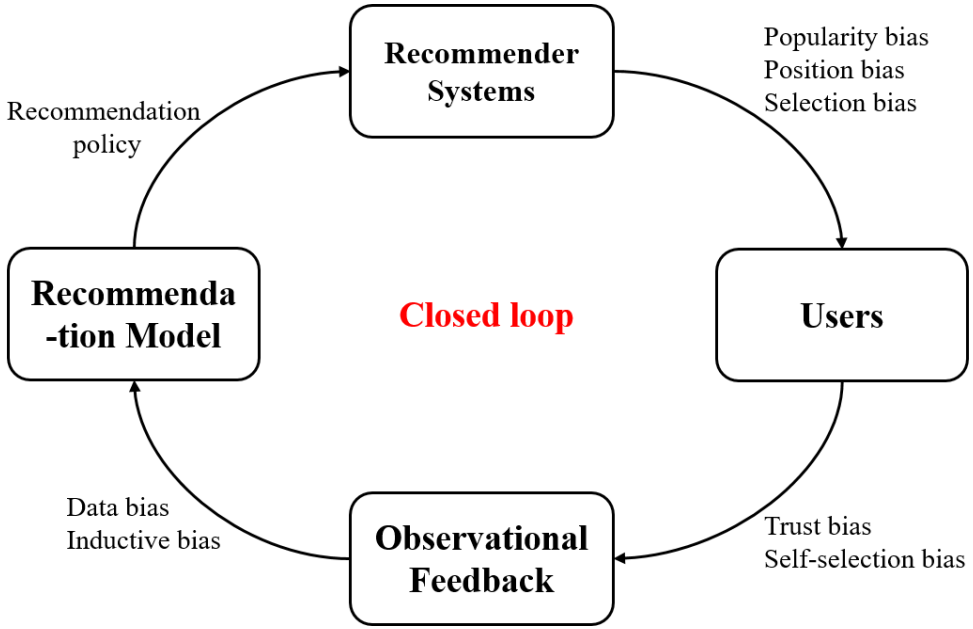
Transfer Learning in Collaborative Recommendation for Bias Reduction

Zinan Lin[1], Dugang Liu[1], Weike Pan[1]*, and Zhong Ming[1]*

lzn87591@gmail.com, dugang.ldg@gmail.com, panweike@szu.edu.cn, mingz@szu.edu.cn

[1]College of Computer Science and Software Engineering
Shenzhen University, Shenzhen, China

Background (1/2)



- In a typical closed-loop recommender system, a user's feedback or interaction is often influenced by the items' displaying positions and popularity, etc, which means that the collected data are biased.
- Most current collaborative recommendation models are built with the biased data only, which may not suit the users' tastes well.

Background (2/2)

- Previous works show that introducing the unbiased data has the potential to mitigate the bias in users' feedback.
- Existing research work includes proposing a domain adaptation algorithm [Bonner and Vasile, 2018], developing a knowledge-distillation framework [Liu et al., 2020], and designing a meta learning algorithm [Chen et al., 2021], etc.
- Most 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.

As a response, we propose a novel transfer learning solution called transfer via joint reconstruction (TJR).

Notations

Table: Some notations and explanations.

$\mathcal{U}$	user set, $u \in \mathcal{U}$
$\mathcal{I}$	item set, $i, \in \mathcal{I}$
$S^{\mathcal{A}} = \{(u, i)\}$	logged data collected by a non-uniform policy
$S^{\mathcal{T}} = \{(u, i)\}$	logged data collected by a uniform (i.e., random) policy
$y_u^{\mathcal{A}}$	the observed labels for user u from $S_u^{\mathcal{A}} (S_u^{\mathcal{A}} \subseteq S^{\mathcal{A}})$
$y_u^{\mathcal{T}}$	the observed labels for user u from $S_u^{\mathcal{T}} (S_u^{\mathcal{T}} \subseteq S^{\mathcal{T}})$

Problem Definition

- **Input**: 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 big and biased data collected by a commonly deployed non-uniform policy in a typical online recommender system and $S^{\mathcal{T}}$ is a small and unbiased data collected by a specially designed uniform (i.e., random) policy.
- **Goal**: To reduce the bias in $S^{\mathcal{A}}$ in order to fully exploit these two types of data with different properties.

Overall of Our Solution (1/2)

- We convert the modeling of the biased and unbiased data into a transfer learning problem, where the big biased data is taken as the auxiliary data and the small unbiased data is taken as the target data.
- Intuitively, the learned latent features, whether they represent users' preferences or bias information, ultimately affect the prediction of the model.
- 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.

Overall of Our Solution (2/2)

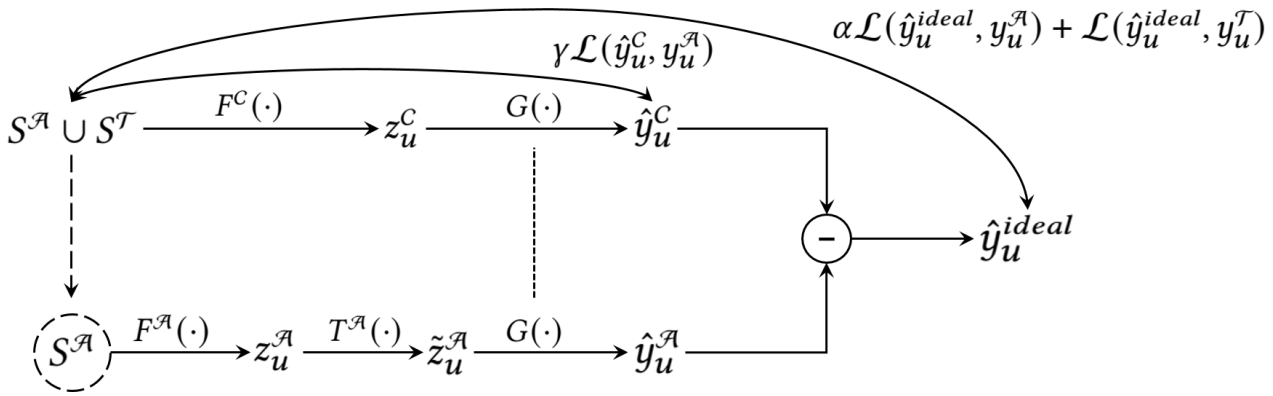


Figure: Illustration of our transfer via joint reconstruction (TJR). We extract the corresponding information from two different data. $F^{\mathcal{C}}(\cdot)$ is used to extract confusing latent features $z_u^{\mathcal{C}}$ that contain both user preferences and bias information, and both $F^{\mathcal{A}}(\cdot)$ and $T^{\mathcal{A}}(\cdot)$ are used to extract latent features $z_u^{\mathcal{A}}$ about 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 biased and unbiased data, to achieve the effect of joint reconstruction.

Upper Branch

- We use the union of $S^{\mathcal{A}}$ and $S^{\mathcal{T}}$ to obtain the confusing latent features $z_u^{\mathcal{C}}$ that contain user preferences and bias information through the function for latent features extraction, i.e., $F^{\mathcal{C}}(\cdot)$.
- We then 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, $F^{\mathcal{A}}(\cdot)$ refers to the encoder and $G(\cdot)$ refers to the decoder.

Bottom Branch

- 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 that the latent features of bias are usually non-linear and high-dimensional.
- We then use $G(\cdot)$ to get the prediction $\hat{y}_u^{\mathcal{A}}$.
- Notice that $G(\cdot)$ in the bottom branch is shared with the the one in the upper branch generates $\hat{y}_u^{\mathcal{C}}$.

Unbiased Prediction

To an obtain unbiased prediction $\hat{y}_u^{ideal}$, we intuitively use a linear method through $\hat{y}_u^{\mathcal{C}}$ and $\hat{y}_u^{\mathcal{A}}$,

$$\hat{y}_u^{ideal} = \hat{y}_u^{\mathcal{C}} - \hat{y}_u^{\mathcal{A}}. \tag{1}$$

Objective Function (1/2)

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,

$$\alpha \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}),$$

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 [Liu et al., 2020] and introduce a hyper-parameter α in the above loss function.

Objective Function (2/2)

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,

$$\mathcal{L}_{\text{TJR}} = \alpha\mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{A}}) + \mathcal{L}(\hat{y}_u^{ideal}, y_u^{\mathcal{T}}) + \gamma\mathcal{L}(\hat{y}_u^{\mathcal{C}}, y_u^{\mathcal{A}}), \tag{2}$$

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 $S_u^{\mathcal{T}}(S_u^{\mathcal{T}} \subseteq S^{\mathcal{T}})$.

Datasets (1/2)

- **Yahoo! R3**. Yahoo! R3 is a (user, song) rating data with a biased user subset and an unbiased random subset, involving 15400 users and 1000 songs. The user subset is biased, collected under a common recommendation policy and the random subset is an unbiased data, collected under a uniform policy.
- **Coat Shopping**. Coat Shopping 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.

Datasets (2/2)

For both datasets, 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.

Table: 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	
	#Feedback	P/N	#Feedback	P/N
$S^{\mathcal{A}}$	311,704	67.02%	6960	37.69%
$S^{\mathcal{T}}$	5,400	9.05%	464	28.89%
S_{va}	5,400	9.31%	464	23.40%
S_{te}	43,200	9.76%	3712	21.94%

Baselines (1/2)

We use three representative methods as backbone models (BMs), i.e., variational autoencoders (VAE), matrix factorization (MF) and neural collaborative filtering (NCF).

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.

Baselines (2/2)

- BM($S^{\mathcal{A}}$)
- BM($S^{\mathcal{T}}$)
- BM($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)
- Inverse propensity score (IPS) [Schnabel et al., 2016]
- CausE [Bonner and Vasile, 2018]
- The Bridge strategy [Liu et al., 2020]
- The Weight strategy [Liu et al., 2020]
- Autodebias [Chen et al., 2021]

Implementation Details (1/3)

- Both Yahoo! R3 and Coat shopping are rating data. We follow a previous work [Liu et al., 2020], and regard the ratings larger than 3 as positive feedback.
- 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.
- For AutoDebias-MF, we implement it via PyTorch 1.1 using the MSE loss and parameter configurations following the original paper [Chen et al., 2021]. For all the other methods, we implement them via TensorFlow 1.2 using the cross-entropy loss and batch training.

Implementation Details (2/3)

- We conduct a grid search to tune the hyper-parameters of the methods by checking the AUC performance on the validation data S_{va}.
- We choose the embedding size $rank \in \{50, 100, 200\}$, the regularization hyper-parameter $\lambda \in \{1e^{-5}, 1e^{-4} \cdots 1\}$, the tradeoff hyper-parameters $\alpha \in \{0.1, 0.2, ..., 1.0\}$ and $\gamma \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$.
- 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 $\{0.2, 0.3, 0.4, 0.5\}$.

Implementation Details (3/3)

- For the methods using MF (except AutoDebias-MF) and 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.
- The range of the values for batch size to be tuned is as follows. Notice that the numbers in brackets are the parameter range when using $S^{\mathcal{T}}$ to train the model, because the size of $S^{\mathcal{T}}$ is very small.

	Yahoo! R3	Coat Shopping
VAE-based Models	100 (100)	5,10,15,20 (5,10,15,20)
MF-based Models	512 (32)	32,64,128,256 (8,16,32,64)
NCF-based Models	512 (32)	32,64,128,256 (8,16,32,64)

Evaluation Metrics

- AUC
- NDCG@50

Main Results (1/3)

Table: Recommendation performance on Yahoo! R3 and Coat Shopping, 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 matrix factorization as the backbone model in Autodebias, since it does not support other backbone models such as VAE and NCF.

Dataset	Metrics	VAE($S^{\mathcal{A}}$)	VAE($S^{\mathcal{T}}$)	VAE($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	IPS-VAE	CausE-VAE	Bridge-VAE	Weight-VAE	Autodebias-VAE	TJR-VAE
Yahoo! R3	AUC	0.7666	0.5770	0.7709	0.7470	0.7673	0.7711	<u>0.7723</u>	-	**0.7804**
	NDCG@50	0.1009	0.0258	<u>0.1014</u>	0.0809	0.1013	0.1007	0.0998	-	**0.1023**
Coat Shopping	AUC	0.6210	0.5413	<u>0.6269</u>	0.5757	0.6210	0.6210	0.6245	-	**0.7603**
	NDCG@50	0.0921	0.0807	<u>0.0952</u>	0.0856	0.0922	0.0932	0.0949	-	**0.1239**

Dataset	Metrics	MF($S^{\mathcal{A}}$)	MF($S^{\mathcal{T}}$)	MF($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	IPS-MF	CausE-MF	Bridge-MF	Weight-MF	Autodebias	TJR-MF
Yahoo! R3	AUC	0.7329	0.5684	0.7409	0.7346	0.7285	<u>0.7524</u>	0.7465	0.7472	**0.7696**
	NDCG@50	0.0382	0.0304	0.0439	0.0426	0.0445	0.0615	0.0494	**0.0870**	<u>0.0705</u>
Coat Shopping	AUC	0.7606	0.5231	0.7631	0.7636	0.7611	**0.7653**	0.7636	0.6965	<u>0.7646</u>
	NDCG@50	0.0990	0.0578	<u>0.1016</u>	0.0965	0.0985	0.1005	0.1012	0.0999	**0.1027**

Dataset	Metrics	NCF($S^{\mathcal{A}}$)	NCF($S^{\mathcal{T}}$)	NCF($S^{\mathcal{A}} \cup S^{\mathcal{T}}$)	IPS-NCF	CausE-NCF	Bridge-NCF	Weight-NCF	Autodebias-NCF	TJR-NCF
Yahoo! R3	AUC	0.7245	0.6050	0.7268	0.7273	0.7283	0.7367	<u>0.7380</u>	-	**0.7420**
	NDCG@50	0.0279	0.0275	0.0327	0.0304	0.0284	<u>0.0439</u>	0.0383	-	**0.0454**
Coat Shopping	AUC	0.7507	0.5840	0.7508	0.7337	0.7516	<u>0.7522</u>	0.7509	-	**0.7537**
	NDCG@50	<u>0.0976</u>	0.0781	0.0969	0.0902	0.0961	0.0969	0.0946	-	**0.0995**

Main Results (2/3)

Observations

- Overall, our TJR outperforms all the baselines on both datasets, which clearly shows the advantage of our transfer learning solution in jointly modeling the biased and unbiased data. Notice that we take AUC as the main metric and use it in the process of tuning the hyper-parameters.
- Our TJR is flexible and could be easily extended to different backbone models such as MF, VAE and NCF.
- The performance of both CausE and the Bridge strategy are limited by the pre-trained model obtained by $S^{\mathcal{T}}$, and are 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.

Main Results (3/3)

- 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 analyse 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, which is caused by the nature of the dataset.

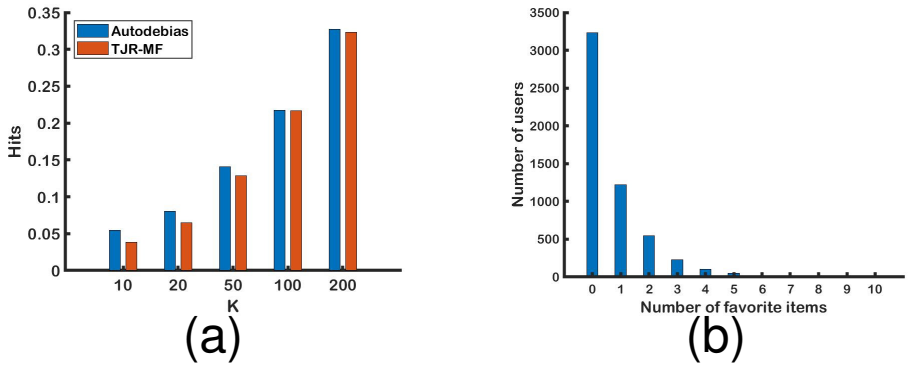
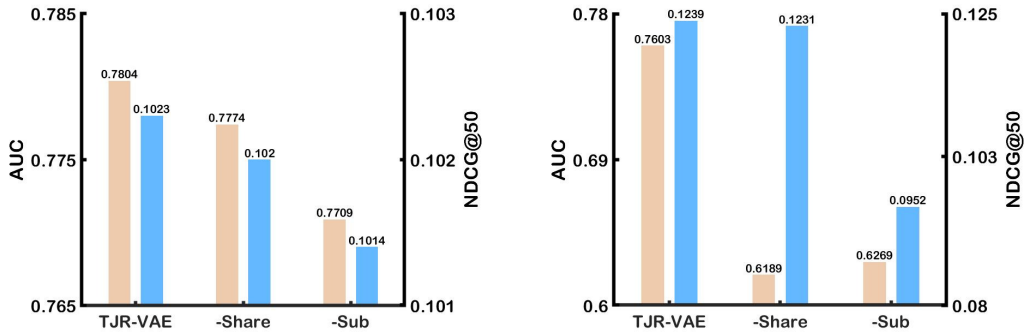


Figure: The percentage of hits of Autodebias and our TJR-MF (a), and the distribution of the number of users over the number of favorite items in the test set (b), for Yahoo! R3.

Ablation Studies (1/2)

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.



(a) Yahoo! R3 (b) Coat

Figure: 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.

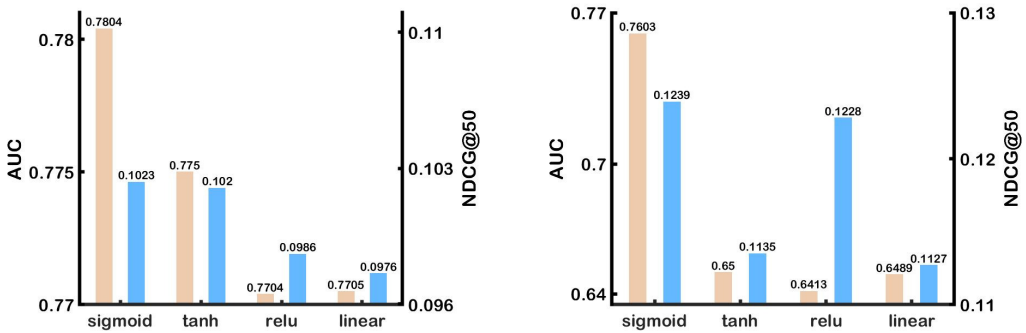
Ablation Studies (2/2)

Observations

- 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.
- We remove the bottom branch, 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.

Impact of the Transform Function (1/2)

We again use VAE as the backbone model and further study the impact of the transform function used to extract bias information. We report the the recommendation performance of our TJR with different transform functions including sigmoid, tanh, relu and linear.



(a) Yahoo! R3 (b) Coat

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

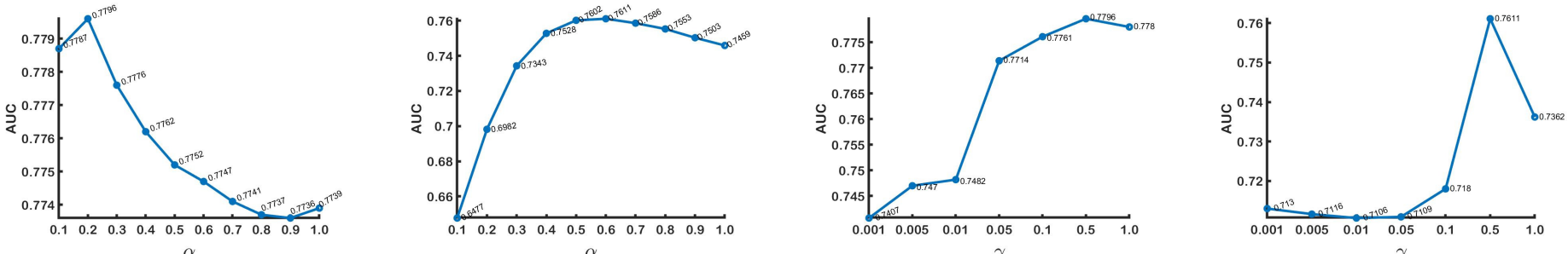
Impact of the Transform Function (2/2)

Observations

- 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.

Impact of the Hyper-Parameters (1/2)

We use VAE as the backbone model and study the impact of the hyper-parameters α and γ in our TJR.



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

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

Impact of the Hyper-Parameters (2/2)

Observations

- For α, we can see that the best values on Yahoo! R3 and Coat Shopping are 0.2 and 0.6, respectively. The reason is that the ratio $|S^{\mathcal{A}}|/|S^{\mathcal{T}}|$ of Yahoo! R3 is larger than that of Coat Shopping. In order to reduce the effect of the loss on $S^{\mathcal{A}}$, α shall be smaller.
- For γ, we can see that the best values on Yahoo! R3 and Coat Shopping are both 0.5. 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. The role of γ is to control the 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 bottom branch in our TJR will be weakened.

Conclusion

- We study an emerging and important problem called collaborative recommendation with a big biased data and a small unbiased data.
- We view this problem from a transfer learning perspective, and propose a novel transfer learning solution to achieve knowledge transfer between the two different data, aiming to reduce the bias and improve the recommendation performance.
- We design an end-to-end transfer learning framework, including two different but related models to extract latent features that represent users' preferences and bias information, a bias reduction component and a shared prediction model, optimized by a joint reconstruction loss, which is thus called transfer via joint reconstruction (TJR).
- We conduct extensive empirical studies on two public datasets, and find that our TJR performs significantly better than some very competitive baseline methods in most cases.

Future Works

For future works, we are interested in further generalizing our transfer learning solution to include more information such as temporal dynamics and item descriptions.

Thank you!

We thank the support of National Natural Science Foundation of China Nos. 61836005 and 61872249, and Shenzhen Basic Research Fund No. JCYJ20200813091134001.

Bonner, S. and Vasile, F. (2018).

Causal embeddings for recommendation.

In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 104–112.

Chen, J., Dong, H., Qiu, Y., He, X., Xin, X., Chen, L., Lin, G., and Yang, K. (2021).

Autodebias: Learning to debias for recommendation.

arXiv preprint arXiv:2105.04170.

Liu, D., Cheng, P., Dong, Z., He, X., Pan, W., and Ming, Z. (2020).

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*, pages 831–840.

Schnabel, T., Swaminathan, A., Singh, A., Chandak, N., and Joachims, T. (2016).

Recommendations as treatments: Debiasing learning and evaluation.

arXiv preprint arXiv:1602.05352.