A Revisiting Study of Appropriate Offline Evaluation for Top-*N* Recommendation Algorithms

WAYNE XIN ZHAO and ZIHAN LIN, Renmin University of China, China
ZHICHAO FENG and PENGFEI WANG, Beijing University of Posts and Telecommunications, China
JI-RONG WEN, Renmin University of China, China

In recommender systems, top-*N* recommendation is an important task with implicit feedback data. Although the recent success of deep learning largely pushes forward the research on top-*N* recommendation, there are increasing concerns on appropriate evaluation of recommendation algorithms. It therefore is important to study how recommendation algorithms can be reliably evaluated and thoroughly verified. This work presents a large-scale, systematic study on six important factors from three aspects for evaluating recommender systems. We carefully select 12 top-*N* recommendation algorithms and eight recommendation datasets. Our experiments are carefully designed and extensively conducted with these algorithms and datasets. In particular, all the experiments in our work are implemented based on an open sourced recommendation library, Recbole [139], which ensures the reproducibility and reliability of our results. Based on the large-scale experiments and detailed analysis, we derive several key findings on the experimental settings for evaluating recommender systems. Our findings show that some settings can lead to substantial or significant differences in performance ranking of the compared algorithms. In response to recent evaluation concerns, we also provide several suggested settings that are specially important for performance comparison.

CCS Concepts: • **Information systems → Recommender systems;**

Additional Key Words and Phrases: Top-*N* recommendation, evaluation, experimental setup

ACM Reference format:
Wayne Xin Zhao, Zihan Lin, Zhichao Feng, Pengfei Wang, and Ji-Rong Wen. 2022. A Revisiting Study of Appropriate Offline Evaluation for Top-*N* Recommendation Algorithms. *ACM Trans. Inf. Syst.* 41, 2, Article 32 (December 2022), 41 pages.
https://doi.org/10.1145/3545796

This work was partially supported by National Natural Science Foundation of China under Grant No. 61872369 and 61802029, Beijing Natural Science Foundation under Grant No. 4222027, and Beijing Outstanding Young Scientist Program under Grant No. BJJWZYJH012019100020098. This work was also partially supported by Beijing Academy of Artificial Intelligence (BAAI).

Authors' addresses: W. X. Zhao, Gaoling School of Artificial Intelligence, Beijing Key Laboratory of Big Data Management and Analysis Methods, Renmin University of China, Beijing, 100872, China; email: batmanfly@gmail.com; Z. Lin, School of Information, Beijing Key Laboratory of Big Data Management and Analysis Methods, Renmin University of China, Beijing, 100872, China; email: zhlin@ruc.edu.cn; Z. Feng and P. Wang, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, 100876, China; emails: fzcbupt@gmail.com, wangpengfei@bupt.edu.cn; J.-R. Wen (corresponding author), Gaoling School of Artificial Intelligence, School of Information, Beijing Key Laboratory of Big Data Management and Analysis Methods, Renmin University of China, Beijing, 100872, China; email: jrwen@ruc.edu.cn.

1046-8188/2022/12-ART32 $15.00
https://doi.org/10.1145/3545796

1 INTRODUCTION

Since the early 2000s, recommender systems have been an important research field, which aim to recommend suitable information resources to target users given the context [100, 131]. The core task of recommender systems is to learn a parametric [44, 47, 118] or non-parametric [53, 93] recommendation function that is able to better predict user preference over the resource pool of items. According to the type of user feedback data, the recommendation is based on either *implicit feedback* [90, 129] or *explicit feedback* [103]. Specifically, from implicit feedback, top-N item recommendation [90] has been a widely studied task that aims to identify a small set of the most possible items that a user may prefer from a large collection.

In the literature, various top-N recommendation algorithms have been developed [1, 15, 47, 90, 113, 118]. At the early stage, collaborative filtering approaches have been proposed to make recommendations according to similar interests or tastes. For example, UserKNN and ItemKNN recommend items that are from similar user preference or with similar item characteristics [1]. Such an approach lays the foundation of early solutions for recommender systems. Afterwards, matrix factorization has been widely adopted to better characterize the user-item interaction [47, 53, 63]. The basic idea is to project users, items, and related context features into low-dimensional semantic space, so that the factors involved in a specific interaction are drawn to be close [93]. Recently, with the success of deep learning, a number of neural recommendation algorithms [131] have been proposed, making important progress in recommender systems.

To verify the effectiveness of a recommendation algorithm, one needs to construct reliable evaluation experiments with thorough comparisons. Typically, such an evaluation procedure consists of a series of setup steps on datasets, metrics, baselines, and other protocols. As each setup step can be conducted with different options, it is essential to develop and design appropriate criterions for standardizing the experimental settings [138]. There is substantial divergence on the used settings or options in existing studies. With these inconsistent settings, researchers have found that the claimed improvement through offline evaluation sometimes might not be convincing or reliable [92]. Some studies even questioned the actual progress of recommender systems reported in an unreproducible evaluation setting [24, 57].

In the meantime, there has been increasing research on standardized or reproducible evaluation criterion for top-N item recommendation [58, 107]. These studies either focus on some specific factor [58] or the entire evaluation process [107]. However, they mainly adopt traditional recommendation algorithms as the objects to study. It is not clear whether some specific findings still hold when neural algorithms are involved in evaluation. As another limit, prior studies may not well respond to recent concerns [64] about evaluation protocols on neural recommendation algorithms, and the studied or compared settings in References [87, 99] do not align with the major divergence from current debate. Besides, existing studies usually use very few comparison algorithms or datasets. Therefore, there is a need to thoroughly revisit experimental settings of substantial divergence in recent literature, considering both traditional and neural algorithms.

In this work, we aim to present a comprehensive, systematic study on various experimental settings for evaluating top-N item recommendation algorithms. Our focus is to empirically examine whether various settings will lead to substantially different comparison results. Furthermore, we aim to draw some conclusions or findings on these evaluation steps, so that the evaluation process can be conducted in a more reproducible and reliable way. Specifically, we focus on studying three very important aspects, including evaluation metrics, dataset construction, and model optimization. For evaluation metrics, we study the correlation of comparison results on different evaluation metrics to examine whether some metrics are redundant or requisite. In particular, we study a recent concern about sampled metrics [9, 64], where the performance results are computed

according to a partial candidate item list. For dataset construction, we study various filtering, ordering, and splitting strategies for constructing the evaluation sets, and the purpose is to compare different combinations of these strategies and identify more reliable settings. For model optimization, we discuss two important issues related to the performance, including optimization function and hyper-parameter search. The goal is to derive empirical suggestions to help reproduce the optimal performance of recommendation algorithms.

To study the above three aspects, we select 12 recommendation algorithms covering both traditional and neural-based algorithms. These recommendation algorithms correspond to distinct architectures or designs, which are widely adopted as baselines in the literature of recommender systems. We further select eight benchmark datasets for constructing the experiments. The eight datasets correspond to different amounts of interaction data from diverse domains. To develop our study, at each time, our idea is to design a complete configuration consisting of some options for evaluation metrics, dataset construction, and model optimization. Then we vary and compare different options in two or more configurations and measure their differences in actual performance rankings. Through our experiments, we try to find out the options that lead to significant change in performance rankings and discuss what are possibly the suitable options given the rest fixed. To make these experiments themselves reproducible and standardized, all the studied or compared options are implemented based on an open sourced recommendation library *RecBole* [139]. Researchers can conduct fair evaluation for recommendation algorithms or make further studies on the evaluation protocol with the help of this library. We share all the scripts or codes to run *RecBole* for our experiments.

In summary, the main contributions of this work can be listed as follows:

- We present a large-scale, systematic study on six important factors from three aspects for evaluating recommender systems. To conduct our study, we extensively collect the research papers on top-N recommendation and then analyze and summarize the most divergent settings in different factors. This work presents a comprehensive study for appropriately evaluating top-N recommendation algorithms.
- We carefully select 12 top-N recommendation algorithms and eight recommendation datasets. Our experiments are carefully designed and extensively conducted with these algorithms and datasets. In particular, we utilize an open sourced recommendation library *Recbole* [139] to conduct all the experiments, ensuring that our experiments can be reproduced from scratch. We create a GitHub page to report all the running details about our experiments at this link: https://github.com/RUCAIBox/RecSysEvaluation.
- Based on the large-scale experiments and detailed analyses, we derive several key findings on the experimental settings for evaluating recommender systems (see Table 1 for a brief summary of these findings). Our findings show that some settings can lead to substantial or significant difference in performance rankings of compared algorithms. In response to recent evaluation concerns, we also provide some suggested settings that are specially important for performance comparison.

The rest of this article is organized as follows. The related work is reviewed in Section 2. In Section 3, we first present the overall procedure and then introduce the experimental settings for this study. As the major contents, we discuss three studied aspects in Sections 4, 5, and 6, respectively. Section 7 concludes this work and presents the future work.

2 RELATED WORK

In this section, we first review the related work in three aspects and finally discuss the differences between our work and existing studies.

Table 1. A Brief Summary of the Key Findings in This Work

Aspect	New Findings	Re-verified Findings
Metrics	• For performance comparison, AUC and other beyond-accuracy metrics are very different from the top-N ranking metrics. • Some recently proposed debiasing methods may not effectively reduce the bias from sampled metrics.	• Top-N ranking metrics capture different characteristics of algorithm performance, forming three coherent groups. • Sampled metrics would yield different performance rankings in evaluation compared to full-ranking metrics.
Dataset	• Dataset filtering is an important factor that is likely to significantly affect the performance rankings. • Performance ranking is more sensitive to the way of data ordering than the splitting strategy.	• Existing dataset construction methods might lead to data leakage in global timelines, which often introduces a gap between offline evaluation and real-world recommendation scenario.
Optimization	• The performance of algorithms originally with the BPR loss would decrease when it uses BCE loss, and vice versa. • The search range of hyper-parameters is likely to affect the final results, which is often neglected by previous work. Moreover, an efficient search strategy, i.e., sequential search, can achieve comparable accuracy.	• For optimization functions, the non-sampling method has intrinsic advantage when the item set is large. • For sampling-based algorithms, different loss functions (BPR or BCE) will lead to different performance and the loss function of algorithm should be carefully selected.

Re-verified findings refer that they were also discussed in previous work, but we performed new experiments to verify them.

2.1 Top-N Recommendation Algorithms

As one of the most fundamental tasks in recommender systems, top-N recommendation has attracted a large amount of research attention [107]. There are different ways to categorize existing recommendation algorithms. Here we consider a literature classification way tailored to our experiments, namely traditional algorithms, neural-based algorithms, and non-sampling algorithms.

2.1.1 Traditional Recommendation Algorithms. Early recommendation algorithms are mainly built on the idea of collaborative filtering [13], such as UserKNN and ItemKNN [1], which recommend items from someone with similar tastes or items similar to the liked ones. It was originally proposed for modeling explicit preference with rating data, and Deshpande and Karypis [28] presented a discussion on how to extend item-oriented approach with implicit feedback data. Among existing traditional approaches, **Matrix Factorization (MF)** is the most popular one for recommender systems, which is also the basis of many effective algorithms [15, 47]. Early MF-based algorithms [63] focused on explicit feedback (e.g., ratings), and such a task setting is often called *rating prediction*. They learn to map users and items to a latent factor space, so that user-item relationships can be re-constructed via the inner product between latent factors. A number of variants based on MF have been proposed by incorporating additional context information, including neighborhood [62], temporal information [126], social links [36], and text information [66]. To incorporate general side information, Factorization Machines [89] can model high-order feature interaction based on the matrix factorization approach, where users and items were also considered as two kinds of specific features. Since explicit feedback is not always available, a large body of studies have focused on implicit feedback recommendation, which is a ranking task in essence. As a representative work, Rendle et al. [90] proposed a classic pairwise learning algorithm, BPR, to optimize relative preference of a user over pairs of items based on negative sampling. Several

efforts [48, 71] further focused on the weighting scheme to better select negative samples from the unobserved items. These traditional recommendation algorithms are efficient to optimize and usually robust in various settings. However, they use relatively simple fitting functions that have limited capacity for modeling complicated user preference.

2.1.2 Neural Algorithms. With the success of deep learning, a significant number of neural algorithms were proposed over the past few years [131]. As a representative algorithm, **neural collaborative filtering (NCF)** [47] proposed a **multi-layer perceptron (MLP)** to fit the interaction function between users and items, which is also integrated with an additional matrix factorization module for achieving a better performance. Furthermore, Xue et al. [122] extended the classic item-based CF method by modeling the complex relation among items with neural networks. Hidasi et al. [51] introduced the GRU network to capture the sequential pattern from the session-based interaction data. Besides, various neural network architectures were adapted to develop more effective recommendation algorithms, such as attention mechanisms [5, 52, 112], memory networks [14, 32, 108], and convolutional neural network [119, 121]. More recently, graph neural networks were also utilized to capture complex user-item interactions [44, 113]. These algorithms typically cast user-item interactions as interaction graphs, and aim to learn effective node representations from the topology structure of the interaction graph. Several advanced architectures were proposed to better characterize the underlying graph structure of the interaction data. For example, Chen et al. [18] introduced attention mechanism into GCN-based model in a self-supervised paradigm. Yang et al. [125] proposed to utilize mutual information maximization to enhance the learning of graph-based collaborative filtering. Although neural algorithms are more capable of modeling complex data patterns, they are more sensitive to the training algorithm and the quality of training data.

2.1.3 Non-sampling Model Optimization. The aforementioned implicit feedback recommendation algorithms are mostly based on negative sampling, where we sample random negative items for optimization. Apart from this kind of algorithms, non-sampling algorithms have been proposed by taking a different solution. The core idea is to treat all non-interacted items as negative items, so that all the unobserved interactions with them can be used by the recommendation algorithm. Several algorithms were proposed based on this whole-data-based strategy [29, 71]. For example, WMF [53] considered all the unobserved interactions as negative and built a pointwise regression approach with all the data. With the development of deep learning, auto-encoder has been a popular architecture for developing non-sampling recommendation algorithms. For example, Wu et al. [118] proposed **collaborative denoising auto-encoders (CDAE)** by integrating user-specific bias, Zhang et al. [133] designed AutoSVD++ that utilized a contrastive autoencoder to model side information by extending the original SVD++, and Liang et al. [72] proposed **Variational Autoencoders for Collaborative Filtering (MultiVAE)** that employed a principled Bayesian approach. To improve model robustness, Chen and de Rijke [20] further utilized a variational autoencoder to incorporate side information for collective modeling. Besides, another category of non-sampling algorithms focused on adapting the objective function for matrix factorization algorithms. As a representative algorithm, Chen et al. [15] proposed an **efficient neural matrix factorization (ENMF)** framework that can be optimized by an alternating-based efficient learning algorithm, and similar methods are also used for multi-behavior recommendation [16].

2.2 Evaluation of Recommender System

2.2.1 Overall Studies. The concerns about the evaluation have been discussed along with the development of recommender system. Before the prevalence of neural recommendation algorithms, Shani and Gunawardana [99] studied how to design suitable evaluation experiments for comparing

and selecting a number of candidate recommendation algorithms. Besides the basic accuracy metrics, they also emphasized several important measurements that should be considered, including scalability, adaptivity, and privacy. Recently, the evaluation on neural recommendation algorithms has attracted the attention from the research community. Said and Bellogín [98] presented a tutorial on discussing reproducibility and replicability of the evaluation and results. Zhao et al. [138] studied the evaluation of top-N recommendation algorithms in three different aspects, namely dataset splitting, sampled metrics, and domain selection. Furthermore, Sun et al. [107] systematically reviewed recent papers and studied the essential factors related to evaluation such as metrics, sampling, and hyper-parameters and also released an open source library for facilitating the reliable evaluation of recommender systems. Cañamares et al. [11] extensively discussed methodological decisions for designing recommender systems and the corresponding choices for different steps.

2.2.2 Evaluation Metrics. For top-N recommendation, classic evaluation metrics mainly follow the ranking measures, such as Recall, Precision, and **F1-measure (F1)** [101], and there are increasing studies on the effectiveness of these metrics [78, 107, 109]. Also, the correlation of these metrics are investigated. For example, Sun et al. [107] examined pairwise metric correlation based on a number of recommendation algorithms and found that Recall has a weak correlation with other metrics. Besides Recall, the relation between true- and false-positive metrics was also discussed in Reference [78], where they found that systematic disagreements existed between the two types of metrics. Valcarce et al. [109] further examined the robustness and discriminative power of evaluation metrics, showing that Precision and NGCG offer high robustness and best discriminative power, respectively, in their study. Apart from accuracy-based metrics, beyond-accuracy metrics, such as novelty, diversity, and surprise [101], have been recently studied. When both accuracy and beyond-accuracy metrics are involved, recommendation algorithms are encouraged to seek a good tradeoff between the two kinds of metrics [85]. Interestingly, a conflict between the two kinds of metrics was also found in Reference [104]. A more recent concern about metrics is that a large number of research works compute the metrics based on an incomplete candidate list under sampling [47, 70, 81, 120]. It has shown that most of the sampled metrics produce inconsistent rankings compared with original metrics [64], suggesting that sampled metrics should not be used for evaluation. To alleviate this sampling bias, several unbiased estimators of the sampled metric were introduced in References [9, 64] through theoretical analysis. Furthermore, Li et al. [69] developed a novel mapping function for Hit-Ratio [69], which provides a safe approach to using sampled Hit-Ratio metrics. Further analysis [9] also considered the effect of different sampling sizes on the informativeness and consistency of experiments and found that a better size may lie in between the maximum and minimum target sets.

2.2.3 Dataset Construction. Similarly to other machine learning tasks, a typical approach for dataset construction is to split the original data into train, validation, and test sets [79]. The impact of pre-processing strategies was studied in a recent paper [107], where they found that the performance of algorithms generally improves when there is more training data for one user. Besides, various data splitting strategies were discussed and used [80, 131]. Meng et al. [80] observed that different splitting strategies may be biased toward specific recommendation models, due to the different data distributions in the evaluation sets. A comparison between random and time-aware splitting strategies was conducted in Reference [138], where different combinations between data ordering and splitting strategies are compared and studied. To better predict online performance, the effect of the temporal information was discussed for improving offline evaluation strategies [54]. Besides, an alternative offline evaluation procedure was proposed in Reference [55], with special considerations on the chronological order of interaction data. Considering the

temporal effect, three splitting strategies were analyzed [10], and they suggested that temporal global timeline splitting should be used but they were seldom used before [79].

2.2.4 Model Optimization. The model optimization approach becomes increasingly important for neural recommendation algorithms. As a common strategy, an algorithm is trained with true positive interactions and sampled false interactions from non-interacted items, such as the classic BPR loss [90, 113] and the BCE loss [47]. Two simple popularity-based item samplers (i.e., low popularity sampler and high-popularity sampler) were considered. They were compared with uniform sampler, showing that uniform sampling performs best in most of the cases [107]. Moreover, combining uniform and popularity-based sampling in a proper way can potentially improve the recommendation accuracy [50]. Furthermore, a simplified and robust negative sampling approach with score-based memory and variance-based sampling criterion was proposed for high-quality true negative instances [30]. Meanwhile, an adaptive sampler based on Bayesian pointwise optimization was designed for implicit feedback data with noisy labels [129]. As another alternative optimization approach, several non-sampling optimization methods are proposed in the literature [15], where they treat all the non-interacted items as negative instances and optimize the algorithm by an objective function with all the items.

2.2.5 Online Evaluation. To evaluate the performance of recommender systems, online evaluation is the most desired, since it can provide accurate results of how good our system is with real users [94]. However, it is typically difficult to conduct online evaluation, which requires significant efforts on integration and deployment in real applications. In industry, a commonly adopted strategy is the online A/B test [61]. By fixing the experimental setting, we compare a studied algorithm and a reference algorithm using two different (sampled) populations of users. Then, the comparison is usually measured with some industry metrics such as Click-Through-Rate [59] and Conversion Rate [86]. Since it is very costly to conduct online evaluation, several researchers further investigated the correlation between online and offline performance [55, 96]. Rossetti et al. [96] empirically showed that there might be inconsistent ranking results between offline and online evaluation, even for the same set of users. Since offline evaluation is likely to induce the evaluation bias [17], there are also studies focusing on mitigating the bias from offline evaluation [17, 39, 124]. For example, for the selection bias introduced by implicit-feedback, Yang et al. [124] proposed an unbiased offline evaluator for implicit feedback datasets, based on inverse propensity scoring. In addition, the bias in traditional offline estimator was alleviated by Gilotte et al. [39] with a new counterfactual estimator.

2.3 Reproducibility of Recommendation Algorithms

With the rapid progress made on recommendation algorithms, there is an increasing concern on the reproducibility of the reported results in evaluation. A recent study [24] analyzed a number of recent neural algorithms with a systemic comparison under the reproduced results, showing that only a part of recently proposed algorithms can be reproduced with reasonable effort. Similar concerns were also proposed by another work [25], raising a series of potential issues with recent progress, including limited reproducibility, unreliable comparison, and experimental arbitrariness.

A major difficulty in reproducing these baselines is that implementation details have been missing in the original papers. In Reference [23], through extensive experiments, it is shown that more complicated neural algorithms may not always outperform relatively simple algorithms. Another concern is that the compared baselines lack proper optimization, and many algorithms only give sub-optimal performance. In particular, the efficacy of MLP-based algorithms has been doubted recently, where it was shown that simple matrix factorization algorithms may not be well tuned with a proper hyper-parameter selection [91, 92].

To reproduce various recommendation algorithms, a number of open source recommender system libraries have been publicly released. At the early stage, Mymedialite [38], LightFM [65], RankSys [110], and LibRec [40] were released for rapid prototyping and testing of recommendation algorithms. In particular, LibRec [40] provides an Java library with a number of non-neural recommendation algorithms and built-in evaluation metrics. Then, with the resurge of **deep learning (DL)**, several DL-based libraries have been developed by incorporating neural network algorithms for recommendation. For example, NeuRec [116] implemented many neural algorithms and incorporated sequential and social recommendation tasks into an unified framework. DeepRec [41] was released with at-scale recommendation inference. Furthermore, DaisyRec [107], RecBole [139], QRec [127], and ELLIOT [3] largely improved the quality of recommender system libraries with more supporting implementations for the entire evaluation pipeline, such as data filtering/splitting operations and hyper-parameter tuning features.

2.4 Differences with Existing Works

There are a large number of studies on evaluating recommender algorithms [58, 99, 107, 138] from an overall [99, 107, 138] or factor-specific [54, 58] perspective. However, a majority of these prior studies mainly focus on non-neural algorithms, and it is not clear whether the derived findings still apply to neural algorithms. We are also aware that several recent studies have examined both traditional and neural recommendation algorithms [107, 138]. However, these works are usually limited to a small number of comparison algorithms or datasets.

Our work is highly built on these works, aiming to incorporate the widely discussed or studied factors from the literature (instead of exploring new experimental settings). As a comparison, we provide a comprehensive discussion about multiple factors, especially those with increasing concerns, based on large-scale experiments with a number of comparison algorithms and datasets. In particular, this work is supported by the open sourced recommendation library RecBole, so that the entire running details or studied settings can be reproduced with RecBole.

The most relevant study to our work is the DaisyRec framework [107], which has extensively studied various settings and their implementations. Some of our experimental settings or design are borrowed from Reference [107]. For example, we follow Reference [107] to collect a number of top-tired recommender system papers for identifying feasible options, and calculate the proportions of different options. As a major difference in research methodology, their work focuses on the absolute performance of recommendation algorithms, while our work mainly considers the performance rankings and explores the effect of different settings on performance rankings. Besides, they only used one neural recommendation algorithm in comparison, and several recent settings or issues were not considered (e.g., non-sampling VAE loss, sampled metrics, beyond-accuracy metrics, and temporal data leakage).

This work is a significant extension of a previously published short paper [138], where it has studied only three factors based on the Amazon datasets. As a comparison, currently, we study six factors and use more datasets from other domains. This work presents new experimental design or results compared with Reference [138], even for the same factor.

3 OVERALL EXPERIMENTAL SETTINGS

In this section, we first present an overview of the procedure to conduct our study for evaluating top-N recommendation algorithms, and then describe the details to set up our experiments.

3.1 Studied Factors

In this work, we study three important aspects for evaluating top-N recommendation algorithms, including *evaluation metrics*, *dataset construction*, and *model optimization*. The three aspects play

a key role in conducting empirical comparison of recommendation algorithms, which correspond to three different components of *measure*, *data*, and *optimization* in recommender systems [97], respectively. In recent literature, there is substantial divergence or even debate on the related factors from the three aspects. Therefore, our study is targeted on the divergent options for each aspect and aims to provide important guidelines to instruct the evaluation of recommender systems.

For the first aspect of evaluation metrics, we study how to select appropriate evaluation measurements for the recommendation performance. For top-N recommendation algorithms, it typically returns a ranked list of items that are of potential interests to a target user. The evaluation task therefore becomes how to measure the recommendation quality according to the generated recommendation list of an algorithm. It has been widely recognized that evaluation measures are particularly important to consider when designing the experiments in recommender systems [99]. Following existing studies [64, 107], we investigate the consistency of *evaluation metrics* and impact of *sampled metrics*. Although the first factor has been widely studied, prior studies usually adopt a small number of comparison algorithms [107] or mainly consider non-neural algorithms [78, 85]. With the increasing number of neural recommendation algorithms, it is necessary to revisit this factor using a considerable number of neural algorithms. The second factor recently attracts much concerns from the research community. Several studies find that it is biased or improper to use sampled metrics for performance comparison [64, 69].

For the second aspect of dataset construction, although it is a fundamental step to set up the experiments, there is still not a widely recognized standard for selecting or generating the evaluation sets. A large number of research papers simply adopt two or three datasets for evaluation. It lacks a thorough evaluation with more diverse datasets [107]. Furthermore, even for the same dataset, there are also many options to preprocess and split the datasets. With different options, the comparison results might become substantially different [107]. Based on these considerations, we aim to perform a systematic study on the possible strategies of *dataset selection and preprocessing* and *dataset splitting* [54, 138]. The first factor focuses on how to select and preprocess the datasets, and the second factor discusses how to derive evaluation sets by splitting the datasets. It aims to derive some empirical suggestions for preparing datasets in evaluation.

For the third aspect of model optimization, we study how to appropriately optimize the recommendation algorithms. Compared with traditional recommendation algorithms, model optimization plays a more important role for neural-based algorithms in practice [131], since there are more components or parameters to tune. Indeed, this aspect is highly related to the issue of reproducibility of recommendation algorithms, which has become the worrying concern in the field of recommender systems [25, 91, 92]. In this aspect, we study the impact of two important factors, namely *objective function* and *hyper-parameter search*. The first factor focuses on the suitable optimization loss function for recommendation algorithms, and the second factor focuses on how to set the hyper-parameters in an effective and efficient way.

3.2 Paper Selection with Option Categorization

In the above, we have introduced the three aspects with two factors for each aspect. For each factor, existing works usually have diverse options for experimental setup. To revisit the debate or divergence in existing works, we take a literature survey approach to analyzing and comparing the widely used options from the recently published recommender system papers of top-tiered conferences.

Following Reference [107], we focus on eight conferences, including KDD, SIGIR, WWW, IJCAI, AAAI, WSDM, CIKM, and RecSys, and collect 93 representative papers[1] on the topic of top-N

[1]Due to the limit for the number of the references, we do not cite all the collected papers in this article but present the detailed information of them at the link: https://github.com/RUCAIBox/RecSysEvaluation.

Table 2. The Categorization of Experimental Settings in the Collected Papers

Aspect	Factors	Options	Papers
Evaluation metrics	Metrics incorporated in evaluation	Recall	[14, 33, 52, 56, 67, 70, 73, 77, 83, 105, 111–114, 117, 130, 134, 135, 140]
		Hit	[5, 19, 22, 27, 31, 32, 37, 45, 47, 56, 74, 81, 82, 108, 115, 119–121, 132, 134, 136, 137]
		NDCG	[5, 14, 19, 27, 31–33, 45, 47, 52, 56, 68, 73, 74, 81–83, 105, 108, 112–115, 117, 119–121, 130, 132, 134]
		AUC	[19, 83]
		Beyond-accuracy	[37, 140]
	Metric calculation	Sampled	[5, 27, 32, 47, 52, 70, 74, 77, 81, 82, 108, 115, 120, 121, 132, 136]
		Non-sampled	[14, 19, 22, 31, 33, 45, 56, 67, 68, 105, 111–114, 119, 128, 134, 135, 137, 140]
Dataset construction	Number of used datasets	≥3	[5, 14, 22, 27, 32, 33, 37, 45, 52, 68, 73, 74, 81–83, 108, 111–114, 117, 119, 130, 132, 135, 137]
		<3	[19, 31, 47, 56, 67, 70, 77, 105, 115, 120, 121, 128, 134, 136, 140]
	Dataset filtering	n-core filtering	[5, 14, 19, 27, 31, 33, 37, 45, 47, 56, 67, 70, 73, 74, 108, 112, 113, 115, 121, 128, 130, 132, 134, 135, 137, 140]
		Original or neglection	[32, 52, 68, 77, 81–83, 105, 111, 114, 117, 119, 120, 136]
	Dataset splitting	RO_RS	[14, 33, 52, 56, 67, 68, 73, 74, 77, 111–114, 117, 128, 130, 134–136]
		RO_LS	[22, 37, 45, 70, 81–83, 105, 120, 137]
		TO_RS	[19, 117, 140]
		TO_LS	[5, 27, 31, 32, 45, 47, 108, 115, 119, 120, 132]
Model optimization	Objective function	BPR-based	[5, 14, 19, 31, 32, 45, 56, 67, 77, 83, 105, 108, 112–114, 119–121, 128, 130, 134]
		BCE-based	[22, 27, 47, 52, 68, 74, 81, 111, 117, 132, 135, 140]
	Hyper-parameters	Reported	[5, 19, 22, 33, 67, 68, 70, 73, 111, 113, 117, 128, 134, 137]
		Not reported	[14, 27, 31, 32, 45, 47, 52, 56, 74, 77, 81–83, 105, 108, 112, 114, 115, 119–121, 130, 132, 135, 140]

recommendation during the period of year 2017–2020. Our selection criterion is mainly based on the citation count, while taking more diverse algorithms into consideration. For each paper, we first identify the possible options for the six studied factors from the three aspects. Then, we categorize or group similar options and then assign the paper to the corresponding option slot. In this way, we obtain the major options to set up the six factors.

Based on this collection and the corresponding categorization, we can compare and analyze the divergence of options for each factor. The categorization of the paper collection is listed in Table 2.

3.3 Experimental Settings

To examine the impact of these key factors, we conduct large-scale experiments to compare and analyze different options of them. We first introduce the details of datasets and algorithms and then describe the experimental protocols of our study.

Table 3. Statistics of the Used Datasets for Our Experiments

Dataset		ML-1M	Netflix	Last.FM	Yelp	AMZ_Movie	AMZ_Elec	AMZ_Video	AMZ_Toys
original	#User	6,040	248,873	1,893	1,542,657	1,881,064	3,510,477	688,894	1,151,803
	#Item	3,629	17,493	17,633	202,668	189,334	433,476	46,944	305,729
	#Interaction	836,478	5,754,113	92,834	6,102,153	4,034,282	6,466,395	1,094,400	1,943,977
	Sparsity%	96.1838	99.8678	99.7219	99.998	99.9989	99.9996	99.9966	99.9994
	#AIU	138.51	23.12	49.07	3.96	2.14	1.84	1.59	1.69
	#AII	230.56	328.96	5.27	30.11	21.31	14.92	23.31	6.36
5-core	#User	6,039	140,075	1,860	245,368	105,027	150,524	18,814	15,529
	#Item	3,308	15,034	2,824	99,897	44,211	52,030	8,692	9,697
	#Interaction	835,789	5,535,469	71,355	3,821,892	1,406,666	1,312,570	177,572	133,837
	Sparsity%	95.8162	99.7371	98.6415	99.9844	99.9697	99.9832	99.8912	99.9111
	#AIU	138.42	39.52	38.38	15.58	13.39	8.72	9.44	8.62
	#AII	252.73	368.22	25.28	38.26	31.82	25.23	20.43	13.8
10-core	#User	6,034	103,286	1,798	90,211	26,969	13,456	1,782	828
	#Item	3,124	12,370	1,508	50,667	18,564	8,361	1,448	811
	#Interaction	834,449	5,269,251	62,376	2,521,784	762,957	2,34521	32,375	16,080
	Sparsity%	95.5733	99.6876	97.6995	99.9448	99.8476	99.7915	98.7453	97.6054
	#AIU	138.31	51.02	34.71	27.95	28.29	17.43	18.18	19.44
	#AII	267.19	426	41.39	49.77	41.1	28.05	22.37	19.85
Timestamp		✓	✓	×	✓	✓	✓	✓	✓
Rating		✓	✓	×	✓	✓	✓	✓	✓
Domain		Movie	Movie	Music	Shopping	Shopping	Shopping	Shopping	Shopping

#AIU denotes the average number of interactions per user, and #AII denotes the average number of interactions per item.

3.3.1 Datasets. By surveying the collected papers, we select eight representative datasets. Among the eight datasets, MovieLens-1M, Netflix, Last.FM, and Yelp are mostly used by existing works. Another four datasets are from the Amazon data collection [75], containing multiple domains. The original Amazon data collection consists of 24 domains, and we follow Reference [138] to select four representative domains with different characteristics. Next, we present the details of the eight datasets.

- **MovieLens-1M** [42] is a widely used benchmark dataset in movie recommendations, which contains 1,000,209 explicit ratings (an integer between 1 and 5) from 6,040 users on 3,629 movies.
- **Netflix** [7] is a large real-world movie dataset from Netflix, which is collected between November 1999 to December 2005. It contains 100,480,507 explicit ratings (an integer between 1 and 5) from 480,189 users on 17,770 movies.
- **Last.FM** [12] contains listening records of users from the online music system Last.fm. It contains 92,834 clicks from 1,892 users on 17,632 music.
- **Yelp** is the Yelp Challenge Dataset released by Yelp, which contains user check-ins at local businesses, together with user reviews and local businesses information network. It contains 8,021,122 explicit ratings (an integer between 1 and 5) from 1,968,703 users on 209,393 businesses.
- **Amazon Movies_and_TV, Electronics, Toys_and_Games, Video_Games** [75] are obtained from the Amazon review dataset in different categories. These datasets include a significant amount of user-item interaction data.

We summarize the basic statistics of these datasets in Table 3. For the top-N recommendation, we transform explicit feedback datasets into implicit feedback datasets by setting a binarized threshold of 3. Overall, the selected datasets cover diverse domains and contain varying numbers of users, items, and interaction records.

3.3.2 Comparison Algorithms. To carry out our experiments, we further select 12 top-N recommendation algorithms. Among them, 4 are non-neural algorithms, and 8 are neural algorithms.

According to the architecture, we categorize these 12 algorithms into five groups, namely traditional algorithms, MF-based algorithms, similarity-based algorithms, GNN-based algorithms, and non-sampling algorithms.

(1) Traditional algorithms
- **Popularity:** This algorithm ranks items based on their popularity, evidenced by the number of interactions in the training set. This is a non-personalized algorithm.
- **ItemKNN** [1]: This is an item neighborhood-based collaborative filtering algorithm. It exploits cosine item-item similarities to produce recommendation results.

(2) Matrix factorization-based algorithms
- **SVD++** [62]: It is a variant of singular value decomposition, which projects users and items into latent factors and reconstruct the interaction matrix.
- **Bayesian Personalized Ranking-based Matrix Factorization (BPRMF)** [90]: It is a classic algorithm for learning pairwise personalized rankings from users' implicit feedback data.
- **NCF** [47]: It combines the MF algorithm with a MLP to learn the user-item interaction function.

(3) Item similarity-based algorithms
- **Factored Item Similarity Model (FISM)** [60]: It is an item-based CF algorithm in which the user preference vector is derived by the sum of historical item vectors instead of looking up the user embedding matrix.
- **Neural Attentive Item Similarity Model (NAIS)** [46]: It generalizes the factored item similarity model by employing an attention mechanism.

(4) Graph neural network-based algorithms
- **Neural Graph Collaborative Filtering (NGCF)** [113]: It adopts three GNN layers on the user-item interaction graph, aiming to refine user and item representations via at most three-hop neighbors' information.
- **Simplifying and Powering Graph Convolution Network (LightGCN)** [44]: It is a special GNN-based algorithm for recommendation, which discards the feature transformation and the nonlinear activation functions in the GCN aggregator.

(5) Non-sampling algorithms
- **CDAE** [118]: It is specifically optimized for implicit feedback recommendation tasks by utilizing the idea of denoising auto-encoders (DAE).
- **MultiVAE** [72]: It extends variational and denoising autoencoders to collaborative filtering using a multinomial likelihood.
- **ENMF** [15]: It is a recently proposed non-sampling neural recommendation algorithm. It is a state-of-the-art algorithm for top-N recommendation that is only based on the historical feedback information.

These algorithms are widely adopted as the comparison baselines in existing research, and well cover the major categories of recommendation algorithms. Another reason to select these baselines is that they have released official implementation codes with optimization details.

3.3.3 Configuration and Implementation. To examine the impact of the options for the studied factors, we do not focus on specific recommendation performance of algorithms, since even the same algorithm might have very different results under different settings. Instead, we mainly consider the change of the overall ranking of different comparison algorithms. We introduce the term "*configuration*" to denote a kind of combination for different settings of the studied factors. In this way, our study is to measure and analyze how the rankings of recommendation algorithms

change under different configurations. Given a configuration, we can obtain a ranked list of the 12 comparison algorithms, called *performance ranking*, according to the descending order of their performance based on some metric. We next study how to measure the correlation or similarity degree between two performance rankings. We consider two measures to quantitatively characterize the correlation degree between two performance rankings:

• ***Overlap Ratio at top-k positions (OR@k).*** We compute the overlap ratio of top k algorithms between two ranked lists:

$$OR@k(c_1, c_2) = \frac{\left|R_k^{(c_1)} \cap R_k^{(c_2)}\right|}{k}, \tag{1}$$

where c_1 (or c_2) is a configuration, $R_k^{(c_1)}$ (or $R_k^{(c_2)}$) is the set of algorithms that are ranked at top k positions under configuration c_1 (or c_2), and $|R_k^{(c_1)} \cap R_k^{(c_2)}|$ is the number of algorithms in the intersection set. It equal to 1 when there are totally the same algorithms that are ranked at top k. In our experiments, we set $k = 5$, namely $OR@5$.

• ***Spearman Rank Correlation (SRC).*** SRC is commonly used to measure the association between two ranked lists:

$$SRC(c_1, c_2) = 1 - \frac{6 \sum_{a \in \mathcal{A}} \left(r_a^{(c_1)} - r_a^{(c_2)}\right)^2}{|\mathcal{A}|^3 - |\mathcal{A}|}, \tag{2}$$

where $\mathcal{A}$ is the set of all compared algorithm (with size $|\mathcal{A}|$), and $r_a^{(c_1)}$ is the ranking number of algorithm a under configuration c_1.

The first measure focuses on the top-ranked algorithms and the second measure focuses on the overall rank correlation. In the comparison of recommendation algorithms, we might concern either the top performers or the overall ranking. These two measures are able to describe the impact of different options on the comparison of algorithms. To examine the effect of a studied factor, we will accordingly generate multiple configurations by varying its possible settings. Then, we compute the correlation degree between the rankings under two different configurations using the above two measures. Finally, the correlation results will be averaged over multiple configuration comparisons.

3.3.4 Experimentation with Recbole. To ensure the reliability and reproducibility of the results, we conduct all the experiments with a latest open sourced recommender system library, *RecBole*[2] [139]. It is a unified, comprehensive and efficient framework that has integrated 78 recommendation algorithms and 28 datasets, covering all the comparison algorithms and datasets in our experiments. All the algorithms are released after a strict code review and test process. In this section, we give a brief introduction of how to set the studied factors to reproduce the experimental results with *RecBole* and then present some examples.

Set the studied factors in *Recbole*. For this work, RecBole has supported most of the our studied options, and our experiments can be implemented in a unified framework, making the results reproducible. A detailed introduction of the library can be found in the paper for RecBole [139].

The studied factors can be easily configured in command line parameters or configuration files. For example, the target metrics can be set by the *metrics* parameter. Besides, the splitting of dataset and candidate sampling strategy can be set via the *eval_setting* parameter that support all the five splitting methods and three candidate generation methods. For dataset preprocessing, long-tailed users and items can be further filtered by the *min_user_num* and *min_item_num* parameter. We summarize the detailed commands for various setting with RecBole in Table 4. In the following,

[2]GitHub Page: https://github.com/RUCAIBox/RecBole.

Table 4. Example usage of RecBole with Various Settings for Evaluating Recommender Systems

Factors	Options	Command
Dataset	ML-1M, Yelp, Netflix, Amazon, Pinterest, Taobao, etc.	`-dataset=Yelp`
Algorithm	BPR, ItemKNN, NCF, NGCF, ENMF, NAIS, etc.	`-model=BPR`
Filtering	Filter user of item under any number of integer	`-min_user_num=5` `-min_item_num=5`
Splitting	RO_RS, RO_LS, TO_RS, TO_LS, etc.	`-eval_setting=RO_RS,full`
Cancidates	full, uni100, uni500, uni1000, pop100, pop500, etc.	
Metrics	Recall, NDCG, MAP, MRR, Hit, Precision, AUC	`-metrics=[''Recall'',''GAUC'']`

we will present two examples on how to run customized data and perform hyperparameter search with RecBole.

Examples of configuration. With Recbole, we can either set the parameters of evaluation with a single-line command or through a readable configuration file. In this part, we present an illustrative example for how to train and evaluate algorithms with customized settings to reproduce the results. For example, we can train and evaluate BPR algorithm on ML-1M dataset with TO_RS splitting by the following command:

```
python run_recbole.py --dataset=ml-1m --model=BPR --eval_setting=TO_RS,full
```

Furthermore, we can set filtering strategy and metrics detailedly in a YAML file as

```
eval_setting: TO_RS,full
min_user_inter_num: 5
min_item_inter_num: 5
metrics: ["Recall","NDCG","Hit"]
```

Then run the following command to obtain the results:

```
python run_recbole.py --dataset=ml-1m --config_files=config.yaml --model=BPR
```

To find optimal hyper-parameters, we can configure the hyper-parameters and corresponding range in a file with the following format:

```
learning_rate choice [5e-5,1e-4,5e-4,1e-3]
dropout_prob choice [0.0,0.1,0.3]
```

And the optimal hyper-parameters and evaluation results can obtained by the following command:

```
python run_hyper.py --dataset=ml-1m --model=NGCF --params_file=range.hyper
```

Unless specified, the results of the comparison algorithms are reported with an exhaustive hyper-parameter search. To support this work, we have set up a GitHub page at https://github.com/RUCAIBox/RecSysEvaluation that includes all the possible commands or scripts to reproduce all the results with RecBole.

4 EVALUATION METRICS

In this section, we study the impact of the settings for evaluation metrics, which is a fundamental factor to be determined before evaluation.

4.1 Consistency of Evaluation Metrics

In the literature, various metrics have been proposed to evaluate the performance of recommendation algorithms [101]. However, it is common that a research paper only adopts a small number of

metrics for evaluation (e.g., limited to paper length), either based on subjective selection or following previous work. In this part, we first review nine widely used evaluation metrics, and analyze their impact on the performance rankings of recommendation algorithms.

4.1.1 Reviewing Evaluation Metrics. In principle, top-N recommendation can be considered as a ranking task [64], where a good recommendation algorithm should be able to rank proper items at top positions. In this manner, many ranking-oriented measures are used for evaluating recommender systems. Here we give a simple review of several widely used evaluation metrics for recommendation.

Specifically, suppose there is a set of items to be ranked. Given a user u, let $\hat{R}(u)$ represent a ranked list of items that an algorithm produces, and $R(u)$ represent a ground-truth set of items that user u has interacted with. For top-N recommendation, only top-ranked items are important to consider. Therefore, we simplify the evaluation scenarios by truncating the recommendation list with a length k. Hence, the following evaluation metrics are defined and computed based on this truncated list. We present the specific formulations for different evaluation metrics as follows:

- **Truncated Precision at top k positions (Precision@k)** [101]: It is a metric that computes the fraction of correct items by an algorithm for top k recommendations:

$$Precision@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\hat{R}(u) \cap R(u)|}{|\hat{R}(u)|}, \tag{3}$$

  where $|\hat{R}(u)|$ represents the item count of $\hat{R}(u)$ and $\mathcal{U}$ denotes the user set with size $|\mathcal{U}|$. Here, $|\hat{R}(u)| = k$.
- **Truncated Recall at top k positions (Recall@k):** It is a metric for computing the fraction of relevant items out of all relevant items:

$$Recall@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\hat{R}(u) \cap R(u)|}{|R(u)|}, \tag{4}$$

  where $|R(u)|$ represents the item count of $R(u)$.
- **F1:** F1-measure is the harmonic mean of precision and recall, which is a widely used measure in recommendation:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{5}$$

  where we can compute $F1$ according to truncated or original precision/recall values.
- **Hit-Ratio at top k positions (HR@k):** Hit-Ratio is an all-but-one measure used in recommendation [35]. If there is at least one item that falls in the ground-truth set, then we call it a hit. HR@k is calculated in the following way:

$$HR@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} I\Big(\hat{R}(u) \cap R(u) \neq \emptyset\Big), \tag{6}$$

  where $I(\cdot)$ is an indicator function and $\emptyset$ denotes the empty set.
- **Mean Average Precision (MAP):** It measures the precision at all ranks that hold a relevant item [64]:

$$MAP = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|R(u) \cap \hat{R}(u)|} \sum_{j=1}^{|\hat{R}(u)|} I\Big(\hat{R}_j(u) \in R(u)\Big) \cdot Precision_u@j, \tag{7}$$

  where $\hat{R}_j(u)$ is the jth item in the recommendation list of $\hat{R}(u)$.

- **Mean Reciprocal Rank (MRR):** It computes the reciprocal rank of the first relevant item found by an algorithm:

$$MRR = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{rank_u^*}, \tag{8}$$

where $rank_u^*$ is the rank position of the first relevant item found by an algorithm for a user u.

- **Normalized Discounted Cumulative Gain (NDCG):** It is a metric from information retrieval, where positions are discounted logarithmically [95]. It accounts for the position of the hit by assigning higher scores to hits at top ranks [47]:

$$NDCG = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{Z} \sum_{j=1}^{|\hat{R}(u)|} \frac{2^{I(\hat{R}_j(u) \in R(u))} - 1}{\log_2(j+1)}, \tag{9}$$

where $\hat{R}_j(u)$ stands for the item recommended in at jth position, and Z is a normalized factor that denotes the ideal value of DCG given $R(u)$. Note that, for implicit feedback, we only consider two relevance levels, namely relevant and non-relevant here.

- AUC measures how many positive items are ranked above negative items, and is normalized by the total number of possible pairs. Its value is 1 for a perfect ranking (i.e., each positive item is ranked above all negative items):

$$AUC = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|R(u)|(|\mathcal{I}| - |R(u)|)} \sum_{j \in R(u)} \sum_{j' \notin R(u)} I(p(j) > p(j')), \tag{10}$$

where $p(j)$ stands for the ranking position of item j.

Besides the above metrics, researchers have an increasing attention on beyond-accuracy metrics [3], which are designed to measure the performance of non-accuracy aspects for recommender systems, including novelty [140], diversity [94], unexpectedness [76], serendipity, and coverage [101]. Here, we mainly consider using two widely used beyond-accuracy metrics, namely novelty and coverage. More metrics can be found in Reference [101].

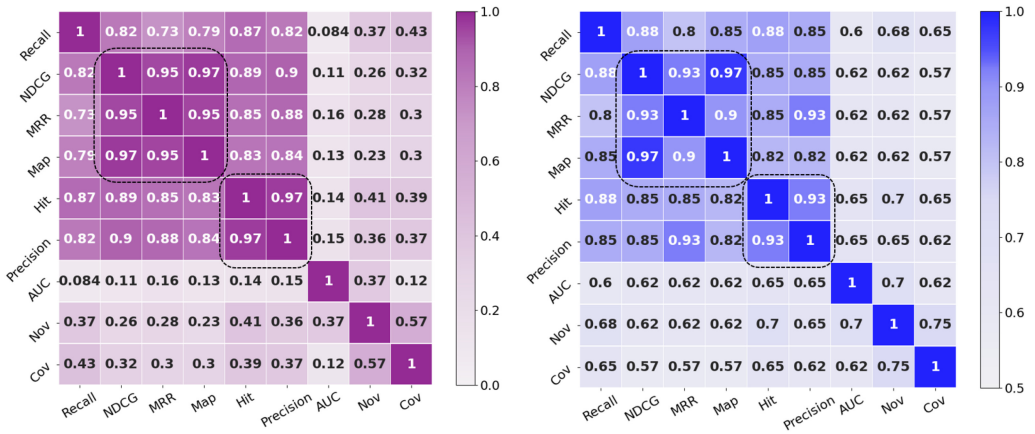
- **Novelty:** This metric advocates the recommendations with novel items that are different from existing ones. Following [101], it can be defined as follows:

$$Novelty@k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{j \in \hat{R}(u)} \frac{\log(\text{pop}(\hat{R}_j(u)) + 1)}{k}, \tag{11}$$

where $\hat{R}_j(u)$ represents the jth item recommended to user u, and $\text{pop}(\hat{R}_j(u))$ represents the popularity of item $\hat{R}_j(u)$ (e.g., the number of interactions related to $\hat{R}_j(u)$). In general, novelty can be defined in other ways, such as the proportion of unknown items in the prediction list for each user. In this work, we adopt the above simple version that calculates the popularity of recommended items. Here, a smaller is better is preferred for Novelty.

- **Coverage:** Coverage targets on the whole recommender systems instead of a single recommendation list. There are three kinds of coverage measures proposed in existing studies [101], including item space coverage, user space coverage, and genre space coverage. Following Reference [101], we select item coverage for study, since it is more frequently used in our collected papers. It is defined formally as

$$Coverage = \frac{|\bigcup_{u \in \mathcal{U}} \hat{R}(u)|}{|\mathcal{I}|}, \tag{12}$$



(a) Correlation on SRC (b) Correlation on OR@5

Fig. 1. Visualization of metric correlations on SRC (in purple) and OR@5 (in blue). Each cell indicates the computed SRC or OR@5 score between two metrics (a darker color indicates a larger correlation). We omit the @10 postfix for all metrics except AUC.

where $|\mathcal{I}|$ represents the count of all items. In a real recommender system, there are usually a large number of items to be recommended. Coverage encourages the algorithm to recommend more diverse items, so it can be an indicator of algorithm's capacity in covering more items of the item set. Here a larger value is preferred for Coverage.

4.1.2 Research Question and Experimental Setup. Given a number of available evaluation metrics, we study the following two questions:

- Whether different metrics lead to similar performance rankings of recommendation algorithms?
- How to choose proper metrics for evaluating recommendation algorithms?

To study the above questions, the 12 algorithms (Section 3.3.2) are tested on the eight datasets (Section 3.3.1). Specifically, for each dataset, we run all the compared algorithms to obtain their performance scores and then produce a ranked list of algorithms for each metric. Then, for each pair of two metrics, we compute the SRC (Equation (2)) and OR@5 (Equation (1)) between the corresponding two ranked lists. We repeat this process on each dataset and average the ranking correlation scores as the consistency degree between two metrics. In this way, we can derive a correlation heatmap among all pairs of evaluation metrics.

The detailed configuration for the rest factors is listed below. Yelp and Netflix datasets are processed with 10-core filtering as they are very large, and the other datasets are processed with 5-core filtering. The interaction of each user is ordered by timestamp, and the entire dataset is split to train/validation/test sets by a ratio of 0.8/0/1/0.1. To optimize the algorithm, we employ the validation set for parameter search. To generate the candidate item list, we treat all the items that a user has not interacted with as candidates. All the metrics are computed based on a truncated length of 10, except AUC, which is computed on the whole list.

4.1.3 Observation and Discussion. Figure 1 presents the average correlation results among different metrics on SRC and OR@5, where a darker color indicates a stronger correlation. We have the following observations:

(1) Overall, the heatmaps in both figures reveal that the metrics at the top left (Recall, MRR, NDCG, HR, Precision, and MAP) have higher correlations with each other (called *the first part*), while the three metrics at the bottom right have relatively weak correlations with the rest metrics. Interestingly, the first part of metrics are further divided into three dense groups, namely {*Recall*}, {*MRR*, *NDCG*, *HR*} and {*Precision*, *MAP*}, where the metrics in each group have a correlation degree that is close to one. In general, it is suggested that a research study in recommender system should use as more metrics as possible. However, it is usually infeasible to report the results of all the metrics due to space limitation. This finding indicates that if the number of adopted metrics are limited, then we should cover different groups and select representative metrics on the whole. Actually, similar correlation results are also found in previous studies [107, 109]. Sun et al. [107] found that Recall tends to be poorly correlated with other metrics, while the rankings by Precision, HR, MAP, MRR, and NDCG show a fairly strong correlation.

(2) Among the rest metrics (AUC, Novelty, and Coverage), AUC is a commonly adopted metric in industry [106]. However, we do observe that recent recommender system papers often neglect this metric for evaluation. As we see, it has very weak correlations with other traditional metrics (the part of metrics at the top left). A possible reason is that those ranking metrics are computed with a small cutoff that only concerns the top-k results. In the extended experiments (reported in the Appendix from the website of this work), we find that the SRC between AUC and NDCG increases from 0.11 to 0.73 when the cutoff length varies from 10 to 1000: The SRC value is still not high for a large cutoff length of 1000. It indicates that AUC (a pairwise metric) is essentially different from other ranking-based metrics. Therefore, the above results suggest that AUC should be used as evaluation metrics, since it measures the recommendation results in a substantially different way compared with other metrics.

(3) As for the Novelty and Coverage metrics, both of them show a relatively weak positive correlation with the other metrics (shown in the two bottom rows of the heatmaps). This finding has also been reported in previous studies [37, 140], where beyond-accuracy metrics take a different perspective to evaluate the recommendation quality. In recent studies, there are increasing interests in beyond-accuracy metrics [3], since it has been found that beyond-accuracy metrics are particularly useful to guide the improving of recommendation algorithms in non-accuracy aspects, such as diversity and novelty of recommendation [109]. We suggest using both accuracy and beyond-accuracy metrics in these works.

4.2 Sampled Metrics

To test the performance of a recommendation algorithm on a given dataset, it will be time-consuming to take all the items from the item set as the candidates when the size of item set is large. An alternative way is to sample a small set of items as irrelevant items, so that the performance scores can be computed based on a partial list of the ground-truth item and sampled negative items. Such a way is usually called *sampled metrics* [64]. Now, sampled metrics have been widely used to reduce the time cost of enumerating the entire item set. However, more recently, several studies [9, 64, 69] raised serious concerns on sampled metrics and argued that sampled metrics tend to lead to inconsistent ranking results. In this part, we investigate into the impact of sampled metrics on performance ranking, and study how different configurations affect the final comparison results.

4.2.1 Sampling Method. To set up our experiments, we first consider the possible sampling methods for sampled metrics. Actually, the sampling-based evaluation was first used in Reference [62], where 1,000 movies are sampled uniformly from the item set with the ground-truth movie for each user. We refer to this method by *uniform sampling*. Besides, there are also some variants that

are either user based [49] or popularity based [84]. Here we mainly consider uniform sampling and popularity-based sampling. To configure the sampling method, we can also prepare different numbers of negative items, and vary the number of negative items in the set {100, 200, 500, 1000}. Note that our sampling method corresponds to the test stage instead of the learning stage. The sampling issue related to the learning stage will be left as future work.

It has been found that existing sampling methods tend to introduce bias in performance ranking. A few amendment measures have been proposed to alleviate this issue [64]. Here we adopt the amendment measure of debiased estimator from Reference [64] as an improved sampling method:

$$\hat{f}(\tilde{r}) = f\left(1 + \frac{(n-1)(r-1)}{m}\right), \tag{13}$$

where r represents the ranking of positive instance on sampled candidates, $\tilde{r}$ is the correspondingly estimated ranking for whole candidates, m and n denote the amount of candidates under sampling and no-sampling respectively, and $f(\cdot)$ represents any metric that calculates on the rankings.

4.2.2 Research Question and Experimental Setup. Given the concerns on sampled metrics [64], we aim to study the following research questions:

- whether sampled metrics will lead to different performance rankings compared with non-sampling ranking?
- how different factors of the sampling method affect the performance ranking?

For these research questions, we consider the two kinds of sampling methods, namely uniform sampling and popularity-based sampling (Section 4.2.1), and use different numbers of irrelevant items in the set {100, 200, 500, 1000}. Given a combination of sampling method and negative number, we can derive a candidate list of items for each dataset, rank different algorithms according to their performance on both *sampled ranking list* and *full-ranking list*, and compute the SRC and OR@5 values between the two kinds of performance rankings. We repeat this process on each dataset and report the average results. Moreover, we adopt an amendment function on the sampled metrics (Equation (13)) to evaluate the performance under amendment for comparison.

Since previously we have found high correlations among some metrics, we only use three representative metrics (Recall@10, NDCG@10, AUC) for ranking. The rest procedures (e.g., data preprocessing) follows the same way as reported in Section 4.1.2.

4.2.3 Observations and Discussion. Figures 2 and 3 present the results of SRC values and OR@5 on eight datasets respectively, where each dataset corresponds to four lines containing two sampling methods and two debiased versions according to Equation (13). We show the changing trend across different numbers of negative items. The following observations can be found as follows:

(1) First, for uniform sampling (Figure 2), almost all the SRC scores show an increasing trend when we enlarge the size of candidates with uniform sampling (red lines in eight subfigures). After examining the SRC scores, we can find that only Last.FM dataset shows a strong correlation score (>0.9), while the rest datasets show a relatively weak correlation score. Therefore, it is suggested that the entire item set should be considered when testing. Similar findings have been reported in References [64, 69]. However, these studies mainly focus on theoretical analysis, and they only consider a small number of algorithms for comparing the performance ranking. Our results provide empirical evidence with a number of recommendation algorithms on large-scale datasets for this finding.

(2) Second, the amendment measures (Figure 2) seem to have limited capacity in eliminating the ranking bias yielded from incomplete candidate lists. Specifically, when the original SRC is high (>0.6), the debiased SRC is equal to or lower than it; while, when the original SRC is relatively low,

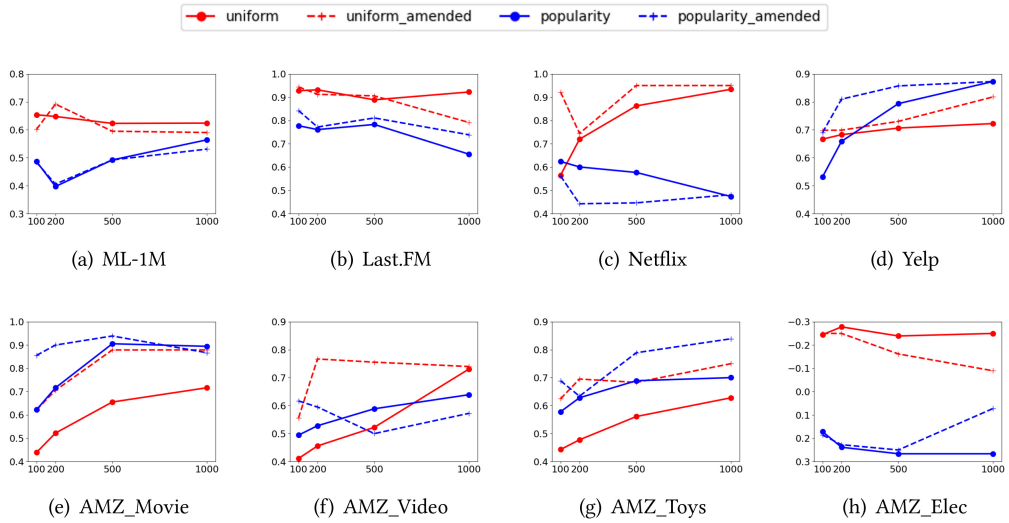


Fig. 2. SRC results of the performance ranking with different sizes of candidates on eight datasets.

the debiased SRC scores are substantially improved. Based on these empirical results, it seems such a corrected estimation method cannot fully resolve the issue of sampled metrics. We speculate that the original proof and derivation are based on strict assumptions and limited to some specific kind of algorithms. The effectiveness of target sampling and the effect of target sets are also discussed in Reference [9], and they find that tie analysis is more informative than traditional statistical significance tests in comparative evaluation.

(3) Last, from the OR@5 results (Figure 3), the change of these lines are more mitigatory than that for SRC results in Figure 2. Major findings from SRC results are similar for OR@5 results. It is interesting to see that there is little impact on top-ranked algorithms if a large number of negative samples are used. Overall, uniform sampling leads to a more similar ranking compared with the full-ranking list (red line is above or overlaps with the blue line in five of the eight datasets). Except the last two datasets (AMZ_Toys and AMZ_Elec), the rest datasets correspond to a value equal to or larger than 0.8, which means that four of the top five algorithms in full-ranking comparison remain among the top five positions with sampled metrics on average. These findings might suggest that sampled metrics mainly affect the performance of weak recommendation algorithms.

5 DATASET CONSTRUCTION

In this section, we study the impact of different dataset construction strategies on performance evaluation. Data construction refers to a series of steps on how to select and preprocess the original data and construct the training, validation, and test sets.

5.1 Dataset Selection and Preprocessing

As the first step, to construct the evaluation sets, we first need to select the datasets and make proper preprocessing on original data copies. Before our discussions, we first make a brief review on existing evaluation datasets.

5.1.1 Existing Evaluation Datasets. By taking a thorough examination of the collected recommender system papers, we find that each paper conducts the evaluation with 2.9 public datasets

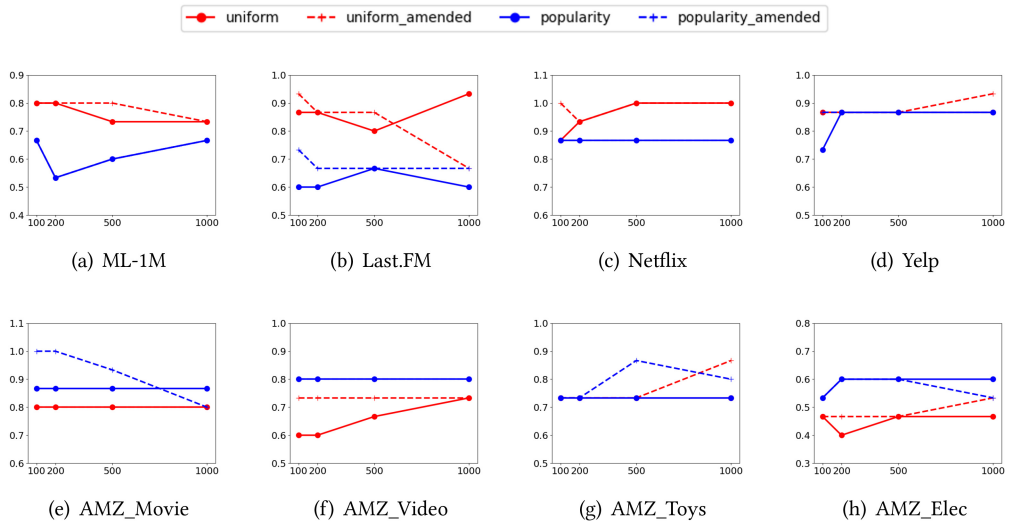


Fig. 3. OR@5 of the performance ranking with different sizes of candidates on eight datasets.

on average. Among these papers,[3] 27% of the papers use more than three datasets, and 14% of the papers only report the performance on a single dataset. According to our statistics, ML-1M, Last.FM, Netflix, and Yelp are the four most frequently used datasets. Besides, Amazon datasets are also widely used. As mentioned in Section 3.3.1, our experiments include the four most frequently used datasets and four Amazon datasets of different domains to cover both diverse and frequent datasets. It is suggested that researchers should use more diverse datasets for evaluation [107, 138].

To preprocess the datasets, a commonly used strategy is to filter users or items with a very small number of interactions, and only keep users or items with at least n interactions. Such a strategy is usually called *n-core filtering*. Based on our paper collection, 44% of the papers offer no information about the preprocessing step, and 34% of the papers adopt 5-core or 10-core filtering. However, even for those with n-core filtering, there are also inconsistent processing methods: Some studies perform the filtering once and other studies repeat removing users or items until all the users or items have at least n interactions. In this work, we adopt the latter approach. We also find that some papers [123, 132] only perform the filtering in an unsymmetrical way, e.g., only for users or items. Besides, several papers [27, 34] set a relatively large value for n (e.g., $n = 25, 30$), which can derive a more dense dataset. However, it might significantly change the overall distributions of user-item interactions of the original dataset.

5.1.2 Research Question and Experimental Setup. Based on the above discussions, we study the following two research questions:

- How do different data filtering strategies affect the performance ranking?
- How to select proper datasets for evaluation?

To conduct the studies, for each dataset, we first consider applying one of two filtering strategies {5-core, 10-core} to obtain the filtered datasets. Then, for each of the two filtering strategies, we compute and report their ranking correlation (using the SRC measurement in Equation (2)) with the original data without filtering. Such a setting essentially follows Reference [107], while

[3]Throughout this article, the reported percentage is calculated based on our collected papers. Please refer to the GitHub page for a complete list of these papers.

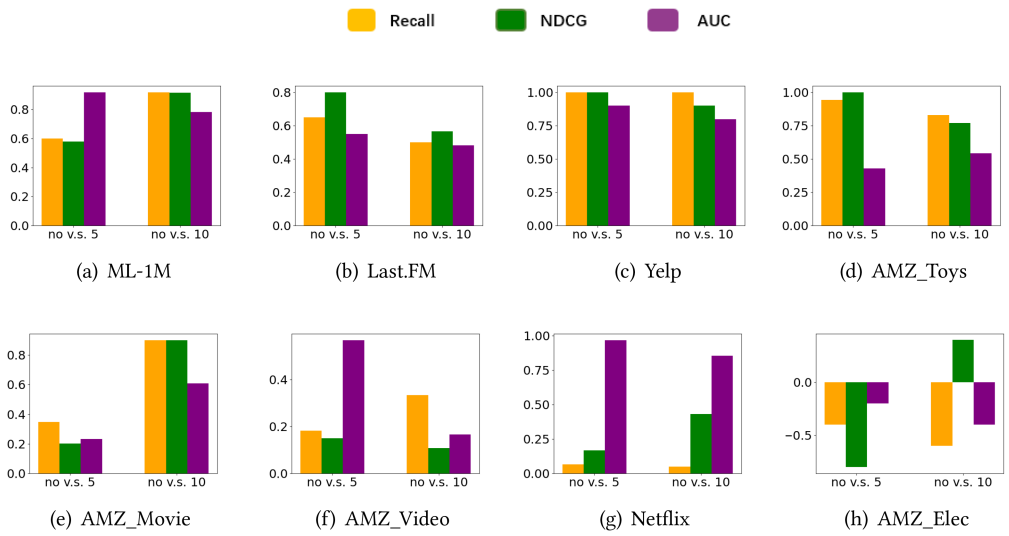


Fig. 4. SRC results of the performance ranking correlation on eight datasets. For each dataset, we compare the rankings with filtered or original datasets.

we adopt the overall correlation measurement with more comparison algorithms. We report the results with three representative metrics (Recall@10, NDCG@10, and AUC) for ranking. The rest setup follows the same way as reported in Section 4.1.2.

5.1.3 Observations and Discussion. Figure 4 presents the SRC values on eight datasets, where each dataset corresponds to the comparisons between different filtering strategies: no-filtering vs. 5-core filtering and no-filtering vs 10-core filtering.

The following observations can be found as follows:

(1) First, the two filtering strategies seem to lead to very different rankings for four of the eight datasets (bottom part), especially on the Amazon electronic dataset. The major difference between different processed data copies lies in the proportion of infrequent users or items and their sparsity levels. To show this, we present the detailed statistics of the eight datasets in Table 5. We can see that different filtering strategies can significantly change the original data characteristics and distributions. For example, the number of contained users will be dramatically reduced, and the overall sparsity levels seem to have substantial variations. In particular, it can be observed that the performance with respective to AUC is less sensitive (except the Amazon electronic dataset) for the filtering strategies than the other metrics. Such an observation is somehow related to our previous finding in Section 4.1: AUC seems to have a weak correlation with the other metrics.

(2) It is relatively difficult to draw a conclusion on the characteristics of the datasets that are relatively insensitive to the filtering strategies. Empirically, we find that the change of the measure $\frac{\#AII}{\#AIU}$ (computed by the ratio between the average number of interactions per item and the average number of interactions per user) are indicative of the ranking correlations to some extent. As shown in Table 5, it seems that the less insensitive datasets (with large correlation values) have relatively smaller values for $\frac{\#AII}{\#AIU}$. Such a measure reflects the variation of the sparsity levels from the perspectives of item and user, respectively. A more "stable" dataset tends to have a value of $\frac{\#AII}{\#AIU}$ more close to 1, which means that the distribution of interactions are balanced in terms of both users and items. Given a dataset with a large value $\frac{\#AII}{\#AIU}$, the original data distribution is likely to change if data filtering strategies apply, which may further affect the performance ranking.

Table 5. Detailed Statistics of Eight Datasets under Different Filtering Strategies

Dataset		ML-1M	Last.FM	AMZ_Toys	Yelp		AMZ_Movie	AMZ_Elec	AMZ_Video	Netflix
original	#User	6,040	1,893	1,151,803	1,542,657		1,881,064	3,510,477	688,894	248,873
	Sparsity%	96.1838	99.7219	99.9994	99.998		99.9989	99.9996	99.9966	99.8678
	#AII/#AIU	1.66	0.11	3.76	7.6		9.96	8.11	14.66	14.23
5-core	#User	6,039	1,860	15,529	245,368		105,027	150,524	18,814	140,075
	Sparsity%	95.8162	98.6415	99.9111	99.9844		99.9697	99.9832	99.8912	99.7371
	User Drop%	0.02	1.74	98.66	84.1		94.42	95.72	97.27	43.72
	#AII/#AIU	1.83	0.66	1.6	2.46		2.37	3.24	2.16	9.32
10-core	#User	6,034	1,798	828	90,211		26,969	13,456	1,782	103,286
	Sparsity%	95.5733	97.6995	97.6054	99.9448		99.8476	99.7915	98.7453	99.6876
	User Drop%	0.1	5.02	99.93	94.16		98.57	99.62	99.74	58.5
	#AII/#AIU	1.93	1.19	1.02	1.78		1.45	1.61	1.23	8.35

The percentage of removed users and the ratio between average interaction per item (#AII) and average interaction per user (#AIU) are listed for each dataset.

In existing studies [2, 107, 138], researchers suggest using multiple diverse datasets (e.g., more domains [138] and more sparsity levels [2]) for evaluating recommender systems. Note that the four Amazon datasets are selected based on the correlation results from Reference [138], where they find that there are four coherent groups (we select one representative dataset from each group). Based on these studies, we further experimentally find that different data filtering strategies have substantial effect on performance ranking, depending on the characteristics of the specific datasets. In practice, we suggest the researchers use multiple datasets for evaluation, considering domain, sparsity and other data characteristics. It is also suggested that we should be careful to apply the data filtering strategies, which will change the data characteristics and potentially lead to different performance ranking results. More recently, there are also several studies that attempt to analyze the characteristics of evaluation datasets in recommender systems and their effect on the recommendation algorithms [21, 26]. The readers can refer to these studies for a deep understanding of dataset effect for evaluating recommender systems.

5.2 Dataset Splitting

Data splitting aims to divide the original data into training, validation and test sets, having important impact on performance ranking [54, 57]. The major limit of previous studies is that only a small number of comparison algorithms are used. Here, we revisit this factor and conduct the study with more compared algorithms.

5.2.1 Splitting Methods. To split the dataset, we need to consider two processing steps, namely data ordering and splitting. For data ordering, we can consider either an ordering of randomly shuffled data and a chronological ordering of interaction data. For splitting, we can consider either ratio-based splitting or leave-one-out splitting (a special case of the ratio-based splitting). Following References [107, 138], we consider the following four combinations for comparison:

- **Random Ordering Ratio-based Splitting (RO_RS).** It first arranges the interactions of a user randomly, and then splits the sorted sequence into three parts for training, validation and test, respectively, according to predefined ratio (e.g., 0.8/0/1/0.1).
- **Random Ordering Leave-one-out Splitting (RO_LS).** It first arranges the interactions of a user randomly, and then selects one ground-truth item as test set and another one as validation set, while the rest items are considered as training set.
- **Temporal Ordering Ratio-based Splitting (TO_RS).** It first sorts the interactions of a user by timestamp, and then splits the dataset into three parts for training, validation and test, respectively, according to predefined ratio (e.g., 0.8/0/1/0.1).

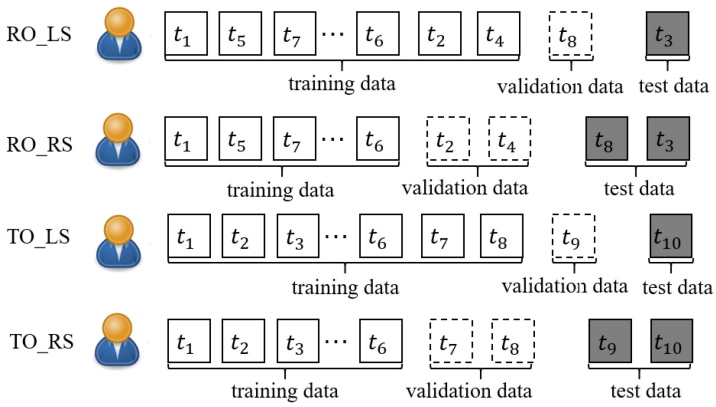


Fig. 5. An illustrative example for four splitting strategies. The user has interacted with 10 items. The timestamp of each interaction is denoted by t and a smaller index indicates a smaller timestamp. We use normal, dash-lined and grey boxes to denote the training, validation and test sets, respectively. This figure is reused from Reference [138].

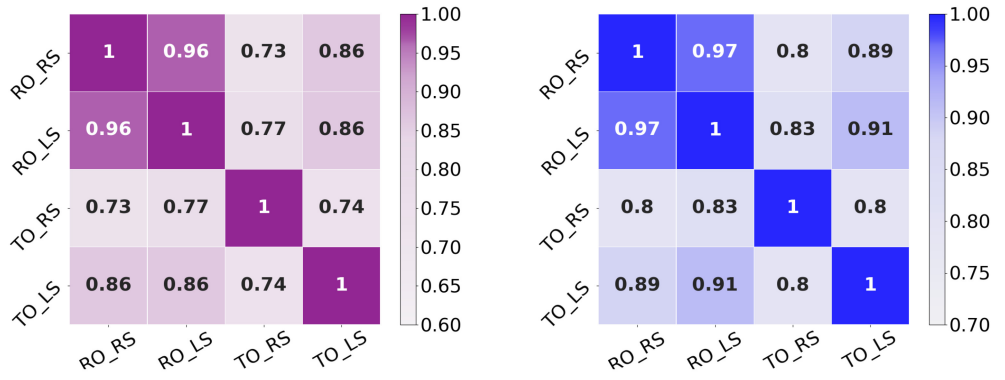
- **Temporal Ordering Leave-one-out Splitting (TO_LS).** It first sorts the interactions of a user by timestamp, and then selects one ground-truth item as test set and another one as validation set, while the rest items are considered as training set.

We present an illustrative example for the four splitting strategies in Figure 5. Based on our statistics, 90% of the papers use one of the four splitting methods, where RO_RS is the frequently used splitting [107] and leave-one-out splitting becomes increasingly more common in experiments. In existing time-based splitting methods, splitting is mainly performed based on relative time, i.e., time index. A recent study [58] argues that global or absolute timestamps should be used when considering splitting the data: The splitting should be conducted on the whole dataset instead of each user. Specifically, each interaction in a dataset should be assigned by the global timeline, so that it can ensure that all the test instances fall behind any training instance in time. Following this suggestion, another possible setting is to split the interaction data according to global timestamps. However, we find such a splitting way is easy to lead to cold-start users. For example, about 40% users do not appear in the training set for Netflix and AMZ datasets by a split ratio of 9:1 on the global timeline. Since our comparison methods are not specially designed for cold-start recommendation, it may be meaningless to evaluate their performance under the globally temporal split. Considering this, we leave a deep analysis with the effect of global time splitting as future work.

5.2.2 Research Questions and Experimental Setup. Based on the above discussions, we study the following two questions:

- How do the performance rankings vary with the four splitting methods?
- Do the four splitting methods make invalid recommendations violating global time constraints?

For the first question, we rank the list of compared algorithms given a splitting method. And, we measure the rank correlations between two splitting methods. The process is repeated for all the datasets except Last.FM dataset that does not contain timestamps. The rest procedures (e.g., data preprocessing) follows the same way as reported in Section 4.1.2. For simplicity, we compute the rank correlations with the metrics of Recall@10.



(a) Correlation on SRC (b) Correlation on OR@5

Fig. 6. Visualization of correlations among four splitting methods. Each cell indicates the computed SRC score (in purple) or OR@5 (in blue) score between two splitting methods (a darker color indicates a larger value).

For the second question, we are inspired by a recent study [58], where the authors carefully study the severity of data leakage when splitting by relative time index of each user. Here, we mainly focus on the average invalid recommendations of each user that violate the constraints of global timing, which is formally computed as follows:

$$\#invalid = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{i \in \hat{R}_u} I\left(T_i^f \geq T_u^l\right), \tag{14}$$

where $\hat{R}_u$ denotes a recommendation list for user u by a method and we truncate it to 10 in this experiment, T_i^f is the first time that item i appears in training set, T_u^l is the time of the last interaction from user u in training set, and $I(\cdot)$ is an indicator function that returns 1 when the condition is true and 0 otherwise.

5.2.3 Observations and Discussion. In this part, we analyze and compare the effect of different ordering and splitting strategies in performance ranking.

Figure 6 presents the average SRC and OR@5 values among four splitting methods on seven datasets. As we can see, different splitting methods are likely to lead to the change of the performance ranking. It becomes more significant when we use different data ordering options to organize the interaction. In practice, the researchers should carefully select the ordering way based on their task focus. When the task is time sensitive, it is better to use temporal ordering instead of random ordering. As a comparison, there seems to be less difference between ratio-based splitting and leave-one-out splitting when the data are randomly ordered. A similar finding has also been reported in Reference [138], where data ordering has more impact on the final performance ranking than data splitting. Furthermore, we find that the difference of splitting strategies (ratio-based splitting and leave-one-out splitting) is more significant under the temporal ordering. When using temporal ordering, we suggest using the ratio-based splitting instead of the leave-one-out splitting. As a comparison, when using random ordering, we can adopt the leave-one-out splitting (the ranking correlation is close to 1) when data are limited. Another observation is that OR@5 is higher than SRC (>0.8), which indicates at most one result is different among the top five rankings (i.e., top results are relatively stable under different splitting settings).

Furthermore, Table 6 shows the number of invalid recommendations under different splitting settings. We can observe that temporal ordering produces more invalid recommendations than

Table 6. The Number of Invalid Recommendations of Each User under the Four Splitting Methods (Averaged over Different Algorithms)

Dataset	#invalid			
	RO_RS	RO_LS	TO_RS	TO_LS
ML-1M	0.054 ± 0.004	0.049 ± 0.004	0.059 ± 0.004	0.054 ± 0.004
Yelp	0.067 ± 0.005	0.053 ± 0.005	0.078 ± 0.006	0.062 ± 0.005
Netflix	0.036 ± 0.000	0.034 ± 0.000	0.074 ± 0.000	0.056 ± 0.000
AMZ_Movie	0.169 ± 0.014	0.247 ± 0.023	0.195 ± 0.015	0.277 ± 0.023
AMZ_Video	0.283 ± 0.022	0.491 ± 0.038	0.414 ± 0.039	0.433 ± 0.040
AMZ_Toys	0.167 ± 0.014	0.353 ± 0.026	0.233 ± 0.020	0.403 ± 0.024
AMZ_Elec	0.079 ± 0.006	0.228 ± 0.019	0.269 ± 0.018	0.361 ± 0.021

random ordering. A major reason is that temporal ordering organizes the items according to the relative interaction time of users. While, the users are active at different time spans in the global timeline, and it is possible that an interaction from the training set has a later timestamp than some interaction at the test set. Since the recommendation algorithms are learned on training set, they tend to be injected into temporal information when learning user preference, thus recommending future items to users [58].

The above findings show that ordering and splitting has important impact on the performance comparison, which should be carefully considered when one prepares the experimental setup.

6 MODEL OPTIMIZATION

Besides evaluation metrics and datasets, another important aspect to consider is model optimization. Since more and more neural recommendation algorithms are used in experimental comparison, it becomes increasingly important to tune and optimize them in a proper way.

6.1 Objective Function

We first study the impact of objective function on the performance of algorithms, which is one of the most important parts for various recommendation algorithms.

6.1.1 Mainstream Objective Functions. Different from rating prediction, for implicit feedback, we usually only have the signals for "*positive data*" (i.e., the items that a user interacts with), while negative data are usually obtained via negative sampling. Following this way, two widely adopted objective functions are the BPR loss [90] and BCE loss [47]. The BPR loss aims to optimize the comparison between a positive item and a negative item, and the BCE loss aims to optimize the probabilities of both positive and sampled negative items. Next, we present the formulations for these two functions:

$$\mathcal{L}_{bpr} = \sum_{(u,i),(u,j)\in\tilde{S}} -\log\sigma(\hat{r}_{ui} - \hat{r}_{uj}), \tag{15}$$

$$\mathcal{L}_{bce} = \sum_{(u,i)\in\tilde{S}} -r_{ui}\cdot\log(\hat{r}_{ui}) - (1-r_{ui})\cdot\log(1-\hat{r}_{ui}), \tag{16}$$

where $\tilde{S}$ is a set of training instances consisting of positive interactions $\{(u,i)|r_{ui}=1\}$ and sampled negative interactions $\{(u,j)|r_{uj}=0\}$, and $\hat{r}_{ui}$ is the predicted label for the (u,i) instance. Thereinto, BPR loss is more commonly used than BCE loss, which accounts for 40% and 30% in our collected papers, respectively. Apart from the mainstream sampling-based objective function, several recent studies proposed to apply non-sampling strategies to fully leverage the unobserved interaction

Table 7. The Optimization and Training Efficiency of the Studied Recommendation Algorithms

Algorithm	Approach	Optimization	Training Efficiency
Pop	Counting	No	★★★★★
ItemKNN	Similarity	No	★★★★★
SVD++	MF-based	BCE	★★★★
BPRMF	MF-based	BPR	★★★★★
NCF	MF+Neural	BCE	★★
FISM	Similarity+MF	BCE	★★
NAIS	Similarity+Neural	BCE	★
NGCF	GNN-based	BPR	★★★
LightGCN	GNN-based	BPR	★★★
CDAE	Non-sampling	Reconstruction	★★★★
MultiVAE	Non-sampling	Reconstruction	★★★★
ENMF	Non-sampling	Whole data	★★★★

We adopt a five-star scheme to denote the levels of training efficiency, and set five corresponding bins according to the training time {<60 s, 60 s–180 s, 180 s–360 s, 360 s–600 s, >600 s} based on ML-1M dataset with our server (more stars indicates more efficient).

data [15, 72, 118]. A key assumption is that it fits the labels of the interactions with the entire item set (all non-interacted items are considered to be negative), which is formally defined as

$$\mathcal{L}_{ae} = \sum_{u \in \mathcal{U}} \sum_{i \in I} \frac{c_{ui}}{2} (r_{ui} - \hat{r}_{ui})^2, \tag{17}$$

where c_{ui} is an optional weight for the interaction of user u and item i, r_{ui} is a binary value that represents whether user u interacts with item i or not, and $\hat{r}_{ui}$ is the predicted value of r_{ui} by a recommendation algorithm. It is computed based on all the items for each user instead of sampled items. To have an overall comparison of these algorithms, we present the optimization approach and training efficiency of the studied algorithms in Table 7. Since these algorithms are built on different architectures (involving very different parameters), it is difficult to formally compare their time complexities in a unified way. Here we present a rough estimation of the training efficiency according to the training time based on the ML-1M dataset. Since Pop and ItemKNN are mainly based on counting statistics or sorting, we do not include them in the following discussions.

6.1.2 Research Question and Experimental Setup. For model optimization, we study the following two questions:

- Which kind of objective functions generally achieves better performance, sampling or non-sampling methods?
- Is there a significant difference between BPR and BCE losses in performance?

Recall that we have presented 12 recommendation algorithms in Section 3.3.2. Except popularity and ItemKNN, the remaining 10 algorithms are implemented with one of the three optimization functions. To conduct the experiments, we categorize 10 algorithms into three categories according to their objective functions, including *Non-sampling algorithm* (CDAE, MultiVAE and ENMF), *BPR-based algorithm* (BPRMF, NGCF, LightGCN), and *BCE-based algorithm* (NCF, SVD++, FISM, and NAIS). For the first question, we aim to examine which category of recommendation algorithms generally has a better performance. For the second question, since BPR and BCE losses are

Table 8. Algorithm Performance with Different Types of Loss Functions

Category	Algorithm	ML-1M			Netflix		
		Recall@10	NDCG@10	AUC	Recall@10	NDCG@10	AUC
Non-Sampling	a	0.0637	0.0711	0.8488	0.0442	0.0284	0.9132
	b	0.0688	0.0709	**0.8653**	0.0433	0.0283	0.9218
	c	0.0681	0.0726	0.8224	0.0426	0.0273	0.9052
BCE based	d_o	0.0632	0.0735	0.8589	0.0356	0.0238	0.9220
	d_m	0.0502	0.0643	0.8328	0.0307	0.0219	0.8710
	e_o	**0.0735**	**0.0760**	0.8585	0.0415	0.0265	0.9144
	e_m	0.0606	0.0705	0.8549	**0.0570**	**0.0374**	0.8844
	f_o	0.0709	0.0752	0.8510	0.0454	0.0293	0.9071
	f_m	0.0379	0.0471	0.8116	0.0459	0.0272	0.8669
BPR based	g_o	0.0666	0.0750	0.8618	0.0379	0.0252	**0.9258**
	g_m	0.0653	0.0734	0.8593	0.0370	0.0245	0.9219
	h_o	0.0684	0.0755	0.8586	0.0373	0.0242	0.9169
	h_m	0.0686	0.0753	0.8586	0.0368	0.0240	0.9196
	i_o	0.0689	0.0740	0.8571	0.0407	0.0264	0.9205
	i_m	0.0662	0.0769	0.8583	0.0384	0.0252	0.9226

The subscript o represents original algorithm and m represents the corresponding modified algorithm (if the algorithm originally used the BPR loss, we would replace it with the BCE loss, and vice versa).

similar, we aim to examine whether changing BCE or BPR loss will lead to a substantial performance change for the latter two categories.

To set up the experiments, we select the ML-1M and Netflix datasets for evaluation, which are the smallest and largest datasets, respectively. We take three metrics (Recall@10, NDCG@10, and AUC) as evaluation metrics, since they have been shown to be correlated with at least one of the rest metrics. The rest procedures (e.g., data preprocessing) follow the same way as reported in Section 4.1.2. The results are illustrated in Table 8. And the best results under each metric are bolded.

6.1.3 Observations and Discussion. The results of the nine algorithms on two datasets are listed in Table 8. Since we are not interested in specific ranking positions of recommendation algorithms, we shuffle the algorithms within each category and mask their names with the alphabets from a to i. For sampling-based algorithms, we also include a variant that is replaced with the other sampling-based optimization function. The following observations can be found as follows.

(1) As we can observe in Table 8, the three non-sampling algorithms generally perform better on Netflix dataset, while they are worse than sampling-based algorithms on the small ML-1M dataset. Except the AUC score, almost all the other cases on ML-1M dataset are won by one of the sampling-based algorithms. In contrast, on the large Netflix dataset, non-sampling methods outperform the sampling-based algorithms except for algorithm f. These results indicate that the superiority seems to be related to the number of items in a dataset: The larger the number of items is, the better performance non-sampling algorithms achieve. Similar findings are also found in knowledge base completion tasks [43, 111], where we can either sample negative entities or consider all the entities that are not in training triples as negative entities. A possible reason is that a user is typically interested in a small number of items, and most items of a large item pool are probable to be negative items for a user. A non-sampling algorithm has intrinsic advantage in capturing the semantics of "*dislike*."

(2) Comparing the two categories of sampling-based algorithms, there seems to be no significant difference between their performance, except that algorithm f gives the best performance on the Netflix dataset. However, when comparing the original implementations (denoted by the subscript

o) and the modified implementations (denoted by the subscript m) by replacing the optimization function, we can see the performance decreases to some extent. In particular, for BCE-based algorithms, their modifications with the BPR functions have led to significant performance drop. These findings show that when using sampling-based algorithms as baselines, we should follow the original optimization functions as possible. Although the two sampling-based optimization functions are similar in formulation, they have different implications in essence. We will study the difference between the two loss functions in future work.

6.2 Hyper-parameter Search

Besides simple statistics-based algorithms, most of the recommendation algorithms have several parameters to tune. Here we discriminate parameters that need to be learned through optimization algorithms from those that are needed to be set before training the algorithms. We refer the latter as *hyper-parameters*. In particular, neural-based algorithms typically need to set more hyper-parameters, including the number of layers, embedding size, and dropout rate. Recently, a number of studies have complained that the comparison baselines are not equipped with well-tuned parameters [23] in published research papers, leading to unfair comparison. In this part, we conduct the study on optimal parameter search for recommendation algorithms.

6.2.1 Search Methodology. In principle, optimal hyper-parameters should be obtained through the tuning on validation sets. However, we do find that a number of research papers do not use validation sets. In our paper collection, for those with validation sets, 36% papers directly provide the hyper-parameters in their experiments without further search (e.g., reuse the originally reported values), and 30% of papers offer no information about hyper-parameters in their experiment sections.

By reviewing existing studies, *grid search* is the most widely used parameter search approach, in which we first set the search range and then perform an exhaustive procedure to evaluate all the possible parameter settings. Grid search is easy to be implemented, however, it is very time-consuming when the number of hyper-parameters is large. To alleviate this problem, several parameter optimization strategies have been proposed, such as random search [8], Bayesian HyperOpt [102], and asynchronous model-based search [4], which try to seek a balance between efficiency and effectiveness. However, these methods mainly focus on general parameter search, which might be too complicated to be used in recommender systems.

Inspired by sequential parameter optimization [6], we consider a sequential search procedure [130] (called *sequential search*) that gradually optimizes the parameters one by one. In this method, we tune one parameter at each time while fixing the rest parameters, in a sequential manner. Sequential search dramatically reduces the time complexity to enumerate possible parameter combinations. Besides, we further design a simple variant for sequential search by incorporating the early-stop mechanism [88], called *sequential search with early stop*. In this way, we can stop searching for some parameters when the performance gap between two examined values is small, which is useful to reduce tuning time.

For parameter tuning, given a hyper-parameter, we first set the search set by selecting several representative values. A larger search set indicates a more number of values to be searched. We arrange the parameters requiring more search values at a later order, and it is more robust when sequentially searching a number of parameters.[4] Thus, we sequentially search the parameters according to the ascending order of their searching range.

Next, we construct a detailed study of parameter search on recommender systems.

[4]Intuitively, the parameters with fewer search values are relatively easier to set than those with more search values.

6.2.2 Research Question and Experimental Setup. In recent years, there are increasing concerns on the reproducible results of recommendation algorithms [2, 23–25], where proper hyper-parameter setting is one of the most important factors for the algorithm reproducibility. We consider studying the following two questions:

- Are there any shortcut rules as empirical experiences to set these hyper-parameters?
- Is there an approach to balancing the effectiveness and efficiency in hyper-parameter search?

For different recommendation algorithms, the involved parameters are quite different. Here we mainly consider studying four major hyper-parameters in neural recommendation algorithms, including the learning rate, embedding size, hidden layer size, and regularization weight. These four kinds of hyper-parameters are commonly used in various recommendation algorithms. For the other parameters to be tuned, we still adopt the grid search approach to optimize them but omit their tuning results in this article. To perform the experiments, the dataset is preprocessed by five-core filtering and split with the TO_RS option (Section 5.2), where 80% of data is used for training and the remaining 20% of data is equally divided for validating and testing. Due to a space limit, we only report the optimal parameters on ML-1M datasets, and the results of more datasets can be found in our GitHub page for this work.

6.2.3 Observations and Discussion. We first report the search set and optimal hyper-parameters found by the *grid search* in Table 9. As a comparison, we present the hyper-parameters that are found by sequential search in Table 10. Here the optimal performance obtained from grid search is considered as a performance reference for the two sequential search methods. In Table 9, the following observations can be found as follows.

(1) The optimal value for *embedding size* seems to be varying for different search methods. Eight of 10 comparison algorithms achieve their best when *embedding size* is equal or greater than 128. It indicates that these recommendation algorithms might need a large embedding size to achieve the optimal performance in experimental comparison. However, in existing recommender systems, they typically set the embedding size to 64 or 128 in baseline algorithms, such as the most widely adopted baseline BPR. These observations show that we need to consider a large search range for the embedding size in parameter tuning. However, the setting of embedding size depends on the actual hardware support, since some algorithms might take a significant amount of space to run with a large embedding size. In this case, we recommend that the authors should indicate that the parameters might not be optimal for performance comparison.

(2) Compared with embedding size, *learning rate* seems to be more stable for different algorithms, where the optimal values are around the scale between 1e-4 and 1e-5 for most of sampling-based algorithms. A majority of algorithms (except FISM and non-sampling algorithms) obtain their best performance when *learning rate* falls in $[1e-5, 1e-4]$. While the *learning rate* of those non-sampling algorithms is usually larger and falls in $[1e-3, 1e-2]$. A possible reason is that sampling and non-sampling algorithms use different optimization ways. For sampling-based algorithms, they usually sample a batch of interactions for updating the parameters iteratively; for non-sampling algorithms, they consider the entire item set in one update, so the learning rate can be set to a larger value for accelerating the optimization. These findings can be used as empirical guidelines for parameter tuning.

(3) Next, we continue to analyze the results of sequential search in Table 11. Here we take the results of grid search as a reference and report the relative change in both accuracy and efficiency. Although the optimal parameters obtained from sequential search are somehow different from

Table 9. Optimal Hyper-parameters Found by Grid Search on ML-1M Dataset and the Corresponding Search Set

Algorithm	Optimal hyper-parameters	Search set
BPR	embedding_size=2048 learning_rate=1e-4	embedding_size in [16,32,64,128,256,512, 1024,2048,4096] learning_rate in [5e-5,1e-4,5e-4,1e-3, 5e-3,1e-2,5e-2]
SVD++	embedding_size=128 learning_rate=5e-4 reg_weight=5e-4	embedding_size in [64,128,256,512] learning_rate in [1e-4,5e-4,1e-3,5e-3,1e-2] reg_weight in [0,5e-5,5e-4,5e-3,5e-2]
NCF	dropout_prob=0.3 embedding_size=64 learning_rate=5e-4 mlp_hidden_size=[256,256,256]	dropout_prob in [0.1,0.2,0.3,0.5] embedding_size in [32,64,128,256,512] learning_rate in [5e-5,1e-4,5e-4,1e-3] mlp_hidden_size in [[256,128,256],[128,128,128], [256,256,256],[64,64,64]]
NAIS	embedding_size=32 learning_rate=5e-5 reg_weights=0	embedding_size in [16,32,64,128] learning_rate in [1e-5,5e-5,1e-4,5e-4,1e-3] reg_weights in [1e-7,1e-3,0,10]
FISM	embedding_size=128 learning_rate=1e-3 reg_weights=1e-3	embedding_size in [64,128,256] learning_rate in [5e-5,1e-4,5e-4,1e-3] reg_weights in [1e-7,0,1e-1,1e-3,1e-5]
NGCF	embedding_size=2048 hidden_size=256 learning_rate=5e-4 reg_weight=1e-6	embedding_size in [512,1024,2048] hidden_size in [256,512,1024] learning_rate in [1e-4,5e-4,1e-3,5e-3] reg_weight in [0,1e-6,1e-4,1e-2]
LightGCN	embedding_size=1024 learning_rate=5e-4 reg_weight=1e-2	embedding_size in [256,512,1024,2048] learning_rate in [1e-4,5e-4,1e-3,5e-3,1e-2] reg_weight in [1e-6,1e-5,1e-4,1e-2,0]
ENMF	dropout_prob=0.3 embedding_size=256 learning_rate=1e-3 negative_weight=0.5	dropout_prob in [0.5,0.3,0.1] embedding_size in [64,256,512] learning_rate in [5e-5,1e-4,5e-4,1e-3,5e-3,1e-2] negative_weight in [5e-3,5e-2,0.5]
CDAE	embedding_size=4096 learning_rate=5e-3	embedding_size in [16,32,64,128,256,512, 1024,2048,4096] learning_rate in [5e-5,1e-4,5e-4,1e-3,5e-3, 1e-2,5e-2]
MultiVAE	latent_dimension=128 learning_rate=1e-2 mlp_hidden_size=600	latent_dimension in [128,512,1024,2048] learning_rate in [5e-5,1e-4,5e-4,1e-3,5e-3,1e-2] mlp_hidden_size in [100,300,600,1200,2400]

Here, hyper-parameters are denoted with the options in RecBole.

those obtained from grid search (shown in Table 10), they are able to retain comparable accuracy, even a slight improvement in some cases (except the case of FISM using sequential search with early stop). Furthermore, for efficiency consideration, approximate sequential search is able to largely reduce the expected time costs. Besides, when comparing the two sequential search variants, we find that the incorporation of early stop leads to unstable change on accuracy (the case of FISM in Table 11). Overall, these observations indicate that sequential search for parameter tuning can be applied when it is time-consuming to compare many recommendation algorithms, which can yield comparable accuracy with grid search.

Table 10. Optimal Hyper-parameters Found by Sequential Search in the Search Set

Algorithms	Sequential search	Sequential search with early stopping
BPR	embedding_size=4096 learning_rate=5e-5	embedding_size=4096 learning_rate=5e-5
SVD++	embedding_size=64 learning_rate=1e-3	embedding_size=64 learning_rate=1e-3
NCF	dropout_prob=0.2 mlp=[256,128,256] embedding_size=32	dropout_prob=0.2 mlp=[256,128,256] embedding_size=32
NAIS	reg_weights=1e-7 embedding_size=16 learning_rate=5e-4	reg_weights=1e-7 embedding_size=16 learning_rate=5e-4
FISM	embedding_size=64 learning_rate=5e-4	reg_weights=[0.1,0.1] learning_rate=5e-3
NGCF	hidden_size=1024 embedding_size=512	hidden_size=1024 embedding_size=512
LightGCN	reg_weight=1e-6	reg_weight=1e-6
ENMF	dropout_prob=0.5 learning_rate=5e-4	dropout_prob=0.5 learning_rate=5e-4
CDAE	None	embedding_size=64 learning_rate=5e-2
MultiVAE	mlp_hidden_size=[300]	mlp_hidden_size=1200 learning_rate=1e-3

Hyper-parameters are denoted with the options in RecBole.

Table 11. Effectiveness and Efficiency Comparison between Sequential Search and Grid Search

Algorithm	Sequential search		Sequential search with early stopping	
	Accuracy	Efficiency	Accuracy	Efficiency
BPR	+0.25%	+74.61%	+0.2%	82.54%
SVD++	+0.26%	+86%	+0.26%	+90%
NCF	−0.83%	+95%	−0.83%	97.25%
NAIS	−1.87%	+80%	−1.87%	+85%
FISM	+1.22%	+43.66%	−23.24%	+62.5%
NGCF	+5.87%	+88.89%	+5.87%	+91.67%
LightGCN	−0.13%	+52%	−0.13%	+72%
ENMF	−2.12%	+86.12%	−2.12%	+88.89%
CDAE	0%	+68.26%	+2.37%	+76.2%
MultiVAE	−2.97%	+83.34%	−2.12%	+86.84%

Here, we take the optimal accuracy on Recall@10 and training time with grid search as the reference, and report the relative change ratio.

7 CONCLUSION

In this work, we conducted a large-scale, systematic study on six important factors from three aspects for evaluating recommender systems. We extensively collect the research papers on top-N recommendation and analyze and summarize the most divergent settings in different

factors. Our experiments are thoroughly designed and extensively performed. In particular, all the experimental results in this article can be reproduced with an open sourced recommendation library Recbole [139]. To conclude this article and highlight our findings, we summarize them as follows:

- *Evaluation metrics*. We find that evaluation metrics can form into several coherent groups: the metrics from the same group have very strong correlations, and inter-group metrics are not consistent in ranking measurement. In particular, beyond-accuracy metrics are very different from accuracy metrics. It is suggested the researchers should select a number of diverse metrics for evaluation, at least covering different groups of metrics. For sampled metrics, it is likely to yield very different performance rankings. With more sampled negatives, the correlation degree becomes larger. Specifically, there are more variations in the overall ranking than the top ranked algorithms. We also test some recently proposed debiasing method for sampled metrics. However, we find that it may not effectively reduce the bias from sampled metrics. In practice, the researchers should avoid the use of sampled metrics. If it was used, then it is suggested to use a large number of negative items as possible.
- *Dataset construction*. We find that the influencing factors for selecting datasets are very diverse. In experiments, the researchers are recommended to select more diverse datasets covering different characteristics in scale, sparsity or domain. Furthermore, data filtering tends to produce different performance rankings. Even with the classic n-core filtering, the researchers should carefully examine the statistics and distribution of the interaction data, to obtain similar comparison results as the original datasets. Given the selected datasets, we compare four combinations to construct the training, validation and test sets. We find that performance ranking is more sensitive to the way of data ordering than the splitting strategy. We also find that all the four data construction methods lead to data leakage in global timelines, which should be taken special considerations for avoiding this issue.
- *Model optimization*. We analyze two kinds of optimization functions widely adopted in existing recommendation algorithms, namely sampling-based and non-sampling methods. Overall, the non-sapling method has intrinsic advantage when the item set is large. We further study the BPR loss and BCE loss based on the sampling-based methods. It is suggested that the implemented baselines should follow the original optimization functions. Otherwise, the performance might be sub-optimal or not comparable to the originally reported result. Furthermore, we extensively discuss the hyper-parameter search. We list several empirical guidelines to reduce the search range for hyper-parameters, and also present a sequential search procedure that can effectively reduce the search cost.

As future work, we will consider conducting more studies on beyond-accuracy metrics, which have different effects on the performance rankings compared with accuracy metrics. Besides, considering that full-ranking evaluation is very time-consuming, we will consider designing better sampled metrics, so that the correlation with the full-ranking performance can be largely improved. Another interesting direction is to study how these findings generalize to online evaluation. Currently, there is not a thorough discussion about the connection and difference between online and offline evaluation. We will investigate into this topic in the future.

Furthermore, to conduct fair, reproducible evaluation, we believe a fundamental solution in practice is to develop a standardized, open sourced evaluation platform (e.g., RecBole) with all the evaluation setup steps. We will consider continually improving the RecBole library for including more commonly used evaluation options or settings.

ACKNOWLEDGMENTS

The authors gratefully appreciate the anonymous reviewers for their valuable and detailed comments that greatly helped to improve the quality of this article.

REFERENCES

[1] Fabio Aiolli. 2013. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys'13)*. Association for Computing Machinery, New York, NY, 273–280. https://doi.org/10.1145/2507157.2507189

[2] Zafar Ali, Irfan Ullah, Amin Khan, Asim Ullah Jan, and Khan Muhammad. 2021. An overview and evaluation of citation recommendation models. *Scientometrics* 126, 5 (2021), 4083–4119.

[3] Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. Elliot: A comprehensive and rigorous framework for reproducible recommender systems evaluation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'21)*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 2405–2414. https://doi.org/10.1145/3404835.3463245

[4] Prasanna Balaprakash, Michael Salim, Thomas D. Uram, Venkat Vishwanath, and Stefan M. Wild. 2018. DeepHyper: Asynchronous hyperparameter search for deep neural networks. In *Proceedings of the IEEE 25th International Conference on High Performance Computing (HiPC'18)*. 42–51. https://doi.org/10.1109/HiPC.2018.00014

[5] Oren Barkan, Yonatan Fuchs, Avi Caciularu, and Noam Koenigstein. 2020. Explainable recommendations via attentive multi-persona collaborative filtering. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys'20)*. Association for Computing Machinery, New York, NY, 468–473. https://doi.org/10.1145/3383313.3412226

[6] T. Bartz-Beielstein, C. W. G. Lasarczyk, and M. Preuss. 2005. Sequential parameter optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Vol. 1. 773–780. https://doi.org/10.1109/CEC.2005.1554761

[7] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of the KDD Cup and Workshop*, Vol. 2007. Citeseer, 35.

[8] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, null (Feb. 2012), 281–305.

[9] Rocío Cañamares and Pablo Castells. 2020. On target item sampling in offline recommender system evaluation. In *Proceedingsof the 14th ACM Conference on Recommender Systems (RecSys'20)*. Association for Computing Machinery, New York, NY, 259–268. https://doi.org/10.1145/3383313.3412259

[10] Pedro G. Campos, Fernando Díez, and Manuel Sánchez-Montañés. 2011. Towards a more realistic evaluation: Testing the ability to predict future tastes of matrix factorization-based recommenders. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys'11)*. Association for Computing Machinery, New York, NY, 309–312. https://doi.org/10.1145/2043932.2043990

[11] Rocío Cañamares, Pablo Castells, and Alistair Moffat. 2020. Offline evaluation options for recommender systems. *Inf. Retriev. J.* (2020), 1–24.

[12] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2011. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys'11)*. ACM, New York, NY.

[13] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. 2001. Rectree: An efficient collaborative filtering method. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*. Springer, 141–151.

[14] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2019. Social attentional memory network: Modeling aspect- and friend-level differences in recommendation. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining (WSDM'19)*. Association for Computing Machinery, New York, NY, 177–185. https://doi.org/10.1145/3289600.3290982

[15] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient neural matrix factorization without sampling for recommendation. *ACM Trans. Inf. Syst.* 38, 2, Article 14 (Jan. 2020), 28 pages. https://doi.org/10.1145/3373807

[16] Chong Chen, Min Zhang, Yongfeng Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. 2020. Efficient heterogeneous collaborative filtering without negative sampling for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 19–26.

[17] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and debias in recommender system: A survey and future directions. arXiv:2010.03240. Retrieved from https://arxiv.org/abs/2010.03240.

[18] Weijian Chen, Yulong Gu, Zhaochun Ren, Xiangnan He, Hongtao Xie, Tong Guo, Dawei Yin, and Yongdong Zhang. 2019. Semi-supervised user profiling with heterogeneous graph attention networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'19)*, Vol. 19. 2116–2122.

[19] Yihong Chen, Bei Chen, Xiangnan He, Chen Gao, Yong Li, Jian-Guang Lou, and Yue Wang. 2019. λ Opt: Learn to regularize recommender models in finer levels. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*. Association for Computing Machinery, New York, NY, 978–986. https://doi.org/10.1145/3292500.3330880

[20] Yifan Chen and Maarten de Rijke. 2018. A collective variational autoencoder for top-n recommendation with side information. In *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*. 3–9.

[21] Jin Yao Chin, Yile Chen, and Gao Cong. 2022. The datasets dilemma: How much do we really know about recommendation datasets? In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*. 141–149.

[22] Evangelia Christakopoulou and George Karypis. 2018. Local latent space models for top-n recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. Association for Computing Machinery, New York, NY, 1235–1243. https://doi.org/10.1145/3219819.3220112

[23] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. 2021. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Trans. Inf. Syst.* 39, 2, Article 20 (January 2021), 49 pages. https://doi.org/10.1145/3434185

[24] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys'19)*. Association for Computing Machinery, New York, NY, 101–109. https://doi.org/10.1145/3298689.3347058

[25] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2020. Methodological issues in recommender systems research. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*. 4706–4710.

[26] Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Felice Antonio Merra. 2020. How dataset characteristics affect the robustness of collaborative recommendation models. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 951–960.

[27] Zhi-Hong Deng, Ling Huang, Chang-Dong Wang, Jian-Huang Lai, and Philip S. Yu. 2019. DeepCF: A unified framework of representation learning and matching function learning in recommender system. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 61–68. https://doi.org/10.1609/aaai.v33i01.330161

[28] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.* 22, 1 (2004), 143–177.

[29] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. 2015. Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 189–198.

[30] Jingtao Ding, Yuhan Quan, Quanming Yao, Yong Li, and Depeng Jin. 2020. Simplify and robustify negative sampling for implicit collaborative filtering. arXiv:2009.03376. Retrieved from https://arxiv.org/abs/2009.03376.

[31] Jingtao Ding, Guanghui Yu, Yong Li, Xiangnan He, and Depeng Jin. 2020. Improving implicit recommender systems with auxiliary data. *ACM Trans. Inf. Syst.* 38, 1, Article 11 (Feb. 2020), 27 pages. https://doi.org/10.1145/3372338

[32] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'18)*. Association for Computing Machinery, New York, NY, 515–524. https://doi.org/10.1145/3209978.3209991

[33] Ehtsham Elahi, Wei Wang, Dave Ray, Aish Fenton, and Tony Jebara. 2019. Variational low rank multinomials for collaborative filtering with side-information. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys'19)*. Association for Computing Machinery, New York, NY, 340–347. https://doi.org/10.1145/3298689.3347036

[34] Ehtsham Elahi, Wei Wang, Dave Ray, Aish Fenton, and Tony Jebara. 2019. Variational low rank multinomials for collaborative filtering with side-information. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 340–347.

[35] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 278–288. https://doi.org/10.1145/2736277.2741667

[36] Soude Fazeli, Babak Loni, Alejandro Bellogin, Hendrik Drachsler, and Peter Sloep. 2014. Implicit vs. explicit trust in social matrix factorization. In *Proceedings of the 8th ACM Conference on Recommender systems*. 317–320.

[37] Evgeny Frolov and Ivan Oseledets. 2019. HybridSVD: When collaborative information is not enough. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys'19)*. Association for Computing Machinery, New York, NY, 331–339. https://doi.org/10.1145/3298689.3347055

[38] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A free recommender system library. In *Proceedings of the 5th ACM Conference on Recommender Systems*. 305–308.

[39] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline a/b testing for recommender systems. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 198–206.

[40] Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. 2015. LibRec: A java library for recommender systems. In *UMAP Workshops*, Vol. 4. Citeseer.

[41] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, 982–995.

[42] F. Maxwell Harper and Joseph A. Konstan. 2015. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2015), 1–19.

[43] Gaole He, Junyi Li, Wayne Xin Zhao, Peiju Liu, and Ji-Rong Wen. 2020. Mining implicit entity preference from user-item interaction data for knowledge graph completion via adversarial learning. In *Proceedings of the Web Conference 2020*. 740–751.

[44] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 639–648.

[45] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'18)*. Association for Computing Machinery, New York, NY, 355–364. https://doi.org/10.1145/3209978.3209981

[46] X. He, Z. He, J. Song, Z. Liu, Y. Jiang, and T. Chua. 2018. NAIS: Neural attentive item similarity model for recommendation. *IEEE Trans. Knowl. Data Eng.* 30, 12 (2018), 2354–2366. https://doi.org/10.1109/TKDE.2018.2831682

[47] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the World Wide Web Conference (WWW'17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 173–182. https://doi.org/10.1145/3038912.3052569

[48] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 549–558.

[49] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback *(SIGIR'16)*. Association for Computing Machinery, New York, NY, 549–558. https://doi.org/10.1145/2911451.2911489

[50] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*. Association for Computing Machinery, New York, NY, 843–852. https://doi.org/10.1145/3269206.3271761

[51] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. arXiv:1511.06939. Retrieved from https://arxiv.org/abs/1511.06939.

[52] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S. Yu. 2018. Leveraging meta-path based context for top- n recommendation with a neural co-attention model. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. Association for Computing Machinery, New York, NY, 1531–1540. https://doi.org/10.1145/3219819.3219965

[53] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining*. IEEE, 263–272.

[54] Olivier Jeunen. 2019. Revisiting offline evaluation for implicit-feedback recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys'19)*. Association for Computing Machinery, New York, NY, 596–600. https://doi.org/10.1145/3298689.3347069

[55] Olivier Jeunen, Koen Verstrepen, and Bart Goethals. 2018. Fair offline evaluation methodologies for implicit-feedback recommender systems with MNAR data. In *Proceedings of the REVEAL 18 Workshop on Offine Evaluation, October 2018*, Vancouver, Canada.

[56] Shuyi Ji, Yifan Feng, Rongrong Ji, Xibin Zhao, Wanwan Tang, and Yue Gao. 2020. Dual channel hypergraph collaborative filtering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20)*. Association for Computing Machinery, New York, NY, 2020–2029. https://doi.org/10.1145/3394486.3403253

[57] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2020. A re-visit of the popularity baseline in recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1749–1752.

[58] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2021. A critical study on data leakage in recommender system offline evaluation. arXiv:2010.11060 [cs.IR]. Retrieved from https://arxiv.org/abs/2010.11060.

[59] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 43–50.
[60] Santosh Kabbur, Xia Ning, and George Karypis. 2013. FISM: Factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. Association for Computing Machinery, New York, NY, 659–667. https://doi.org/10.1145/2487575.2487589
[61] Ron Kohavi and Roger Longbotham. 2017. *Online Controlled Experiments and A/B Testing*. 922–929. https://doi.org/10.1007/978-1-4899-7687-1_891
[62] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. Association for Computing Machinery, New York, NY, 426–434. https://doi.org/10.1145/1401890.1401944
[63] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[64] Walid Krichene and Steffen Rendle. 2020. On sampled metrics for item recommendation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.
[65] Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations. arXiv:1507.08439. Retrieved from https://arxiv.org/abs/1507.08439.
[66] Volodymyr Kysenko, Karl Rupp, Oleksandr Marchenko, Siegfried Selberherr, and Anatoly Anisimov. 2012. GPU-accelerated non-negative matrix factorization for text mining. In *Proceedings of the International Conference on Application of Natural Language to Information Systems*. Springer, 158–163.
[67] Dung D. Le and Hady W. Lauw. 2017. Indexable bayesian personalized ranking for efficient top-k recommendation. In *Proceedings of the ACM on Conference on Information and Knowledge Management (CIKM'17)*. Association for Computing Machinery, New York, NY, 1389–1398. https://doi.org/10.1145/3132847.3132913
[68] DongSheng Li, Chao Chen, Qin Lv, Li Shang, Stephen Chu, and Hongyuan Zha. 2017. ERMMA: Expected risk minimization for matrix approximation-based recommender systems. In *Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence*.
[69] Dong Li, Ruoming Jin, Jing Gao, and Zhi Liu. 2020. On sampling top-k recommendation evaluation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20)*. Association for Computing Machinery, New York, NY, 2114–2124. https://doi.org/10.1145/3394486.3403262
[70] Xiaopeng Li and James She. 2017. Collaborative variational autoencoder for recommender systems. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, 305–314. https://doi.org/10.1145/3097983.3098077
[71] Dawen Liang, Laurent Charlin, James McInerney, and David M. Blei. 2016. Modeling user exposure in recommendation. In *Proceedings of the 25th International Conference on World Wide Web*. 951–961.
[72] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the World Wide Web Conference (WWW'18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 689–698. https://doi.org/10.1145/3178876.3186150
[73] Chen Ma, Peng Kang, Bin Wu, Qinglong Wang, and Xue Liu. 2019. Gated attentive-autoencoder for content-aware recommendation. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining (WSDM'19)*. Association for Computing Machinery, New York, NY, 519–527. https://doi.org/10.1145/3289600.3290977
[74] Jingwei Ma, Jiahui Wen, Mingyang Zhong, Liangchen Liu, Chaojie Li, Weitong Chen, Yin Yang, Hongkui Tu, and Xue Li. 2019. DBRec: Dual-bridging recommendation via discovering latent groups. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM'19)*. Association for Computing Machinery, New York, NY, 1513–1522. https://doi.org/10.1145/3357384.3357892
[75] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 43–52.
[76] Sean M. McNee, John Riedl, and Joseph A. Konstan. 2006. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*. 1097–1101.
[77] Lei Mei, Pengjie Ren, Zhumin Chen, Liqiang Nie, Jun Ma, and Jian-Yun Nie. 2018. An attentive interaction network for context-aware recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*. Association for Computing Machinery, New York, NY, 157–166. https://doi.org/10.1145/3269206.3271813
[78] Elisa Mena-Maldonado, Rocío Cañamares, Pablo Castells, Yongli Ren, and Mark Sanderson. 2020. Agreement and disagreement between true and false-positive metrics in recommender systems evaluation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'20)*. Association for Computing Machinery, New York, NY, 841–850. https://doi.org/10.1145/3397271.3401096

[79] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring data splitting strategies for the evaluation of recommendation models. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys'20)*. Association for Computing Machinery, New York, NY, 681–686. https://doi.org/10.1145/3383313.3418479

[80] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring data splitting strategies for the evaluation of recommendation models. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys'20)*. Association for Computing Machinery, New York, NY, 681–686. https://doi.org/10.1145/3383313.3418479

[81] Athanasios N. Nikolakopoulos, Dimitris Berberidis, George Karypis, and Georgios B. Giannakis. 2019. Personalized diffusions for top-n recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys'19)*. Association for Computing Machinery, New York, NY, 260–268. https://doi.org/10.1145/3298689.3346985

[82] Athanasios N. Nikolakopoulos and George Karypis. 2019. RecWalk: Nearly uncoupled random walks for top-n recommendation. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining (WSDM'19)*. Association for Computing Machinery, New York, NY, 150–158. https://doi.org/10.1145/3289600.3291016

[83] Rasaq Otunba, Raimi A. Rufai, and Jessica Lin. 2017. MPR: Multi-objective pairwise ranking. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys'17)*. Association for Computing Machinery, New York, NY, 170–178. https://doi.org/10.1145/3109859.3109903

[84] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. 2008. One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on Data Mining*. 502–511. https://doi.org/10.1109/ICDM.2008.16

[85] Bibek Paudel, Thilo Haas, and Abraham Bernstein. 2017. Fewer flops at the top: Accuracy, diversity, and regularization in two-class collaborative filtering. In *Proceedings of the 11th ACM Conference on Recommender Systems (RecSys'17)*. Association for Computing Machinery, New York, NY, 215–223. https://doi.org/10.1145/3109859.3109916

[86] Changhua Pei, Xinru Yang, Qing Cui, Xiao Lin, Fei Sun, Peng Jiang, Wenwu Ou, and Yongfeng Zhang. 2019. Value-aware recommendation based on reinforced profit maximization in e-commerce systems. arXiv:1902.00851. Retrieved from https://arxiv.org/abs/1902.00851.

[87] Nikolaos Polatidis, Stelios Kapetanakis, Elias Pimenidis, and Konstantinos Kosmidis. 2018. Reproducibility of experiments in recommender systems evaluation. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 401–409.

[88] Lutz Prechelt. 1998. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Netw.* 11, 4 (1998), 761–767.

[89] Steffen Rendle. 2010. Factorization machines. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 995–1000.

[90] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*. 452–461.

[91] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 240–248.

[92] Steffen Rendle, Li Zhang, and Yehuda Koren. 2019. On the difficulty of evaluating baselines: A study on recommender systems. arXiv:1905.01395. Retrieved from https://arxiv.org/abs/1905.01395.

[93] Jasson D. M. Rennie and Nathan Srebro. 2005. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine Learning*. 713–719.

[94] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender Systems Handbook*. Springer, 1–35.

[95] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2010. *Recommender Systems Handbook* (1st ed.). Springer-Verlag, Berlin.

[96] Marco Rossetti, Fabio Stella, and Markus Zanker. 2016. Contrasting offline and online results when evaluating recommendation algorithms. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys'16)*. Association for Computing Machinery, New York, NY, 31–34. https://doi.org/10.1145/2959100.2959176

[97] Alan Said and Alejandro Bellogín. 2014. Comparative recommender system evaluation: Benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems*. 129–136.

[98] Alan Said and Alejandro Bellogín. 2015. Replicable evaluation of recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys'15)*. Association for Computing Machinery, New York, NY, 363–364. https://doi.org/10.1145/2792838.2792841

[99] Guy Shani and Asela Gunawardana. 2011. Evaluating recommendation systems. In *Recommender Systems Handbook*. Springer, 257–297.

[100] Lalita Sharma and Anju Gera. 2013. A survey of recommendation system: Research challenges. *Int. J. Eng. Trends Technol.* 4, 5 (2013), 1989–1992.

[101] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. 2019. How good your recommender system is? A survey on evaluations in recommendation. *Int. J. Mach. Learn. Cybernet.* 10, 5 (2019), 813–831.

[102] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc., 2951–2959.

[103] Harald Steck. 2013. Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*. 213–220.

[104] Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, and Mark Coates. 2020. A framework for recommending accurate and diverse items using bayesian graph convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20)*. Association for Computing Machinery, New York, NY, 2030–2039. https://doi.org/10.1145/3394486.3403254

[105] Rui Sun, Xuezhi Cao, Yan Zhao, Junchen Wan, Kun Zhou, Fuzheng Zhang, Zhongyuan Wang, and Kai Zheng. 2020. Multi-modal knowledge graphs for recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM'20)*. Association for Computing Machinery, New York, NY, 1405–1414. https://doi.org/10.1145/3340531.3411947

[106] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Yu Chen, and Chi Xu. 2017. MRLR: Multi-level representation learning for personalized ranking in recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17)*. 2807–2813.

[107] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. 2020. Are we evaluating rigorously? Benchmarking recommendation for reproducible evaluation and fair comparison. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 23–32.

[108] Thanh Tran, Xinyue Liu, Kyumin Lee, and Xiangnan Kong. 2019. Signed distance-based deep memory recommender. In *Proceedings of the World Wide Web Conference (WWW'19)*. Association for Computing Machinery, New York, NY, 1841–1852. https://doi.org/10.1145/3308558.3313460

[109] Daniel Valcarce, Alejandro Bellogín, Javier Parapar, and Pablo Castells. 2018. On the robustness and discriminative power of information retrieval metrics for top-n recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'18)*. Association for Computing Machinery, New York, NY, 260–268. https://doi.org/10.1145/3240323.3240347

[110] Saúl Vargas. 2014. Novelty and diversity enhancement and evaluation in recommender systems and information retrieval. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. 1281–1281.

[111] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. 2019. Multi-task feature learning for knowledge graph enhanced recommendation. In *Proceedings of the World Wide Web Conference (WWW'19)*. Association for Computing Machinery, New York, NY, 2000–2010. https://doi.org/10.1145/3308558.3313411

[112] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*. Association for Computing Machinery, New York, NY, 950–958. https://doi.org/10.1145/3292500.3330989

[113] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. Association for Computing Machinery, New York, NY, 165–174. https://doi.org/10.1145/3331184.3331267

[114] Xiang Wang, Yaokun Xu, Xiangnan He, Yixin Cao, Meng Wang, and Tat-Seng Chua. 2020. Reinforced negative sampling over knowledge graph for recommendation. In *Proceedings of the World Wide Web Conference 2020 (WWW'20)*. Association for Computing Machinery, New York, NY, 99–109. https://doi.org/10.1145/3366423.3380098

[115] Zengmao Wang, Yuhong Guo, and Bo Du. 2018. Matrix completion with preference ranking for top-n recommendation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. International Joint Conferences on Artificial Intelligence Organization, 3585–3591. https://doi.org/10.24963/ijcai.2018/498

[116] Bin Wu, Zhongchuan Sun, Xiangnan He, Xiang Wang, and Jonathan Staniforth. 2017. NeuRec. Retrieved from https://github.com/wubinzzu/NeuRec.

[117] Ga Wu, Maksims Volkovs, Chee Loong Soon, Scott Sanner, and Himanshu Rai. 2019. Noise contrastive estimation for one-class collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. Association for Computing Machinery, New York, NY, 135–144. https://doi.org/10.1145/3331184.3331201

[118] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM'16)*. Association for Computing Machinery, New York, NY, 153–162. https://doi.org/10.1145/2835776.2835837

[119] Xin Xin, Bo Chen, Xiangnan He, Dong Wang, Yue Ding, and Joemon Jose. 2019. CFM: Convolutional factorization machines for context-aware recommendation. In *Proceedings of the 28th International Joint Conference on Artificial*

Intelligence (IJCAI'19). International Joint Conferences on Artificial Intelligence Organization, 3926–3932. https://doi.org/10.24963/ijcai.2019/545

[120] Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. 2019. Relational collaborative filtering: Modeling multiple item relations for recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. Association for Computing Machinery, New York, NY, 125–134. https://doi.org/10.1145/3331184.3331188

[121] Fengli Xu, Jianxun Lian, Zhenyu Han, Yong Li, Yujian Xu, and Xing Xie. 2019. Relation-aware graph convolutional networks for agent-initiated social e-commerce recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM'19)*. Association for Computing Machinery, New York, NY, 529–538. https://doi.org/10.1145/3357384.3357924

[122] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. 2019. Deep item-based collaborative filtering for top-n recommendation. *ACM Trans. Inf. Syst.* 37, 3 (2019), 1–25.

[123] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep matrix factorization models for recommender systems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17)*, Vol. 17. Melbourne, Australia, 3203–3209.

[124] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys'18)*. Association for Computing Machinery, New York, NY, 279–287. https://doi.org/10.1145/3240323.3240355

[125] Yonghui Yang, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2021. Enhanced graph learning for collaborative filtering via mutual information maximization. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 71–80.

[126] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. 2016. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Proceedings of the Conference and Workshop on Neural Information Processing Systems (NIPS'16)*. 847–855.

[127] Junliang Yu, Min Gao, Jundong Li, Hongzhi Yin, and Huan Liu. 2018. Adaptive implicit friends identification over heterogeneous network for social recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 357–366.

[128] Lu Yu, Chuxu Zhang, Shichao Pei, Guolei Sun, and Xiangliang Zhang. 2018. WalkRanker: A unified pairwise ranking model with multiple relations for item recommendation.

[129] Wenhui Yu and Zheng Qin. 2020. Sampler design for implicit feedback data by noisy-label robust learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'20)*. Association for Computing Machinery, New York, NY, 861–870. https://doi.org/10.1145/3397271.3401155

[130] Wenhui Yu, Huidi Zhang, Xiangnan He, Xu Chen, Li Xiong, and Zheng Qin. 2018. Aesthetic-based clothing recommendation. In *Proceedings of the World Wide Web Conference (WWW'18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 649–658. https://doi.org/10.1145/3178876.3186146

[131] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.* 52, 1 (2019), 5:1–5:38.

[132] Shuai Zhang, Lina Yao, Lucas Vinh Tran, Aston Zhang, and Yi Tay. 2019. Quaternion collaborative filtering for recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. International Joint Conferences on Artificial Intelligence Organization, 4313–4319. https://doi.org/10.24963/ijcai.2019/599

[133] Shuai Zhang, Lina Yao, and Xiwei Xu. 2017. AutoSVD++ An efficient hybrid collaborative filtering model via contractive auto-encoders. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 957–960.

[134] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W. Bruce Croft. 2017. Joint representation learning for top-n recommendation with heterogeneous information sources. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM'17)*. Association for Computing Machinery, New York, NY, 1449–1458. https://doi.org/10.1145/3132847.3132892

[135] Yuan Zhang, Xiaoran Xu, Hanning Zhou, and Yan Zhang. 2020. Distilling structured knowledge into embeddings for explainable and accurate recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM'20)*. Association for Computing Machinery, New York, NY, 735–743. https://doi.org/10.1145/3336191.3371790

[136] Yan Zhang, Hongzhi Yin, Zi Huang, Xingzhong Du, Guowu Yang, and Defu Lian. 2018. Discrete deep learning for fast content-aware recommendation. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM'18)*. Association for Computing Machinery, New York, NY, 717–726. https://doi.org/10.1145/3159652.3159688

[137] Feipeng Zhao and Yuhong Guo. 2017. Learning discriminative recommendation systems with side information. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. 3469–3475. https://doi.org/10.24963/ijcai.2017/485

[138] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. 2020. Revisiting alternative experimental settings for evaluating top-n item recommendation algorithms. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2329–2332.

[139] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, et al. 2021. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4653–4664.

[140] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*. Association for Computing Machinery, New York, NY, 1079–1088. https://doi.org/10.1145/3219819.3219826

Received 14 May 2021; revised 10 March 2022; accepted 28 May 2022