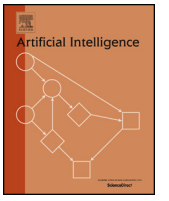
Artificial Intelligence 324 (2023) 103992

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Transfer learning for collaborative recommendation with biased and unbiased data ☆

Zinan Lin [a], Dugang Liu [a], Weike Pan [a,*], Qiang Yang [b], Zhong Ming [a]

[a] College of Computer Science and Software Engineering, Shenzhen University, China
[b] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, China

ARTICLE INFO

Article history:
Received 15 December 2021
Received in revised form 30 July 2023
Accepted 5 August 2023
Available online 23 August 2023

Keywords:
Transfer learning
Collaborative recommendation
Bias reduction
Unbiased data

ABSTRACT

In a recommender system, a user's interaction is often biased by the items' displaying positions and popularity, as well as the user's self-selection. Most existing recommendation models are built using such a biased user-system interaction data alone. In this paper, we introduce an additional specially collected unbiased data, and then have a new problem called collaborative recommendation with biased and unbiased data.

We first formalize the studied problem and list three challenges, including the bias challenge, the heterogeneity challenge and the unbalance challenge. Then we propose a novel transfer learning-based AI solution, i.e., transfer via joint reconstruction (TJR), to achieve knowledge transfer and sharing between the biased data and unbiased data. Specifically, in our TJR, we use two different models to extract the users' preferences and bias information, and then refine the prediction via the latent features containing the bias information in order to obtain a more accurate and unbiased recommendation. We further integrate the two data by reconstructing their interaction in a joint learning manner. Moreover, in order to better address the unbalance challenge, we introduce a bias regularization term and integrate bidirectional knowledge distillation. Finally, we adopt four representative methods, i.e., variational autoencoders, matrix factorization, neural collaborative filtering and graph convolution network, as the backbone models of our TJR and conduct extensive empirical studies on three public datasets, showcasing the effectiveness of our transfer learning solution over some very competitive baselines.

☆ This work is an extension of our previous work [1]. Compared with our previous work, we have added the following new contents in this paper. (i) We have developed three variants of our TJR, i.e., TJR++ in Section 5, TJR-PT in Section 6 and TJR-BiKD in Section 7, which denote extracting the bias information from a pre-trained model obtained using the biased data, adding a bias regularizer and integrating bidirectional knowledge distillation, respectively. (ii) We have described in detail the specialization of our TJR using four backbone models, i.e., variational autoencoders, matrix factorization, neural collaborative filtering and graph convolution network, in Sections 4.1, 4.2, 4.3 and 4.4, respectively. (iii) We have included more datasets, empirical studies and associated discussions in Section 8. (iv) We have included more related works in Section 2. And (v) we have made many improvements (e.g., figure illustration, notations, descriptions and discussions) throughout the whole paper.

* Corresponding author.
E-mail addresses: lzn87591@gmail.com (Z. Lin), dugang.ldg@gmail.com (D. Liu), panweike@szu.edu.cn (W. Pan), qyang@cse.ust.hk (Q. Yang), mingz@szu.edu.cn (Z. Ming).

https://doi.org/10.1016/j.artint.2023.103992
0004-3702/

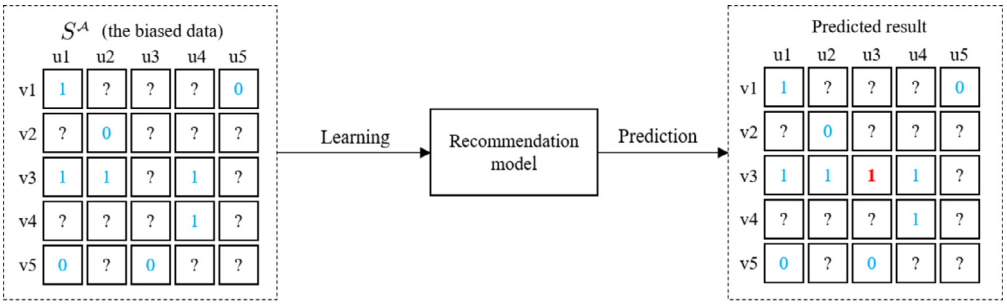
1. Introduction

For collaborative recommendation, we usually learn users' preferences from their historical interaction with the system-recommended items. There is an important issue in such a preferences learning setting, i.e., the bias in users' interaction or feedback [2,3]. For example, in a typical closed-loop recommender system, a user's feedback or interaction is often influenced by the position in which an item is displayed, the algorithm the system adopted, etc., which means that the collected data are biased [2–4]. Specifically, whether in the field of search, recommendation or advertising, related works state a phenomenon that the top ranked items often correspond to higher click-through rates, i.e., the displayed positions will affect the click-through rates of the items, which may not be related to the real interest of the user, but only to the attention of the user [5,6]. In other words, the premise is that an item is interacted with by a user if only it can be exposed to and observed by the user. However, due to a series of factors such as the large number of candidate items, the limitation of the displayed page, the requirement of engineering speed, and the need of business growth, the final exposure probability of each item does not only depend on the relevance between the user and the item. Currently, most collaborative recommendation models are built with the biased data only, which may not suit the users' tastes well. In other words, reducing bias plays an important role in improving the intelligence of a recommender system and its interactivity with the users. In this paper, we turn to leverage an additional unbiased data for users' preferences learning, which is thus called collaborative recommendation with biased and unbiased data. The difference between our studied problem and traditional collaborative recommendation is shown in Fig. 1. In industry, YouTube researchers reconstruct the model by considering the position bias, resulting in a 0.24% improvement on the engagement metric compared with the baseline model [5]. As another example, Huawei researchers mitigate the bias problem by introducing an unbiased data, and improve the AUC metric by 1.56% and 2.98% in their offline and online experiments, respectively [4].

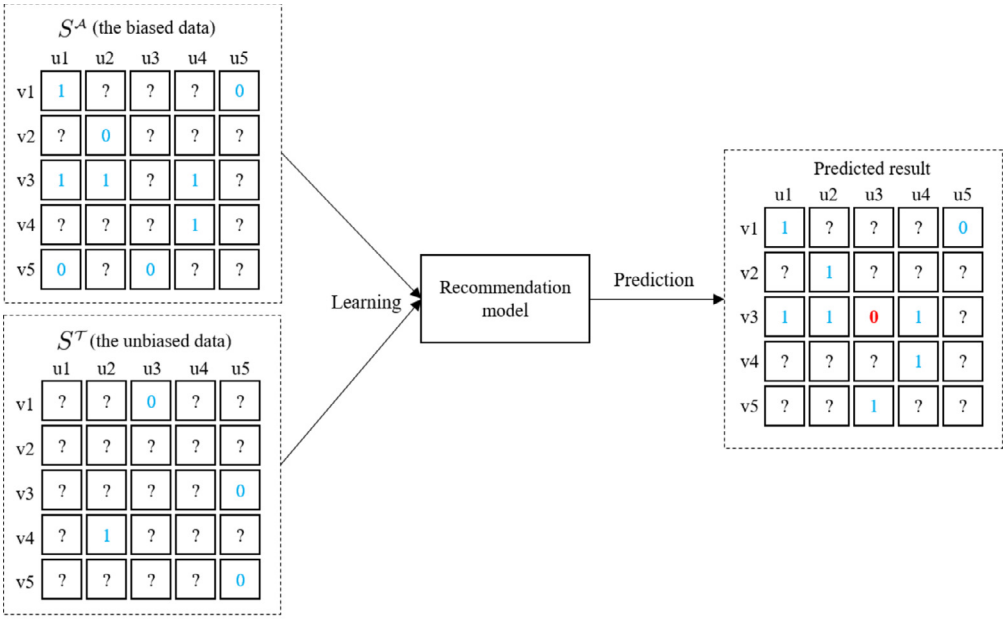
For the studied recommendation problem with both biased and unbiased data, previous works show that the unbiased data has the potential to mitigate the bias in users' feedback [7,4,8]. However, the unbiased data is usually rather small due to the high expense of collection in a deployed online system, which makes it difficult to help reduce the bias in the relatively larger biased data. Some researchers have made some pioneering efforts in this direction, including proposing a domain adaptation algorithm [9], developing a knowledge-distillation framework [4], and designing a meta learning algorithm [10], etc. However, these methods do not fully consider the difference between the biased and unbiased data in the generation process, which may not address the bias challenge in the biased data and the heterogeneity challenge of the two different data well.

Bias comes from the collected data, so when we make use of such data for model training, the bias may be reflected in different levels: (i) in the label level [4,11,12], including the interactions between users and items, as well as the prediction of the model; (ii) in the feature level [4,13,5], including the input features and the latent features; (iii) in the model's parameters level [4,14]; (iv) in the gradient level [10], including the size and the direction. Existing works solve the bias problem from the aforementioned perspectives, which can be considered as solving the problem from an explicit perspective. In addition, there are many works that deal with the bias problem from an implicit perspective, such as modifying the overall loss functions of the model, including weighting the loss functions and adding some auxiliary loss functions [2,15–17]. So far, there have been few research efforts that combine both the explicit and implicit perspectives to solve the problem. The explicit method depends on the rationality of the assumptions used, which may make the model have unstable performance in different training processes, while the implicit method may lead to some issues not being adequately addressed.

As a response, in this paper, we propose a novel transfer learning-based AI solution called transfer via joint reconstruction (TJR), which is believed to provide a more intelligent and more interactive recommender system. Our TJR can break the feedback loop of a recommender system, which help alleviate the bias in the data. In other words, our method does not distinguish one specific type of bias, and can address a variety of biases, including exposure bias, popularity bias, position bias and selection bias. Specifically, we first convert the modeling of the biased and unbiased data into a transfer learning problem, where the larger biased data is taken as the auxiliary data and the small unbiased data is taken as the target data. We extract the latent features that represent users' preferences and bias information from two different data. Our TJR can be divided into two upper and bottom branches, where the upper branch is used to extract the users' preferences and the bottom branch is used to extract the bias information. Intuitively, regardless of what the latent features represent, they ultimately affect the prediction of the model. Therefore, we refine the prediction by the latent features containing bias information to obtain a more accurate and unbiased prediction, which thus alleviates the bias problem. In order to effectively achieve knowledge transfer between the auxiliary data and the target data for bias reduction, we propose a joint reconstruction loss, aiming to guide the reconstructed output to fit both the target data and auxiliary data well. In this way, the unbiased target data implicitly plays a role of high-quality data that guides the learning task of the biased data, which addresses both the bias challenge and the heterogeneity challenge. In addition, we also propose three variants of our TJR, i.e., TJR-PT, TJR++ and TJR-BiKD, which denotes extracting the bias information from a pre-trained model obtained by the biased data, adding a bias regularizer and integrating bidirectional knowledge distillation, respectively. In our preliminary study, we have a simple motivation. From the perspective of machine learning, a recommendation model is able to learn the properties of users' feedback data. If there is bias in the data, the user features learned by the model will contain some bias information. At this point, these users' features undoubtedly give us a source of bias information, with which we design a variant called TJR-PT to isolate the bias information directly from the users' latent features. Since the unbiased data usually only covers fewer users, we introduce a bias constraint and propose an extension TJR++ to allow more users to benefit



(a) Traditional Collaborative Recommendation



(b) Collaborative Recommendation with Biased and Unbiased Data

Fig. 1. The schematic of traditional collaborative recommendation (a), and that of collaborative recommendation with biased and unbiased data (b). In general, the sparsity of the unbiased data is higher than that of the biased data, and using the unbiased data can alleviate the bias problem. For example, (a) shows that the item v3 is a popular item, and traditional recommendation models tend to predict the label of the pair (u3, v3) as 1 based on the attributes of the users and items. However, combining the unbiased data, the label of the pair (u3, v3) may be predicted as 0 considering the similarity between u5 and u3, and the fact that u5 is not interested in v3 as shown in (b).

from the bottom branch used for extracting the bias feature. Inspired by the literatures [4,18], we combine our TJR and the bidirectional knowledge distillation technology to mitigate the impact caused by the small size of the unbiased data.

Compared with some related works [19–21], we have the following differences. Firstly, the focuses are different. The above three works belong to the category of bias phenomenon analysis. In these works, the authors use a large space to study and analyze the bias phenomenon. However, we focus on providing a mitigation solution for the bias problem. Secondly, the biases are different. The bias mentioned in the first work and the third work is the bias of recommendation actions, and that in the second work is the selection bias of users. However, we study the data bias of a recommender system. Data bias is a type of bias that includes a wide range of biases, such as user selection bias, popular bias, and position bias. Finally, the techniques are different. The above three works mainly use auxiliary methods such as causal inference and data simulation to analyze the phenomenon of bias, and analyze how the bias and the recommendation system affect each other. In contrast, we design a new method to alleviate the bias, and set up fair experiments to demonstrate the superiority of our proposed method.

We summarize the main contributions of our work as follows: (i) we study an emerging recommendation problem called collaborative recommendation with a larger biased data and a small unbiased data; (ii) we convert the studied problem to a transfer learning (i.e., a subfield of AI) problem, and propose a novel and generic solution called transfer via joint reconstruction (TJR), and the modeling idea we propose has a certain degree of versatility, e.g., noise reduction; (iii) we propose a unique loss function, which is based on the consideration of the bias problem, and construct a loss function that shall be aligned with both the biased and unbiased data. This loss function contains a balanced consideration, which is quite different from other existing works on the studied problem; (iv) in order to further improve the performance, we propose

three variants of our TJR, including extracting the bias information from a pre-trained model obtained using the biased data, designing a bias regularizer and incorporating bidirectional knowledge distillation; and (v) we adopt four representative methods as the backbone models, i.e., variational autoencoders (VAE), matrix factorization (MF), neural collaborative filtering (NCF) and graph convolution network (GCN), and conduct extensive empirical studies on three public datasets, which shows that our TJR is very competitive comparing with the state-of-the-art methods.

The organization of our paper is as follows. In Section 2, we discuss about some related works, and then give some preliminaries of the studied problem in Section 3, including the problem definition, the challenges and our overall solution. We describe our solution in detail in Section 4, and give three improved variants of our solution in Sections 5, 6 and 7. In Section 8, we first introduce three real-world datasets used in the experiments and describe the implementation details, and then present the experimental results and conduct a detailed analysis. Finally, we give some concluding remarks and future directions in Section 9.

2. Related work

2.1. Collaborative recommendation

For a user u, a recommender system aims to find some most relevant or interested items from a candidate pool. The basic recommendation techniques can be divided into the content-based methods and collaborative filtering methods, where the former are implemented through the relevance between items, while the latter are usually achieved by learning a preferences score between a user u and each item i, i.e., $\hat{r}_{ui}$. In order to estimate $\hat{r}_{ui}$, we may use a neighborhood-based method [22–26], a factorization-based method [27–31], a deep network-based method [32–36], or a graph neural networks-based method [37–40].

A neighborhood-based method first calculates the similarities among the users (or items) and then predicts the preferences $\hat{r}_{ui}$ by aggregating the preferences of the neighbors of user u to item i (or the preferences of user u to the neighbors of item i). However, this method is relatively simple and does not model the complex relationships between users and items well. A factorization-based method often factorizes the user-item interaction matrix to learn the latent representations of the users and items, which is a linear model and cannot represent the non-linear relationship between users and items. A deep neural network-based method is able to capture the non-linear interaction between the latent features of users and items. This method can also be classified as an encoder-decoder-based method and a multilayer-perceptron based method, among which variational autoencoders (VAE) [33,35,41,42] is a very competitive recommendation method, using a distribution instead of a single vector to represent the latent factors of a user's preferences. Recently, there are also some graph neural network based methods [37,38,40], which use graph representation learning to learn the latent features of users and items by constructing a relational graph between users and items. Although this family of methods can learn the non-linear relationship between users and items, their computational complexity is relatively high.

2.2. Transfer learning in collaborative recommendation

Transfer learning refers to the transfer of knowledge from a source domain to a target domain, which is widely used in many fields such as recommender systems, computer vision, and natural language processing [43]. The approaches used for transfer learning can be classified into four categories [43], including instance-based methods [44,45], feature representation-based methods [46,47], parameter-based methods [48,49] and relational knowledge-based methods [50]. An instance-based method transfers knowledge from the perspective of samples such as weighting the samples from a source domain. A feature representation-based method refers to the knowledge transferred from the perspective of feature such as projecting the features of a source domain and that of a target domain into a common feature space. A parameter-based method completes knowledge transfer from the perspective of model parameters such as sharing a portion of the model parameters. A relational knowledge-based method transfers knowledge through matching the relationship between a source domain and a target domain.

In collaborative recommendation, there are several research topics suitable for transfer learning, e.g., collaborative recommendation with auxiliary data [51–53] and cross-domain recommendation [54–56]. Collaborative recommendation with auxiliary data aims to improve the recommendation performance and alleviate the sparsity problem by using some auxiliary data, including content information, social or information networks and additional feedback. The common transfer learning-based methods used for this problem can be divided into three categories [52], including adaptive methods [57,58], collective methods [59,60] and integrative methods [61,62]. Cross-domain recommendation is intended to alleviate the sparsity problem and cold-start problem by combining the feedback in different scenarios, including two heterogeneous applications such as music and movie, and two homogeneous ones like MovieLens and Netflix. The common transfer learning-based methods used for this problem can be classified into three categories [56], i.e., content-based methods [63,64], embedding-based methods [65,66] and rating pattern-based methods [67,68], etc.

2.3. Bias reduction in collaborative recommendation

In recent years, the bias problem in collaborative recommendation has gradually received more attention by the researchers and practitioners from the academia and industry. Existing works for bias reduction can be categorized into three

classes [4], including counterfactual learning-based methods [2,16,69], heuristic-based methods [70,71] and unbiased data augmentation methods [4,9,10]. One of the most popular counterfactual learning-based methods is inverse propensity score (IPS) [2,14], where each sample is assigned a weight via a theoretically unbiased prediction model. Ideally, if all the true propensity scores can be obtained, IPS can alleviate all types of biases, including popularity bias, position bias and social bias (e.g., gender, race, etc.) However, the true inverse propensity scores are not known and are usually evaluated by the methods such as calculating the popularity of items and learning via neural networks. However, these evaluation methods are susceptible to high variance [15,17,72]. Heuristic-based methods are relatively few, which often make certain assumptions about the data being missing not at random. In general, this class of methods can only alleviate a specific type of bias. However, the factors that cause bias are numerous and complex. It is thus usually difficult to solve the problem by using some heuristic-based methods. The data augmentation methods alleviate the bias problem by introducing an unbiased data collected by a specific random policy, which is recognized as a very promising approach. Compared with the above methods, the introduction of an unbiased data undoubtedly brings more valuable information, and naturally gives a clear data advantage to the recommendation model. In addition, this type of methods can alleviate multiple types of biases because most biases end up being reflected in the attributes of the users' feedback data. Collecting an unbiased data requires high expense in a deployed online system, making the size of the data small. Hence, how to make full use of the unbiased data becomes a highly important issue.

Some research works combine inverse propensity scores and the unbiased data to obtain more accurate propensity scores [73,74]. Similarly, this type of methods may not get the correct propensity scores. In [9], the authors propose a domain adaptation algorithm to learn two models from the biased data and the unbiased data, respectively, and then introduce a regularization term to constrain the parameters of both models. It is more difficult to align both models' parameters when they are of high dimensionality. In [4], the authors then propose a knowledge-distillation framework to distill some useful information from the unbiased data through four directions, i.e., label-based, feature-based, sample-based, and model structure-based. However, a single strategy does not guarantee the full utilization of the unbiased data. In [10], the authors propose a meta learning algorithm. It is an improvement of the doubly robust method, which applies the unbiased data to help the model learn better weights and imputation values from the gradient level. However, the computational complexity of this method is relatively high.

Our TJR aims to transfer knowledge between a biased data and an unbiased data for joint preferences modeling, and reduce the bias via a specifically designed prediction refinement component.

3. Collaborative recommendation with biased and unbiased data

3.1. Problem definition

In our studied problem, bias is a phenomenon that is common in recommender systems and affects the accuracy of recommendation results. Essentially, the bias problem points out that a user's feedback usually does not truly represent his or her preferences, which has been overlooked in most previous studies. Concentrating on the problem of collaborative recommendation with biased and unbiased data, we have a set of users and a set of items, denoted by $\mathcal{U} = \{u\} = \{1, 2, ..., n\}$ and $\mathcal{I} = \{i\} = \{1, 2, ..., m\}$, respectively. Moreover, we have two different sets of user-item interaction feedback, i.e., $S^{\mathcal{A}} = \{(u, i)\}$ and $S^{\mathcal{T}} = \{(u, i)\}$ obtained by two different policies. Specifically, $S^{\mathcal{A}}$ is a larger and biased data collected by a commonly deployed non-uniform policy in a typical online recommender system, and the non-uniform policy means using models and strategies to recommend items to users, i.e., each item is exposed to users with an unequal probability [2,75]. $S^{\mathcal{T}}$ is a small and unbiased data collected by a specifically designed uniform (i.e., random) policy. On the contrary, the uniform policy refers to expose each item to users with an equal opportunity [2,75]. Our goal is then to combine $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$ to obtain an improved and unbiased model, which alleviates the bias problem and provides a more accurate personalized recommendation list of items for each user. The experimental setup follows the previous related works [4,9,10].

We list the commonly used notations in the paper in Table 1.

3.2. Challenges

For the studied problem of collaborative recommendation with a larger biased data and a small unbiased data, we have to address the following three specific challenges. Firstly, the bias challenge, i.e., how to reduce the bias in the larger and biased data $S^{\mathcal{A}}$. In reality, bias is ubiquitous, hidden and unmeasurable. Because recommender systems are often large and complex systems, the whole process, including selecting the target items from massive candidates and then showing them to a user, requires different engineering and algorithmic efforts. In addition, because recommender systems are closed-loop systems, the bias in the training data weakens a recommendation model to learn the users' real preferences, which will affect the next recommendation and even the health of the entire ecology. Secondly, the heterogeneity challenge, i.e., how to jointly model these two different types of feedback $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$ in a complementary way. The distributions of the biased and unbiased data are different, mainly for the following two reasons. One is that the biased data cannot truly represent the users' preferences due to the existence of bias, while the unbiased data is opposite since it is being collected by the uniform policy. The other is that the (user, item) pairs in both data are usually not the same. Finally, the unbalance challenge, i.e., the scale of the biased data and the unbiased data differs greatly. The unbiased data is usually quite small, and some users

Table 1
Notations.

Notation	Explanation
n	user number
m	item number
$u \in \{1, 2, ..., n\}$	user ID
$i \in \{1, 2, ..., m\}$	item ID
$\mathcal{U}$	whole set of users
$\mathcal{I}$	whole set of items
$\mathcal{D} = \mathcal{U} \times \mathcal{I}$	whole set of (user, item) pairs
$S^{\mathcal{A}} = \{(u, i)\}$	a larger and biased data collected by a commonly deployed non-uniform policy
$S^{\mathcal{T}} = \{(u, i)\}$	a small and unbiased data collected by a specifically designed uniform (i.e., random) policy
$y_u^{\mathcal{A}}$	an observed click w.r.t. user u from $S^{\mathcal{A}}$
$y_u^{\mathcal{T}}$	an observed click w.r.t. user u from $S^{\mathcal{T}}$
$\hat{y}_u^{\mathcal{C}}$	a predicted click w.r.t. user u from the confusing latent features
$\hat{y}_u^{\mathcal{A}}$	a predicted click w.r.t. user u from the latent features of bias information
$\hat{y}_u^{ideal}$	an unbiased predicted click w.r.t. user u by combining $\hat{y}_u^{\mathcal{C}}$ and $\hat{y}_u^{\mathcal{A}}$
$F^{\mathcal{C}}(\cdot)$	the function for extracting the confusing latent features
$F^{\mathcal{A}}(\cdot)$	the function for extracting the latent features of bias information
$T^{\mathcal{A}}(\cdot)$	a transform function (e.g., sigmoid function)
$G(\cdot)$	the function to get the prediction from the latent features
$z_u^{\mathcal{C}}$	the confusing latent features that contain the user's preferences and bias information for user u
$z_u^{\mathcal{A}}$	the latent features of bias information for user u
$\tilde{z}_u^{\mathcal{A}}$	the latent features of bias information for user u constrained by $T^{\mathcal{A}}(\cdot)$
$\mathcal{L}(\cdot, \cdot)$	an arbitrary loss function such as the cross-entropy loss
$\mathcal{L}_{\text{TJR}}$	the total loss of our TJR
$\alpha, \gamma, \omega, \eta$	the hyper-parameters

even do not have any unbiased sample. The unbiased data will hurt the users' experience and the system revenue during the collection process, which is thus often small, making it severely different from the biased data in scale. Therefore, this comes two difficulties, one is that it is difficult to extract useful information from the unbiased data, and the other is that the recommendation model tends to be guided by the biased data.

3.3. Overview of our solution

As a response to the aforementioned three challenges, we propose a novel transfer learning-based AI solution, which projects the studied problem to a transfer learning view, and proposes an effective knowledge transfer mechanism with bias reduction to address the three challenges. Firstly, for the bias challenge in the auxiliary data $S^{\mathcal{A}}$, we adopt two different models to extract the latent features of users' interests and the latent features of users' bias information, respectively, and then correct the predictions by the models in a linear manner. Secondly, for the heterogeneity challenge, we design a joint loss function to reconstruct both the target data and auxiliary data in a seamless manner. Finally, for the unbalanced challenge, we design two approaches, one of which introduces a bias regularization, and the other incorporates the technology of bidirectional knowledge distillation.

As we all know, in a typical closed-loop recommender system, the bias continuously affects the performance of the recommender system like snowballs. Our TJR can enhance the overall intelligence and interactivity of the recommender system through effectively alleviating the bias and generating a more reasonable recommendation list, which can better meet the users' individual needs, and make the interaction between users and the system smoother.

4. Our solution: transfer via joint reconstruction

In this section, we show the details and motivation of our proposed solution, and how to apply it to four commonly used recommendation models, i.e., variational autoencoders (VAE), matrix factorization (MF), neural collaborative filtering (NCF) and graph convolution network (GCN). From the perspective of transfer learning [43], we regard the larger and biased data $S^{\mathcal{A}}$ as the auxiliary data, and the small and unbiased data $S^{\mathcal{T}}$ as the target data. From the idea of latent variable model, we can learn users' latent features from $S^{\mathcal{A}}$ via sufficient training. However, due to the existence of bias, these latent features are mixed with features containing bias information. Meanwhile, we can not obtain users' unbiased latent features well from $S^{\mathcal{T}}$ because its scale is often too small.

In order to make full use of the information in $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$, we propose a novel transfer learning solution, i.e., transfer via joint reconstruction (TJR), which is illustrated in Fig. 2. Intuitively, the learned latent features, whether they represent users' preferences or bias information, ultimately affect the prediction of the model. Therefore, we use two different models to extract the latent features that represent users' preferences and bias information, and then refine the prediction in a linear manner to alleviate bias problem.

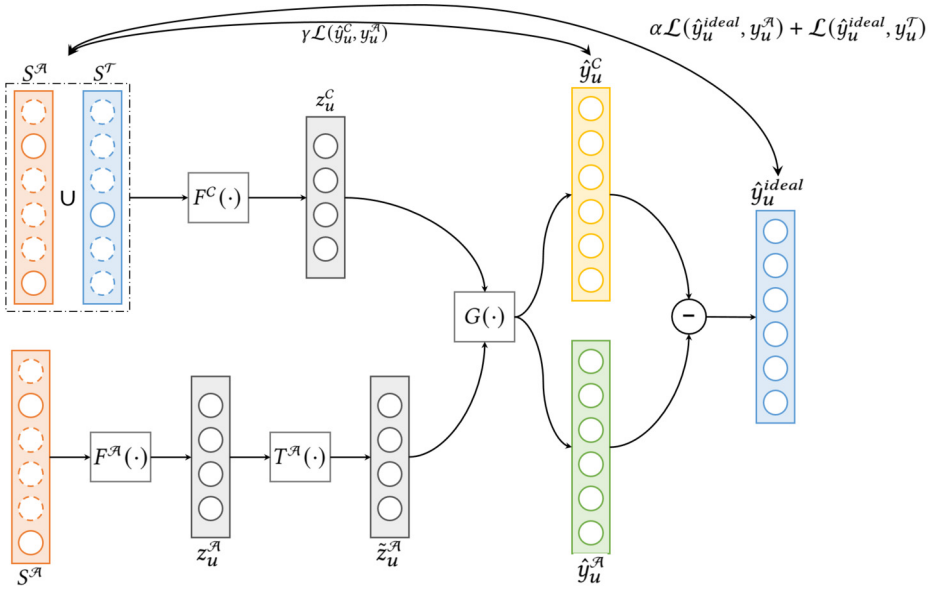


Fig. 2. Illustration of our transfer via joint reconstruction (TJR). We extract the corresponding information from two different data. $F^{\mathcal{C}}(\cdot)$ is used to extract the confusing latent features $z_u^{\mathcal{C}}$ that contain both a user's preferences and bias information, and both $F^{\mathcal{A}}(\cdot)$ and $T^{\mathcal{A}}(\cdot)$ are used to extract latent features $z_u^{\mathcal{A}}$ about the bias information. Based on $z_u^{\mathcal{C}}$, $z_u^{\mathcal{A}}$ and $G(\cdot)$, we can get the unbiased prediction $\hat{y}_u^{ideal}$. Notice that $G(\cdot)$ is shared and the arcs denote the loss functions to be minimized simultaneously, i.e., reconstructing both the biased and unbiased data, to achieve the effect of joint reconstruction.

Specifically, we use the union of $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$ to obtain the confusing latent features $z_u^{\mathcal{C}}$ that contain the user's preferences and bias information through the function for latent feature extraction, i.e., $F^{\mathcal{C}}(\cdot)$. After getting $z_u^{\mathcal{C}}$, we use $G(\cdot)$ to get the prediction $\hat{y}_u^{\mathcal{C}}$. Notice that $\hat{y}_u^{\mathcal{C}}$ is usually biased. In particular, if we use VAE as the backbone model, $F^{\mathcal{C}}(\cdot)$ refers to the encoder and $G(\cdot)$ refers to the decoder.

We use $S^{\mathcal{A}}$ to extract latent features of bias information $\tilde{z}_u^{\mathcal{A}}$ through $F^{\mathcal{A}}(\cdot)$ and $T^{\mathcal{A}}(\cdot)$. Notice that $F^{\mathcal{A}}(\cdot)$ is the same as $F^{\mathcal{C}}(\cdot)$ in structure and $T^{\mathcal{A}}(\cdot)$ is a transform function. The reason for introducing the transform function is to constrain the latent features of the bias within a controllable range. We use sigmoid as the default transform function in our TJR and also study the impact of different ones on the recommendation performance. After obtaining $\tilde{z}_u^{\mathcal{A}}$, we use $G(\cdot)$ to get the prediction $\hat{y}_u^{\mathcal{A}}$. Notice that $G(\cdot)$ in the bottom branch is shared with the one in the upper branch that generates $\hat{y}_u^{\mathcal{C}}$. The reason is that $G(\cdot)$ serves to map the values in the latent feature space to the label space, and the mapping relationship is theoretically unchanging regardless of whether the latent features are biased or unbiased.

To obtain an unbiased prediction $\hat{y}_u^{ideal}$, we intuitively use a linear method through $\hat{y}_u^{\mathcal{C}}$ and $\hat{y}_u^{\mathcal{A}}$, i.e., $\hat{y}_u^{ideal} = \hat{y}_u^{\mathcal{C}} - \hat{y}_u^{\mathcal{A}}$. Finally, in order to train our TJR, we naturally design a joint reconstruction loss function, i.e., reconstructing both the biased and unbiased data. For user u, we can easily obtain the loss function, i.e., $\alpha\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}})$, where $\mathcal{L}(\cdot,\cdot)$ denotes an arbitrary loss function such as the cross-entropy loss. Notice that we follow the idea of the Weight strategy [4] and introduce a hyper-parameter α in the above loss function. To learn the biased features $z_u^{\mathcal{C}}$ better, we introduce an additional loss function, i.e., $\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}})$. Finally, we have the overall loss function of our TJR,

$$\mathcal{L}_{\text{TJR}} = \alpha\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}), \tag{1}$$

where α and γ are the hyper-parameters, $y_u^{\mathcal{A}}$ are the observed labels from $S_u^{\mathcal{A}}(S_u^{\mathcal{A}} \subseteq S^{\mathcal{A}})$ and $y_u^{\mathcal{T}}$ from $S_u^{\mathcal{T}}(S_u^{\mathcal{T}} \subseteq S^{\mathcal{T}})$.

4.1. Specialization via variational autoencoders

Variational autoencoders (VAE) is an important type of generative model [76], which has been widely used in computer vision. Given a certain amount of data, it learns the intrinsic properties of the data by combining variational inference and autoencoders, and then generates some data that approximates the original input. In [33], variational autoencoders is introduced to model users' implicit feedback and achieves very promising results in a recommender system. Notice that unlike other models, variational autoencoders obtains the distribution of the users' latent features rather than some specific values, which is the main reason why it is usually more effective.

In this section, we discuss how to apply our TJR when we use variational autoencoders as a backbone model. Variational autoencoders includes an encoder and a decoder, where the former is an inference model and the latter is a generative model. In a recommender system, the goal of the encoder is to produce the mean and standard deviation of the distribution of a user's latent features, and the function of the decoder is to reconstruct the user's interaction vector from the latent

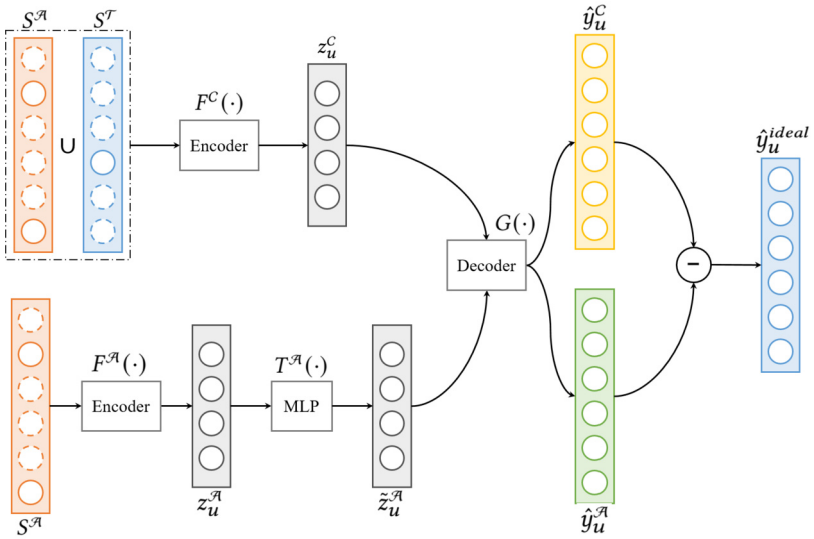


Fig. 3. Illustration of our transfer via joint reconstruction (TJR) with the backbone model of variational autoencoders (VAE).

feature distribution. In order words, taking the users' observed interaction vector $\mathbf{y}_u$ as input, we first obtain the users' latent features through an encoder, and then get the users' prediction $\hat{\mathbf{y}}_u$ via a decoder,

$$\hat{\mathbf{y}}_u = Decoder(Encoder(\mathbf{y}_u)) \tag{2}$$

where $Encoder(\cdot)$ is an encoder and $Decoder(\cdot)$ is a decoder.

For our TJR in Fig. 2, $F^{\mathcal{C}}(\cdot)$ and $F^{\mathcal{A}}(\cdot)$ refer to two different encoders, and $G(\cdot)$ corresponds to the decoder. The model structure is shown in the Fig. 3. In our experiments, the encoder and decoder in our solution have the same structure as that in VAE [33].

4.1.1. The upper branch

In our TJR, $F^{\mathcal{C}}(\cdot)$ is an encoder to produce the distribution of a user's interest latent features. We use $\mathbf{z}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent representation of user u, and $\boldsymbol{\mu}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ and $\boldsymbol{\sigma}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the corresponding mean and standard deviation of $\mathbf{z}_u^{\mathcal{C}}$, respectively, where k is the dimension of the latent features. Taking the user u's observed interaction vector $\mathbf{y}_u^{\mathcal{A}} \in \{0,1\}^{1\times m}$ from $S^{\mathcal{A}}$ and $\mathbf{y}_u^{\mathcal{T}} \in \{0,1\}^{1\times m}$ from $S^{\mathcal{T}}$ as input, we can obtain $\mathbf{z}_u^{\mathcal{C}}$ by sampling from a variational distribution $q_\phi(\mathbf{z}_u^{\mathcal{C}}|\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}})$,

$$q_\phi(\mathbf{z}_u^{\mathcal{C}}|\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}}) = \mathcal{N}(\boldsymbol{\mu}_u^{\mathcal{C}}, \mathrm{diag}(\boldsymbol{\sigma}_u^{\mathcal{C}^2})), \tag{3}$$

where $\boldsymbol{\mu}_u^{\mathcal{C}} = \mathrm{Relu}((\mathbf{y}_u^{\mathcal{A}} + \mathbf{y}_u^{\mathcal{T}})\mathbf{W}_\mu^{\mathcal{C}} + \mathbf{b}_\mu^{\mathcal{C}})$ is the mean and $\boldsymbol{\sigma}_u^{\mathcal{A}} = \exp^{((\mathbf{y}_u^{\mathcal{A}}+\mathbf{y}_u^{\mathcal{T}})\mathbf{W}_\sigma^{\mathcal{C}}+\mathbf{b}_\sigma^{\mathcal{C}})}$ is the standard deviation. Notice that $\mathbf{W}_\mu^{\mathcal{C}} \in \mathbb{R}^{m\times k}$ and $\mathbf{W}_\sigma^{\mathcal{C}} \in \mathbb{R}^{m\times k}$ are the weight matrices, and $\mathbf{b}_\mu^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ and $\mathbf{b}_\sigma^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ are the bias vectors.

During training, it is difficult to derive the gradients due to the sampling process, so we adopt the commonly used re-parameterization trick to avoid this issue [76,77], i.e., we re-parameterize $\mathbf{z}_u^{\mathcal{C}}$ by $\boldsymbol{\mu}_u^{\mathcal{C}} + \varepsilon \odot \boldsymbol{\sigma}_u^{\mathcal{C}}$, where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathrm{diag}(1))$. In addition, according to variational inference, we add a constraint that minimizes the Kullback-Leiber (KL) divergence between $q_\phi(\mathbf{z}_u^{\mathcal{C}}|\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}})$ and the standard normal distribution $p(\mathbf{z}_u)$,

$$\mathcal{L}_{\mathrm{KL}}(\mathbf{z}_u^{\mathcal{C}}) = \mathrm{KL}(q_\phi(\mathbf{z}_u^{\mathcal{C}}|\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}})||p(\mathbf{z}_u)). \tag{4}$$

After that, given $\mathbf{z}_u^{\mathcal{C}}$, we could get a reconstructed interaction vector denoted as $\hat{\mathbf{y}}_u^{\mathcal{C}}$ through the decoder $G(\cdot)$,

$$\hat{\mathbf{y}}_u^{\mathcal{C}} \sim p_\theta(\mathbf{y}_u^{\mathcal{C}}|\mathbf{z}_u^{\mathcal{C}}). \tag{5}$$

4.1.2. The bottom branch

$F^{\mathcal{A}}(\cdot)$ is an encoder to produce the distribution of a user's bias latent features. Notice that $F^{\mathcal{A}}(\cdot)$ and $F^{\mathcal{C}}(\cdot)$ have different roles. Although the two encoders can have different structures, for simplicity, we let them have the same structure. Therefore, we use $\tilde{\mathbf{z}}_u^{\mathcal{A}} \in \mathbb{R}^{1\times k}$ to denote the latent representation of bias information for user u, and $\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}} \in \mathbb{R}^{1\times k}$ and $\tilde{\boldsymbol{\sigma}}_u^{\mathcal{A}} \in \mathbb{R}^{1\times k}$ to denote the corresponding mean and standard deviation of $\tilde{\mathbf{z}}_u^{\mathcal{A}}$, respectively. Given the user u's observed interaction vector $\mathbf{y}_u^{\mathcal{A}} \in \{0,1\}^{1\times m}$ from $S^{\mathcal{A}}$ as input, we can get $\tilde{\mathbf{z}}_u^{\mathcal{A}}$ by sampling from a variational distribution $q_{\phi'}(\tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}})$,

$$q_{\phi'}(\tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}}) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}}, \mathrm{diag}(\tilde{\boldsymbol{\sigma}}_u^{\mathcal{A}2})), \tag{6}$$

where $\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}} = \mathrm{Sigmoid}(\mathbf{y}_u^{\mathcal{A}}\mathbf{W}_\mu^{\mathcal{A}} + \mathbf{b}_\mu^{\mathcal{A}})$ is the mean and $\tilde{\boldsymbol{\sigma}}_u^{\mathcal{A}} = \exp^{(\mathbf{y}_u^{\mathcal{A}}\mathbf{W}_\sigma^{\mathcal{A}} + \mathbf{b}_\sigma^{\mathcal{A}})}$ is the standard deviation. Notice that $\mathbf{W}_\mu^{\mathcal{A}} \in \mathbb{R}^{m\times k}$, $\mathbf{W}_\sigma^{\mathcal{A}} \in \mathbb{R}^{m\times k}$, $\mathbf{b}_\mu^{\mathcal{A}} \in \mathbb{R}^{1\times k}$, and $\mathbf{b}_\sigma^{\mathcal{A}} \in \mathbb{R}^{1\times k}$.

Similarly, we could obtain $\tilde{\mathbf{z}}_u^{\mathcal{A}}$ by $\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}} + \varepsilon \odot \tilde{\boldsymbol{\sigma}}_u^{\mathcal{A}}$ based on the re-parameterization trick, and we again use the KL divergence to constrain $q_{\phi'}(\tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}})$ and the standard normal distribution $p(\mathbf{z}_u)$,

$$\mathcal{L}_{\mathrm{KL}}(\tilde{\mathbf{z}}_u^{\mathcal{A}}) = \mathrm{KL}(q_{\phi'}(\tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}})||p(\mathbf{z}_u)). \tag{7}$$

Let $\hat{\mathbf{y}}_u^{\mathcal{A}}$ denote a reconstructed interaction vector caused by the bias information through the decoder $G(\cdot)$,

$$\hat{\mathbf{y}}_u^{\mathcal{A}} \sim p_\theta(\mathbf{y}_u^{\mathcal{A}}|\tilde{\mathbf{z}}_u^{\mathcal{A}}). \tag{8}$$

4.1.3. Loss function

In VAE, we expect that the reconstruction loss function is minimized so as to ensure that the latent features of users' interest or bias information are learned well during the training process. Following [33], in our solution, we adopt a multinomial conditional likelihood function to achieve $\hat{\mathbf{y}}_u^{ideal}$ as close to $\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}}$ as possible, and $\hat{\mathbf{y}}_u^{\mathcal{C}}$ also as close to $\mathbf{y}_u^{\mathcal{A}}$ as possible, where $\hat{\mathbf{y}}_u^{ideal} = \hat{\mathbf{y}}_u^{\mathcal{C}} - \hat{\mathbf{y}}_u^{\mathcal{A}}$. Specifically, for user u, we can obtain the reconstruction loss,

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) \equiv \mathbb{E}_{q_\phi(\mathbf{z}_u^{\mathcal{C}}, \tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}}, \mathbf{y}_u^{\mathcal{T}})}[\log p_\theta(\mathbf{y}_u^{\mathcal{A}}|\mathbf{z}_u^{\mathcal{C}}, \tilde{\mathbf{z}}_u^{\mathcal{A}})], \tag{9}$$

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) \equiv \mathbb{E}_{q_\phi(\mathbf{z}_u^{\mathcal{C}}, \tilde{\mathbf{z}}_u^{\mathcal{A}}|\mathbf{y}_u^{\mathcal{A}}, \mathbf{y}_u^{\mathcal{T}})}[\log p_\theta(\mathbf{y}_u^{\mathcal{T}}|\mathbf{z}_u^{\mathcal{C}}, \tilde{\mathbf{z}}_u^{\mathcal{A}})], \tag{10}$$

$$\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}) \equiv \mathbb{E}_{q_\phi(\mathbf{z}_u^{\mathcal{C}}|\mathbf{y}_u^{\mathcal{A}} \cup \mathbf{y}_u^{\mathcal{T}})}[\log p_\theta(\mathbf{y}_u^{\mathcal{A}}|\mathbf{z}_u^{\mathcal{C}})]. \tag{11}$$

Taking all the above derivations together, for user u, we reach an overall loss function to be minimized,

$$\mathcal{L}_{\mathrm{TJR-VAE}} = \alpha\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}) + \beta\left(\mathcal{L}_{\mathrm{KL}}(\mathbf{z}_u^{\mathcal{C}}) + \mathcal{L}_{\mathrm{KL}}(\tilde{\mathbf{z}}_u^{\mathcal{A}})\right), \tag{12}$$

where α, γ and β are the hyper-parameters.

4.2. Specialization via matrix factorization

Matrix factorization (MF) is a classical model in a recommender system. Its essence is to decompose the interaction matrix between users and items, e.g., the rating matrix or the click matrix, into the product of two low-rank matrices, one of which represents the latent vectors of the users, and the other represents the latent vectors of the items. Notice that matrix factorization is an improvement of collaborative filtering, which can alleviate the data sparsity problem to a certain extent. At the same time, it also belongs to a machine learning algorithm, which can generally be learned by gradient descent or alternating least squares.

In this section, we explore how to apply our TJR when we use matrix factorization as a backbone model. There are several variants of matrix factorization, such as considering rating bias, using square loss function, and using cross entropy loss function. In our experiments, we consider the items' rating bias, and use the cross entropy as the loss function. In other words, we consider the predicted feedback $\hat{y}_{ui}$ of user u on item i as follows,

$$\hat{y}_{ui} = \mathbf{U}_u \cdot \mathbf{V}_i^T + b_i, \tag{13}$$

where $\mathbf{U}_u$ denotes the latent feature vector of user u and $\mathbf{V}_i$ denotes the latent feature vector of item i.

For our TJR in Fig. 2, $F^{\mathcal{C}}(\cdot)$ and $F^{\mathcal{A}}(\cdot)$ refer to two different embedding layers, and $G(\cdot)$ corresponds to the inner product. The model structure is shown in the Fig. 4.

4.2.1. The upper branch

We use $\mathbf{U}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the user's information, and use $\mathbf{V}_i^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the item's information, and use $b_i^{\mathcal{C}} \in \mathbb{R}^{1\times 1}$ to denote the item bias, where k is the dimension of the latent vectors. In our experiment, we randomly initialize $\mathbf{U}_u^{\mathcal{C}}$ and $\mathbf{V}_i^{\mathcal{C}}$, and initialize $b_i^{\mathcal{C}}$ to zero. According to the principle of matrix factorization, for a (u, i) pair, we can obtain the prediction through their inner product,

$$\hat{y}_{ui}^{\mathcal{C}} = \mathbf{U}_u^{\mathcal{C}} \cdot \mathbf{V}_i^{\mathcal{C}T} + b_i^{\mathcal{C}}. \tag{14}$$

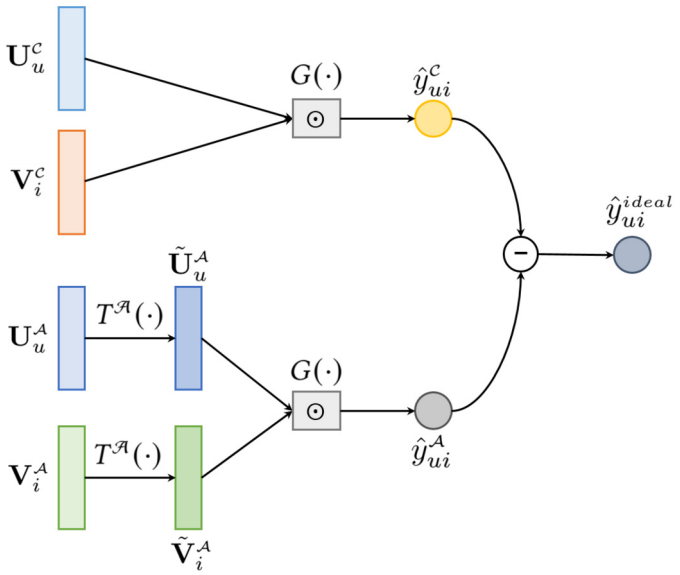


Fig. 4. Illustration of our transfer via joint reconstruction (TJR) with the backbone model of matrix factorization (MF).

4.2.2. The bottom branch

Similarly, let $\tilde{\mathbf{U}}_u^{\mathcal{A}} \in \mathbb{R}^{n\times k}$ denote the latent vector of the user's bias information, and $\tilde{\mathbf{V}}_i^{\mathcal{A}} \in \mathbb{R}^{m\times k}$ denote the latent vector of the item's bias information. In our experiments, we first randomly initialize two variables $\mathbf{U}_u^{\mathcal{A}}$ and $\mathbf{V}_i^{\mathcal{A}}$. Then we use a transform function to obtain $\tilde{\mathbf{U}}_u^{\mathcal{A}}$ and $\tilde{\mathbf{V}}_i^{\mathcal{A}}$,

$$\tilde{\mathbf{U}}_u^{\mathcal{A}} = \text{Sigmoid}(\mathbf{U}_u^{\mathcal{A}}), \tilde{\mathbf{V}}_i^{\mathcal{A}} = \text{Sigmoid}(\mathbf{V}_i^{\mathcal{A}}). \tag{15}$$

Based on the principle of matrix factorization, we can obtain the prediction through the inner product related to by the bias information. Notice that we do not consider the impact of the item bias for simplicity.

$$\hat{y}_{ui}^{\mathcal{A}} = \tilde{\mathbf{U}}_u^{\mathcal{A}} \cdot \tilde{\mathbf{V}}_i^{\mathcal{A}T}. \tag{16}$$

4.2.3. Loss function

According to our solution, for (u, i) pair, we could obtain the unbiased prediction denoted as $\hat{y}_{ui}^{ideal}$ through $\hat{y}_{ui}^{\mathcal{C}}$ and $\hat{y}_{ui}^{\mathcal{A}}$, i.e., $\hat{y}_{ui}^{ideal} = \hat{y}_{ui}^{\mathcal{C}} - \hat{y}_{ui}^{\mathcal{A}}$. In our experiments, we adopt the cross-entropy loss to train the model. Therefore, for user u, we can obtain the specific loss,

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) = \sum_{(u,i)\in S_u^{\mathcal{A}}} -y_{ui}^{\mathcal{A}} log(\sigma(\hat{y}_{ui}^{ideal})) - (1 - y_{ui}^{\mathcal{A}}) log(1 - \sigma(\hat{y}_{ui}^{ideal})), \tag{17}$$

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) = \sum_{(u,i)\in S_u^{\mathcal{T}}} -y_{ui}^{\mathcal{T}} log(\sigma(\hat{y}_{ui}^{ideal})) - (1 - y_{ui}^{\mathcal{T}}) log(1 - \sigma(\hat{y}_{ui}^{ideal})), \tag{18}$$

$$\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}) = \sum_{(u,i)\in S_u^{\mathcal{A}}} -y_{ui}^{\mathcal{A}} log(\sigma(\hat{y}_{ui}^{\mathcal{C}})) - (1 - y_{ui}^{\mathcal{A}}) log(1 - \sigma(\hat{y}_{ui}^{\mathcal{C}})), \tag{19}$$

where $\sigma(\cdot)$ is a sigmoid function.

Finally, the loss function when our TJR is applied to MF as follows,

$$\mathcal{L}_{\text{TJR-MF}} = \alpha \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma \mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}). \tag{20}$$

4.3. Specialization via neural collaborative filtering

Neural collaborative filtering (NCF) is a model that combines deep learning and matrix factorization [34]. It retains the latent vectors of users and the latent vectors of items along the lines of matrix factorization, and replaces the simple inner product with multilayer perceptron. Multilayer perceptron can improve the model's nonlinear modeling ability, making the NCF more expressive.

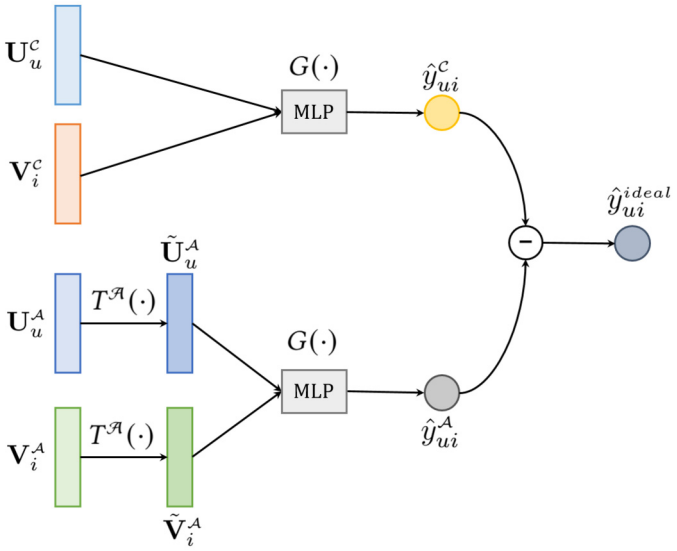


Fig. 5. Illustration of our transfer via joint reconstruction (TJR) with the backbone model of neural collaborative filtering (NCF).

In this section, we describe how to apply our TJR when we use neural collaborative filtering as a backbone model. In [34], the authors also propose generalized matrix factorization, which simultaneously uses inner product and multilayer perceptron to calculate the similarity between users and items. In our experiments, we only use multilayer perceptron to calculate the similarity. In other words, for a (u, i) pair, we predict the feedback $\hat{y}_{ui}$ between them as follows,

$$\hat{y}_{ui} = (tanh([\mathbf{U}_u; \mathbf{V}_i]\mathbf{W}_1 + \mathbf{b}_1))\,\mathbf{W}_2, \tag{21}$$

where $\mathbf{U}_u$ denotes the latent feature vector of user u, $\mathbf{V}_i$ denotes the latent feature vector of item i, $\mathbf{W}_1$ and $\mathbf{W}_2$ are the weight matrices, and $\mathbf{b}_1$ is the bias vector.

For our TJR in Fig. 2, $F^{\mathcal{C}}(\cdot)$ and $F^{\mathcal{A}}(\cdot)$ refer to two different embedding layers, and $G(\cdot)$ corresponds to the multilayer perceptron. The model structure is shown in the Fig. 5.

4.3.1. The upper branch

Similar to matrix factorization, we use $\mathbf{U}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to represent the latent vector of the user's information, and use $\mathbf{V}_i^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the item's information. In our experiments, we use two embedding matrices to represent $\mathbf{U}_u^{\mathcal{C}}$ and $\mathbf{V}_i^{\mathcal{C}}$, and initialize them randomly. According to Eq. (21), for a (u, i) pair, we can obtain the prediction between them as follows,

$$\hat{y}_{ui}^{\mathcal{C}} = \left(tanh([\mathbf{U}_u^{\mathcal{C}}; \mathbf{V}_i^{\mathcal{C}}]\mathbf{W}_1 + \mathbf{b}_1)\right)\mathbf{W}_2, \tag{22}$$

where $\mathbf{W}_1 \in \mathbb{R}^{2k\times k}$ and $\mathbf{W}_2 \in \mathbb{R}^{k\times 1}$ are the weight matrices, and $\mathbf{b}_1 \in \mathbb{R}^{k\times 1}$ is the bias vector.

4.3.2. The bottom branch

We use $\tilde{\mathbf{U}}_u^{\mathcal{A}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the user's bias information, and use $\tilde{\mathbf{V}}_i^{\mathcal{A}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the item's bias information. Similar to matrix factorization, we initialize two variables randomly, and get $\tilde{\mathbf{U}}_u^{\mathcal{A}}$ and $\tilde{\mathbf{V}}_i^{\mathcal{A}}$ through the transform function,

$$\tilde{\mathbf{U}}_u^{\mathcal{A}} = \text{Sigmoid}(\mathbf{U}_u^{\mathcal{A}}), \tilde{\mathbf{V}}_i^{\mathcal{A}} = \text{Sigmoid}(\mathbf{V}_i^{\mathcal{A}}), \tag{23}$$

where $\mathbf{U}_u^{\mathcal{A}}$ and $\mathbf{V}_i^{\mathcal{A}}$ are two intermediate variables.

After that, based on our solution, we can obtain the prediction related to the bias information for a (u, i) pair,

$$\hat{y}_{ui}^{\mathcal{A}} = \left(tanh([\tilde{\mathbf{U}}_u^{\mathcal{A}}; \tilde{\mathbf{V}}_i^{\mathcal{A}}]\mathbf{W}_1 + \mathbf{b}_1)\right)\mathbf{W}_2. \tag{24}$$

Notice that the multilayer perceptron is shared with that in the upper branch.

4.3.3. Loss function

For a (u, i) pair, we can get the unbiased prediction through the connection between $\hat{y}_{ui}^{\mathcal{C}}$ and $\hat{y}_{ui}^{\mathcal{A}}$, i.e., $\hat{y}_{ui}^{ideal} = \hat{y}_{ui}^{\mathcal{C}} - \hat{y}_{ui}^{\mathcal{A}}$, where $\hat{y}_{ui}^{ideal}$ denotes the unbiased prediction. Similar to matrix factorization, we adopt the cross-entropy loss to training the model. Therefore, for user u, we have the following loss function,

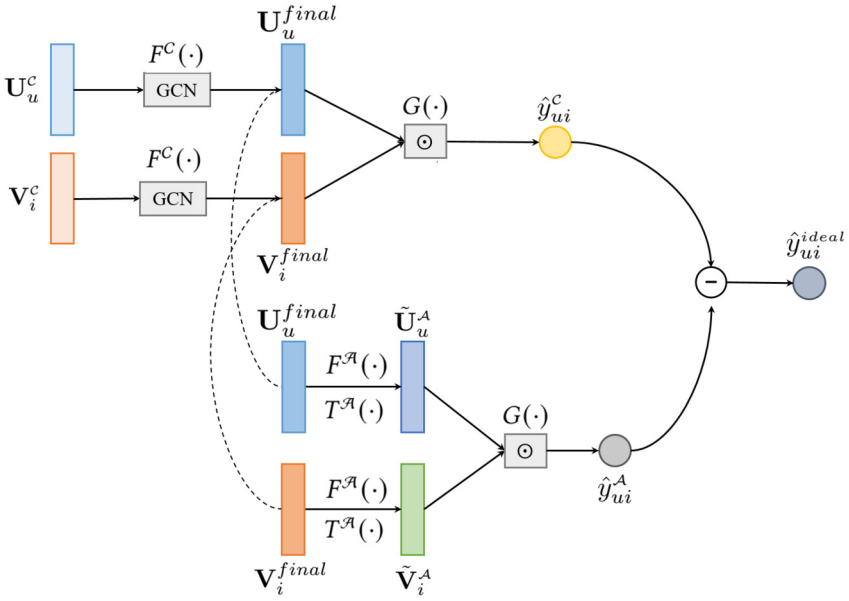


Fig. 6. Illustration of our transfer via joint reconstruction (TJR) with the backbone model of graph convolution network (GCN). The dashed line denotes the two embedding layers are the same.

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) = \sum_{(u,i)\in S_u^{\mathcal{A}}} -y_{ui}^{\mathcal{A}} log(\sigma(\hat{y}_{ui}^{ideal})) - (1 - y_{ui}^{\mathcal{A}}) log(1 - \sigma(\hat{y}_{ui}^{ideal})), \tag{25}$$

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) = \sum_{(u,i)\in S_u^{\mathcal{T}}} -y_{ui}^{\mathcal{T}} log(\sigma(\hat{y}_{ui}^{ideal})) - (1 - y_{ui}^{\mathcal{T}}) log(1 - \sigma(\hat{y}_{ui}^{ideal})), \tag{26}$$

$$\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}) = \sum_{(u,i)\in S_u^{\mathcal{A}}} -y_{ui}^{\mathcal{A}} log(\sigma(\hat{y}_{ui}^{\mathcal{C}})) - (1 - y_{ui}^{\mathcal{A}}) log(1 - \sigma(\hat{y}_{ui}^{\mathcal{C}})), \tag{27}$$

where $\sigma(\cdot)$ is a sigmoid function.

In summary, for user u, the total loss function is,

$$\mathcal{L}_{\text{TJR}-\text{NCF}} = \alpha \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma \mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}). \tag{28}$$

4.4. Specialization via graph convolution network

In this section, we show how to apply our TJR when we use graph convolution network as the backbone model. In recent years, graph neural network is one of the research hotspots in academia and industry, using graph technology to establish the relationship between users and items. In collaborative filtering, LightGCN [38] is a relatively well-known method, which is improved on the basis of NGCF [37]. Therefore, we use LightGCN as the backbone model to explore the application of our TJR to graph neural networks. For our TJR in Fig. 2, $F^{\mathcal{C}}(\cdot)$ refers to the graph neural network, $F^{\mathcal{A}}(\cdot)$ refers to the multilayer perceptron and $G(\cdot)$ corresponds to the inner product. The model structure is shown in the Fig. 6.

4.4.1. The upper branch

We use $\mathbf{U}_u^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the user's information, and $\mathbf{V}_i^{\mathcal{C}} \in \mathbb{R}^{1\times k}$ to denote the latent vector of the item's information, where k is the dimension of the latent vectors. For LightGCN, through the message aggregation function, the final embeddings of users and items can be written as follows,

$$\mathbf{U}_u^{final} = \sum_{l=0}^{L} w_l \mathbf{U}_u^{\mathcal{C}(l)}, \tag{29}$$

$$\mathbf{V}_i^{final} = \sum_{l=0}^{L} w_l \mathbf{V}_i^{\mathcal{C}(l)}, \tag{30}$$

where l denotes the l-th layer, L is the total number of layers, and w_l represents the weight of the l-th layer embedding in composing the final embedding.

4.4.2. The bottom branch

Under the guidance of our TJR, we will rebuild a brand-new graph model to extract the latent features of the bias information from the biased data. The complexity of the graph neural network model is high, and the modification mentioned above further increases the complexity of the model and the difficulty of training the model. Therefore, we make a light modification, i.e., we employ the feature extractor $F^{\mathcal{A}}(\cdot)$ and the transform function $T^{\mathcal{A}}(\cdot)$ from the final embeddings $\mathbf{U}_u^{final}$ and $\mathbf{V}_i^{final}$. In our experiments, we employ a single-layer fully connected network with a sigmoid function to extract the latent features of the bias information,

$$\tilde{\mathbf{U}}_u^{\mathcal{A}} = \text{Sigmoid}(\mathbf{U}_u^{final}\mathbf{W}_{gcn(u)} + \mathbf{b}_{gcn(u)}), \tag{31}$$

$$\tilde{\mathbf{V}}_i^{\mathcal{A}} = \text{Sigmoid}(\mathbf{V}_i^{final}\mathbf{W}_{gcn(v)} + \mathbf{b}_{gcn(v)}), \tag{32}$$

where $\mathbf{W}_{gcn(u)} \in \mathbb{R}^{k\times k}$ and $\mathbf{W}_{gcn(v)} \in \mathbb{R}^{k\times k}$ are the weight matrices, and $\mathbf{b}_{gcn(u)})$ and $\mathbf{b}_{gcn(v)}$ are the bias vectors.

4.4.3. Loss function

Similarly, for a (u, i) pair, we could obtain the unbiased prediction $\hat{y}_u^{ideal}$,

$$\hat{y}_{ui}^{\mathcal{C}} = \mathbf{U}_u^{final} \cdot \mathbf{V}_i^{final^T}, \tag{33}$$

$$\hat{y}_{ui}^{\mathcal{A}} = \tilde{\mathbf{U}}_u^{\mathcal{A}} \cdot \tilde{\mathbf{V}}_i^{\mathcal{A}T}, \tag{34}$$

$$\hat{y}_{ui}^{ideal} = \hat{y}_{ui}^{\mathcal{C}} - \hat{y}_{ui}^{\mathcal{A}}, \tag{35}$$

where $\hat{y}_u^{\mathcal{C}}$ is the prediction of the upper branch in our TJR and $\hat{y}_u^{\mathcal{A}}$ is the prediction of the bottom branch.

In addition, following the original work [38], we use the BPR loss function. For user u, we have the following loss function,

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) = -\sum_{u\in\mathcal{U}}\sum_{i\in S_u^{\mathcal{A}}}\sum_{j\notin(S_u^{\mathcal{A}}\cup S_u^{\mathcal{T}})} \ln \text{Softplus}(\hat{y}_{ui}^{\text{ideal}} - \hat{y}_{uj}^{\text{ideal}}), \tag{36}$$

$$\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) = -\sum_{u\in\mathcal{U}}\sum_{i\in S_u^{\mathcal{T}}}\sum_{j\notin(S_u^{\mathcal{A}}\cup S_u^{\mathcal{T}})} \ln \text{Softplus}(\hat{y}_{ui}^{\text{ideal}} - \hat{y}_{uj}^{\text{ideal}}), \tag{37}$$

$$\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}) = -\sum_{u\in\mathcal{U}}\sum_{i\in(S_u^{\mathcal{A}}\cup S_u^{\mathcal{T}})}\sum_{j\notin(S_u^{\mathcal{A}}\cup S_u^{\mathcal{T}})} \ln \text{Softplus}(\hat{y}_{ui}^{\mathcal{C}} - \hat{y}_{uj}^{\mathcal{C}}). \tag{38}$$

Finally, the loss function when our TJR is applied to LightGCN is as follows,

$$\mathcal{L}_{\text{TJR}-\text{GCN}} = \alpha\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}). \tag{39}$$

4.5. Discussions

In this paper, we study on common recommender systems, besides, there are some special recommender systems such as fashion recommender systems [78], multimedia recommender systems [79], music recommender systems [80] and sequence-aware recommender systems [81]. To alleviate the bias in above recommender systems through our TJR, we first need to collect an unbiased data, and then modify the backbone model lightly. In fashion recommender systems, common biases include seasonal bias, gender bias, popularity bias, etc. In multimedia recommender systems, common biases include multimodal differences, popularity bias and fairness. The music recommender systems and sequence-aware recommender systems are similar to other common recommender systems, including popularity bias, location bias, and community bias. Although different recommender systems face different types of biases, most of them are reflected in users' feedback data, so they can be considered as data bias. Therefore, to alleviate the bias, we can first get an unbiased data by setting a random policy or creating virtual unbiased data. Among them, setting a random policy refers to displaying randomly selected items to the users, and creating virtual unbiased data means to use methods such as IPS to construct a virtual unbiased data.

In addition to the recommender systems, some tasks such as vision tasks and sequential tasks also have biases, and we can also use transfer learning to alleviate them. Biases have commonality across different tasks, and can be classified into three categories, i.e., biases in the data (e.g., sampling bias, measurement bias and label bias), biases in the model (e.g., imputation bias, confounding bias and sample selection bias), and biases in the validation evaluation process. Transfer learning is mainly used to mitigate biases in the data and in the model with the following strategies.

- Instance-based. We can use data generation methods to obtain more data or even unbiased data, and then combine them to train a more unbiased model. For example, we may (i) use multiple models to predict the same instance, and improve the reliability of the prediction by setting a threshold; (ii) combine the inverse propensity score method to generate some relatively unbiased instances; or (iii) use a large model or a generative model to generate more instances.

- Parameters-based. We can use parameter alignment methods or parameter reuse methods to train a more unbiased model. For example, we may use a large amount of data to train a big model, and then use knowledge distillation or fine-tuning techniques to get a more unbiased model.
- Feature-based. We can use methods such as feature alignment, feature mutual exclusion and feature fusion to get more unbiased latent features, and then obtain more unbiased predictions. For example, we may (i) design and train two independent sets of feature extractors with a biased and an unbiased data as a way to obtain unbiased latent features; or (ii) design a common feature extractor or a constraint function to extract common latent features from a biased data and an unbiased data.

5. Bias regularizer

In practice, the latent features of the bias information are quite cryptic, and the learning of bias latent features by the loss $\mathcal{L}_{\text{TJR}}$ may have certain limitations, because the scale of $S^{\mathcal{T}}$ is usually limited. To alleviate this issue, we introduce a bias constraint and obtain a further extension TJR++. Specifically, in each iteration, for a user u, we randomly select a user r from $\mathcal{U}$ to construct a user pair. Then we get the following constraint,

$$\mathcal{L}_{\text{bias}} = |cos(z_u^{\mathcal{C}}, z_r^{\mathcal{C}}) - \omega cos(\tilde{z}_u^{\mathcal{A}}, \tilde{z}_r^{\mathcal{A}})|, \tag{40}$$

where $z_u^{\mathcal{C}}$ and $z_r^{\mathcal{C}}$ denote the confused latent features contained both the user's preferences and bias information of user u and r, respectively, $\tilde{z}_u^{\mathcal{A}}$ and $\tilde{z}_r^{\mathcal{A}}$ denote the latent features of the bias information of user u and r, respectively, $|\cdot|$ is an absolute function, $cos(\cdot,\cdot)$ is a cosine similarity, and ω is a hyper-parameter.

The intuition behind the bias constraint is that the difference between two users' preferences will be preserved in the bias. In particular, if two users have close preferences, they may be treated equally by the system, resulting in similar bias effects. By transferring the knowledge of the preferences to the bias, all users may benefit from the branch that extracts latent features of the bias information, especially those who lack $S_u^{\mathcal{T}}$. Finally, we optimize the model by the sum of the loss functions in Eq. (1) and Eq. (40). We denote this variant as TJR++, and the total loss function of TJR++ is as follows,

$$\mathcal{L}_{\text{TJR++}} = \mathcal{L}_{\text{TJR}} + \mathcal{L}_{\text{bias}}. \tag{41}$$

Next, we discuss how to apply our proposed bias regularizer to different backbone models. For variational autoencoders, in the preliminary experiment, we find that the improvement in model performance is not significant when using $\mathbf{z}_u^{\mathcal{C}}$ and $\tilde{\mathbf{z}}_u^{\mathcal{A}}$ for the bias constraint. We believe that this is because they actually represent the distribution of the users' latent features. During the model training process, $\mathbf{z}_u^{\mathcal{C}}$ and $\tilde{\mathbf{z}}_u^{\mathcal{A}}$ are not stable because of the presence of variance, i.e., $\boldsymbol{\sigma}_u^{\mathcal{C}}$ and $\tilde{\boldsymbol{\sigma}}_u^{\mathcal{A}}$. Therefore, we adopt the corresponding mean of $\mathbf{z}_u^{\mathcal{C}}$ and the mean of $\tilde{\mathbf{z}}_u^{\mathcal{A}}$ for the bias regularizer,

$$\mathcal{L}_{\text{bias-VAE}} = |cos(\boldsymbol{\mu}_u^{\mathcal{C}}, \boldsymbol{\mu}_r^{\mathcal{C}}) - \omega cos(\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}}, \tilde{\boldsymbol{\mu}}_r^{\mathcal{A}})|, \tag{42}$$

where $\boldsymbol{\mu}_u^{\mathcal{C}}$ and $\tilde{\boldsymbol{\mu}}_u^{\mathcal{A}}$ are the mean of $\mathbf{z}_u^{\mathcal{C}}$ and $\tilde{\mathbf{z}}_u^{\mathcal{A}}$, respectively, and $\boldsymbol{\mu}_r^{\mathcal{C}}$ and $\tilde{\boldsymbol{\mu}}_r^{\mathcal{A}}$ are the mean of $\mathbf{z}_r^{\mathcal{C}}$ and $\tilde{\mathbf{z}}_r^{\mathcal{A}}$, respectively.

For matrix factorization and neural collaborative filtering, we place the bias constraints on both of the users' and items' embedding vectors. For (u,i) pair, we randomly select a (user, item) pair (p,q), and then have the following equation,

$$\mathcal{L}_{\text{bias-MF}} = \mathcal{L}_{\text{bias-NCF}} = |cos(\mathbf{U}_u^{\mathcal{C}}, \mathbf{U}_p^{\mathcal{C}}) - \omega cos(\tilde{\mathbf{U}}_u^{\mathcal{A}}, \tilde{\mathbf{U}}_p^{\mathcal{A}})| + |cos(\mathbf{V}_u^{\mathcal{C}}, \mathbf{V}_q^{\mathcal{C}}) - \omega cos(\tilde{\mathbf{V}}_u^{\mathcal{A}}, \tilde{\mathbf{V}}_q^{\mathcal{A}})|. \tag{43}$$

6. Extracting bias information via pre-trained model

In the models described above, it is necessary to jointly learn the latent features of the bias information, the latent features of the preferences, as well as the relationship between them. This may lead to insufficient training of the model. Inspired by a typical machine learning model, a recommendation model is able to learn the properties of the users' feedback data. If there is bias information in the data, the user features learned by the model will contain some bias information. Therefore, we try to extract the latent features of the bias information from the model pre-trained using $S^{\mathcal{A}}$, because it naturally contains the bias information. In other words, we modify the initial information for learning the latent features of the bias. In the above model, we start training to learn the latent features of the bias information from the biased data, while now we extract the bias information from the embeddings pre-trained using $S^{\mathcal{A}}$.

Specifically, we first use $S^{\mathcal{A}}$ to train the model to get the latent features $z_u^{\mathcal{A}}$. According to the nature of $S^{\mathcal{A}}$, we consider that $z_u^{\mathcal{A}}$ includes the users' interest and bias information. Therefore, we can design a specific model to extract the latent features of the bias information from $z_u^{\mathcal{A}}$. We denote it as a bias separation module. Notice that all users share the same bias separation module, and the pre-trained latent features $z_u^{\mathcal{A}}$ are frozen during the joint training,

$$\tilde{z}_u^{\mathcal{A}} = f(z_u^{\mathcal{A}}), \tag{44}$$

where $f(\cdot)$ is an arbitrary function.

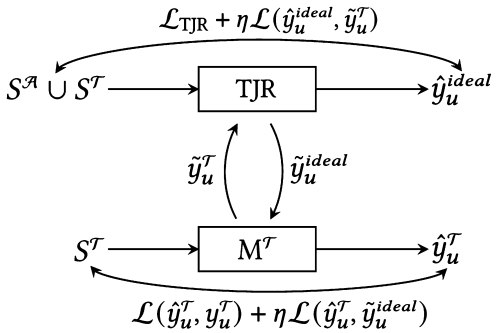


Fig. 7. Illustration of TJR-BiKD, which combines our TJR and bidirectional knowledge distillation. We introduce an auxiliary model denoted as $\text{M}^{\mathcal{T}}$ trained with $S^{\mathcal{T}}$ only, aiming to provide more unbiased pseudo labels denoted as $\tilde{y}_u^{\mathcal{T}}$ to assist the training of our TJR. At the same time, our TJR offers unbiased pseudo labels denoted as $\tilde{y}_u^{ideal}$ for $\text{M}^{\mathcal{T}}$ to facilitate the training of $\text{M}^{\mathcal{T}}$.

However, it is difficult and complicated to extract the bias latent features because of its non-obviousness. Fortunately, neural networks have a powerful function fitting capability. Hence, we employ a single-layer fully connected network with an activation function in our experiments. Finally, we also adopt the joint reconstruction loss function shown in Eq. (1), and denote this variant as TJR-PT. Notice that for matrix factorization and neural collaborative filtering, the bias separation module used for the users is different from that for the items.

7. Knowledge distillation for enhanced label information

To further stimulate the ability of our proposed model, we exploit knowledge distillation to extend our TJR. From the literature [4], it is known that the use of knowledge distillation can extract useful information from $S^{\mathcal{T}}$ to alleviate the bias problem. It is worth noting that the authors [4] propose the Bridge strategy, which exploits the pseudo labels generated by the model $\text{M}^{\mathcal{T}}$ obtained by training only using $S^{\mathcal{T}}$. That method is limited by the scale of $S^{\mathcal{T}}$ because the small size of $S^{\mathcal{T}}$ will lead to the low confidence for those pseudo labels. If we have more unbiased data, the performance of the model can become better. However, it is obviously unrealistic, because collecting unbiased data will hurt the user's experience and economic revenue. Hence, we turn to generate more reliable pseudo labels by an unbiased model, i.e., our TJR, and then exploit them to further mitigate the effect of bias.

Inspired by the Bridge strategy [4] and the work [18], we design a new approach to combine our TJR and knowledge distillation, and illustrate the schematic diagram in Fig. 7. Specifically, we introduce an auxiliary model denoted as $\text{M}^{\mathcal{T}}$, i.e., a model trained with $S^{\mathcal{T}}$ only. Similar to the Bridge strategy, it serves to provide more unbiased pseudo labels to TJR, aiming to improve its performance. Unlike the Bridge strategy, in our approach, we use the labels generated by our TJR to update $\text{M}^{\mathcal{T}}$ so that $\text{M}^{\mathcal{T}}$ can produce more pseudo labels with higher confidence. It is worth noting that in the Bridge strategy, $\text{M}^{\mathcal{T}}$ is not updated during the model training.

In short, our TJR uses the information of $\text{M}^{\mathcal{T}}$ to assist its own training, and $\text{M}^{\mathcal{T}}$ also uses the information of TJR to facilitate its own training. This process can be considered as bidirectional knowledge distillation. We thus denote our proposed approach as TJR-BiKD.

The information is exchanged between TJR and $\text{M}^{\mathcal{T}}$ in the same way as the Bridge strategy. We denote $\mathcal{D}$ as the entire dataset of all the (user, item) pairs, including both the observed and unobserved samples. For simplicity, we randomly sample an auxiliary set of (u, i) pairs denoted as S^a from $\mathcal{D}$, which has the same scale as $S^{\mathcal{A}}$. Notice that the sparsity of the datasets is high, so the picked (u, i) pairs are mostly unobserved samples. In addition, in our experiments, the (u, i) pairs used for TJR and $\text{M}^{\mathcal{T}}$ are the same for simplicity. For user u, the pseudo labels obtained by TJR are denoted as $\tilde{y}_u^{ideal}$, and that provided by $\text{M}^{\mathcal{T}}$ are denoted as $\tilde{y}_u^{\mathcal{T}}$. Finally, similar to the Bridge strategy, we have the overall loss function of our TJR-BiKD,

$$\mathcal{L}_{\text{TJR-BiKD}} = \mathcal{L}_{\text{TJR}} + \eta\mathcal{L}(\hat{y}_u^{ideal}, \tilde{y}_u^{\mathcal{T}}), \tag{45}$$

$$\mathcal{L}_{\text{M}^{\mathcal{T}}} = \mathcal{L}(\hat{y}_u^{\mathcal{T}}, y_u^{\mathcal{T}}) + \eta\mathcal{L}(\hat{y}_u^{\mathcal{T}}, \tilde{y}_u^{ideal}), \tag{46}$$

where η is a hyper-parameter, $y_u^{\mathcal{T}}$ are the observed labels from $S_u^{\mathcal{T}}$ ($S_u^{\mathcal{T}} \subseteq S^{\mathcal{T}}$).

In order to train our TJR-BiKD, we naturally think of using alternative training. In our experiments, we first train TJR by Eq. (45) and then train $M^{\mathcal{T}}$ through Eq. (46). More experimental details can be found in Section 8.6.

8. Experiments

In this section, we conduct extensive empirical studies on three real-world datasets aiming to investigate the following five research questions (RQs).

Table 2
Statistics of the datasets used in the experiments. Notice that P/N denotes the ratio between the numbers of positive feedback and negative feedback.

	Yahoo! R3		Coat Shopping		KuaiRec	
	#Feedback	P/N	#Feedback	P/N	#Feedback	P/N
$S^{\mathcal{A}}$	311,704	67.02%	6,960	37.69%	987,174	186.57%
$S^{\mathcal{T}}$	5,400	9.05%	464	28.89%	11,266	264.83%
S_{va}	5,400	9.31%	464	23.40%	11,266	269.62%
S_{te}	43,200	9.76%	3,712	21.94%	90,132	266.99%

RQ1: How does our TJR perform compared with the state-of-the-art methods?
RQ2: How do the reduced models perform, i.e., the ablation studies of our TJR?
RQ3: What is the performance of our TJR when using different transform functions?
RQ4: How do the hyper-parameters affect the performance of our TJR?
RQ5: How to alleviate the bias when TJR meets the knowledge distillation technology?
RQ6: What is the performance of our TJR when facing different degrees of bias problem?
RQ7: What is the performance of our TJR when meeting graph neural network?

8.1. Experimental setup

8.1.1. Datasets

For the studied problem of collaborative recommendation with biased data and unbiased data, there are only three public datasets available, i.e., Yahoo! R3 [82], Coat Shopping [2] and KuaiRec [83], which are thus included in our experiments. We show the statistics of the processed datasets used in the experiments in Table 2.

- **Yahoo! R3** [82]. It is a (user, song) rating data with a biased user subset and an unbiased random subset. The user subset contains over 300 K ratings from 15400 users to 1000 songs, which is biased because it is collected by normal interactions, i.e., users select and rate items as they wish when interacting with the recommender system. The random subset is an unbiased data, which is collected by the first 5400 users of those 15400 users, each of whom is asked to provide ratings to 10 songs randomly selected from those 1000 songs. For any user, the randomly selected strategy can ensure that each item has the same exposure opportunity. We follow a previous work [4], and regard the ratings larger than 3 as positive feedback. In our experiments, we regard the biased user subset as the auxiliary data ($S^{\mathcal{A}}$) and randomly split the unbiased random subset into three subsets: 10% as the target data ($S^{\mathcal{T}}$) for training, 10% as the validation data (S_{va}) to tune the hyper-parameters, and the rest 80% as the test data (S_{te}) for performance evaluation.
- **Coat Shopping** [2]. It is a (user, coat) rating data collected from 290 Amazon Mechanical Turk (AMT) workers on an inventory of 300 coats. Similar to Yahoo! R3, it contains a biased user subset and an unbiased random subset. Unlike Yahoo! R3, the random subset is collected by asking all the 290 workers to provide ratings to 16 coats randomly selected from the 300 coats. The preprocessing and split of the two subsets are the same as that on Yahoo! R3.
- **KuaiRec** [83]. It is a (user, video) interaction data with a sparse subset and a dense subset. The density of the sparse subset is 16.3%, which contains over ten million interactions from 7176 users to 10728 videos, and that of the dense subset is 99.9%, which contains over one million interactions from 1411 users to 3327 videos. The sparse subset can be considered as a biased data since it is missing at not random, and the dense subset can be considered as an unbiased data. Through data analysis and statistics, we find that KuaiRec does not directly conform to our studied problem, and the density of the dataset is much higher than that of other datasets. In our studied problem, the scale of the unbiased data is extremely small, since it is very expensive to obtain such a data in real applications. Therefore, we conduct random sampling from the raw data. In addition, in the experiments, we find the values of some metrics are small because the gap between the item range of the test set and that of the candidate set is very large. Finally, we keep the set of items common to both the sparse and dense datasets, i.e., 3327 items in total. After data processing, we get a biased user subset and an unbiased random subset from the sparse subset and the dense subset, respectively. The preprocessing and split of the two subsets are the same as that on Yahoo! R3.

8.1.2. Evaluation metrics

For evaluation of collaborative recommendation, we adopt two widely used metrics in the community of recommender systems, including the area under the ROC curve (AUC), normalized discounted cumulative gain (NDCG@K), precision (P@K) and recall (R@K), where AUC is the main metric. In our experiments, K is set to 50.

- **AUC** is a performance metric to measure the classification effectiveness of the model. Its calculation formula is as follows,

$$AUC = \frac{\sum_{(i,j)\in\Omega}^{L_p} Rank_{ij} - \binom{L_p}{2}}{(L_p(L - L_p))} \tag{47}$$

where Ω denotes the set of all positive feedback in the test set (or the validation set in the process of selecting parameters), L_p denotes the total number of positive feedback in Ω, $Rank_{ij}$ is the position of a positive feedback (i, j) among all the feedback according to the descending order of the prediction results, and L is the total number of feedback in the test set (or validation set for parameter selection), including the positive and negative feedback.

- **NDCG@K** is an evaluation metric widely used to measure the accuracy of the ranking list. Its calculation formula is as follows,

$$NDCG@K = \frac{\sum_{u\in\mathcal{U}^{te}} NDCG_u@K}{|\mathcal{U}^{te}|} \tag{48}$$

$$NDCG_u@K = \frac{DCG_u@K}{IDCG_u} \tag{49}$$

$$DCG_u@K = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)} \tag{50}$$

where $\mathcal{U}^{te}$ denotes the set of users in the test set (or the validation set in the process of selecting parameters), $NDCG_u@K$ denotes the normalized discounted cumulative gain of user u, $DCG_u@K$ is the discounted cumulative gain of user u, $IDCG_u$ is the discounted cumulative gain of user u in the ideal case (obtained by calculating $DCG_u@K$ based on the returned results which are reordered according to the true correlation), K is the length of the recommendation list, and rel_i is the relevance of the recommended items. In our experiments, K is set to 50.

- **P@K** is also an evaluation metric widely used in recommender systems to measure the quality of the recommended lists. Its calculation formula is as follows,

$$P@K = \frac{\sum_{u\in\mathcal{U}^{te}} |R(u) \cap T(u)|}{\sum_{u\in\mathcal{U}^{te}} |R(u)|} \tag{51}$$

where $R(u)$ is the recommended list generated by the model for user u, and $T(u)$ is the recommended list formed by the feedback of the user u in the test set (or the validation set in the process of selecting parameters).

- **R@K** has the same function as P@K. Its calculation formula is as follows,

$$R@K = \frac{\sum_{u\in\mathcal{U}^{te}} |R(u) \cap T(u)|}{\sum_{u\in\mathcal{U}^{te}} |T(u)|} \tag{52}$$

8.1.3. Baselines

We use four representative methods as the backbone models (BMs), i.e., variational autoencoders (VAE), matrix factorization (MF), neural collaborative filtering (NCF) and graph convolution network (GCN). Notice that for MF and NCF, we use $S^{\mathcal{A}} \cup S^{\mathcal{T}}$ instead of $S^{\mathcal{A}}$ for learning the bias information in order to avoid the situation when a sampled triple is included in $S^{\mathcal{A}} \cup S^{\mathcal{T}}$ but not in $S^{\mathcal{A}}$, because the latent features are learned by randomly sampling one single (user, item, rating) triple rather than all the rating records of a user in VAE. Moreover, for the unobserved (user, item) pairs, when VAE is used as the backbone model, we treat them as negative feedback, and when MF and NCF are used as the backbone models, we treat them as missing values.

Firstly, we adopt several traditional classical models, i.e., PureSVD [84], PLRec [85] and AutoRec [36]. Our TJR is closely related with the backbone models, and thus use BM($S^{\mathcal{A}}$), BM($S^{\mathcal{T}}$) and BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$) to denote the data sources we used, i.e., training only with $S^{\mathcal{A}}$, training only with $S^{\mathcal{T}}$, and training with both $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$, respectively, where BM is a specific backbone model. We also compare the representative methods proposed in recent years, including inverse propensity score (IPS) [2] and CausE [9], as well as the Bridge and Weight strategies [4]. For IPS, since we use implicit feedback instead of ratings, we estimate the propensity score w.r.t. an item i via the naive Bayes estimator that considers the exposure of the item [2,69], $P_i = \frac{P(\mathcal{I}=i|O=1)P(O=1)}{P(\mathcal{I}=i)}$, where $P(\mathcal{I} = i|O = 1)$ denotes the ratio of positive feedback w.r.t. item i given the observed feedback, $P(O = 1)$ denotes the ratio of the observed feedback, and $P(\mathcal{I} = i)$ represents the ratio of the feedback w.r.t. item i in an unbiased set. In addition, a model called AutoDebias has recently been proposed [10], which is also included as one of the baselines. However, AutoDebias only uses MF and NCF as the backbone models, and it is difficult to be extended to VAE. Hence, we do not compare with AutoDebias using VAE as the backbone model.

8.1.4. Reproducibility

We follow the public source code for the implementation of the traditional models of PureSVD and PLRec,[1] as well as the recent work AutoRec[2] For AutoDebias, we implement it via PyTorch 1.1 using the MSE loss and parameter configurations following the original paper [10]. For all the other methods, we implement them via TensorFlow 1.2 using the cross-entropy loss and batch training. We conduct a grid search to tune the hyper-parameters of all the methods by checking the AUC performance on the validation data S_{va}. Specifically, we choose the embedding size $rank \in \{50, 100, 200\}$, the hyper-parameter on the regularization term $\lambda \in \{1e^{-5}, 1e^{-4}, \ldots, 1\}$, and the tradeoff hyper-parameters $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$, $\gamma \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$ and $\omega \in \{0.01, 0.05, 0.1, 0.5, 1.0\}$. For the methods using VAE as the backbone model, we use RMSProp as the optimizer (the learning rate is fixed as 0.0001), set the weight on the KL divergence as 0.2, fix the iteration number as 300, and choose the dropout rate from $\{0.2, 0.3, 0.4, 0.5\}$. For the methods using MF (except AutoDebias-MF) and NCF (except AutoDebias-NCF) as the backbone models, we use Adam as the optimizer (the learning rate is fixed as 0.001), and set the iteration number as 100. Notice that we adopt an early stopping strategy with the patience set to 5 times for the methods using NCF as the backbone model. In addition, due to the relatively small size of the validation data, the selection of the optimal values is not always very stable. Therefore, we run ten times, and then select the one with the best average performance. Notice that the source code and scripts to reproduce all the results are publicly available at http://csse.szu.edu.cn/staff/panwk/publications/TJR/.

8.2. RQ1: performance comparison

We report the main results in Table 3 and Table 4, from which we can have the following observations.

Firstly, overall, our TJR outperforms all the baselines on all datasets, which clearly shows the advantage of our transfer learning solution in jointly modeling the biased and unbiased data. Compared with the traditional models such as PureSVD, PLRec and AutoRec, our proposed model has a clear advantage, and achieves better recommendation performance than the backbone model at the same time. This again shows the better adaptability of the model. Our TJR can effectively reduce the impact of the bias, and improve the interactivity between users and the recommender system. Notice that we take AUC as the main metric and use it in the process of tuning the hyper-parameters.

Secondly, the recommendation performance of our TJR and its variants have a certain relationship with the adopted backbone model and the scale of the dataset. When VAE or MF is used as the backbone model, the variant TJR-PT performs worse than TJR on Yahoo! R3 and KuaiRec in general, but is better on Coat Shopping. We think that this is likely related to the scale of the datasets. The model complexity of TJR-PT is lower than that of TJR, so in general its expressive ability is weaker than TJR. But when the dataset is small, the more complicated the model is, the easier it is to be overfitted, resulting in poor performance. For another variant TJR++, it consistently outperforms TJR across all datasets, which shows the effectiveness of the bias regularizer described in Section 5. When using NCF as the backbone model on Coat Shopping, the performance of TJR++ is slightly weaker than our TJR, which is related to the nature of the backbone model itself and the small size of the dataset.

Thirdly, the loss function of our TJR is similar to that of the Weight strategy [4], but the performance of our model is better, which further illustrates the validity of our TJR. Moreover, our TJR is flexible and could be easily extended to different backbone models such as MF, VAE and NCF, and still maintains relatively good performance.

Finally, the performance of both CausE and the Bridge strategy is limited by the pre-trained model obtained by $S^{\mathcal{T}}$, and is also limited by the scale of $S^{\mathcal{T}}$. Notice that our TJR jointly reconstructs $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$, which alleviates the problem of small size of $S^{\mathcal{T}}$ to a certain extent. In addition, in the experiments with matrix factorization as the backbone model, the performance of our TJR is worse than AutoDebias on NDCG@50 on Yahoo! R3. We analyze the prediction of AutoDebias and our TJR-MF, and find that the hits of the two models are not very different and most users' hits are 0s as shown in Fig. 8, which is caused by the nature of the dataset. Specifically, in the test set, the number of positive feedback of each user is rather small. This situation is very different from most recommendation settings on public biased datasets in previous works. Therefore, for bias reduction, using NDCG as an evaluation metric may not be the best, which may only be used as an auxiliary metric.

8.3. RQ2: ablation studies

In order to gain some deep understanding of our TJR, we use VAE as the backbone model and conduct some ablation studies by removing some components from the framework of our TJR, and report the results in Fig. 9. Firstly, we cut off the sharing path of $G(\cdot)$, denoted as "-Share". We can see that sharing $G(\cdot)$ can improve the performance of our model because that could force the users' latent feature space and the bias's latent feature space to be as close as possible, which is beneficial for our TJR to alleviate the bias problem from the same angle. In addition, we remove the branch that extracts latent features of bias information so that our TJR degenerates into VAE($S^{\mathcal{A}} \cup S^{\mathcal{T}}$), denoted as "-Sub". We can see that the recommendation performance of our TJR without the bias branch degrades significantly, which shows the usefulness of the designed bias reduction component.

[1] https://github.com/wuga214/NCE_Projected_LRec.
[2] https://github.com/dgliu/SIGIR20_KDCRec.

Table 3
Recommendation performance on Yahoo! R3, Coat Shopping and KuaiRec where the best results are marked in bold and the second best results are marked in underline. Notice that we follow the original paper and only use MF and NCF as the backbone models in AutoDebias, since it is difficult to support other backbone models such as VAE.

Yahoo! R3												
Model	VAE				MF				NCF			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
BM($S^{\mathcal{A}}$)	0.7666	0.1009	0.0083	0.2715	0.7329	0.0382	0.0045	0.1417	0.7245	0.0279	0.0033	0.1053
BM($S^{\mathcal{T}}$)	0.5770	0.0258	0.0029	0.0873	0.5684	0.0304	0.0032	0.0984	0.6050	0.0275	0.0030	0.0929
BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	0.7709	0.1014	0.0083	0.2713	0.7409	0.0439	0.0048	0.1516	0.7268	0.0327	0.0036	0.1156
IPS	0.7470	0.0809	0.0073	0.2341	0.7346	0.0426	0.0046	0.1441	0.7273	0.0304	0.0034	0.1064
CausE	0.7673	0.1013	0.0083	0.2724	0.7285	0.0445	0.0045	0.1421	0.7283	0.0284	0.0032	0.1050
Bridge	0.7711	0.1007	0.0082	0.2702	0.7524	0.0615	0.0061	0.1924	0.7367	0.0439	0.0044	0.1405
Weight	0.7723	0.0998	0.0081	0.2668	0.7465	0.0494	0.0052	0.1664	0.7380	0.0383	0.0043	0.1359
AutoDebias	-	-	-	-	0.7472	**0.0870**	**0.0069**	**0.2277**	0.7340	**0.0576**	**0.0053**	**0.1711**
TJR-PT	0.7804	0.1011	0.0082	0.2730	0.7442	0.0612	0.0057	0.1797	0.7363	0.0363	0.0039	0.1250
TJR	0.7804	0.1023	0.0081	0.2718	0.7696	0.0705	0.0064	0.2096	0.7420	0.0454	0.0048	0.1524
TJR++	**0.7838**	**0.1054**	**0.0084**	**0.2787**	**0.7742**	0.0747	**0.0069**	0.2267	**0.7429**	0.0489	0.0049	0.1584

Coat Shopping												
Model	VAE				MF				NCF			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
BM($S^{\mathcal{A}}$)	0.6210	0.0921	0.0156	0.2467	0.7606	0.0990	0.0155	0.2589	0.7507	0.0976	0.0166	**0.2846**
BM($S^{\mathcal{T}}$)	0.5413	0.0807	0.0123	0.1968	0.5231	0.0578	0.0103	0.1686	0.5840	0.0781	0.0120	0.1924
BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	0.6269	0.0952	0.0161	0.2528	0.7631	0.1016	0.0150	0.2644	0.7508	0.0969	0.0160	0.2684
IPS	0.5757	0.0856	0.0141	0.2171	0.7636	0.0965	0.0153	0.2541	0.7337	0.0902	0.0139	0.2373
CausE	0.6210	0.0922	0.0156	0.2467	0.7611	0.0985	0.0153	0.2557	0.7516	0.0961	0.0156	0.2646
Bridge	0.6210	0.0932	0.0156	0.2467	0.7653	0.1005	0.0156	0.2625	0.7522	0.0969	**0.0169**	0.2750
Weight	0.6245	0.0949	0.0157	0.2498	0.7636	0.1012	0.0158	0.2681	0.7509	0.0946	0.0153	0.2580
AutoDebias	-	-	-	-	0.6965	0.0999	0.0161	0.2706	0.7463	0.0944	0.0155	0.2512
TJR-PT	**0.7647**	0.1265	0.0177	0.3296	**0.7689**	0.0966	0.0153	0.2510	0.7519	0.0918	0.0152	0.2528
TJR	0.7603	0.1239	0.0177	0.3304	0.7646	**0.1027**	**0.0162**	**0.2789**	**0.7537**	**0.0995**	0.0163	0.2755
TJR++	0.7621	**0.1272**	**0.0179**	**0.3410**	0.7666	0.1015	0.0160	0.2771	0.7523	0.0988	0.0161	0.2729

KuaiRec												
Model	VAE				MF				NCF			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
BM($S^{\mathcal{A}}$)	0.7463	0.0186	0.0178	0.0192	0.8645	0.0189	0.0178	0.0193	0.8508	0.0191	0.0182	0.0197
BM($S^{\mathcal{T}}$)	0.7041	0.0180	0.0170	0.0182	0.7504	0.0142	0.0156	0.0168	0.7707	0.0178	0.0170	0.0184
BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	0.7481	**0.0187**	**0.0179**	**0.0194**	0.8682	0.0185	0.0174	0.0189	0.8475	**0.0197**	**0.0185**	**0.0200**
IPS	0.7137	0.0173	0.0165	0.0176	0.8671	0.0181	0.0174	0.0188	0.8600	0.0194	0.0180	0.0196
CausE	0.7464	0.0185	0.0177	0.0191	0.8657	**0.0193**	**0.0183**	**0.0197**	0.8560	0.0191	0.0182	0.0197
Bridge	0.7463	0.0185	0.0177	0.0191	0.8648	0.0189	0.0181	0.0195	0.8531	0.0187	0.0177	0.0191
Weight	0.7482	0.0182	0.0173	0.0187	0.8684	0.0192	0.0180	0.0195	0.8574	0.0194	0.0182	0.0197
AutoDebias	-	-	-	-	0.8530	0.0189	0.0181	0.0195	0.8612	0.0190	0.0181	0.0195
TJR-PT	0.7542	0.0181	0.0173	0.0186	**0.8690**	0.0189	0.0181	0.0196	**0.8637**	0.0195	0.0181	0.0196
TJR	0.7620	0.0184	0.0177	0.0191	**0.8690**	0.0190	0.0182	0.0196	0.8624	0.0193	0.0180	0.0194
TJR++	**0.7637**	0.0184	0.0175	0.0191	0.8685	0.0185	0.0173	0.0187	0.8624	0.0194	0.0181	0.0196

8.4. RQ3: impact of the transform function

In this subsection, we again use VAE as the backbone model and further study the impact of the transform function used to extract the bias information. Specifically, we report the recommendation performance of our TJR with different transform functions including sigmoid, tanh, relu and linear, which are shown in Fig. 10. We can see that using a nonlinear activation function improves the performance of our TJR more significantly than a linear one. Among them, using the sigmoid function performs the best, followed by using tanh. The reason is related to the property of the bias itself. Notice that the performance of a recommendation model largely depends on whether it learns the users' interests or not, and the impact of bias on the performance is not very significant. In other words, the value of the bias features could not be too large to ensure the overall performance of our TJR, thus we limit the value via the sigmoid function.

Table 4
Recommendation performance on Yahoo! R3, Coat Shopping and KuaiRec where the best results are marked in bold and the second best results are marked in underline.

Model	Yahoo! R3				Coat Shopping				KuaiRec			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
PureSVD	0.6750	0.0702	0.0061	0.1981	0.5908	0.0912	0.0139	0.2407	0.6117	0.0160	0.0160	0.0173
PLRec	0.6865	0.0723	0.0064	0.2073	0.7069	0.1179	**0.0181**	0.3158	0.6163	0.0160	0.0161	0.0173
AutoRec	0.7123	0.0598	0.0054	0.1714	0.7369	0.0945	0.0157	0.2585	0.7690	0.0184	0.0176	0.0190
VAE	0.7709	0.1014	**0.0083**	0.2713	0.6269	0.0952	0.0161	0.2528	0.7481	0.0187	0.0179	0.0194
MF	0.7409	0.0439	0.0048	0.1516	0.7631	0.1016	0.0150	0.2644	0.8682	0.0185	0.0174	0.0189
NCF	0.7268	0.0327	0.0036	0.1156	0.7508	0.0969	0.0160	0.2684	0.8475	**0.0197**	**0.0185**	**0.0200**
TJR-VAE	**0.7804**	**0.1023**	0.0081	**0.2718**	0.7603	**0.1239**	0.0177	**0.3304**	0.7620	0.0184	0.0177	0.0191
TJR-MF	0.7696	0.0705	0.0064	0.2096	**0.7646**	0.1027	0.0162	0.2789	**0.8690**	0.0190	0.0182	0.0196
TJR-NCF	0.7420	0.0454	0.0048	0.1524	0.7537	0.0995	0.0163	0.2755	0.8624	0.0193	0.0180	0.0194

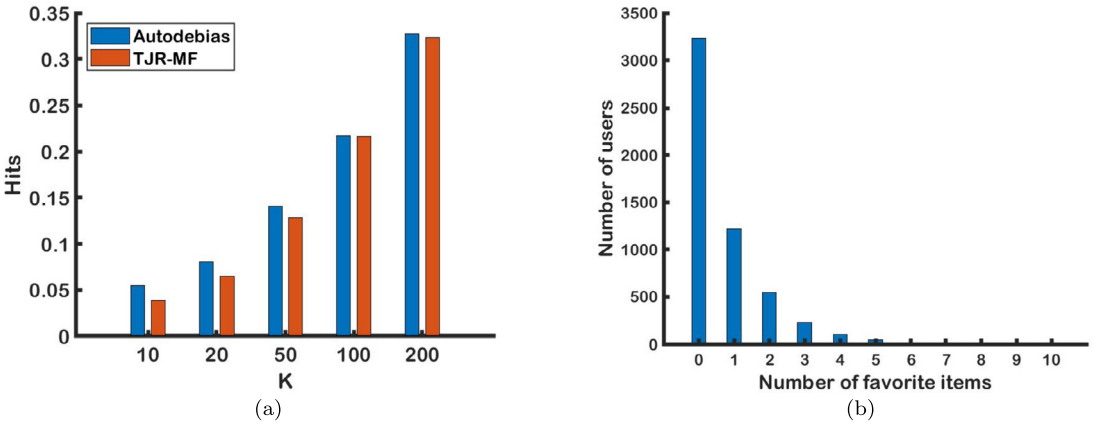


Fig. 8. The percentage of hits of AutoDebias and our TJR-MF (a), and the distribution of the number of users over the number of favorite items in the test set (b), for Yahoo! R3.

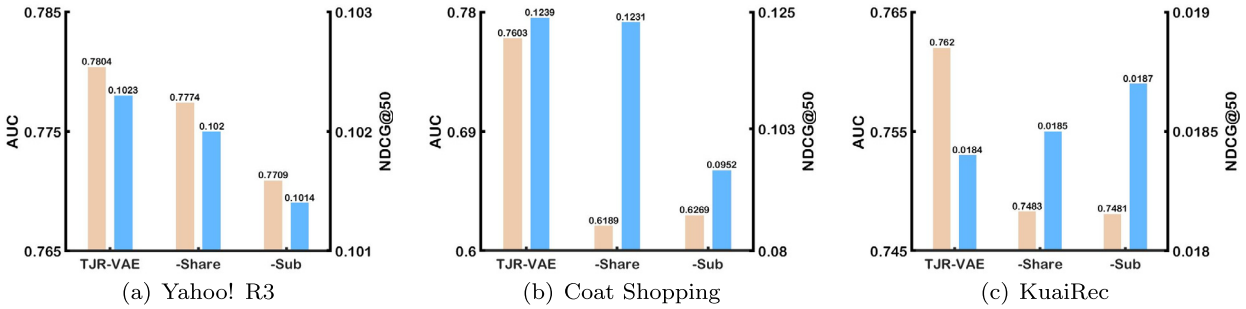
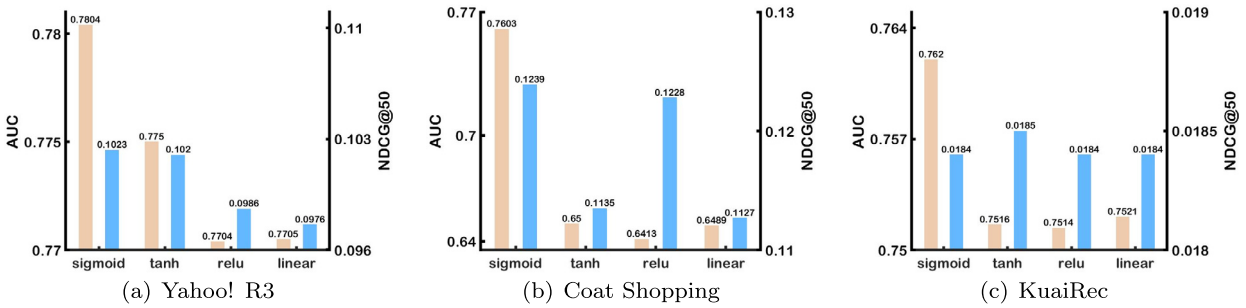


Fig. 9. Recommendation performance of our TJR by removing different components (i.e., "-Share" and "-Sub"). Notice that "-Share" and "-Sub" denotes removing the sharing path of $G(\cdot)$ and removing the branch used to extract the bias information, respectively.

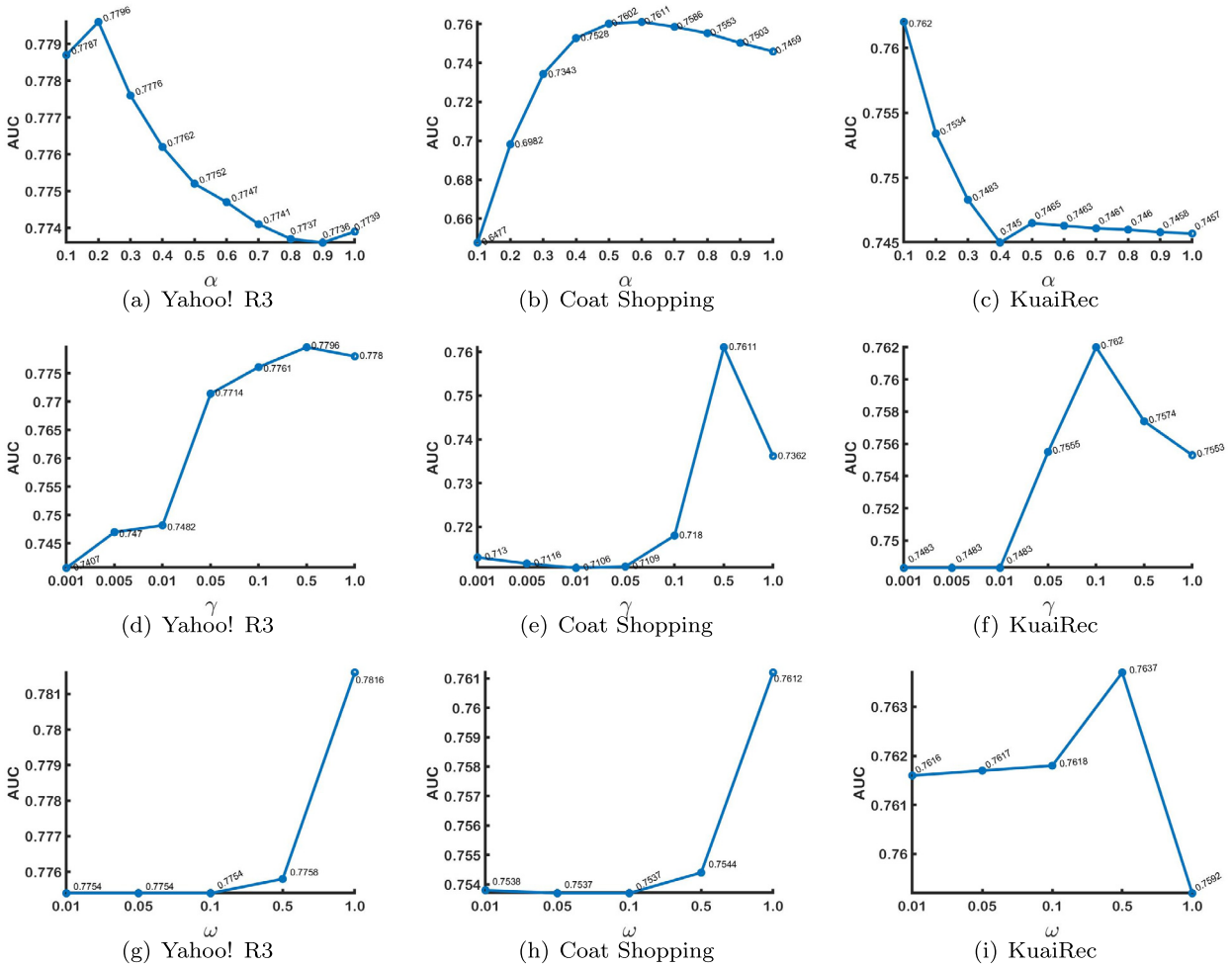
8.5. RQ4: impact of the hyper-parameters

In this subsection, we use VAE as the backbone model and study the impact of the hyper-parameters α and γ in our TJR, and show the results in Fig. 11 (a-f). Specifically, we first fix γ as the optimal value, and report the recommendation performance with different values of $\alpha \in \{0.1, 0.2, ..., 1.0\}$. We can see that the best values on Yahoo! R3, Coat Shopping and Kuairec are 0.2, 0.6 and 0.1, respectively. The reason is that the ratio $|S^{\mathcal{A}}|/|S^{\mathcal{T}}|$ of Yahoo! R3 and Kuairec are larger than that of Coat Shopping. In order to reduce the effect of the loss on $S^{\mathcal{A}}$, α shall be smaller. Similarly, we fix α as the optimal value, and report the recommendation performance with different values of $\gamma \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$. We can see that the best values on Yahoo! R3 and Coat Shopping are both 0.5, and that on KuaiRec is 0.1. For user u, $\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}})$ is used to learn the biased features $z_u^{\mathcal{C}}$ better and the role of γ is to control its proportion. If γ is too small, the users' latent features may not be fully trained, and if γ is too large, the effect of the bias branch in TJR will be weakened.



(a) Yahoo! R3 (b) Coat Shopping (c) KuaiRec

Fig. 10. Recommendation performance of our TJR by using different transform functions (i.e., sigmoid, tanh, relu and linear).



(a) Yahoo! R3 (b) Coat Shopping (c) KuaiRec

(d) Yahoo! R3 (e) Coat Shopping (f) KuaiRec

(g) Yahoo! R3 (h) Coat Shopping (i) KuaiRec

Fig. 11. Recommendation performance of our TJR on Yahoo! R3 and Coat Shopping with different values of $\alpha \in \{0.1, 0.2, ..., 1.0\}$, $\gamma \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$ and $\omega \in \{0.01, 0.05, 0.1, 0.5, 1.0\}$, which are shown in (a-b), (c-d) and (e-f), respectively.

In addition, we introduce the hyper-parameter ω in the variant TJR++. And in order to analyze its impact on TJR++, similarly, we use the VAE as backbone model in the experiments. We fix α and γ with the optimal values, and report the recommendation performance with different values of $\omega \in \{0.01, 0.05, 0.1, 0.5, 1.0\}$. The experimental results are shown in Fig. 11 (g-i), from which we can observe the best values on Yahoo! R3 and Coat Shopping are both 1.0, and that on KuaiRec is 0.5. When the value of ω is small, the bias constraint does not work, which makes the performance of the model worse. From the perspective of Eq. (40), the value of ω can affect the learning of the user's latent features, thereby affecting the performance of the model.

Table 5
Recommendation performance on Yahoo! R3 and KuaiRec where the best results are marked in bold.

Yahoo! R3												
Model	VAE				MF				NCF			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
TJR	0.7804	**0.1023**	**0.0081**	**0.2718**	0.7696	0.0705	0.0064	0.2096	0.7420	0.0454	0.0048	0.1524
TJR-BiKD	**0.7808**	0.1007	**0.0081**	0.2685	**0.7753**	**0.0765**	**0.0068**	**0.2214**	**0.7447**	**0.0513**	**0.0051**	**0.1646**
KuaiRec												
Model	VAE				MF				NCF			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
TJR	0.7620	**0.0184**	**0.0177**	**0.0191**	**0.8690**	**0.0190**	**0.0182**	**0.0196**	**0.8624**	0.0193	0.0180	0.0194
TJR-BiKD	**0.7712**	**0.0184**	0.0174	0.0190	**0.8690**	0.0188	0.0180	0.0194	0.8618	**0.0195**	**0.0184**	**0.0199**

8.6. RQ5: extension of TJR

In Section 7, we use knowledge distillation to improve our TJR and propose a new model named TJR-BiKD. In this subsection, we report the experimental details and results. Specifically, our TJR-BiKD introduces a hyperparameter η, for which we choose its value from $\{0.00001, 0.0001, 0.001, 0.01, 0.1\}$ in the experiments. Before training the model, $M^{\mathcal{T}}$ is initialized by using the optimal model parameters trained by $S^{\mathcal{T}}$ only, rather than initialized randomly; and for TJR, it is initialized randomly. However, this may cause a problem that the random initialization of TJR makes its generated pseudo labels with a low confidence at the beginning. In order to solve this problem, we introduce a threshold value. The model $M^{\mathcal{T}}$ starts training only when the AUC on the validation set of TJR is larger than the threshold. Finally, we conduct experiments on Yahoo! R3 and KuaiRec, and report results in Table 5, from which we can see that our TJR-BiKD outperforms TJR in most cases, showing the effectiveness of integration of bidirectional knowledge distillation into our TJR. Notice that we take AUC as the main metric and use it in the process of tuning the hyper-parameters. When VAE or MF is used as the backbone model, the performance of the variant TJR-BiKD is worse than that of TJR on KuaiRec, but their difference is only some thousandths.

8.7. RQ6: impact of the degrees of bias problem

In this subsection, we study how our TJR performs with different levels of biases. In this context, different degrees of bias refer to a data that contains different degrees of bias properties (e.g. exposure bias and popularity bias). To accomplish this goal, we use MovieLens 100 K to create a semi-synthetic data following a previous work [69]. The data synthesis method [69] can not only get the users' feedback in a biased environment, but also the real correlation between users and items, i.e., both biased and unbiased data can be obtained to satisfy the studied problem setting. In addition, this data generation method introduces a hyper-parameter p, which controls the distribution of the biased data. When the value of p is larger, the exposure bias problem is more severe. In our experiments, we choose p in the range of $\{0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$, and adopt representative methods, i.e., BM($S^{\mathcal{A}}$), BM($S^{\mathcal{T}}$), BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$), IPS and Weight. Among them, BM($S^{\mathcal{A}}$), BM($S^{\mathcal{T}}$) and BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$) are used as the most basic methods and IPS as a classic method. The performance of the Weight strategy is the best among most baselines as can be seen from Table 3, which is close to our TJR. Similarly, we use VAE as the backbone model and continue the experimental settings mentioned in Section 8.1 for parameters selection. The final results are shown in Fig. 12.

Firstly, it can be seen from Fig. 12(a) that different values of p not only change the degree of the bias, but also the sparsity of the data. The larger the values of p, the sparser the data. Therefore, for fair comparison, we control the ratio of the biased and the unbiased data around 1/20 when generating the semi-synthetic data. Secondly, from Fig. 12(b) and Fig. 12(c), we can obtain: (1) as the value of p increases, the data becomes more sparse, and the performance of all models (except IPS whose performance is poor) decreases; (2) as a whole, our TJR performs better than all the other models, and achieves superiority at different degrees of bias; (3) when the value of p is equal to 0.4 or 0.5, the AUC of the Weight strategy and our TJR do not perform well, indicating that the benefit brought by debiasing is not obvious, where the main reason may be that the data is too sparse, and the factor that constrains the model performance at this time is the sparsity rather than the bias.

8.8. RQ7: combining TJR and graph neural network

In this section, we study how our TJR performs when we use graph neural network as the backbone model. From the experimental results and analysis in Section 8.2, we find that the Weight strategy has the best performance among all the baselines, which is similar to our TJR. Therefore, we adopt BM($S^{\mathcal{A}}$), BM($S^{\mathcal{T}}$), BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$) and the Weight strategy as the

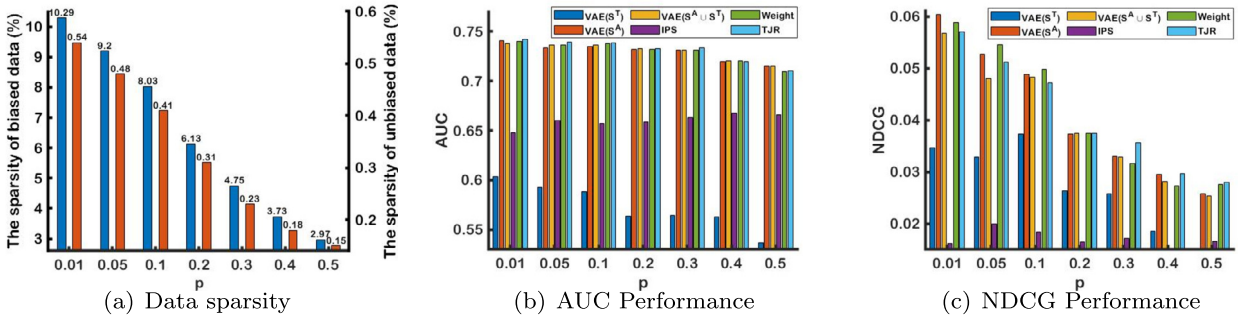


Fig. 12. Recommendation performance of our TJR and some representative baselines on the semi-synthetic datasets with different values of p. Notice that the value of p can be used to adjust the distribution of the exposure parameter and thus control different degrees of bias. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 6
Recommendation performance on Yahoo! R3 and KuaiRec, where the best results are marked in bold and the second best results are marked in underline.

Model	Yahoo! R3				KuaiRec			
	AUC	NDCG@50	P@50	R@50	AUC	NDCG@50	P@50	R@50
LightGCN($S^{\mathcal{A}}$)	0.7430	0.0884	<u>0.0076</u>	<u>0.2511</u>	0.7443	**0.0194**	**0.0179**	**0.0195**
LightGCN($S^{\mathcal{T}}$)	0.6502	0.0536	0.0051	0.1676	0.7445	0.0187	0.0171	0.0183
LightGCN($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	<u>0.7440</u>	0.0883	<u>0.0076</u>	0.2508	0.7448	<u>0.0192</u>	<u>0.0178</u>	<u>0.0194</u>
Weight-LightGCN	0.07412	<u>0.0896</u>	<u>0.0076</u>	0.2509	<u>0.7472</u>	0.0191	0.0176	0.0192
TJR-LightGCN	**0.7444**	**0.0897**	**0.0077**	**0.2542**	**0.7487**	0.0190	0.0175	0.0191

baselines in our experiments. Following the original work [38] and the source code,[3] we fix the number of graph layers as 3 and the embedding size as 64, use Adam as the optimizer (the learning rate is fixed as 0.001 for Yahoo! R3, and as 0.0001 for KuaiRec), and adopt an early stopping strategy with the patience set to 20 times for all the methods. The settings of other hyper-parameters are the same as those described in Section 8.1.4. Similarly, we conduct a grid search to tune the hyper-parameters for all the methods by checking the AUC performance on the validation data S_{va}. Finally, we report the experiments results in Table 6. Notice that in the preliminary experiments, we find that LightGCN performed poorly on Coat Shopping, probably because of the small size of the dataset. Therefore, the results on this dataset are not included. From Table 6, we can see that our TJR outperforms all the baselines in most cases, showing the effectiveness of our TJR with GCN as the backbone model. Notice that we take AUC as the main metric and use it in the process of tuning the hyper-parameters.

9. Conclusions and future work

In this paper, we study an emerging and important problem called collaborative recommendation with a larger biased data and a small unbiased data. As a response, we view this problem from a transfer learning perspective, which is a subfield of artificial intelligence. We then propose a novel transfer learning solution to achieve knowledge transfer between the two different data, aiming to reduce the bias, improve the recommendation performance and enhance the intelligence and interactivity of a recommender system. Specifically, we design (i) an end-to-end transfer learning framework, including two different but related models to extract latent features that represent users' preferences and bias information, respectively, and (ii) a bias reduction component and a shared prediction model, optimized by a joint reconstruction loss, which is thus called transfer via joint reconstruction (TJR). In addition, we also propose three variants of our TJR, i.e., TJR++, TJR-PT and TJR-BiKD, aiming to alleviate the impact caused by the small scale of the unbiased data and further improve the performance of the model. Finally, we conduct extensive empirical studies on three public datasets, and find that our TJR performs significantly better than some very competitive baseline methods in most cases.

For our proposed methods, TJR is a general framework that can be adapted to different backbone models, but its training complexity is relatively high, for which we will further optimize the training process and reduce the computational complexity. The performance of TJR++ is very promising in many cases, which shows the effectiveness of the constraint on bias, and we will study the properties of bias and design more constraints or regularization. TJR-PT performs better on a smaller dataset, which may be because the model uses a pre-training model and some bias information is not learned in the pre-training phase, and we will construct a new training method to fine-tune the pre-trained model. The performance of TJR-BiKD is more obvious on a larger dataset, which illustrates the effectiveness of bidirectional knowledge distillation

[3] https://github.com/kuandeng/LightGCN.

between TJR and the pseudo label generation model, but the credibility of the pseudo labels is not high. We will design a new distillation method and generate more reliable unbiased information.

For future works, we are interested in further generalizing our transfer learning solution to include more information such as temporal dynamics and item descriptions, and improving the interpretation of the delivered recommendations, as well as applying transfer learning to more bias reduction problems and recommendation tasks such as sequential recommendation and large language model (LLM)-based recommendation [71,86,87].

- We will extend TJR to more backbone models, more recommendation tasks and more application scenarios in industry such as click-through rate (CTR) prediction.
- We will adapt transfer learning to more bias reduction tasks. As shown in Section 4.5, transfer learning is mainly used to mitigate biases with three methods, i.e., instance-based, parameters-based and feature-based. Taking conversational recommender systems as an example, we can design a method based on inverse propensity scores by first calculating the bias score of each sample through some heuristics [71] and then fine-tune the model based on the bias scores.
- Bias can be divided into benign and harmful ones [88,89], and it is difficult to identify them. However, we can design a bias separation method based on transfer learning to reward a benign bias and constrain a harmful one. For example, we can build an instance-based method. Specifically, we first use a conventional debiasing method to obtain a base model, and then rebuild a new model, and compare it with the base model during the training process aiming to reduce the weight of harmful samples and increase the weight of favorable ones.
- We will introduce more additional data to alleviate bias, such as user attributes and item attributes. Different users have different bias effects, and similarly, different items have different bias effects. If the user attributes or item attributes are changed, the probability of users' clicking on items may decrease or increase. Therefore, we can first construct some pseudo samples, and then design a method based on transfer learning to alleviate the bias.
- Inspired by the fact that large language models have shown excellent performance in various NLP tasks, we are interested in using transfer learning to take advantage of large language models [87]. For example, we may employ a large language model to generate some virtual data, and let the model infer some items that may be of interest by providing a prompt of items that a user is interested in. Moreover, we may use a large language model to extract richer features for item attributes and user attributes, and integrate them into recommendation models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

We thank the support of National Natural Science Foundation of China (Nos. 62172283, 61836005 and 62272315) and National Key Research and Development Program of China (No. 2018AAA0101100).

References

[1] Z. Lin, D. Liu, W. Pan, Z. Ming, Transfer learning in collaborative recommendation for bias reduction, in: Proceedings of the 15th ACM Conference on Recommender Systems, 2021, pp. 736–740.
[2] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, T. Joachims, Recommendations as treatments: debiasing learning and evaluation, in: Proceedings of the 33rd International Conference on Machine Learning, 2016, pp. 1670–1679.
[3] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, M. Najork, Position bias estimation for unbiased learning to rank in personal search, in: Proceedings of the 11th ACM International Conference on Web Search and Data Mining, 2018, pp. 610–618.
[4] D. Liu, P. Cheng, Z. Dong, X. He, W. Pan, Z. Ming, A general knowledge distillation framework for counterfactual recommendation via uniform data, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 831–840.
[5] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, E. Chi, Recommending what video to watch next: a multitask ranking system, in: Proceedings of the 13th ACM Conference on Recommender Systems, 2019, pp. 43–51.
[6] H. Guo, J. Yu, Q. Liu, R. Tang, Y. Zhang, Pal: a position-bias aware learning framework for ctr prediction in live recommender systems, in: Proceedings of the 13th ACM Conference on Recommender Systems, 2019, pp. 452–456.
[7] D.C. Liu, S. Rogers, R. Shiau, D. Kislyuk, K.C. Ma, Z. Zhong, J. Liu, Y. Jing, Related pins at pinterest: the evolution of a real-world recommender system, in: Proceedings of the 26th International Conference on World Wide Web Companion, 2017, pp. 583–592.
[8] J. Yu, H. Zhu, C.-Y. Chang, X. Feng, B. Yuan, X. He, Z. Dong, Influence function for unbiased recommendation, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 1929–1932.
[9] S. Bonner, F. Vasile, Causal embeddings for recommendation, in: Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 104–112.
[10] J. Chen, H. Dong, Y. Qiu, X. He, X. Xin, L. Chen, G. Lin, K. Yang, Autodebias: learning to debias for recommendation, in: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2021, pp. 21–30.

[11] T. Wei, F. Feng, J. Chen, Z. Wu, J. Yi, X. He, Model-agnostic counterfactual reasoning for eliminating popularity bias in recommender system, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2021, pp. 1791–1800.
[12] Y. Gu, Z. Ding, S. Wang, L. Zou, Y. Liu, D. Yin, Deep multifaceted transformers for multi-objective ranking in large-scale e-commerce recommender systems, in: Proceedings of the 29th ACM International Conference on Information and Knowledge Management, 2020, pp. 2493–2500.
[13] D. Liu, P. Cheng, H. Zhu, Z. Dong, X. He, W. Pan, Z. Ming, Mitigating confounding bias in recommendation via information bottleneck, in: Proceedings of the 15th ACM Conference on Recommender Systems, 2021, pp. 351–360.
[14] D. Liang, L. Charlin, D.M. Blei, Causal inference for recommendation, in: Workshop on Causation: Foundation to Application Co-Located with the 32nd Conference on Uncertainty in Artificial Intelligence, 2016.
[15] X. Wang, R. Zhang, Y. Sun, J. Qi, Doubly robust joint learning for recommendation on data missing not at random, in: Proceedings of the 36th International Conference on Machine Learning, 2019, pp. 6638–6647.
[16] Z. Zhu, Y. He, Y. Zhang, J. Caverlee, Unbiased implicit recommendation and propensity estimation via combinational joint learning, in: Proceedings of the 14th ACM Conference on Recommender Systems, 2020, pp. 551–556.
[17] Y. Saito, Asymmetric tri-training for debiasing missing-not-at-random explicit feedback, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 309–318.
[18] W. Kweon, S. Kang, H. Yu, Bidirectional distillation for top-k recommender system, in: Proceedings of the Web Conference 2021, 2021, pp. 3861–3871.
[19] N. Hazrati, F. Ricci, Choice models and recommender systems effects on users' choices, 2022
.
[20] A. Sharma, J.M. Hofman, D.J. Watts, Estimating the causal impact of recommendation systems from observational data, in: Proceedings of the 16th ACM Conference on Economics and Computation, 2015, pp. 453–470.
[21] D. Fleder, K. Hosanagar, Blockbuster culture's next rise or fall: the impact of recommender systems on sales diversity, Manag. Sci. 55 (5) (2009) 697–712.
[22] M. Deshpande, G. Karypis, Item-based top-N recommendation algorithms, ACM Trans. Inf. Syst. 22 (1) (2004) 143–177.
[23] G. Linden, B. Smith, J. York, Amazon.com recommendations: item-to-item collaborative filtering, IEEE Internet Comput. 7 (1) (2003) 76–80.
[24] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th International Conference on World Wide Web, 2001, pp. 285–295.
[25] C. Kim, J. Kim, A recommendation algorithm using multi-level association rules, in: Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003), IEEE, 2003, pp. 524–527.
[26] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 426–434.
[27] A. Paterek, Improving regularized singular value decomposition for collaborative filtering, in: Proceedings of KDD Cup and Workshop, vol. 2007, 2007, pp. 5–8.
[28] A. Mnih, R.R. Salakhutdinov, Probabilistic matrix factorization, in: Proceedings of the 21st International Conference on Neural Information Processing Systems, 2007, pp. 1257–1264.
[29] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[30] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, Bpr: Bayesian personalized ranking from implicit feedback, arXiv preprint arXiv:1205.2618, 2012.
[31] S. Kabbur, X. Ning, G. Karypis, 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, 2013, pp. 659–667.
[32] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 791–798.
[33] D. Liang, R.G. Krishnan, M.D. Hoffman, T. Jebara, Variational autoencoders for collaborative filtering, in: Proceedings of the 27th International Conference on World Wide Web, 2018, pp. 689–698.
[34] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, 2017, pp. 173–182.
[35] X. Li, J. She, Collaborative variational autoencoder for recommender systems, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 305–314.
[36] S. Sedhain, A.K. Menon, S. Sanner, L. Xie, Autorec: autoencoders meet collaborative filtering, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 111–112.
[37] X. Wang, X. He, M. Wang, F. Feng, T.-S. Chua, Neural graph collaborative filtering, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 165–174.
[38] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang, 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, 2020, pp. 639–648.
[39] S. Wu, F. Sun, W. Zhang, X. Xie, B. Cui, Graph neural networks in recommender systems: a survey, ACM Comput. Surv. (2020).
[40] K. Mao, J. Zhu, X. Xiao, B. Lu, Z. Wang, X. He, Ultragcn: Ultra simplification of graph convolutional networks for recommendation, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 2021, pp. 1253–1262.
[41] Z. Gao, T. Shen, Z. Mai, M.R. Bouadjenek, I. Waller, A. Anderson, R. Bodkin, S. Sanner, Mitigating the filter bubble while maintaining relevance: targeted diversification with vae-based recommender systems, in: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2022, pp. 2524–2531.
[42] G. Wu, M.R. Bouadjenek, S. Sanner, One-class collaborative filtering with the queryable variational autoencoder, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 921–924.
[43] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[44] Y. Yao, G. Doretto, Boosting for transfer learning with multiple sources, in: Proceedings of 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2010, pp. 1855–1862.
[45] B. Tan, Y. Song, E. Zhong, Q. Yang, Transitive transfer learning, in: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, pp. 1155–1164.
[46] S.J. Pan, I.W. Tsang, J.T. Kwok, Q. Yang, Domain adaptation via transfer component analysis, IEEE Trans. Neural Netw. 22 (2) (2010) 199–210.
[47] M. Long, J. Wang, G. Ding, J. Sun, P.S. Yu, Transfer feature learning with joint distribution adaptation, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 2200–2207.
[48] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, arXiv preprint arXiv:1411.1792, 2014.
[49] M. Long, Y. Cao, J. Wang, M. Jordan, Learning transferable features with deep adaptation networks, in: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 97–105.
[50] L. Mihalkova, T. Huynh, R.J. Mooney, Mapping and revising Markov logic networks for transfer learning, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, vol. 7, 2007, pp. 608–614.
[51] W. Pan, Q. Yang, Transfer learning in heterogeneous collaborative filtering domains, Artif. Intell. 197 (2013) 39–55.

[52] W. Pan, A survey of transfer learning for collaborative recommendation with auxiliary data, Neurocomputing 177 (2016) 447–453.
[53] X. Chen, L. Li, W. Pan, Z. Ming, A survey on heterogeneous one-class collaborative filtering, ACM Trans. Inf. Syst. 38 (4) (2020) 1–54.
[54] I. Fernández-Tobías, I. Cantador, M. Kaminskas, F. Ricci, Cross-domain recommender systems: a survey of the state of the art, in: Proceedings of Spanish Conference on Information Retrieval, 2012, pp. 1–12.
[55] L. Zhao, S.J. Pan, Q. Yang, A unified framework of active transfer learning for cross-system recommendation, Artif. Intell. 245 (2017) 38–55.
[56] F. Zhu, Y. Wang, C. Chen, J. Zhou, L. Li, G. Liu, Cross-domain recommendation: challenges, progress, and prospects, arXiv preprint arXiv:2103.01696, 2021.
[57] W. Pan, E. Xiang, N. Liu, Q. Yang, Transfer learning in collaborative filtering for sparsity reduction, in: Proceedings of the 24th AAAI Conference on Artificial Intelligence, vol. 24, 2010, pp. 230–235.
[58] L. Tang, B. Long, B.-C. Chen, D. Agarwal, An empirical study on recommendation with multiple types of feedback, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 283–292.
[59] A.P. Singh, G.J. Gordon, Relational learning via collective matrix factorization, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 650–658.
[60] J. Cheng, T. Yuan, J. Wang, H. Lu, Group latent factor model for recommendation with multiple user behaviors, in: Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2014, pp. 995–998.
[61] Y. Zhen, W.-J. Li, D.-Y. Yeung, Tagicofi: tag informed collaborative filtering, in: Proceedings of the 3rd ACM Conference on Recommender Systems, 2009, pp. 69–76.
[62] W. Pan, M. Liu, Z. Ming, Transfer learning for heterogeneous one-class collaborative filtering, IEEE Intell. Syst. 31 (4) (2016) 43–49.
[63] S. Tan, J. Bu, X. Qin, C. Chen, D. Cai, Cross domain recommendation based on multi-type media fusion, Neurocomputing 127 (2014) 124–134.
[64] H. Kanagawa, H. Kobayashi, N. Shimizu, Y. Tagami, T. Suzuki, Cross-domain recommendation via deep domain adaptation, in: Proceedings of the 41st European Conference on Information Retrieval, Springer, 2019, pp. 20–29.
[65] C.-Y. Li, S.-D. Lin, Matching users and items across domains to improve the recommendation quality, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 801–810.
[66] G. Hu, Y. Zhang, Q. Yang, Transfer meets hybrid: a synthetic approach for cross-domain collaborative filtering with text, in: Proceedings of the 28th International Conference on World Wide Web, 2019, pp. 2822–2829.
[67] S. Gao, H. Luo, D. Chen, S. Li, P. Gallinari, J. Guo, Cross-domain recommendation via cluster-level latent factor model, in: Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 161–176.
[68] F. Yuan, L. Yao, B. Benatallah, Darec: deep domain adaptation for cross-domain recommendation via transferring rating patterns, arXiv preprint arXiv:1905.10760, 2019.
[69] Y. Saito, S. Yaginuma, Y. Nishino, H. Sakata, K. Nakata, Unbiased recommender learning from missing-not-at-random implicit feedback, in: Proceedings of the 13th International Conference on Web Search and Data Mining, 2020, pp. 501–509.
[70] D. Liu, C. Lin, Z. Zhang, Y. Xiao, H. Tong, Spiral of silence in recommender systems, in: Proceedings of the 12th ACM International Conference on Web Search and Data Mining, 2019, pp. 222–230.
[71] T. Shen, J. Li, M.R. Bouadjenek, Z. Mai, S. Sanner, Towards understanding and mitigating unintended biases in language model-driven conversational recommendation, Inf. Process. Manag. 60 (1) (2023) 103139.
[72] Y. Saito, H. Sakata, K. Nakata, Doubly robust prediction and evaluation methods improve uplift modeling for observational data, in: Proceedings of the 2019 SIAM International Conference on Data Mining, SIAM, 2019, pp. 468–476.
[73] T. Schnabel, P.N. Bennett, Debiasing item-to-item recommendations with small annotated datasets, in: Proceedings of the 14th ACM Conference on Recommender Systems, 2020, pp. 73–81.
[74] X. Wang, R. Zhang, Y. Sun, J. Qi, Combating selection biases in recommender systems with a few unbiased ratings, in: Proceedings of the 14th ACM International Conference on Web Search and Data Mining, 2021, pp. 427–435.
[75] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, X. He, Bias and debias in recommender system: a survey and future directions, arXiv preprint arXiv:2010.03240, 2020.
[76] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv preprint arXiv:1312.6114, 2013.
[77] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, arXiv preprint arXiv:1401.4082, 2014.
[78] Y. Deldjoo, F. Nazary, A. Ramisa, J. Mcauley, G. Pellegrini, A. Bellogin, T. Di Noia, A review of modern fashion recommender systems, arXiv preprint arXiv:2202.02757, 2022.
[79] Y. Deldjoo, M. Schedl, B. Hidasi, Y. Wei, X. He, Multimedia recommender systems: algorithms and challenges, in: Recommender Systems Handbook, Springer, 2022, pp. 973–1014.
[80] M. Schedl, P. Knees, B. McFee, D. Bogdanov, Music recommendation systems: techniques, use cases, and challenges, in: Recommender Systems Handbook, Springer, 2022, pp. 927–971.
[81] M. Quadrana, P. Cremonesi, D. Jannach, Sequence-aware recommender systems, ACM Comput. Surv. 51 (4) (2018) 1–36.
[82] B.M. Marlin, R.S. Zemel, Collaborative prediction and ranking with non-random missing data, in: Proceedings of the 3rd ACM Conference on Recommender Systems, 2009, pp. 5–12.
[83] C. Gao, S. Li, W. Lei, J. Chen, B. Li, P. Jiang, X. He, J. Mao, T.-S. Chua, Kuairec: a fully-observed dataset and insights for evaluating recommender systems, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022, pp. 540–550.
[84] P. Cremonesi, Y. Koren, R. Turrin, Performance of recommender algorithms on top-n recommendation tasks, in: Proceedings of the 4th ACM Conference on Recommender Systems, 2010, pp. 39–46.
[85] G. Wu, M. Volkovs, C.L. Soon, S. Sanner, H. Rai, Noise contrastive estimation for one-class collaborative filtering, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 135–144.
[86] A. Rago, O. Cocarascu, C. Bechlivanidis, D. Lagnado, F. Toni, Argumentative explanations for interactive recommendations, Artif. Intell. 296 (2021) 103506.
[87] L. Wu, Z. Zheng, Z. Qiu, H. Wang, H. Gu, T. Shen, C. Qin, C. Zhu, H. Zhu, Q. Liu, et al., A survey on large language models for recommendation, arXiv preprint arXiv:2305.19860, 2023.
[88] A. Lin, J. Wang, Z. Zhu, J. Caverlee, Quantifying and mitigating popularity bias in conversational recommender systems, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022, pp. 1238–1247.
[89] Z. Zhao, J. Chen, S. Zhou, X. He, X. Cao, F. Zhang, W. Wu, Popularity bias is not always evil: disentangling benign and harmful bias for recommendation, IEEE Trans. Knowl. Data Eng. (2022).