar finding also discovered by Rendle et al. [5].

Despite the importance, improving reproducibility in recommender systems research is highly challenging due to the many influential evaluation factors for recommendation performance. Said et al. [2] find large differences in the effectiveness of recommendation methods across different implementation frameworks as well as across evaluation datasets and metrics. A companion toolkit RiVal [3] was released to allow for the control of data splitting and evaluation metrics, while Elliot [185] further improves it by implementing more baselines and incorporating statistic significance tests. Beel et al. [186] find a similar phenomenon in news and research paper recommendation and identify influential factors such as user characteristics and time of recommendation. Valcarce et al. [83] specifically study the properties of evaluation metrics for item ranking, marking precision as the most robust and NDCG presenting the highest discriminative power. More recently, Rendle et al. [5] demonstrate the importance of hyperparameter search in baseline methods, e.g., matrix factorization, and stress the need for standardized benchmarks where methods should be extensively tuned for fair comparison. Sachdeva et al. [187] specifically examine the impact of dataset sampling strategies on model performance, and indicate that sampling methods, including the random ways do matter with regard to final performance of recommendation algorithms.

Existing benchmarks are, however, either restricted to pre-neural methods [2], a single evaluation factor [83], or rating prediction [5] which has been discouraged as a way to formulate the recommendation problem [188]; besides, most of the existing benchmarks consider two or three datasets (including [6]), ignoring the richness of available datasets often chosen by newly published work. The two most recent work, RecBole [154] and Elliot [185], has partially alleviated the aforementioned issues by implementing more baselines (neural ones included), considering varied datasets and recommendation scenarios (e.g., temporal and context-aware ones), and incorporating hyper-parameter optimization strategies. However, similar to other recommender system libraries (e.g., Librec [152], MyMediaLite [189], and Surprise [190]), they strive to provide a unified framework for developing and reproducing algorithms for different scenarios in terms of different evaluation metrics.

Instead, aimed for a full treatment of evaluation issues, our work takes a bottom-up approach analyzing an extensive amount of literature to search for and summarize important evaluation factors, denoted as *hyper-factors* (categorized as model-dependent and model-independent ones), which might influence model performance in model evaluation, towards the goal of performing rigorous evaluation. We further present a benchmark supported by an empirical study at a bigger-than-ever scale with the hope of laying a strong foundation for future research.

6 Conclusion

This paper aims to benchmark recommendation for reproducible evaluation and fair comparison from the angles of both practical theory analysis and empirical study. Regarding theory analysis, 141 recommendation papers published in the four recent years (2017-2020) from eight top tier conferences have been systematically reviewed, whereby we define and extract the hyper-factors affecting recommendation evaluation, classied into model-independent (e.g., dataset splitting methods) and -dependent (e.g., loss function design) factors. Accordingly, different modes for rigorous evaluation are defined and discussed in-depth. To support the empirical study, a user-friendly Python toolkit – DaisyRec 2.0 has been released and updated by seamlessly accommodating the extracted hyper-factors. Thereby, the impacts of different hyper-factors on evaluation are then empirically examined and comprehensively analyzed.

[11] www.acm.org/publications/policies/artifact-review-badging; see also SIGIR's implementation of the policy [177].

[12] aaai.org/Conferences/AAAI-22/reproducibility-checklist/.

[13] sigir.org/sigir2022/call-for-reproducibility-track-papers/.

Lastly, we create benchmarks for rigorous evaluation by proposing standardized procedures and providing the performance of ten well-tuned state-of-the-art algorithms on six widely-used datasets across six metrics as a reference for later study. For the future work, we plan to deepen our investigation by, for example, diving into more diverse (e.g., session/sequential-aware) recommendation tasks, and more evaluation metrics (e.g., diversity, novelty and serendipity).

References

[1] Z. Sun *et al.*, “Research commentary on recommendations with side information: A survey and research directions,” *ECRA*, vol. 37, p. 100879, 2019.

[2] A. Said and A. Bellogín, “Comparative recommender system evaluation: benchmarking recommendation frameworks,” in *RecSys*, 2014, pp. 129–136.

[3] ——, “Rival: a toolkit to foster reproducibility in recommender system evaluation,” in *RecSys*, 2014, pp. 371–372.

[4] Z. Sun *et al.*, “Are we evaluating rigorously? benchmarking recommendation for reproducible evaluation and fair comparison,” in *RecSys*, 2020, pp. 23–32.

[5] S. Rendle *et al.*, “On the difficulty of evaluating baselines: A study on recommender systems,” in *arXiv preprint arXiv:1905.01395*, 2019.

[6] M. F. Dacrema *et al.*, “Are we really making much progress? a worrying analysis of recent neural recommendation approaches,” in *RecSys*, 2019, pp. 101–109.

[7] B. Sarwar *et al.*, “Item-based collaborative filtering recommendation algorithms,” in *WWW*, 2001, pp. 285–295.

[8] Z. Sun *et al.*, “Exploiting both vertical and horizontal dimensions of feature hierarchy for effective recommendation,” in *AAAI*, 2017, pp. 189–195.

[9] D. Li *et al.*, “Ermma: Expected risk minimization for matrix approximation-based recommender systems,” in *AAAI*, 2017, pp. 1403–1409.

[10] L. Yu *et al.*, “Walkranker: A unified pairwise ranking model with multiple relations for item recommendation,” in *AAAI*, 2018, pp. 2596–2603.

[11] M. Wang *et al.*, “Collaborative filtering with social exposure: A modular approach to social recommendation,” in *AAAI*, 2018, pp. 2516–2523.

[12] T. D. T. Do and L. Cao, “Coupled poisson factorization integrated with user/item metadata for modeling popular and sparse ratings in scalable recommendation,” in *AAAI*, 2018, pp. 2918–2925.

[13] J. Zhang *et al.*, “Hierarchical reinforcement learning for course recommendation in moocs,” in *AAAI*, 2019, pp. 435–442.

[14] C. Wang *et al.*, “Camo: A collaborative ranking method for content based recommendation,” in *AAAI*, 2019, pp. 5224–5231.

[15] C. Lin *et al.*, “Non-compensatory psychological models for recommender systems,” in *AAAI*, 2019, pp. 4304–4311.

[16] J. Li *et al.*, “From zero-shot learning to cold-start recommendation,” in *AAAI*, 2019, pp. 4189–4196.

[17] C. Liu *et al.*, “Discrete social recommendation,” in *AAAI*, 2019, pp. 208–215.

[18] Z.-H. Deng *et al.*, “Deepcf: A unified framework of representation learning and matching function learning in recommender system,” in *AAAI*, 2019, pp. 61–68.

[19] L. Hu *et al.*, “Hers: Modeling influential contexts with heterogeneous relations for sparse and cold-start recommendation,” in *AAAI*, 2019, pp. 3830–3837.

[20] X. Wang *et al.*, “Explainable reasoning over knowledge graphs for recommendation,” in *AAAI*, 2019, pp. 5329–5336.

[21] T. Shen *et al.*, “Peia: Personality and emotion integrated attentive model for music recommendation on social media platforms,” in *AAAI*, 2020, pp. 206–213.

[22] Q. Zhu *et al.*, “A knowledge-aware attentional reasoning network for recommendation,” in *AAAI*, 2020, pp. 6999–7006.

[23] C. Chen *et al.*, “Efficient heterogeneous collaborative filtering without negative sampling for recommendation,” in *AAAI*, 2020, pp. 19–26.

[24] G. Guo *et al.*, “Leveraging title-abstract attentive semantics for paper recommendation,” in *AAAI*, 2020, pp. 67–74.

[25] M. Li *et al.*, “Symmetric metric learning with adaptive margin for recommendation,” in *AAAI*, 2020, pp. 4634–4641.

[26] Y. Xu *et al.*, “Multi-feature discrete collaborative filtering for fast cold-start recommendation,” in *AAAI*, 2020, pp. 270–278.

[27] J. Chen *et al.*, “Fast adaptively weighted matrix factorization for recommendation with implicit feedback,” in *AAAI*, 2020, pp. 3470–3477.

[28] D. D. Le and H. W. Lauw, “Stochastically robust personalized ranking for lsh recommendation retrieval,” in *AAAI*, 2020, pp. 4594–4601.

[29] C. Wang *et al.*, “Setrank: A setwise bayesian approach for collaborative ranking from implicit feedback,” in *AAAI*, 2020, pp. 6127–6136.

[30] Y. Zhang *et al.*, “Joint representation learning for top-n recommendation with heterogeneous information sources,” in *CIKM*, 2017, pp. 1449–1458.

[31] D. D. Le and H. W. Lauw, “Indexable bayesian personalized ranking for efficient top-k recommendation,” in *CIKM*, 2017, pp. 1389–1398.

[32] W. Pei *et al.*, “Interacting attention-gated recurrent networks for recommendation,” in *CIKM*, 2017, pp. 1459–1468.

[33] L. Mei *et al.*, “An attentive interaction network for context-aware recommendations,” in *CIKM*, 2018, pp. 157–166.

[34] H. Wang *et al.*, “Ripplenet: Propagating user preferences on the knowledge graph for recommender systems,” in *CIKM*, 2018, pp. 417–426.

[35] T. Tran *et al.*, “Regularizing matrix factorization with user and item embeddings for recommendation,” in *CIKM*, 2018, pp. 687–696.

[36] J. Ma *et al.*, “Dbrec: Dual-bridging recommendation via discovering latent groups,” in *CIKM*, 2019, pp. 1513–1522.

[37] W.-C. Kang and J. McAuley, “Candidate generation with binary codes for large-scale top-n recommendation,” in *CIKM*, 2019, pp. 1523–1532.

[38] F. Xu *et al.*, “Relation-aware graph convolutional networks for agent-initiated social e-commerce recommendation,” in *CIKM*, 2019, pp. 529–538.

[39] B. Chang *et al.*, “Learning graph-based geographical latent representation for point-of-interest recommendation,” in *CIKM*, 2020, pp. 135–144.

[40] B. Chen *et al.*, “Tgcn: Tag graph convolutional network for tag-aware recommendation,” in *CIKM*, 2020, pp. 155–164.

[41] D. Lee *et al.*, “News recommendation with topic-enriched knowledge graphs,” in *CIKM*, 2020, pp. 695–704.

[42] R. Sun *et al.*, “Multi-modal knowledge graphs for recommender systems,” in *CIKM*, 2020, pp. 1405–1414.

[43] S. Kang *et al.*, “De-rrd: A knowledge distillation framework for recommender system,” in *CIKM*, 2020, pp. 605–614.

[44] Y.-N. Chuang *et al.*, “Tpr: Text-aware preference ranking for recommender systems,” in *CIKM*, 2020, pp. 215–224.

[45] Z. Xu *et al.*, “E-commerce recommendation with weighted expected utility,” in *CIKM*, 2020, pp. 1695–1704.

[46] Y. Wang *et al.*, “Disenhan: Disentangled heterogeneous graph attention network for recommendation,” in *CIKM*, 2020, pp. 1605–1614.

[47] Y. Xian *et al.*, “Cafe: coarse-to-fine knowledge graph reasoning for e-commerce recommendation,” in *CIKM*, 2020, pp. 1645–1654.

[48] F. Yuan *et al.*, “Exploring missing interactions: A convolutional generative adversarial network for collaborative filtering,” in *CIKM*, 2020, pp. 1773–1782.

[49] F. Zhao and Y. Guo, “Learning discriminative recommendation systems with side information,” in *IJCAI*, 2017, pp. 3469–3475.

[50] Z. Sun *et al.*, “Mrlr: Multi-level representation learning for personalized ranking in recommendation,” in *IJCAI*, 2017, pp. 2807–2813.

[51] H.-J. Xue *et al.*, “Deep matrix factorization models for recommender systems,” in *IJCAI*, 2017, pp. 3203–3209.

[52] Y. Liu *et al.*, “Dynamic bayesian logistic matrix factorization for recommendation with implicit feedback,” in *IJCAI*, 2018, pp. 3463–3469.

[53] Z. Wang *et al.*, “Matrix completion with preference ranking for top-n recommendation,” in *IJCAI*, 2018, pp. 3585–3591.

[54] W. Zhao *et al.*, “Plastic: Prioritize long and short-term information in top-n recommendation using adversarial training,” in *IJCAI*, 2018, pp. 3676–3682.

[55] J. Ding *et al.*, “Improving implicit recommender systems with view data,” in *IJCAI*, 2018, pp. 3343–3349.

[56] W. Cheng *et al.*, “Delf: A dual-embedding based deep latent factor model for recommendation,” in *IJCAI*, 2018, pp. 3329–3335.

[57] H. Liu *et al.*, "Discrete factorization machines for fast feature-based recommendation," in *IJCAI*, 2018, pp. 3449–3455.

[58] X. Xin *et al.*, "Cfm: convolutional factorization machines for context-aware recommendation," in *IJCAI*, 2019, pp. 3926–3932.

[59] J. Jiang *et al.*, "Convolutional gaussian embeddings for personalized recommendation with uncertainty," in *IJCAI*, 2019, pp. 2642–2648.

[60] G. Guo *et al.*, "Discrete trust-aware matrix factorization for fast recommendation," in *IJCAI*, 2019, pp. 1380–1386.

[61] Y. Xu *et al.*, "Learning shared vertex representation in heterogeneous graphs with convolutional networks for recommendation," in *IJCAI*, 2019, pp. 4620–4626.

[62] S. Zhang *et al.*, "Quaternion collaborative filtering for recommendation," in *IJCAI*, 2019, pp. 4313–4319.

[63] W. Fan *et al.*, "Deep adversarial social recommendation," in *IJCAI*, 2019, pp. 1351–1357.

[64] Z. Wang *et al.*, "Unified embedding model over heterogeneous information network for personalized recommendation," in *IJCAI*, 2019, pp. 3813–3819.

[65] P. Han *et al.*, "Contextualized point-of-interest recommendation," in *IJCAI*, 2020, pp. 2484–2490.

[66] R. Xie *et al.*, "Internal and contextual attention network for cold-start multi-channel matching in recommendation," in *IJCAI*, 2020, pp. 2732–2738.

[67] H. Chen and J. Li, "Neural tensor model for learning multi-aspect factors in recommender systems," in *IJCAI*, 2020, pp. 2449–2455.

[68] R. Liu *et al.*, "Hypernews: Simultaneous news recommendation and active-time prediction via a double-task deep neural network," in *IJCAI*, 2020, pp. 3487–3493.

[69] X. Li and J. She, "Collaborative variational autoencoder for recommender systems," in *KDD*, 2017, pp. 305–314.

[70] H. Zhu *et al.*, "Learning tree-based deep model for recommender systems," in *KDD*, 2018, pp. 1079–1088.

[71] E. Christakopoulou and G. Karypis, "Local latent space models for top-n recommendation," in *KDD*, 2018, pp. 1235–1243.

[72] B. Hu *et al.*, "Leveraging meta-path based context for top-n recommendation with a neural co-attention model," in *KDD*, 2018, pp. 1531–1540.

[73] X. Wang *et al.*, "Kgat: Knowledge graph attention network for recommendation," in *KDD*, 2019, pp. 950–958.

[74] X. Tang *et al.*, "Akupm: Attention-enhanced knowledge-aware user preference model for recommendation," in *KDD*, 2019, pp. 1891–1899.

[75] J. Zhao *et al.*, "Intentgc: A scalable graph convolution framework fusing heterogeneous information for recommendation," in *KDD*, 2019, pp. 2347–2357.

[76] H. Wang *et al.*, "Knowledge-aware graph neural networks with label smoothness regularization for recommender systems," in *KDD*, 2019, pp. 968–977.

[77] Y. Chen *et al.*, "Lambdaopt: Learn to regularize recommender models in finer levels," in *KDD*, 2019, pp. 978–986.

[78] J. Jin *et al.*, "An efficient neighborhood-based interaction model for recommendation on heterogeneous graph," in *KDD*, 2020, pp. 75–84.

[79] C. Ma *et al.*, "Probabilistic metric learning with adaptive margin for top-k recommendation," in *KDD*, 2020, pp. 1036–1044.

[80] S. Ji *et al.*, "Dual channel hypergraph collaborative filtering," in *KDD*, 2020, pp. 2020–2029.

[81] R. Otunba *et al.*, "Mpr: Multi-objective pairwise ranking," in *RecSys*, 2017, pp. 170–178.

[82] D. Rafailidis and F. Crestani, "Learning to rank with trust and distrust in recommender systems," in *RecSys*, 2017, pp. 5–13.

[83] D. Valcarce *et al.*, "On the robustness and discriminative power of information retrieval metrics for top-n recommendation," in *RecSys*, 2018, pp. 260–268.

[84] Z. Sun *et al.*, "Recurrent knowledge graph embedding for effective recommendation," in *RecSys*, 2018, pp. 297–305.

[85] L. Zheng *et al.*, "Spectral collaborative filtering," in *RecSys*, 2018, pp. 311–319.

[86] S. Ouyang *et al.*, "Asymmetric bayesian personalized ranking for one-class collaborative filtering," in *RecSys*, 2019, pp. 373–377.

[87] H. Liu *et al.*, "Deep generative ranking for personalized recommendation," in *RecSys*, 2019, pp. 34–42.

[88] A. N. Nikolakopoulos *et al.*, "Personalized diffusions for top-n recommendation," in *RecSys*, 2019, pp. 260–268.

[89] F. S. d. Costa and P. Dolog, "Collective embedding for neural context-aware recommender systems," in *RecSys*, 2019, pp. 201–209.

[90] E. Frolov and I. Oseledets, "Hybridsvd: when collaborative information is not enough," in *RecSys*, 2019, pp. 331–339.

[91] E. Elahi *et al.*, "Variational low rank multinomials for collaborative filtering with side-information," in *RecSys*, 2019, pp. 340–347.

[92] Y. Zhang *et al.*, "Content-collaborative disentanglement representation learning for enhanced recommendation," in *RecSys*, 2020, pp. 43–52.

[93] D. Liu *et al.*, "Kred: Knowledge-aware document representation for news recommendations," in *RecSys*, 2020, pp. 200–209.

[94] J. P. Zhou *et al.*, "Tafa: Two-headed attention fused autoencoder for context-aware recommendations," in *RecSys*, 2020, pp. 338–347.

[95] J. Chen *et al.*, "Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention," in *SIGIR*, 2017, pp. 335–344.

[96] X. He *et al.*, "Adversarial personalized ranking for recommendation," in *SIGIR*, 2018, pp. 355–364.

[97] T. Ebesu *et al.*, "Collaborative memory network for recommendation systems," in *SIGIR*, 2018, pp. 515–524.

[98] Q. Xu *et al.*, "Graphcar: Content-aware multimedia recommendation with graph autoencoder," in *SIGIR*, 2018, pp. 981–984.

[99] R. Cañamares and P. Castells, "Should i follow the crowd?: A probabilistic analysis of the effectiveness of popularity in recommender systems," in *SIGIR*, 2018, pp. 415–424.

[100] W. Wang *et al.*, "Streaming ranking based recommender systems," in *SIGIR*, 2018, pp. 525–534.

[101] Y. Chen *et al.*, "Bayesian personalized feature interaction selection for factorization machines," in *SIGIR*, 2019, pp. 665–674.

[102] X. Wang *et al.*, "Neural graph collaborative filtering," in *SIGIR*, 2019, pp. 165–174.

[103] G. Wu *et al.*, "Noise contrastive estimation for one-class collaborative filtering," in *SIGIR*, 2019, pp. 135–144.

[104] X. Xin *et al.*, "Relational collaborative filtering: Modeling multiple item relations for recommendation," in *SIGIR*, 2019, pp. 125–134.

[105] Z. Wang *et al.*, "Ckan: Collaborative knowledge-aware attentive network for recommender systems," in *SIGIR*, 2020, pp. 219–228.

[106] C. Chen *et al.*, "Jointly non-sampling learning for knowledge graph enhanced recommendation," in *SIGIR*, 2020, pp. 189–198.

[107] J. Gong *et al.*, "Attentional graph convolutional networks for knowledge concept recommendation in moocs in a heterogeneous view," in *SIGIR*, 2020, pp. 79–88.

[108] C. Hansen *et al.*, "Content-aware neural hashing for cold-start recommendation," in *SIGIR*, 2020, pp. 971–980.

[109] X. He *et al.*, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *SIGIR*, 2020, pp. 639–648.

[110] S. Shi *et al.*, "Beyond user embedding matrix: Learning to hash for modeling large-scale users in recommendation," in *SIGIR*, 2020, pp. 319–328.

[111] C.-Y. Tai *et al.*, "Mvin: Learning multiview items for recommendation," in *SIGIR*, 2020, pp. 99–108.

[112] L. Wu *et al.*, "Joint item recommendation and attribute inference: An adaptive graph convolutional network approach," in *SIGIR*, 2020, pp. 679–688.

[113] D.-K. Chae *et al.*, "Ar-cf: Augmenting virtual users and items in collaborative filtering for addressing cold-start problems," in *SIGIR*, 2020, pp. 1251–1260.

[114] X. Wang *et al.*, "Disentangled graph collaborative filtering," in *SIGIR*, 2020, pp. 1001–1010.

[115] L. Zou *et al.*, "Neural interactive collaborative filtering," in *SIGIR*, 2020, pp. 749–758.

[116] J. Sun *et al.*, "Neighbor interaction aware graph convolution networks for recommendation," in *SIGIR*, 2020, pp. 1289–1298.

[117] Q. Zhao *et al.*, "Multi-product utility maximization for economic recommendation," in *WSDM*, 2017, pp. 435–443.

[118] Y. Zhang *et al.*, "Discrete deep learning for fast content-aware recommendation," in *WSDM*, 2018, pp. 717–726.

[119] Z. Jiang *et al.*, "Recommendation in heterogeneous information networks based on generalized random walk model and bayesian personalized ranking," in *WSDM*, 2018, pp. 288–296.

[120] W. Niu *et al.*, "Neural personalized ranking for image recommendation," in *WSDM*, 2018, pp. 423–431.

[121] C. Ma *et al.*, "Gated attentive-autoencoder for content-aware recommendation," in *WSDM*, 2019, pp. 519–527.

[122] A. N. Nikolakopoulos and G. Karypis, "Recwalk: Nearly uncoupled random walks for top-n recommendation," in *WSDM*, 2019, pp. 150–158.

[123] C. Chen *et al.*, "Social attentional memory network: Modeling aspect-and friend-level differences in recommendation," in *WSDM*, 2019, pp. 177–185.

[124] D. Liu *et al.*, "Spiral of silence in recommender systems," in *WSDM*, 2019, pp. 222–230.

[125] H. Steck *et al.*, "Admm slim: Sparse recommendations for many users," in *WSDM*, 2020, pp. 555–563.

[126] R. Li *et al.*, "Adversarial learning to compare: Self-attentive prospective customer recommendation in location based social networks," in *WSDM*, 2020, pp. 349–357.

[127] F. Liu *et al.*, "End-to-end deep reinforcement learning based recommendation with supervised embedding," in *WSDM*, 2020, pp. 384–392.

[128] Y. Gu *et al.*, "Hierarchical user profiling for e-commerce recommender systems," in *WSDM*, 2020, pp. 223–231.

[129] J. Wang *et al.*, "Key opinion leaders in recommendation systems: Opinion elicitation and diffusion," in *WSDM*, 2020, pp. 636–644.

[130] C. Sun *et al.*, "Lara: Attribute-to-feature adversarial learning for new-item recommendation," in *WSDM*, 2020, pp. 582–590.

[131] H. Zamani and W. B. Croft, "Learning a joint search and recommendation model from user-item interactions," in *WSDM*, 2020, pp. 717–725.

[132] I. Shenbin *et al.*, "Recvae: A new variational autoencoder for top-n recommendations with implicit feedback," in *WSDM*, 2020, pp. 528–536.

[133] C.-K. Hsieh *et al.*, "Collaborative metric learning," in *WWW*, 2017, pp. 193–201.

[134] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *SIGIR*, 2017, pp. 355–364.

[135] W. Yu *et al.*, "Aesthetic-based clothing recommendation," in *WWW*, 2018, pp. 649–658.

[136] D. Liang *et al.*, "Variational autoencoders for collaborative filtering," in *WWW*, 2018, pp. 689–698.

[137] H. Wang *et al.*, "Multi-task feature learning for knowledge graph enhanced recommendation," in *WWW*, 2019, pp. 2000–2010.

[138] W. Ma *et al.*, "Jointly learning explainable rules for recommendation with knowledge graph," in *WWW*, 2019, pp. 1210–1221.

[139] Y. Cao *et al.*, "Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences," in *WWW*, 2019, pp. 151–161.

[140] T. Tran *et al.*, "Signed distance-based deep memory recommender," in *WWW*, 2019, pp. 1841–1852.

[141] H. Wang *et al.*, "Knowledge graph convolutional networks for recommender systems," in *WWW*, 2019, pp. 3307–3313.

[142] C.-M. Chen *et al.*, "Collaborative similarity embedding for recommender systems," in *WWW*, 2019, pp. 2637–2643.

[143] F. Khawar *et al.*, "Learning the structure of auto-encoding recommenders," in *WWW*, 2020, pp. 519–529.

[144] H. Liu *et al.*, "Deep global and local generative model for recommendation," in *WWW*, 2020, pp. 551–561.

[145] A. Javari *et al.*, "Weakly supervised attention for hashtag recommendation using graph data," in *WWW*, 2020, pp. 1038–1048.

[146] C. Wang *et al.*, "Personalized employee training course recommendation with career development awareness," in *WWW*, 2020, pp. 1648–1659.

[147] Q. Tan *et al.*, "Learning to hash with graph neural networks for recommender systems," in *WWW*, 2020, pp. 1988–1998.

[148] C. Chen *et al.*, "Efficient non-sampling factorization machines for optimal context-aware recommendation," in *WWW*, 2020, pp. 2400–2410.

[149] Y. Koren *et al.*, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[150] S. Rendle *et al.*, "Bpr: Bayesian personalized ranking from implicit feedback," in *IUI*, 2009, pp. 452–461.

[151] B. Hidasi *et al.*, "Parallel recurrent neural network architectures for feature-rich session-based recommendations," in *RecSys*, 2016, pp. 241–248.

[152] G. Guo *et al.*, "Librec: A java library for recommender system," in *Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd Conference on UMAP*, 2015.

[153] S. Zhang *et al.*, "Deeprec: An open-source toolkit for deep learning based recommendation," in *arXiv preprint arXiv:1905.10536*, 2019.

[154] W. X. Zhao *et al.*, "Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms," in *CIKM*, 2021, pp. 4653–4664.

[155] P. Cremonesi *et al.*, "Performance of recommender algorithms on top-n recommendation tasks," in *RecSys*, 2010, pp. 39–46.

[156] I. Cantador *et al.*, "The 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec)," in *RecSys*, 2011, pp. 387–388.

[157] J. Tang *et al.*, "eTrust: Understanding trust evolution in an online world," in *KDD*, 2012, pp. 253–261.

[158] C.-N. Ziegler *et al.*, "Improving recommendation lists through topic diversification," in *WWW*, 2005, pp. 22–32.

[159] S. Rendle, "Factorization machines," in *ICDM*, 2010, pp. 995–1000.

[160] Y. Hu *et al.*, "Collaborative filtering for implicit feedback datasets," in *ICDM*, 2008, pp. 263–272.

[161] X. Ning and G. Karypis, "Slim: Sparse linear methods for top-n recommender systems," in *ICDM*, 2011, pp. 497–506.

[162] X. He *et al.*, "Fast matrix factorization for online recommendation with implicit feedback," in *SIGIR*, 2016, pp. 549–558.

[163] H. Zhang *et al.*, "Discrete collaborative filtering," in *SIGIR*, 2016, pp. 325–334.

[164] F. Zhang *et al.*, "Collaborative knowledge base embedding for recommender systems," in *KDD*, 2016, pp. 353–362.

[165] Y. Wu *et al.*, "Collaborative denoising auto-encoders for top-n recommender systems," in *WSDM*, 2016, pp. 153–162.

[166] Q. Zhao *et al.*, "Interpreting user inaction in recommender systems," in *RecSys*, 2018, pp. 40–48.

[167] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.

[168] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.

[169] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *JMLR*, vol. 13, no. 1, pp. 281–305, 2012.

[170] J. Snoek *et al.*, "Practical bayesian optimization of machine learning algorithms," in *NIPS*, 2012, pp. 2960–2968.

[171] L. Yang *et al.*, "Openrec: A modular framework for extensible and adaptable recommendation algorithms," in *WSDM*, 2018, pp. 664–672.

[172] D. Jannach and M. Ludewig, "When recurrent neural networks meet the neighborhood for session-based recommendation," in *RecSys*, 2017, pp. 306–310.

[173] E. Raff, "A step toward quantifying independently reproducible machine learning research," in *NIPS*, 2019.

[174] O. S. Collaboration *et al.*, "Estimating the reproducibility of psychological science," *Science*, vol. 349, no. 6251, 2015.

[175] M. Baker, "Reproducibility crisis," *Nature*, vol. 533, no. 26, pp. 353–66, 2016.

[176] M. R. Munafò *et al.*, "A manifesto for reproducible science," *Nature Human Behaviour*, vol. 1, no. 1, pp. 1–9, 2017.

[177] N. Ferro and D. Kelly, "SIGIR initiative to implement acm artifact review and badging," vol. 52, no. 1, pp. 4–10, 2018.

[178] J. Freire *et al.*, "Report from dagstuhl seminar 16041: reproducibility of data-oriented experiments in e-science," *Dagstuhl Reports*, vol. 6, no. 1, pp. 108–159, 2016.

[179] R. Clancy *et al.*, "Overview of the 2019 open-source ir replicability challenge (osirrc 2019)," in *OSIRRC@SIGIR*, 2019, pp. 1–7.

[180] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle, "Improving reproducibility in machine learning research: a report from the neurips 2019 reproducibility program," *JMLR*, vol. 22, 2021.

[181] A. Hanbury *et al.*, *Advances in information retrieval: 37th european conference on IR research, ECIR 2015, Vienna, Austria, March 29-April 2, 2015. Proceedings*. Springer, 2015, vol. 9022.

[182] *MM '21: Proceedings of the 29th ACM International Conference on Multimedia*, New York, NY, USA, 2021.

[183] A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, and H. Alani, *The Semantic Web–ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings*. Springer Nature, 2021, vol. 12922.

[184] *RecSys '20: Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020.

[185] V. W. Anelli, A. Bellogín, A. Ferrara, D. Malitesta, F. A. Merra, C. Pomo, F. M. Donini, and T. Di Noia, "Elliot: a comprehensive

and rigorous framework for reproducible recommender systems evaluation," in *SIGIR*, 2021.

[186] J. Beel *et al.*, "Towards reproducibility in recommender-systems research," *UMUAI*, vol. 26, no. 1, pp. 69–101, 2016.

[187] N. Sachdeva, C.-J. Wu, and J. McAuley, "On sampling collaborative filtering datasets," in *WSDM*, 2022.

[188] S. M. McNee *et al.*, "Being accurate is not enough: how accuracy metrics have hurt recommender systems," in *SIGCHI*, 2006, pp. 1097–1101.

[189] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Mymedialite: A free recommender system library," in *RecSys*, 2011, pp. 305–308.

[190] N. Hug, "Surprise: A python library for recommender systems," *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020.

[191] B. McFee *et al.*, "The million song dataset challenge," in *WWW*, 2012, pp. 909–916.

Appendix

TABLE 6
Representative datasets in various domains for recommendation, where ‘SNs’ and ‘LBSNs’ are short for social networks and the location-based social networks, respectively.

Domain	Representative Datasets
Movie	MovieLens (100K/1M/10M/20M/25M/Latest), Netflix, Amazon, FilmTrust, Douban, Yahoo!Movie, Flixster, CiaoDVD, EachMovie, IMDB, Watcha [113]
Consumable	Amazon (Clothing, Home, Office, Sports, Electronic, Cell Phones, Beauty, Food, Health, Toy, Baby), Taobao, Beibei, Tmall, Alibaba, Retailrocket, Flipkart [37]
Music	Last.fm, Yahoo!Music, Douban, Amazon-CDs, KKBox, Kollect.fm, EchoNest, MSD [191], TasteProfile [35]
Book	Amazon-Book/Kindle, Douban, Dianping, IntentBooks, Book-Crossing, LibraryThing, GoodReads
News	Bing-News, Baidu-News, Medium, Adressa-News [68], Microsoft News [93]
SNs	Epinions, Delicious, Douban, Last.fm, Yelp, Ciao, Xing, FilmTrust, Twitter, Wechat, Weibo
LBSNs	Foursquare, Yelp, Gowalla, BrightKite
Paper	Aminer, CiteULike, PRSDataset [24]
Image	Pinterest, Flickr, Aesthetic Visual Analysis [135]

TABLE 7
The abbreviation and full names of the eight conferences.

Abbre.	Full Names
AAAI	AAAI Conference on Artificial Intelligence
CIKM	Conference on Information and Knowledge Management
IJCAI	International Joint Conference on Artificial Intelligence
KDD	SIGKDD Conference on Knowledge Discovery and Data Mining
RecSys	ACM Conference on Recommender System
SIGIR	SIGIR Conference on Research and Development in Information Retrieval
WSDM	International Conference on Web Search and Data Mining
WWW	The ACM Web Conference

TABLE 8
The equations for the six evaluation metrics.

$$Precision@N = \frac{1}{|\mathcal{U}|} \sum_u \frac{1}{N} \sum_{j=1}^{N} rel_j$$

$$Recall@N = \frac{1}{|\mathcal{U}|} \sum_u \frac{1}{|T(u)|} \sum_{j=1}^{N} rel_j$$

$$HR@N = \frac{1}{|\mathcal{U}|} \sum_u \delta(R(u) \cap T(u) \neq \emptyset)$$

$$MAP@N = \frac{1}{|\mathcal{U}|} \sum_u \frac{1}{N} \sum_{k=1}^{N} Precision@k$$

$$MRR@N = \frac{1}{|\mathcal{U}|} \sum_u \sum_{j=1}^{N} j^{-1} rel_j$$

$$NDCG@N = \frac{1}{|\mathcal{U}|} \sum_u \frac{DCG@N}{IDCG@N}, \; DCG@N = \sum_{j=1}^{N} \frac{2^{rel_j} - 1}{log_2(j+1)}$$

TABLE 9
The update rules of commonly-used optimizers.

Optimizer	Update Rule
GD	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_\theta \mathcal{L}(\boldsymbol{\theta}_t)$
SGD	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_\theta \mathcal{L}(\boldsymbol{\theta}_t; (u, i))$
MB-SGD	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_\theta \mathcal{L}(\boldsymbol{\theta}_t; B(u, i))$
AdaGrad	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{G}_t + \epsilon}} \cdot \boldsymbol{g}_t; \; \boldsymbol{g}_t = \nabla_\theta \mathcal{L}(\boldsymbol{\theta}_t)$
RMSProp	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} \cdot \boldsymbol{g}_t; \; \mathbb{E}(\boldsymbol{g}^2)_t = \gamma \mathbb{E}[\boldsymbol{g}^2]_{t-1} + (1 - \gamma) \boldsymbol{g}_t^2$
Adam	$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t + \epsilon}} \cdot \hat{\boldsymbol{m}}_t; \; \hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t} \; \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}$ $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t; \; \boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t^2;$
Remark	$\boldsymbol{\theta}$ is the learnable parameter; η is the learning rate; (u, i) is one training sample; $B(u, i)$ is a batch of training samples; $\boldsymbol{G}_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element is the sum of the squares of the gradients w.r.t. $\boldsymbol{\theta}$ up to time step t; ϵ is a smoothing term to avoid division by zero (usually on the order of $1e - 8$); $\gamma = 0.9$ is the momentum term; $\boldsymbol{m}_t$ is the decaying average of past gradients; $\boldsymbol{v}_t$ is the decaying average of past squared gradients;

TABLE 10
The performance of ten baselines on ML-1M ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.2554	0.0530	0.6997	0.1734	0.8290	0.4952
ItemKNN	0.2218	0.0676	0.6521	0.1428	0.6804	0.4261
PureSVD	0.2389	0.0760	0.6840	0.1558	0.7430	0.4562
SLIM	–	–	–	–	–	–
BPRMF	0.2854	0.0819	0.7585	0.1924	0.8932	0.5275
BPRFM	**0.4379**	**0.1156**	**0.8667**	**0.3360**	**1.3718**	**0.6745**
NeuMF	0.4210	0.1023	0.8515	0.3215	1.3105	0.6528
NFM	0.4162	0.0999	0.8426	0.3171	1.2995	0.6473
NGCF	0.3684	0.0971	0.8426	0.2646	1.1560	0.6230
Multi-VAE	0.4201	0.1023	0.8532	0.3218	1.3131	0.6538
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.2558	0.0531	0.7001	0.1737	0.8303	0.4957
ItemKNN	0.2284	0.0666	0.6502	0.1482	0.6987	0.4319
PureSVD	0.2288	0.0626	0.6541	0.1534	0.7062	0.4334
SLIM	–	–	–	–	–	–
BPRMF	0.2858	0.0712	0.6665	0.2068	0.8888	0.4814
BPRFM	**0.4191**	**0.1077**	**0.8627**	**0.3200**	**1.3213**	**0.6651**
NeuMF	0.4184	0.1073	0.8571	0.3157	1.3029	0.6568
NFM	0.4042	0.0972	0.8397	0.3066	1.2672	0.6381
NGCF	0.2151	0.0618	0.6491	0.1328	0.6487	0.4186
Multi-VAE	0.4056	0.1017	0.8487	0.3062	1.2673	0.6415
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.2569	0.0532	0.7024	0.1745	0.8339	0.4975
ItemKNN	0.2208	0.0619	0.6381	0.1443	0.6821	0.4233
PureSVD	0.2270	0.0676	0.6900	0.1416	0.6984	0.4479
SLIM	0.2182	0.0642	0.6832	0.1377	0.6879	0.4450
BPRMF	0.2760	0.0670	0.6421	0.2001	0.8505	0.4601
BPRFM	**0.4032**	**0.0992**	**0.8540**	**0.3034**	**1.2637**	**0.6466**
NeuMF	0.2123	0.0638	0.6471	0.1378	0.6803	0.4322
NFM	0.3898	0.0916	0.8303	0.2923	1.2141	0.6232
NGCF	0.2865	0.0793	0.7909	0.1903	0.9134	0.5494
Multi-VAE	0.3901	0.0913	0.8298	0.2952	1.2335	0.6287

TABLE 11
The performance of ten baselines on LastFM ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0712	0.0716	0.3978	0.0338	0.2509	0.2311
ItemKNN	**0.4219**	**0.4332**	0.9580	**0.3324**	**1.5841**	**0.8026**
PureSVD	0.4173	0.4304	0.9480	0.3321	1.5807	0.7989
SLIM	–	–	–	–	–	–
BPRMF	0.3796	0.3906	0.9411	0.2867	1.4303	0.7624
BPRFM	0.3367	0.3465	0.8970	0.2415	1.2322	0.6897
NeuMF	0.3561	0.3663	0.9320	0.2552	1.3112	0.7279
NFM	0.2656	0.2723	0.8449	0.1772	0.9723	0.6058
NGCF	0.4059	0.4175	**0.9596**	0.3004	1.4575	0.7688
Multi-VAE	0.2662	0.2724	0.8396	0.1772	0.9659	0.5998
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0743	0.0875	0.4067	0.0388	0.2873	0.2450
ItemKNN	0.2410	0.3065	0.8551	**0.1647**	**0.9835**	0.6341
PureSVD	0.2374	0.2959	0.8507	0.1567	0.9453	0.6197
SLIM	–	–	–	–	–	–
BPRMF	0.2368	0.2975	0.8643	0.1551	0.9349	0.6222
BPRFM	0.2429	0.3042	0.8507	0.1586	0.9399	0.6135
NeuMF	0.2452	0.3074	0.8540	0.1627	0.9731	0.6284
NFM	0.1198	0.1428	0.5717	0.0675	0.4543	0.3625
NGCF	**0.2532**	**0.3252**	**0.8880**	0.1631	0.9698	**0.6381**
Multi-VAE	0.1176	0.1405	0.5695	0.0676	0.4515	0.3577
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0677	0.0880	0.4097	0.0349	0.2712	0.2476
ItemKNN	0.1851	0.2621	0.7840	0.1188	0.7708	0.5577
PureSVD	0.1862	0.2522	0.7556	0.1201	0.7673	0.5418
SLIM	**0.2120**	**0.3064**	**0.8369**	**0.1380**	**0.8713**	**0.6018**
BPRMF	0.1915	0.2657	0.8009	0.1193	0.7696	0.5592
BPRFM	0.1891	0.2631	0.7949	0.1157	0.7551	0.5513
NeuMF	0.1690	0.2382	0.7529	0.1001	0.6621	0.5044
NFM	0.0745	0.0967	0.4495	0.0336	0.2574	0.2532
NGCF	0.1821	0.2659	0.8020	0.1060	0.7019	0.5349
Multi-VAE	0.0855	0.1113	0.4768	0.0447	0.3309	0.2901

TABLE 12
The performance of ten baselines on Book-X ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0048	0.0150	0.0411	0.0023	0.0214	0.0249
ItemKNN	0.0214	0.0283	0.1027	0.0159	0.1010	0.0782
PureSVD	0.0776	0.1761	0.3247	0.0560	0.3178	0.2344
SLIM	–	–	–	–	–	–
BPRMF	0.0868	0.2309	0.3924	0.0586	0.3450	0.2687
BPRFM	0.0954	**0.2970**	**0.4603**	0.0636	**0.3836**	**0.3115**
NeuMF	0.0693	0.2067	0.3578	0.0421	0.2655	0.2289
NFM	0.0858	0.2757	0.4278	0.0576	0.3548	0.2919
NGCF	**0.0957**	0.2343	0.3947	**0.0673**	0.3787	0.2778
Multi-VAE	0.0858	0.2762	0.4282	0.0576	0.3550	0.2921
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0097	0.0177	0.0789	0.0047	0.0420	0.0476
ItemKNN	0.0419	0.0661	0.2186	0.0283	0.1937	0.1580
PureSVD	0.1030	0.1833	0.4480	**0.0697**	**0.4115**	0.3105
SLIM	–	–	–	–	–	–
BPRMF	0.1057	0.2127	0.4945	0.0637	0.3911	0.3159
BPRFM	0.1021	0.1994	0.4766	0.0609	0.3747	0.3033
NeuMF	0.0839	0.1619	0.4148	0.0480	0.3041	0.2571
NFM	0.0802	0.1549	0.3971	0.0480	0.3030	0.2529
NGCF	**0.1099**	**0.2311**	**0.5083**	0.0652	0.4021	**0.3249**
Multi-VAE	0.0810	0.1568	0.4001	0.0483	0.3046	0.2542
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0134	0.0192	0.1081	0.0062	0.0547	0.0630
ItemKNN	0.0516	0.0786	0.2780	0.0335	0.2314	0.1921
PureSVD	**0.1079**	**0.1622**	0.4740	**0.0695**	**0.4223**	**0.3216**
SLIM	0.0753	0.1014	0.3475	0.0510	0.3264	0.2504
BPRMF	0.0998	0.1619	**0.4801**	0.0565	0.3594	0.2983
BPRFM	0.0931	0.1401	0.4474	0.0524	0.3332	0.2779
NeuMF	0.0985	0.1490	0.4582	0.0564	0.3476	0.2852
NFM	0.0741	0.1119	0.3841	0.0415	0.2748	0.2380
NGCF	0.0818	0.1415	0.4345	0.0417	0.2788	0.2533
Multi-VAE	0.0751	0.1136	0.3867	0.0416	0.2736	0.2378

TABLE 13
The performance of ten baselines on Epinions ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0016	0.0009	0.0157	0.0005	0.0052	0.0076
ItemKNN	0.0480	0.0193	0.1599	0.0365	0.1753	0.1176
PureSVD	0.0681	0.0413	0.2462	0.0490	0.2424	0.1675
SLIM	–	–	–	–	–	–
BPRMF	0.0721	0.0467	0.2861	0.0483	0.2523	0.1844
BPRFM	0.0774	0.0448	0.3018	0.0494	0.2576	0.1884
NeuMF	**0.0882**	**0.0573**	**0.3519**	**0.0588**	**0.3176**	**0.2329**
NFM	0.0693	0.0416	0.2822	0.0431	0.2322	0.1745
NGCF	0.0819	0.0558	0.3234	0.0553	0.2852	0.2069
Multi-VAE	0.0696	0.0420	0.2822	0.0432	0.2317	0.1743
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0033	0.0039	0.0299	0.0012	0.0111	0.0152
ItemKNN	0.0354	0.0309	0.1668	0.0222	0.1271	0.1037
PureSVD	0.0335	0.0311	0.1651	0.0188	0.1117	0.0962
SLIM	–	–	–	–	–	–
BPRMF	0.0369	0.0378	0.1825	0.0219	0.1317	0.1115
BPRFM	**0.0391**	0.0354	0.2002	0.0220	0.1368	0.1192
NeuMF	0.0382	0.0383	**0.2095**	0.0215	**0.1423**	**0.1279**
NFM	0.0289	0.0252	0.1638	0.0150	0.1008	0.0946
NGCF	0.0385	**0.0406**	0.1777	**0.0230**	0.1335	0.1086
Multi-VAE	0.0283	0.0245	0.1611	0.0148	0.0996	0.0932
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0040	0.0058	0.0370	0.0015	0.0139	0.0189
ItemKNN	0.0261	0.0315	0.1388	0.0150	0.0916	0.0813
PureSVD	0.0238	0.0285	0.1332	0.0124	0.0782	0.0736
SLIM	0.0260	0.0327	0.1423	0.0147	0.0936	0.0840
BPRMF	0.0258	0.0318	0.1407	0.0139	0.0893	0.0820
BPRFM	0.0295	0.0335	0.1636	0.0158	0.1061	0.0979
NeuMF	0.0302	0.0375	0.1772	0.0164	**0.1160**	**0.1084**
NFM	0.0200	0.0219	0.1278	0.0101	0.0756	0.0756
NGCF	**0.0326**	**0.0402**	**0.1876**	**0.0165**	0.1092	0.1052
Multi-VAE	0.0199	0.0216	0.1265	0.0100	0.0748	0.0749

TABLE 14
The performance of ten baselines on Yelp ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0012	0.0058	0.0113	0.0005	0.0048	0.0061
ItemKNN	0.0128	0.0302	0.0699	0.0094	0.0628	0.0524
PureSVD	0.0365	0.1168	0.1938	0.0218	0.1417	0.1222
SLIM	–	–	–	–	–	–
BPRMF	0.0425	0.1573	0.2447	0.0234	0.1594	0.1471
BPRFM	**0.0508**	**0.2144**	**0.3141**	0.0271	**0.1925**	**0.1859**
NeuMF	0.0500	0.2092	0.3048	**0.0272**	0.1916	0.1827
NFM	0.0349	0.1598	0.2431	0.0175	0.1369	0.1425
NGCF	–	–	–	–	–	–
Multi-VAE	0.0352	0.1612	0.2445	0.0176	0.1379	0.1433
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0024	0.0044	0.0199	0.0010	0.0090	0.0109
ItemKNN	0.0646	0.1227	0.3061	0.0452	0.2815	0.2208
PureSVD	0.1066	0.2259	0.4823	0.0671	0.4053	0.3162
SLIM	–	–	–	–	–	–
BPRMF	0.1061	0.2260	0.4900	0.0623	0.3809	0.3060
BPRFM	0.1099	0.2262	0.5079	0.0629	0.3811	0.3088
NeuMF	0.1106	0.2370	0.5259	0.0636	0.3975	0.3264
NFM	0.0612	0.1149	0.3322	0.0317	0.2142	0.1931
NGCF	**0.1274**	**0.2832**	**0.5599**	**0.0764**	**0.4574**	**0.3573**
Multi-VAE	0.0621	0.1168	0.3348	0.0322	0.2165	0.1944
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0031	0.0040	0.0262	0.0012	0.0111	0.0137
ItemKNN	0.1174	0.1765	0.4882	0.0792	0.4600	0.3380
PureSVD	0.1508	0.2239	0.5837	0.0975	0.5493	0.3918
SLIM	0.1128	0.1655	0.4819	0.0778	0.4625	0.3432
BPRMF	0.1455	0.2171	0.5849	0.0886	0.5104	0.3770
BPRFM	0.1448	0.2117	0.5847	0.0866	0.4955	0.3701
NeuMF	0.1562	0.2354	0.6187	0.0961	0.5515	0.4032
NFM	0.0697	0.0894	0.3510	0.0360	0.2322	0.2028
NGCF	**0.1685**	**0.2583**	**0.6347**	**0.1051**	**0.5878**	**0.4187**
Multi-VAE	0.0702	0.0899	0.3539	0.0361	0.2331	0.2041

TABLE 15
The performance of ten baselines on AMZe ($N = 10$) across the six metrics, where the best performance is highlighted in bold.

Origin	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0025	0.0169	0.0245	0.0006	0.0063	0.0104
ItemKNN	0.0018	0.0114	0.0176	0.0008	0.0074	0.0096
PureSVD	0.0202	0.1095	0.1474	0.0117	0.0971	0.0969
SLIM	–	–	–	–	–	–
BPRMF	0.0323	0.2060	0.2642	0.0170	0.1510	0.1655
BPRFM	0.0518	0.3530	0.4249	0.0286	0.2549	0.2744
NeuMF	**0.0518**	**0.3531**	**0.4250**	**0.0286**	0.2550	**0.2745**
NFM	0.0517	0.3524	0.4243	0.0285	0.2546	0.2741
NGCF	–	–	–	–	–	–
Multi-VAE	0.0518	0.3531	0.4249	0.0286	**0.2552**	0.2745
5-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0064	0.0193	0.0612	0.0026	0.0257	0.0333
ItemKNN	0.0072	0.0224	0.0640	0.0041	0.0378	0.0419
PureSVD	0.0614	0.2104	0.4070	0.0348	0.2791	0.2654
SLIM	–	–	–	–	–	–
BPRMF	0.0657	0.2178	0.4335	0.0375	0.3012	0.2852
BPRFM	0.0759	0.2350	0.4776	**0.0442**	**0.3466**	**0.3177**
NeuMF	**0.0761**	0.2354	0.4789	0.0440	0.3453	0.3172
NFM	0.0760	**0.2356**	**0.4792**	0.0441	0.3458	0.3176
NGCF	0.0492	0.1711	0.3449	0.0239	0.1960	0.2038
Multi-VAE	0.0759	0.2349	0.4775	0.0441	0.3461	0.3174
10-filter	Pre	Rec	HR	MAP	MRR	NDCG
MostPop	0.0085	0.0179	0.0793	0.0040	0.0387	0.0470
ItemKNN	0.0136	0.0312	0.1136	0.0067	0.0603	0.0679
PureSVD	0.0758	0.1781	0.4530	0.0429	0.3248	0.2935
SLIM	0.0342	0.0802	0.2426	0.0213	0.1769	0.1702
BPRMF	0.0806	0.1875	0.4738	0.0463	0.3500	0.3117
BPRFM	0.0864	0.1901	0.4919	0.0504	0.3746	0.3264
NeuMF	0.0839	0.1852	0.4861	0.0493	0.3694	0.3244
NFM	0.0866	0.1906	0.4919	0.0509	0.3780	0.3276
NGCF	0.0663	0.1561	0.4127	0.0370	0.2880	0.2664
Multi-VAE	**0.0869**	**0.1909**	**0.4925**	**0.0511**	**0.3790**	**0.3285**

TABLE 16
The opitmal hyper-parameter settings found by Bayesian HyperOpt for different baselines with time-aware split-by-ratio under origin view.

Origin	Parameter	ML-1M	LastFM	Book-X	Epinions	Yelp	AMZe	Searching space	Description
ItemKNN	–maxk	8	49	83	14	31	9	$[1, 100]$	the number of neighbors
PureSVD	–factors	39	24	10	99	96	6	$[1, 100]$	the number of singular values
BPRMF	–num_ng	4	5	5	5	5	4	$[1, 5]$	the number of negative items
	–factors	83	92	48	97	72	74	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0056	0.0080	0.0098	0.0095	0.0089	0.0047	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0002	0.0012	0.0008	0.0002	0.0088	0.0016	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	128	64	64	128	1024	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
BPRFM	–num_ng	4	5	5	5	5	2	$[1, 5]$	the number of negative items
	–factors	30	24	83	33	45	66	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0030	0.0100	0.0093	0.0091	0.0100	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0007	0.0056	0.0006	0.0095	0.0012	0.0037	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	256	512	64	256	256	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NeuMF	–num_ng	4	2	3	3	3	4	$[1, 5]$	the number of negative items
	–factors	15	49	100	78	63	60	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0002	0.0016	0.0005	0.0003	0.0012	0.0006	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0010	0.0016	0.0064	0.0086	0.0061	0.0003	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	3	3	2	3	3	3	$[1, 3]$	the number of layers for MLP
	–dropout	0.9531	0.7890	0.0156	0.0001	0.0010	0.7948	$[0, 1]$	dropout ratio
	–batch_size	64	512	64	64	256	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NFM	–num_ng	1	2	3	5	3	3	$[1, 5]$	the number of negative items
	–factors	11	63	42	5	3	100	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0006	0.0030	0.0005	0.0019	0.0100	0.0007	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0017	0.0006	0.0005	0.0002	0.0067	0.0030	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	3	3	3	2	3	1	$[1, 3]$	the number of layers for MLP
	–dropout	0.7108	0.9386	0.9338	0.7830	0.9957	0.9694	$[0, 1]$	dropout ratio
	–batch_size	256	64	64	256	512	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NGCF	–num_ng	4	4	4	5	4	3	$[1, 5]$	the number of negative items
	–factors	35	53	94	91	84	45	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0007	0.0034	0.0007	0.0008	0.0013	0.0013	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0065	0.0084	0.0001	0.0002	0.0071	0.0088	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–prop_layer	3	3	3	3	3	3	–	the number of propagation layers
	–node_dropout	0.0106	0.4160	0.9431	0.0019	0.0009	0.0015	$[0, 1]$	node dropout ratio
	–mess_dropout	0.0573	0.0023	0.0001	0.0382	0.0048	0.0080	$[0, 1]$	message dropout ratio
	–batch_size	128	128	128	128	512	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Multi-VAE	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0003	0.0006	0.0001	0.0001	0.0007	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0003	0.0062	0.0037	0.0001	0.0010	0.0048	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–kl_reg	0.0012	0.5975	0.0570	0.0009	0.0005	0.4945	$[0, 1]$	vae kl regularization
	–dropout	0.0415	0.6912	0.0001	0.0240	0.0263	0.9603	$[0, 1]$	dropout ratio
	–batch_size	128	64	512	128	512	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Remarks	1. The searching space of batch size on ML-1M, LastFM, Book-X, and Epinions is $[2^6, 2^7, 2^8, 2^9]$; while on Yelp and AMZe is $[2^8, 2^9, 2^{10}]$ to speed up the training. 2. Early-stop mechanism is applied to all approaches. 3. The results of SLIM with origin and 5-filter views are not available due to either lack of computational memory or unreasonable amount of time in searching the best hyper-parameter settings. 4. The following results with origin view are not available due to unreasonable amount of time in searching the best hyper-parameter settings. In particular, we use the optimal parameter settings on 5-filter for NGCF and Multi-VAE on Book-X; Multi-VAE on Epinions; NeuMF and NFM on Yelp; BPRMF, BPRFM, NeuMF, NFM and NGCF on AMZe. We use the optimal parameter settings on 10-filter for NGCF and Multi-VAE on Yelp; and Multi-VAE on AMZe.								

TABLE 17
The opitmal hyper-parameter settings found by Bayesian HyperOpt for different baselines with time-aware split-by-ratio under 5-filter view.

5-filter	Parameter	ML-1M	LastFM	Book-X	Epinions	Yelp	AMZe	Searching space	Description
ItemKNN	–maxk	73	55	65	30	75	41	[1, 100]	the number of neighbors
PureSVD	–factors	2	33	80	70	100	2	[1, 100]	the number of singular values
BPRMF	–num_ng	1	5	4	5	5	4	[1, 5]	the number of negative items
	–factors	100	56	99	70	92	74	[1, 100]	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0018	0.0060	0.0093	0.0098	0.0051	0.0047	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0007	0.0003	0.0014	0.0029	0.0001	0.0016	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	64	128	128	128	512	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
BPRFM	–num_ng	3	5	5	4	5	2	[1, 5]	the number of negative items
	–factors	96	66	98	40	66	66	[1, 100]	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0023	0.0099	0.0080	0.0095	0.0097	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0001	0.0001	0.0010	0.0026	0.0003	0.0037	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	64	256	512	256	512	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NeuMF	–num_ng	5	4	4	3	3	4	[1, 5]	the number of negative items
	–factors	47	67	16	48	63	60	[1, 100]	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0007	0.0002	0.0021	0.0001	0.0012	0.0006	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0039	0.0009	0.0056	0.0001	0.0061	0.0003	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	1	3	3	3	3	3	[1, 3]	the number of layers for MLP
	–dropout	0.5919	0.9973	0.8592	0.0932	0.0010	0.7948	[0, 1]	dropout ratio
	–batch_size	128	512	256	128	256	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NFM	–num_ng	5	5	5	3	3	3	[1, 5]	the number of negative items
	–factors	29	45	2	2	3	100	[1, 100]	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0012	0.0031	0.0041	0.0027	0.0100	0.0007	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0059	0.0062	0.0004	0.0047	0.0067	0.0030	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	2	1	1	3	3	1	[1, 3]	the number of layers for MLP
	–dropout	0.9815	0.9977	0.8360	0.6564	0.9957	0.9694	[0, 1]	dropout ratio
	–batch_size	512	512	256	256	512	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NGCF	–num_ng	2	4	4	5	4	3	[1, 5]	the number of negative items
	–factors	96	97	94	96	84	45	[1, 100]	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0005	0.0018	0.0007	0.0010	0.0013	0.0013	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0003	0.0054	0.0001	0.0035	0.0071	0.0088	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–prop_layer	3	3	3	3	3	3	–	the number of propagation layers
	–node_dropout	0.0192	0.0029	0.9431	0.5553	0.0009	0.0015	[0, 1]	node dropout ratio
	–mess_dropout	0.0178	0.3631	0.0001	0.0013	0.0048	0.0080	[0, 1]	message dropout ratio
	–batch_size	512	256	128	128	512	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Multi-VAE	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0007	0.0002	0.0001	0.0001	0.0007	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0003	0.0001	0.0037	0.0001	0.0011	0.0048	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–kl_reg	0.0004	0.6766	0.0570	0.0009	0.0005	0.4945	[0, 1]	vae kl regularization
	–dropout	0.0222	0.0004	0.0001	0.0240	0.0263	0.9603	[0, 1]	dropout ratio
	–batch_size	256	128	512	128	512	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Remarks	The following results with 5-filter view are not available due to unreasonable amount of time in searching the best hyper-parameter settings. In particular, we use the optimal parameter settings on 10-filter for NGCF and Multi-VAE on Yelp; and Multi-VAE on AMZe.								

TABLE 18
The opitmal hyper-parameter settings found by Bayesian HyperOpt for different baselines with time-aware split-by-ratio under 10-filter view.

10-filter	Parameter	ML-1M	LastFM	Book-X	Epinions	Yelp	AMZe	Searching space	Description
ItemKNN	–maxk	100	28	71	14	73	68	$[1, 100]$	the number of neighbors
PureSVD	–factors	39	12	98	98	93	9	$[1, 100]$	the number of singular values
BPRMF	–num_ng	3	5	3	5	5	3	$[1, 5]$	the number of negative items
	–factors	22	92	88	49	98	97	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0004	0.0057	0.0098	0.0097	0.0099	0.0078	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0002	0.0088	0.0001	0.0050	0.0007	0.0004	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	64	128	256	128	1024	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
BPRFM	–num_ng	4	5	3	5	5	3	$[1, 5]$	the number of negative items
	–factors	37	61	94	72	100	37	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0014	0.0096	0.0067	0.0078	0.0096	0.0013	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0004	0.0010	0.0001	0.0019	0.0004	0.0049	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	128	512	64	64	256	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
SLIM	–elastic	0.4462	0.8330	0.8188	0.3951	0.0459	0.3299	$(0, 1]$	the ElasticNet mixing parameter
	–alpha	0.0001	0.0013	0.0072	0.0022	0.0040	0.0020	$[10^{-4}, 10^{-2}]$	constant to multiply penalty terms
NeuMF	–num_ng	5	2	5	5	5	5	$[1, 5]$	the number of negative items
	–factors	31	93	79	41	68	83	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0008	0.0036	0.0003	0.0012	0.0001	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0008	0.0011	0.0096	0.0057	0.0009	0.0015	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	2	2	3	3	2	2	$[1, 3]$	the number of layers for MLP
	–dropout	0.9592	0.5608	0.4719	0.4771	0.7371	0.7162	$[0, 1]$	dropout ratio
	–batch_size	64	512	64	128	1024	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NFM	–num_ng	5	4	1	2	4	3	$[1, 5]$	the number of negative items
	–factors	19	69	74	88	12	99	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0045	0.0002	0.0019	0.0005	0.0005	0.0016	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0026	0.0006	0.0003	0.0071	0.0009	0.0004	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	3	3	2	2	1	3	$[1, 3]$	the number of layers for MLP
	–dropout	0.9818	0.8898	0.9979	0.8666	0.8630	0.9862	$[0, 1]$	dropout ratio
	–batch_size	256	64	128	64	1024	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NGCF	–num_ng	5	5	4	5	4	2	$[1, 5]$	the number of negative items
	–factors	18	74	36	85	84	68	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0004	0.0010	0.0004	0.0017	0.0013	0.0001	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0006	0.0006	0.0005	0.0066	0.0071	0.0042	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–prop_layer	3	3	3	3	3	3	–	the number of propagation layers
	–node_dropout	0.0005	0.0021	0.0002	0.0169	0.0009	0.3658	$[0, 1]$	node dropout ratio
	–mess_dropout	0.0175	0.0015	0.0016	0.0011	0.0048	0.0115	$[0, 1]$	message dropout ratio
	–batch_size	512	256	128	128	512	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Multi-VAE	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0001	0.0005	0.0003	0.0003	0.0007	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0010	0.0004	0.0050	0.0004	0.0011	0.0048	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–kl_reg	0.0003	0.0408	0.0009	0.0003	0.0005	0.4945	$[0, 1]$	vae kl regularization
	–dropout	0.0018	0.5885	0.0083	0.2841	0.0263	0.9603	$[0, 1]$	dropout ratio
	–batch_size	64	512	256	64	512	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Remarks	The detailed explanation for the parameters of SLIM is available at https://lijiancheng0614.github.io/scikit-learn/modules/generated /sklearn.linear_model.ElasticNet.html								

TABLE 19
The opitmal hyper-parameter settings found by Bayesian HyperOpt for different baselines with time-aware leave-one-out under 10-filter view.

10-filter	Parameter	ML-1M	LastFM	Book-X	Epinions	Yelp	AMZe	Searching space	Description
ItemKNN	–maxk	22	70	100	51	52	40	$[1, 100]$	the number of neighbors
PureSVD	–factors	45	26	89	96	97	11	$[1, 100]$	the number of singular values
BPRMF	–num_ng	5	4	5	5	5	5	$[1, 5]$	the number of negative items
	–factors	49	66	76	56	61	100	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0040	0.0082	0.0098	0.0098	0.0047	0.0051	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0014	0.0007	0.0013	0.0016	0.0012	0.0006	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	128	256	256	64	1024	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
BPRFM	–num_ng	5	5	5	5	5	1	$[1, 5]$	the number of negative items
	–factors	99	71	99	87	72	71	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0074	0.0075	0.0096	0.0091	0.0098	0.0002	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0004	0.0079	0.0011	0.0001	0.0017	0.0007	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–batch_size	256	256	128	256	512	1024	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
SLIM	–elastic	0.6215	0.0126	0.0023	0.2328	0.0754	0.1056	$(0, 1]$	the ElasticNet mixing parameter
	–alpha	0.0084	0.0055	0.0098	0.0072	0.0090	0.0066	$[10^{-4}, 10^{-2}]$	constant to multiply penalty terms
NeuMF	–num_ng	5	5	3	3	3	2	$[1, 5]$	the number of negative items
	–factors	29	65	68	61	84	78	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0010	0.0001	0.0091	0.0002	0.0003	0.0013	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0049	0.0004	0.0093	0.0023	0.0005	0.0050	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	2	1	3	3	3	2	$[1, 3]$	the number of layers for MLP
	–dropout	0.1664	0.0933	0.0008	0.2623	0.5016	0.7352	$[0, 1]$	dropout ratio
	–batch_size	512	512	64	512	256	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NFM	–num_ng	4	1	5	2	1	5	$[1, 5]$	the number of negative items
	–factors	40	2	73	37	1	54	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0026	0.0044	0.0024	0.0019	0.0026	0.0004	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0004	0.0002	0.0003	0.0025	0.0005	0.0003	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–num_layers	1	3	3	3	3		$[1, 3]$	the number of layers for MLP
	–dropout	0.7825	0.9421	0.8430	0.9916	0.1267	0.9864	$[0, 1]$	dropout ratio
	–batch_size	128	512	64	64	256	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
NGCF	–num_ng	5	5	5	5	5	4	$[1, 5]$	the number of negative items
	–factors	75	85	63	74	85	2	$[1, 100]$	the dimension of latent factors
	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0007	0.0010	0.0004	0.0004	0.0008	0.0002	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0061	0.0093	0.0100	0.0096	0.0030	0.0006	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–prop_layer	3	3	3	3	3	3	–	the number of propagation layers
	–node_dropout	0.0067	0.0007	0.9751	0.0001	0.0288	0.0001	$[0, 1]$	node dropout ratio
	–mess_dropout	0.0048	0.9363	0.0032	0.0007	0.0040	0.9876	$[0, 1]$	message dropout ratio
	–batch_size	64	64	256	512	512	256	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Multi-VAE	–epochs	50	50	50	50	50	50	–	the number of epochs
	–lr	0.0006	0.0019	0.0003	0.0007	0.0012	0.0023	$[10^{-4}, 10^{-2}]$	learning rate
	–reg_2	0.0004	0.0017	0.0030	0.0062	0.0002	0.0030	$[10^{-4}, 10^{-2}]$	L2 regularization coefficient
	–kl_reg	0.0517	0.0003	0.0007	0.0014	0.0036	0.0069	$[0, 1]$	vae kl regularization
	–dropout	0.0166	0.6342	0.0111	0.8804	0.0242	0.0165	$[0, 1]$	dropout ratio
	–batch_size	256	256	512	512	512	512	$[2^6, 2^7, 2^8, 2^9, 2^{10}]$	batch size
Remarks	The detailed explanation for the parameters of SLIM is available at https://lijiancheng0614.github.io/scikit-learn/modules/generated /sklearn.linear_model.ElasticNet.html								