A General Knowledge Distillation Framework for Counterfactual Recommendation via Uniform Data

Dugang Liu
Shenzhen University
Shenzhen, China
dugang.ldg@gmail.com

Pengxiang Cheng
Huawei Noah's Ark Lab
Shenzhen, China
chengpengxiang1@huawei.com

Zhenhua Dong
Huawei Noah's Ark Lab
Shenzhen, China
dongzhenhua@huawei.com

Xiuqiang He
Huawei Noah's Ark Lab
Shenzhen, China
hexiuqiang1@huawei.com

Weike Pan*
Shenzhen University
Shenzhen, China
panweike@szu.edu.cn

Zhong Ming*
Shenzhen University
Shenzhen, China
mingz@szu.edu.cn

ABSTRACT

Recommender systems are feedback loop systems, which often face bias problems such as popularity bias, previous model bias and position bias. In this paper, we focus on solving the bias problems in a recommender system via a uniform data. Through empirical studies in online and offline settings, we observe that simple modeling with a uniform data can alleviate the bias problems and improve the performance. However, the uniform data is always few and expensive to collect in a real product. In order to use the valuable uniform data more effectively, we propose a general knowledge distillation framework for counterfactual recommendation that enables uniform data modeling through four approaches: (1) label-based distillation focuses on using the imputed labels as a carrier to provide useful de-biasing guidance; (2) feature-based distillation aims to filter out the representative causal and stable features; (3) sample-based distillation considers mutual learning and alignment of the information of the uniform and non-uniform data; and (4) model structure-based distillation constrains the training of the models from the perspective of embedded representation. We conduct extensive experiments on both public and product datasets, demonstrating that the proposed four methods achieve better performance over the baseline models in terms of AUC and NLL. Moreover, we discuss the relation between the proposed methods and the previous works. We emphasize that counterfactual modeling with uniform data is a rich research area, and list some interesting and promising research topics worthy of further exploration. Note that the source codes are available at https://github.com/dgliu/SIGIR20_KDCRec.

CCS CONCEPTS

• **Information systems → Recommender systems**.

*Co-corresponding authors

KEYWORDS

Counterfactual learning, Recommender systems, Knowledge distillation, Uniform data

ACM Reference Format:
Dugang Liu, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Weike Pan, and Zhong Ming. 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 (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401083

1 INTRODUCTION

Recommender Systems as a feedback loop system may suffer from the bias problems such as popularity bias [1, 6], previous model bias [9, 16, 17] and position bias [3, 28]. Previous studies have shown that models and evaluation metrics that ignore the biases do not reflect the true performance of a recommender system, and that explicitly handling of the biases may help improve the performance [16, 28, 31]. Most of the previous works to solve the bias problems of recommender systems can be classified as counterfactual learning-based [25] and heuristic-based approaches. The former mainly uses the inverse propensity score (IPS) [24] and the counterfactual risk minimization (CRM) principle [25], while the latter mainly makes certain assumptions about the data being missing not at random (MNAR) [15, 17].

A recent work has shown that a uniform data can alleviate the previous model bias problem [16]. But the uniform data is always few and expensive to collect in real recommender systems. To collect a uniform data, we must intervene in the system by using a uniform logging policy instead of a stochastic recommendation policy, this is, for each user's request, we do not use the recommendation model for item delivery, but instead randomly select some items from all the candidate items and rank them with a uniform distribution. The uniform data can then be regarded as a good unbiased agent because it is not affected by a previously deployed recommendation model. However, the uniform logging policy would hurt the users' experiences and the revenue of the platform. This means that it is necessary to constrain the uniform data collection within a particularly small traffic (e.g., 1%).

In this paper, we focus on how to solve the bias problems in a recommender system with a uniform data. Along the line of [16], we conduct empirical studies on a real advertising system and a public dataset to validate the usefulness of the uniform data, where the uniform data is simply combined with the non-uniform data for training models. We observe that such a simple method can alleviate the bias and improve the performance, which motivates us to study more advanced methods that can make better use of the uniform data. Although there are many ways to extract information or knowledge from a uniform data, in this paper we focus on knowledge distillation because of its simplicity and flexibility.

To use the few and valuable uniform data more effectively, we propose a general knowledge distillation framework for counterfactual recommendation (KDCRec), which enables uniform data modeling with four approaches, i.e., label-based distillation, feature-based distillation, sample-based distillation and model structure-based distillation. Each one is based on a different concern, i.e., label-based distillation focuses on using the imputed labels as a carrier to provide useful de-biasing guidance; feature-based distillation aims to filter out the representative unbiased features; sample-based distillation considers mutual learning and alignment of the information of the uniform and non-uniform data; and model structure-based distillation constrains the training of the models from the perspective of embedded representation.

The main contributions of this paper are summarized as follows:

- We show empirical evidence that a uniform data is useful for preference modeling via an online A/B test and an offline evaluation, which justifies the importance of our research questions.
- We propose a general knowledge distillation framework KD-CRec for counterfactual recommendation via a uniform data, including label-based distillation, feature-based distillation, sample-based distillation and model structure-based distillation.
- We conduct extensive experiments on both public and product datasets, demonstrating that the four proposed methods achieve better performance over the baseline models in terms of AUC and NLL.
- We discuss the relation between the proposed methods and the previous works, and list some interesting and promising research directions for further exploration.

2 RELATED WORK

Since we study how to apply knowledge distillation techniques for counterfactual recommendation, we first review some related works on general knowledge distillation. We also include some counterfactual learning methods for recommendation and ranking.

2.1 Knowledge Distillation

Hinton's work first proposes the concept of knowledge distillation [10]. By introducing soft-targets related to teacher networks as part of the objective function, the training of student networks is guided to achieve knowledge transfer [18]. A series of follow-up works develop different distillation structures (e.g., multiple teachers [8] and cascade distillations [4]) and different forms of knowledge (e.g., alignment of the hidden layers [22] or the relation between the hidden layers [32]). Some recent works are no longer limited to model structure, but considers sample-based knowledge distillation [21, 27]. In this paper, we further expand the definition of distillation to include label-based and feature-based forms. The marriage of knowledge distillation and recommender systems has also attracted the attention of the researchers [26, 30, 34]. Most of these works focus on using knowledge distillation to extract some useful knowledge from some auxiliary models to enhance the performance or interpretability of the target recommendation model. In this paper, we focus on using knowledge distillation to solve the bias problems in recommender systems.

2.2 Counterfactual Learning for Ranking

For learning-to-rank tasks, Agarwal et al. [2] provides a general and theoretically rigorous framework with two counterfactual learning methods, i.e., SVM PropDCG and DeepPropDCG. Some position bias estimation methods for ranking are proposed in [3, 28]. IPS is one of the most popular counterfactual approaches for recommendation [24, 31], where each sample is weighted with an IPS, referring to the likelihood of the sample being logged. If there are no unobserved confounders, IPS methods can get an unbiased prediction model in theory. A direct method tries to learn an imputation model, which can infer the labels for both the observed and unobserved samples. The imputation model can be learned by machine learning models [7, 14] with the observed data. A doubly robust method [7] combines the IPS method and the aforementioned direct method together, and the bias can be eliminated if either the direct method part or the IPS method part is unbiased. Wang et al. [29] proposes a doubly robust method for joint learning of rating prediction and error imputation. Moreover, a uniform data is useful for counterfactual learning, such as imputation model learning [33], propensity computation [24] and modeling with uniform data directly [5, 11, 16, 23]. In this paper, we would like to study methods for better use of the uniform data from the perspective of knowledge distillation.

3 MOTIVATION

In a recent work [16], it is shown that a uniform (i.e., unbiased) data can alleviate the previous model bias problem. In this section, to further verify the usefulness of a uniform data, we firstly compare the online performance of two models in a real advertising system, where one model is trained with a biased data, and the other is trained with both a uniform data and a biased data. Next, we conduct some pilot experiments to quantify the effectiveness of an unbiased data using a public dataset.

3.1 Model Performance on a Product Dataset

We conduct an online A/B test on a large-scale advertising system. In the system, there is 1% traffic for "uniform data collection": for these requests, we randomly collect some advertisements from all candidates, and rank them with uniform distribution. The 1% training data is isolated from being influenced by the previously deployed recommendation models, which is thus called *1%-unbiased data*, the other 99% non-uniform traffic is named *99%-biased data*, and all the 100% traffic is named *100%-combined data*. Because logistic regression (LR) is one of the most popular models for CTR prediction, we implement two LR models with the *99%-biased data* and the *100%-combined data*, respectively. Next, we deploy the two models in the advertising system.

Experimental Setting. In our preliminary experiments, we collect training data from an online display advertising system for 30 days, and generate three kinds of data sets: *1%-unbiased data*, *99%-biased data* and *100%-combined data*. We verify the two models' effectiveness through an online A/B test for 30 consecutive days. The ads requests have been split into two groups, each of which contains more than two million ads requests each day. One request group receives recommendations from one of the two models. The candidates ads are ranked by $bid * pCTR$, where the advertiser offers the bid, and our models compute the $pCTR$ values. We thus use the effective cost per mille (eCPM) as the online performance:

$$eCPM = \frac{Total\_Ads\_Income}{Total\_Ads\_Impressions} \times 1000. \quad (1)$$

For the offline experiment, we split the 30-day data sequentially, where the first 28 days for training, the last 2 days for validation and test. Following most CTR prediction studies, we consider the area under the roc curve (AUC) as the offline evaluation metric.

Experiment Results. The experimental results are shown in Table 1, from which we can see that the *100%-combined data* model wins the other model by 1.56% (from 0.7571 to 0.7689) in terms of AUC. Although the income degrades when collecting the 1% randomized training data, the improvement from the uniform data is 2.98%, which is much higher than the loss. We also train a model with the *1%-unbiased data*, but the simulated ads ranking lists do not look well, which is thus not deployed by the product team.

Table 1: Performance comparisons on a product dataset.

Data approach	Offline AUC	Online eCPM
99%-biased data	0.7571	0.0%
100%-combined data	0.7689	2.98% (improvement)

3.2 Model Performance on a Public Dataset

We conduct some pilot experiments on the Yahoo! R3 dataset to validate the usefulness of the unbiased data. Yahoo! R3 contains some user-song ratings, where users were asked to rate a uniformly drawn sample of songs. After processing of the data, we split the public dataset into three training subsets, one validation set and one test set, i.e., *uniform data*, *biased data*, *uniform data* ∪ *biased data*, *uniform validation data* and *uniform test data*. We implement three matrix factorization (MF) models with the three training subsets, respectively, and adopt AUC and the negative logarithmic loss (NLL) as the evaluation metrics. It is worth mentioning that we choose the *uniform test data* as the test set to ensure the unbiasedness of the experiment.

We observe a forward effect about the performance of the *uniform data*. As shown in Table 2, the *uniform data* model has the best NLL score, but its AUC score is not competitive. The model trained with the combination of the *uniform data* and the *biased data* performs better than the model trained only with the *biased data*, which means that the *uniform data* can help to improve the accuracy.

Through the experiments with the product dataset and the public dataset, we find that the *uniform data* can improve the recommendation performance by simply being combined with the biased data, which inspires us to study some more advanced methods.

Table 2: Performance comparisons on a public dataset.

Data approach	AUC	NLL
uniform data	0.5692	-0.50994
biased data	0.7275	-0.58905
uniform data ∪ *biased data*	0.7295	-0.58138

4 THE PROPOSED FRAMEWORK

In order to effectively make use of the uniform data, we propose a general **K**nowledge **D**istillation framework for **C**ounterfactual **Rec**ommendation in this section, **KDCRec** for short. Figure 1 shows the overview of the framework of our KDCRec. In our framework, the uniform data can be modeled with four different methods, including label-based distillation, feature-based distillation, sample-based distillation, and model structure-based distillation. Note that we use a general definition of distillation in the study rather than the past knowledge distillation approaches such as considering the level of sample [21, 27] and model structure [10, 22]. Each method is based on different concerns to mine the potentially useful knowledge from the uniform data, which will be used to improve the learning of the biased data. Next, we will introduce the four methods in turn as different modules. More specifically, in each module, we will give a formal definition of the corresponding method, and list some practical solutions under the guidance of the definition.

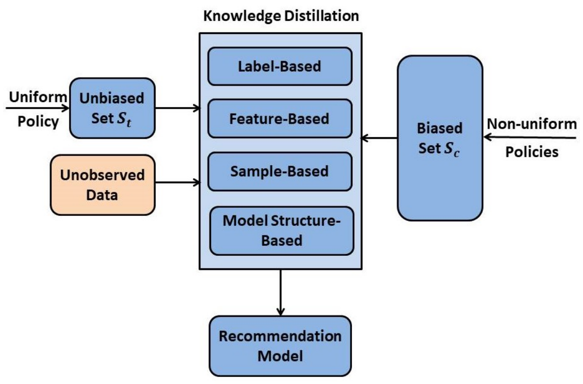


Figure 1: Overview of the KDCRec framework. The scale of the biased set S_c is much larger than that of the unbiased set S_t. Since the unobserved data is only used in some modules, we distinguish it from S_c and S_t using a different color.

4.1 Label-Based Module

Models trained on a non-uniform data S_c tend to produce biased predictions, while predictions from a uniform data S_t are more unbiased. An intuitive idea is that when training a model on S_c, the model receives the imputed labels produced by S_t to correct the bias of its own predictions. Based on this idea, we develop the following formal definition of label-based distillation. Note that on the premise of using the imputed labels, we can also include the labels of S_t. We emphasize the use of the imputed labels to avoid confusion with other distillation methods.

Definition 1 (D1). *A method can be classified as* ***label-based distillation*** *if and only if the training of a non-uniform data S_c can benefit from the imputed labels produced by a uniform data S_t.*

Solutions. Next, we use the two strategies adopted in our experiments as examples to illustrate how label-based distillation can be realized.

- **Bridge Strategy.** Let $\mathcal{D}$ denote the whole set of data, including the non-uniform data S_c, the uniform data S_t and the unobserved data. We first consider a scenario where two models are trained simultaneously, i.e., train the model M_c and M_t in a supervised manner on S_c and S_t, respectively. To correct the bias of M_c, we randomly sample an auxiliary set S_a from $\mathcal{D}$ as a bridge in each iterative training, and expect the predicted output of M_c and M_t on S_a to be close. Note that most of the samples in S_a are unobserved data because of the data sparsity in recommender systems. Due to the unbiased nature of S_t and S_a, this strategy can reduce the bias of M_c. The final objective function of this strategy is,

$$\min_{\mathcal{W}_c,\mathcal{W}_t} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \frac{1}{|S_t|} \sum_{(i,j)\in S_t} \ell\left(y_{ij}, \hat{y}^t_{ij}\right) + \frac{1}{|S_a|} \sum_{(i,j)\in S_a} \ell\left(\hat{y}^c_{ij}, \hat{y}^t_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right) + \lambda_t R\left(\mathcal{W}_t\right), \tag{2}$$

where $\mathcal{W}_c$ and $\mathcal{W}_t$ denote the parameters of M_c and M_t, respectively, and $\ell(\cdot,\cdot)$ is an arbitrary loss function. And y_{ij}, $\hat{y}^c_{ij}$ and $\hat{y}^t_{ij}$ denote the true label, and the predicted labels of M_c and M_t for the sample (i, j), respectively, where (i, j) is associated with user i and item j. Note that $R(\cdot)$ is the regularization term, and λ_c and λ_t are the parameters of the regularization.
- **Refine Strategy.** We next consider a scenario where only one model M_c is trained. The bias of S_c may be reflected in the labels, resulting in models trained on these labels being biased. For example, when generating samples for modeling, all the observed positive feedback are usually labeled as 1, and all the observed negative feedback are labeled as -1. But in fact, they should fit a preference distribution. With S_t, we expect to be able to better infer the true distribution of the labels on S_c and then refine them. Suppose we have obtained a model M_t pre-trained on S_t, and then use it to predict all the samples on S_c. These imputed labels are combined with the original labels of S_c through a weighting parameter, which are then used to train a more unbiased model M_c. Note that in order to avoid the distribution difference between the imputed labels and the original labels, we need to normalize the imputed labels. The final objective function of this strategy is,

$$\min_{\mathcal{W}_c} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij} + \alpha N\left(\hat{y}^t_{ij}\right), \hat{y}^c_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right), \tag{3}$$

where α is a tunable parameter that controls the importance of the imputed labels produced by M_t, and $N(\cdot)$ denotes a normalization function.

4.2 Feature-Based Module

Previous studies find that some features correlate with labels, but the correlation is not a causal relation. For example, from 1999 to 2009, the correlation between "the number of people who drowned by falling into a pool" and "the number of films Nicolas Cage appeared in" is 66.6%. But as we know, if Nicolas Cage does not appear in any film in a year, the number of people who drown in a pool may still not be 0. Hence, we need to learn some causal and stable features. The feature-based module can be divided into two steps, i.e., stable feature selection and biased data correction. Firstly, we filter out causal and stable features via a uniform data through some methods. Then, we need to employ the stable features to train a teacher model that can be used to guide the biased model. Thus, we develop the following formal definition of feature-based distillation.

Definition 2 (D2). *A method can be classified as **feature-based distillation** if and only if the training of a non-uniform data S_c can benefit from the representative causal and stable features produced by a uniform data S_t.*

Solutions. We employ stable feature strategy as an example to reveal how feature-based distillation can be realized.

- **Stable Feature Strategy.** We propose a stable feature distillation module to filter out the causal features for correcting the bias from S_c. Figure 2 illustrates the main idea of stable feature distillation, which consists of a deep global balancing regression (DGBR) algorithm [13], a teacher network and a student network. The DGBR algorithm optimizes a deep autoencoder model for feature selection and a global balancing model for learning the global sample weights and the predicting stability. The main idea of feature-based distillation is to filter out the representative stable features through DGBR from S_t, which are then used to train a teacher network. Next, we train a student network to mimic the output of the teacher model.

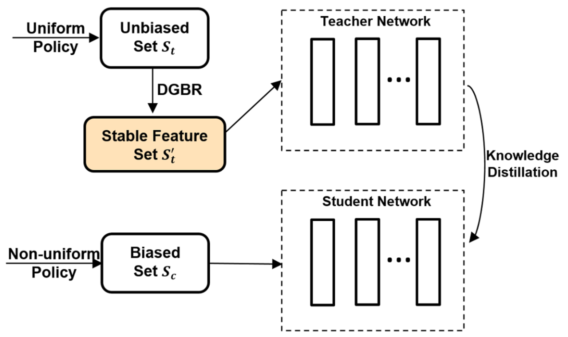


Figure 2: Illustration of stable feature distillation.

4.3 Sample-Based Module

In a real recommender system with a stochastic logging policy, the probability of an item being recommended is different, and the probability of a user making a choice is also different. This means that model M_c may treat some items and users unfairly, because the samples in S_c lack support for these items and users. This unfairness can be corrected to some extent by directly considering the samples in S_t during the training process of M_c, as empirically shown in Section 3. Because the uniform logging policy corresponding to S_t increases the probability of the less popular items being selected, and M_c needs to weigh this difference between S_c and S_t. Based on this idea, we develop the following formal definition of sample-based distillation, Definition 3 (D3). *A method can be classified as **sample-based distillation** if and only if a uniform data S_t is directly applied to help learning on all the samples without generating some imputed labels.*

Solutions. Next, we use the three strategies adopted in our experiments as examples to illustrate how sample-based distillation can be realized.

- **Causal Embedding Strategy (CausE).** The causal embedding method [5] first considers the scenario of training M_c and M_t simultaneously. It designs an additional alignment term to explicitly represent the learning of M_c for M_t. Causal embedding defines this alignment term as the pairwise difference between the parameters of M_c and M_t, which is then included in the object function to be minimized. When the value of the alignment term becomes small, it means that M_c learns the causal information contained in S_t, which helps correct the bias in learning on S_c. Note that it is difficult to dynamically optimize the differences between all the parameters of the two neural networks, so we only use two low-rank models to implement this strategy in our experiments. The final objective function is,

$$\min_{\mathcal{W}_c,\mathcal{W}_t} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \frac{1}{|S_t|} \sum_{(i,j)\in S_t} \ell\left(y_{ij}, \hat{y}^t_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right) + \lambda_t R\left(\mathcal{W}_t\right) + \lambda^{CausE}_{tc} \left\|\mathcal{W}_t - \mathcal{W}_c\right\|^2_F, \tag{4}$$

where λ^{CausE}_{tc} is the regularization parameter for the alignment term of M_c and M_t.
- **Weighted Combination Strategy (WeightC).** How to effectively introduce the samples from S_t to help M_c? Inspired by modeling of heterogeneous implicit feedback [20], we add a confidence parameter to each sample of S_c and S_t to indicate whether it is unbiased. Naturally, the confidence of the samples in S_t is set to 1, and the confidence of the samples in S_c has two schemes to be used. The first scheme is a global setting, i.e., we set a confidence value in advance for all the samples of S_c. The second scheme is a local setting, i.e., each sample of S_c has a confidence value that needs to be learned by M_c. The confidence of each sample is related to the corresponding loss function. The final objective function of this strategy is,

$$\min_{\mathcal{W}_c} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \alpha_{ij}\ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \frac{1}{|S_t|} \sum_{(i,j)\in S_t} \ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right), \tag{5}$$

where $\alpha_{ij} \in [0, 1]$ is a parameter used to control the confidence that we believe the sample (i, j) is unbiased. When considering the global setting, α_{ij} shares a parameter value that we preset for all the samples in S_c, but in the local setting, α_{ij} is an independent parameter value learned by M_c.
- **Delayed Combination Strategy (DelayC).** Instead of introducing a confidence parameter, we propose a strategy called delayed combination. This strategy directly applies the data of S_c and S_t to the training of M_c in an alternative manner. Specifically, in the S_c step of each iteration, M_c is trained on the data of s batches in S_c. In the S_t step, we randomly sample one batch of data from S_t to train M_c. We repeat these two steps until all the data of S_c are used. The batch ratio is set to $s : 1$, which can better ensure the training of M_c itself and the correction under the guidance of S_t. The final objective function of this strategy is,

$$\begin{cases} \min\limits_{\mathcal{W}_c} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right), & S_c \text{ step.} \\ \min\limits_{\mathcal{W}_c} \frac{1}{|S_t|} \sum_{(i,j)\in S_t} \ell\left(y_{ij}, \hat{y}^c_{ij}\right) + \lambda_c R\left(\mathcal{W}_c\right), & S_t \text{ step.} \end{cases} \tag{6}$$

4.4 Model Structure-Based Module

Finally, we return to the model itself through considering how to directly use the pre-trained model M_t to help the learning of M_c. This is the most commonly adopted distillation strategy in existing works. In order to help M_c with the guidance from M_t, we assume that some embedded representations of M_c correspond to some embedded representations of M_t. We constrain the selected embedded representations in M_c to be similar to their corresponding embedded representations in M_t. As a result, M_c will have a similar pattern to M_t and thus may benefit from it. Note that the selected embedded representations of M_c and M_t do not necessarily have the same index. For example, suppose A is a 4-layer network and B is an 8-layer network, we may specify that each layer of A corresponds to an even layer of B, namely 2, 4, 6 and 8. Based on this idea, we develop the following formal definition of model structure-based distillation. For the sake of discussion, as shown in Figure 3, we classify all the embedded representations into three types with different functions.

Definition 4 (D4). *A method can be classified as **model structure-based distillation** if and only if instead of using the labels and data, the embedded representation trained on a uniform data S_t is used to help the learning of a non-uniform data S_c.*

Solutions. Next, we use the three strategies adopted in our experiments as examples to illustrate how model structure-based distillation can be realized.

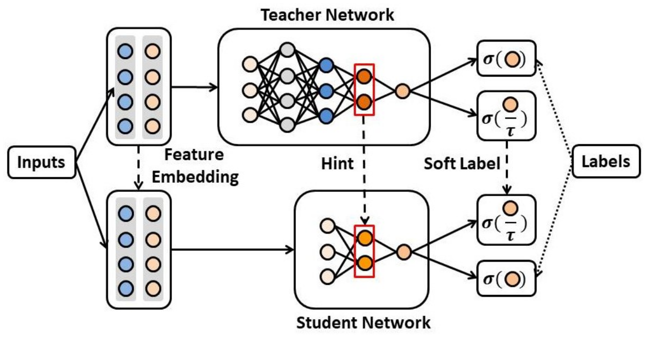


Figure 3: Illustration of three types of model structure-based distillations, including feature embedding, hint and soft label. We use dotted arrows to indicate the matched pairs considered by different types of distillations.

- **Feature Embedding Strategy (FeatE).** Feature embedding are embedded representations that are directly connected to the users and items. In a neural network, it is usually the result of a one-hot coding after a *lookup* operation; and in a low-rank model, it is the users' preference vector u and the items' attribute vector v.

As a special example, we think that the feature embedding of the autoencoder refers to the weights related to the number of items in the first layer and the last layer of the network. It may be unreasonable to directly match the feature embedding in M_c with the that in M_t, because M_t may not learn sufficiently on these user- and item-related embedded representations due to the small data size. We propose the following two alternatives to use the feature embedding in M_t, including initialization of M_c, and concatenation with the parameters of M_c,

Initialization. We have three options to choose the type of feature embedding as the initialization of M_c, including using only user-related, only item-related, and both. In addition, if we know which of the user-related and item-related ones is trained better, we can further use the information from M_t by setting their update steps to 1 (for the better one) and s (for the other, > 1), respectively. We call it the FeatE-*alter*.

Concatenation. After the parameters of M_c are randomly initialized, the feature embedding of M_t will be concatenated with these parameters to form new parameters to train M_c. Note that the features embedded of M_t in the parameters will not be updated during the training process.

- **Hint Strategy.** Hint refers to the hidden layer in a neural network, also known as feature map [22]. They contain higher-order non-linear relations between users or items. Note that in the experiments we must use deep neural networks to implement this strategy. After we specify hint for alignment in M_c and M_t, we explicitly model the difference between the two hints on the objective function of M_c. The final objective function of this strategy is,

$$\begin{aligned}\min_{\mathcal{W}_c} \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij}, \hat{y}_{ij}^{c}\right) + \lambda_c R\left(\mathcal{W}_c\right) \\ + \lambda_{tc}^{hint} \left\| y_t^{hint} - y_c^{hint} \right\|_F^2,\end{aligned} \tag{7}$$

where y_c^{hint} and y_t^{hint} are the output of M_c and M_t on their respective designated hint layers.

- **Soft Label Strategy.** Previous works have shown that training the student network to mimic the output of the teacher network on hard-labeled objectives does not bring much useful information to the student network. But, by introducing softmax and temperature operations to relax the label, training the student network to keep the same output as the teacher network on a soft label will result in a significant improvement [10]. We follow a similar setup in this strategy. Note that in the experiments we must also use deep neural networks to implement this strategy. The final objective function of this strategy is,

$$\begin{aligned}\min_{\mathcal{W}_c} \frac{\alpha}{|\mathcal{D}|} \sum_{(i,j)\in \mathcal{D}} \ell\left(\text{softmax}\left(\frac{\hat{y}_{ij}^{c}}{\tau}\right), \text{softmax}\left(\frac{\hat{y}_{ij}^{t}}{\tau}\right)\right) \\ + \frac{1}{|S_c|} \sum_{(i,j)\in S_c} \ell\left(y_{ij}, \hat{y}_{ij}^{c}\right) + \lambda_c R\left(\mathcal{W}_c\right),\end{aligned} \tag{8}$$

where τ a is a temperature parameter, and α is a tunable parameter that controls the importance of the soft labels.

4.5 Summary and Remarks

Based on the above description, we can see that different strategies exhibit their own characteristics about how to make use of S_t. Some methods commonly used in counterfactual recommendation can be incorporated into our framework. Label-based distillation includes a direct method for learning an imputation model and its variants. Sample-based distillation includes the IPS method [24, 31] and other approaches as described in Section 2.2. Although we introduce the four distillation methods in different modules, their relations are close. This means that we can design new strategies with different combinations of the four distillation methods, such as the doubly robust method [7] and its variants [33]. Moreover, they are also related to the types of knowledge (instance, feature and model) and strategies (adaptive, collective and integrative) in transfer learning [18, 19].

In addition, we must keep in mind that the different considerations when using these four distillation methods. Although label-based and sample-based distillations are easy to implement, they need to consider the potential factors on the label and sample that may affect the model, such as the differences in sample size and the label distributions. The difference in the label distributions is passed on to the distributions of the predicted labels, so that the strategy of directly using the predicted labels may lead to poor results. The difference in data size means that M_c in a rough strategy can almost ignore the guided information from S_t. Feature-based distillation relies on the accuracy of the method used to filter out the causal and stable features. However, the current research in this direction is still not sufficient, and the existing methods need more time and computing resources. Model structure-based distillation requires only the model itself without regarding to other potential factors. But it is not easy to design an effective distillation structure or select some good embedded representations.

5 EMPIRICAL EVALUATION

In this section, we conduct experiments with the aim of answering the following two key questions.

- RQ1: How do the proposed methods perform against baselines?
- RQ2: How does S_t improve the model trained on S_c?

5.1 Experiment Setup

5.1.1 Datasets. To evaluate the recommendation performance of the proposed framework, the selected dataset must have a uniform subset for training and test. We consider the following datasets in the experiments, where the statistics are described in Table 3.

- **Yahoo! R3** [17]: This dataset contains ratings collected from two different sources on Yahoo! Music services, involving 15,400 users and 1000 songs. The Yahoo! user set consists of ratings supplied by users during normal interactions, i.e., users pick and rate items as they wish. This can be considered as a stochastic logging policy by following [24, 31], and thus the user set is biased. The Yahoo! random set consists of ratings collected during an online survey, when each of the first 5400 users is asked to provide ratings on ten songs. The random set is different because the songs are randomly selected by the system instead of by the users themselves. The random set corresponds to a uniform logging policy and can be considered as the ground truth without bias. We binarize the ratings based on a threshold $\epsilon = 3$. Hence, a rating $r_{ij} > \epsilon$ is considered as a positive feedback (i.e., label $y_{ij} = 1$), otherwise, it is considered as a negative feedback (i.e., label $y_{ij} = -1$). The Yahoo! user set is used as a training set in a biased environment (S_c). For Yahoo! random set, we randomly split the user-item interactions into three subsets: 5% for training in an unbiased environment (S_t), 5% for validation to tune the hyper-parameters (S_{va}), and the rest 90% for test (S_{te}).

- **Product**: This is a large-scale dataset for CTR prediction, which includes three weeks of users' click records from a real-world advertising system. The first two weeks' samples are used for training and the next week's samples for test. To eliminate the effects of the bias problems in our experiments, we only filter out the samples at positions 1 and 2. There exists two polices in this dataset: non-uniform policy and uniform policy which are defined in Section 3.1. We can thus separate this dataset into two parts, i.e., a uniform data and a non-uniform data. The non-uniform data contains around 29 million records and 2.8 million users, which is directly used as a training set named as S_c. Next, we randomly split the uniform data into three subsets by the same way as that of Yahoo! R3, i.e., 5% as training set (S_t), 5% as validation set (S_{va}), and the rest as test set (S_{te}).

Table 3: Statistics of the datasets. P/N represents the ratio between the numbers of positive and negative feedback.

	Yahoo! R3		Product	
	#Feedback	P/N	#Feedback	P/N
S_c	311,704	67.02%	29,255,580	2.12%
S_t	2,700	9.36%	20,751	1.57%
S_{va}	2,700	8.74%	20,751	1.42%
S_{te}	48,600	9.71%	373,522	1.48%

5.1.2 *Evaluation Metrics.* Following the settings of the previous works [5, 33], we employ two evaluation metrics that are widely used in industry recommendation, including the negative logarithmic loss (NLL) and the area under the roc curve (AUC). The NLL evaluates the performance of the predictions,

$$\mathrm{NLL} \equiv -\frac{1}{L} \sum_{(i,j)\in\Omega}^{L} \log\left(1 + e^{-y_{ij}\hat{y}_{ij}}\right), \tag{9}$$

where Ω denotes the validation set (when tuning the parameters) or the test set (in evaluation), and L denotes the number of feedback in Ω. The AUC evaluates the performance of rankings and is defined as follows,

$$\mathrm{AUC} \equiv \frac{\sum_{(i,j)\in\Omega^+}^{L_p} \mathrm{Rank}_{ij} - \binom{L_p}{2}}{\left(L_p\left(L - L_p\right)\right)}, \tag{10}$$

where Ω^+ denotes a subset of the positive feedback in Ω, and L_p denotes the number of feedback in Ω^+. Rank_{ij} denotes the rank of a positive feedback (i, j) in all the L feedback, which are ranked in a descending order according to their predicted values. Note that most users in the validation set S_{va} and test set S_{te} may only have negative samples.

5.1.3 *Baselines.* To demonstrate the effectiveness of our proposed framework, we include with the following baselines which are widely used in recommendation scenarios.

Low Rank Baselines:

Biased Matrix Factorization (biasedMF). We first consider the case where the proposed framework is implemented using a low-rank model. We use biased matrix factorization (biasedMF) [12] as the baseline, which is one of the most classic basic models in recommender systems. In this method, a user i's preference for an item j is formalized as $\hat{Y}_{ij} = U_i^T V_j + bu_i + bv_j$. We directly learn user, item and bias representations using the squared loss. All strategies in the framework are implemented when M_c and M_t are a biasedMF model.

Inverse-Propensity-Scored Matrix Factorization (IPS-MF). To test and compare the performance of the propensity-based causal inference, we use a representative counterfactual-based recommendation method as the second low-rank baseline, i.e., IPS-MF [24]. Note that we estimate the propensity scores via the naïve Bayes estimator,

$$P\left(O_{i,j} = 1 | Y_{i,j} = y\right) = \frac{P\left(Y = y, O = 1\right)}{P\left(Y = y\right)}, \tag{11}$$

where $y = \{-1, 1\}$ is the label, $P(Y = y, O = 1)$ denotes the ratio of the feedback labeled as y in the observed feedback, and $P\left(Y = y\right)$ denotes the ratio of the feedback labeled as y in an unbiased set. They are counted by $S_c \cup S_t$ and S_t, respectively, and the subscripts are dropped to reflect that the parameters are tied across all i and j.

Neural Networks Baselines:

AutoEncoder (AE). We next consider the case where the proposed framework is implemented using a neural network model. We choose the autoencoder as the baseline to include more model choices. Except for the hint and soft label strategies where we use a five-layer autoencoder, we use the original three-layer autoencoder by default. All strategies in the framework are also implemented when M_c and M_t are an autoencoder model. Note that in the FeatE-*user* strategy, we use the weights of the first layer of the autoencoder, and in the FeatE-*item* strategy, we use the weights of the last layer of the autoencoder.

Deep Logistic Regression (DLR). Since the DGBR model used in feature-based distillation requires logistic regression components, autoencoder are not suitable. Hence, we use DLR as a baseline in feature-based distillation. This approach consists of two parts: i) deep autoencoder model, which reconstructs the input-vectors in a high-dimensional space and encodes it into low-dimensional codes, and ii) logistic regression model, which handles the manual feature codes and optimizes the model parameters. Considering the odds of deep autoencoder on non-linear dimensionality reduction, we employ it to convert the high-dimensional data into some low-dimensional codes by defining a three-level encoder network and a three-level decoder network. Then we feed the output of this deep autoencoder model to the LR model.

5.1.4 *Implementation Details.* We implement all the methods on TensorFlow[1]. We perform grid search to tune the hyper-parameters for the candidate methods by evaluating the AUC on the validation

[1]https://www.tensorflow.org

set S_{va}, where the range of the values of the hyper-parameters are shown in Table 4.

Table 4: Hyper-parameters tuned in the experiments.

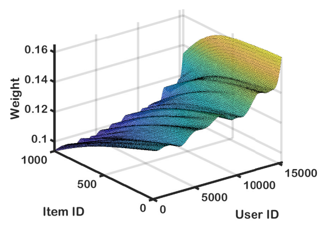
Name	Range	Functionality
$rank$	$\{10, 50, 100, 200\}$	Embedded dimension
λ	$\{1e^{-5}, 1e^{-4} \cdots 1e^{-1}\}$	Regularization
α	$\{0.1, 0.2 \cdots 0.9\}$	Loss weighting
l	$\{2^5, 2^6 \cdots 2^9\}$	Batch size
s	$\{1, 3, 5 \cdots 19, 20\}$	Alternating steps
τ	$\{2, 5, 10, 20\}$	Temperature

5.2 RQ1: How Do the Proposed Methods Perform Against Baselines?

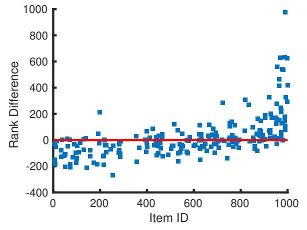
The comparison results are shown in Table 5 and Table 6. Because feature-based distillation requires a special baseline DLR as described in Section. 5.1.3, we list its results separately in Table 6. As shown in the tables, our methods perform better than all the compared methods in most cases. More specifically, we have the following observations: (1) The sample combination of S_c and S_t improves the performance in all cases. The propensity-based method and the method using only S_t have similar performance, i.e., they have superior NLL and uncompetitive AUC on Yahoo! R3, but on Product, their NLL will also deteriorate. One possible reason is that S_c and S_t of Product dataset have a close ratio between the positive and negative feedback. (2) The trends of AUC and NLL metrics may be inconsistent. For example, some of our strategies have a better AUC value but a poor NLL value, while the uniform strategy is the opposite. Since the NLL value is susceptible to the difference in label distribution between the training and test sets, we mainly consider AUC. (3) Most of the bad cases of our proposed methods appear in the feature embedding strategy. This may be because the feature embedding in S_t is not sufficiently trained as described in Section 4.4. We can also see that FeatE-*alter* can effectively alleviate this issue. In addition, a special bad case appears when using WeightC-*local* on Product. We think it is still a challenge that modeling the local weights with a large-scale dataset. (4) The improvements brought by all the proposed strategies vary in different model implementations and different data scales. It means that each strategy's ability to use S_t depends on distinct scenarios. We will conduct in-depth research on some strategies separately in the future.

5.3 RQ2: How Do S_t Improve the Model Trained on S_c?

To explore the form of the useful knowledge provided by S_t, we conduct an in-depth analysis using the first three best strategies implemented with low-rank models on the Yahoo! R3 dataset as an example, i.e., WeightC-*local*, DelayC and Refine. Figure 4(a) shows a visualization of the weight parameters learned from the WeightC-*local* strategy. User IDs and item IDs are sorted in ascending order w.r.t. the user activity and item popularity, respectively. As the item popularity increases, the weight value decreases, and this trend will gradually be weaken as the user activity increases. This means that the useful knowledge provided by S_t is to enhance the contribution of the active users and tail items.



(a)



(b)

Figure 4: (a) Visualization of the weight parameters learned by the WeightC-*local* strategy. (b) The ranking difference between the Refine strategy and the Base strategy for the positive samples in the validation set.

The original DelayC strategy randomly samples one batch of data from S_t to guide M_c. We can control the sampling method to analyze the efficacy of different types of data. Table 7 shows the results under different sampling methods. The *head users* refers to that we only sample the data corresponding to the first 50% of the most active users in S_t, while the *tail users* means that the last 50% of users are sampled. The *head items* and *tail items* are defined in a similar way. We find that although the performance of the four sampling methods is not as good as random sampling, the *head users* and *tail items* are closer to the performance of random sampling than the other two sampling methods. This is consistent with the findings of Figure 4(a).

Finally, we examine the ranking difference between the Refine strategy and the Base strategy for positive samples in the validation set. The results are shown in Figure 4(b). Item IDs are sorted in the ascending order of popularity. We find that the Refine strategy follows the intuition that a popular item is more likely to get feedback than a tail item. It tries to lower the ranking of tail items that may be recommended to the top and raise the ranking of popular items that may be recommended to the tail, as shown on the both sides of Figure 4(b). In the middle of Figure 4(b), we find that the rank difference of most items is not large, which means that a less popular item still has an opportunity to catch up with a more popular item. Since the Refine strategy achieves the best performance, we believe it is a good strategy to combine the advantages of S_c and S_t.

6 FUTURE WORKS

We have proposed some approaches about how to mine some useful knowledge from a uniform data to improve the modeling of a non-uniform data. Counterfactual recommendation via a uniform data is still a rich research field. In this section, we discuss some interesting and promising future directions.

Label-Based Module. Because S_t collected from different scenarios may have different label distributions, the distribution difference between S_t and S_c can be large or small. It is necessary to design some more robust strategies for addressing the difference.

Table 5: Comparison results on Yahoo! R3 and Product except for the feature-based distillation, which are reported in Table 6. (∪) in the Strategy column indicates that the used data is $S_c \cup S_t$.

		Yahoo! R3				Product			
		Low Rank (MF)		Neural Nets (AE)		Low Rank (MF)		Neural Nets (AE)	
Module	**Strategy**	**AUC**	**NLL**	**AUC**	**NLL**	**AUC**	**NLL**	**AUC**	**NLL**
Baseline	Base (S_c)	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%
	Uniform (S_t)	-21.76%	+13.43%	-24.88%	**+11.78%**	+0.68%	-43.98%	+2.65%	+33.17%
	Combine (∪)	+0.27%	+1.30%	+0.38%	+0.84%	+0.62%	+1.00%	+2.31%	+2.31%
	Propensity (∪)	-0.71%	**+23.86%**	—	—	-9.17%	-110.48%	—	—
Label	Bridge (∪)	+0.48%	+2.74%	+1.02%	+4.86%	**+8.74%**	-12.02%	+4.54%	**+36.85%**
	Refine(∪)	**+1.50%**	-12.09%	+0.56%	-0.45%	+0.22%	-0.46%	+2.58%	+25.70%
Sample	CausE (∪)	+0.22%	+1.07%	—	—	+3.85%	+1.87%	—	—
	WeightC-*local* (∪)	+0.68%	+6.62%	+0.65%	+1.40%	-13.28%	+2.10%	-1.76%	+18.23%
	WeightC-*global* (∪)	+0.54%	+3.50%	+0.92%	+1.84%	+6.22%	**+2.54%**	+5.59%	+25.63%
	DelayC (∪)	+0.74%	+5.10%	+0.49%	+0.88%	+1.62%	+1.48%	**+6.06%**	+20.02%
Model Structure	FeatE-*item* (∪)	-0.03%	+0.41%	-1.00%	-0.50%	-1.07%	+1.10%	-3.00%	+22.34%
	FeatE-*user* (∪)	+0.11%	+0.36%	+0.43%	+2.02%	-0.27%	-10.96%	-2.15%	+2.16%
	FeatE-*both* (∪)	+0.34%	+1.46%	-1.58%	-1.15%	+0.91%	-42.63%	+2.36%	-0.08%
	FeatE-*alter* (∪)	+0.59%	+2.70%	+0.83%	+1.86%	+0.72%	-41.78%	+4.37%	+2.14%
	FeatE-*concat* (∪)	+0.34%	+1.46%	+0.05%	+1.80%	+0.54%	-41.98%	+5.07%	+34.83%
	Hint (∪)[a]	—	—	+1.04%	-6.20%	—	—	+2.80%	-56.87%
	Soft Label (∪)[a]	—	—	**+1.10%**	+3.34%	—	—	+3.84%	-43.73%

[a] Note that since these strategies rely on deep networks, we use the deep version of the base strategy as a reference to report the results, which is Deep AutoEncoder.

Table 6: Comparison results of the feature-based distillation.

Module	Strategy (DLR)	Yahoo! R3		Product	
		AUC	Logloss	AUC	Logloss
Feature	Base (S_c)	+0.00%	+0.00%	+0.00%	+0.00%
	Uniform (S_t)	-11.61%	+73.14%	-30.26%	-26.64%
	Combine (∪)	+0.69%	+0.06%	+0.66%	+0.88%
	DGBR (∪)	+1.66%	+32.50%	+1.64%	+0.96%

Table 7: Comparison results of the DelayC strategy with different ways of constructing the batch data from S_t.

Strategy	Sampling Method	AUC	NLL
DelayC	Random	0.7329	-0.5590
	Head Users	0.7303	-0.5706
	Tail Users	0.7251	-0.5630
	Head items	0.7252	-0.5740
	Tail items	0.7306	-0.5599

We can learn different imputation models with S_t, among which one promising direction is about how to ensemble the imputed labels from different imputation models to correct the labels of the biased samples. How to better combine the imputed label with the true label of S_c in a more sophisticated manner is another promising direction.

Feature-Based Module. The current stable feature approach [13] needs much time and computing resources. For implementing the industry recommender system, we need more efficient methods to learn the stable features. Besides, the current approach only makes use of the feature information in each sample to learn the stable features, while the label in each sample from S_t is more stable and unbiased. So how to filter out the stable features with both labels and features in S_t is another interesting research question.

Sample-Based Module. The difference between the data size of S_t and that of S_c is a challenge for sample-based methods. This difference increases the difficulty of model training, e.g., M_t may converge faster than M_c because the number of S_t is much smaller. A large difference in the number means that S_t has very little corrective effect on S_c, which may also weaken the guiding role of S_t. One promising direction is to use the information in S_t to filter out a more unbiased subset from S_c, or use the information in S_c to perform data augmentation on S_t. Instead of using the label information, another promising direction is that we can consider modeling the preference ranking relation between S_t and S_c.

Model Structure-Based Module. The feature embeddings obtained by S_t are often not fully trained due to the size of S_t. A promising direction is to design a good mutual learning strategy for M_t and M_c instead of pre-training M_t before using it to train M_c. The current distillation structure selection methods are based on enumeration or empirical methods. How to effectively design a good distillation structure is another promising direction, for which AutoML has the potential to find a reasonable model structure based on S_t.

Others. There are also many other directions closely related to the framework. For example, the visualization or interpretation of the useful information (or knowledge) learned from S_t; further exploration of the results at a micro level, i.e., the impact on each user or each item; and the relation between the size of S_t and the performance of the model. In addition, we would like to further investigate the trade-off of training on S_c introduced by S_t and gain more theoretical insight into why it is effective. These theoretical insights can also inspire us to design better distillation strategies.

7 CONCLUSIONS

In this work, motivated by the observation that simply modeling with a uniform data can alleviate the bias problems, we propose a general knowledge distillation framework for counterfactual recommendation via uniform data, i.e., KDCRec, including label-based, feature-based, sample-based and model structure-based distillations. We conduct extensive experiments on both public and product datasets, demonstrating that the proposed four methods can achieve better performance over the baseline models. We also analyze the proposed methods in depth, and discuss some promising directions worthy of further exploration.

ACKNOWLEDGMENTS

The authors would like to thank Mr. Bowen Yuan for his helpful comments and discussions, and Prof. Kun Kuang from Tsinghua University for providing the source code of the DGBR algorithm. This work is supported by the National Natural Science Foundation of China Nos. 61872249, 61836005 and 61672358.

REFERENCES

[1] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. 2017. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the 11th ACM Conference on Recommender Systems*. 42–46.

[2] Aman Agarwal, Kenta Takatsu, Ivan Zaitsev, and Thorsten Joachims. 2019. A general framework for counterfactual learning-to-rank. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 5–14.

[3] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. 2019. Estimating position bias without intrusive interventions. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 474–482.

[4] Hessam Bagherinezhad, Maxwell Horton, Mohammad Rastegari, and Ali Farhadi. 2018. Label refinery: Improving imagenet classification through label progression. *arXiv preprint arXiv:1805.02641* (2018).

[5] Stephen Bonner and Flavian Vasile. 2018. Causal embeddings for recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 104–112.

[6] Rocío Cañamares and Pablo Castells. 2018. Should I follow the crowd?: A probabilistic analysis of the effectiveness of popularity in recommender systems. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. 415–424.

[7] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601* (2011).

[8] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. 2017. Efficient knowledge distillation from an ensemble of teachers. In *Interspeech*. 3697–3701.

[9] James J Heckman. 1979. Sample selection bias as a specification error. *Econometrica: Journal of the Econometric Society* (1979), 153–161.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[11] Nathan Kallus, Aahlad Manas Puli, and Uri Shalit. 2018. Removing hidden confounding by experimental grounding. In *Advances in Neural Information Processing Systems*. 10888–10897.

[12] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[13] Kun Kuang, Peng Cui, Susan Athey, Ruoxuan Xiong, and Bo Li. 2018. Stable prediction across unknown environments. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1617–1626.

[14] Carolin Lawrence, Artem Sokolov, and Stefan Riezler. 2017. Counterfactual learning from bandit feedback under deterministic logging: A case study in statistical machine translation. *arXiv preprint arXiv:1707.09118* (2017).

[15] Dugang Liu, Chen Lin, Zhilin Zhang, Yanghua Xiao, and Hanghang Tong. 2019. Spiral of silence in recommender systems. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 222–230.

[16] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 583–592.

[17] Benjamin M Marlin and Richard S Zemel. 2009. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the 3rd ACM Conference on Recommender systems*. 5–12.

[18] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[19] Weike Pan. 2016. A survey of transfer learning for collaborative recommendation with auxiliary data. *Neurocomputing* 177 (2016), 447–453.

[20] Weike Pan, Hao Zhong, Congfu Xu, and Zhong Ming. 2015. Adaptive Bayesian personalized ranking for heterogeneous implicit feedbacks. *Knowledge-Based Systems* 73 (2015), 173–180.

[21] Nikolaos Passalis and Anastasios Tefas. 2018. Learning deep representations with probabilistic knowledge transfer. In *Proceedings of the 15th European Conference on Computer Vision*. 268–284.

[22] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[23] Nir Rosenfeld, Yishay Mansour, and Elad Yom-Tov. 2017. Predicting counterfactuals from large historical data and small randomized trials. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 602–609.

[24] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. 1670–1679.

[25] Adith Swaminathan and Thorsten Joachims. 2015. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research* 16, 1 (2015), 1731–1755.

[26] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2289–2298.

[27] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset distillation. *arXiv preprint arXiv:1811.10959* (2018).

[28] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. 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*. 610–618.

[29] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. 2019. Doubly robust joint learning for recommendation on data missing not at random. In *Proceedings of the 36th International Conference on Machine Learning*. 6638–6647.

[30] Chen Xu, Quan Li, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Hanxiao Sun, and Wenwu Ou. 2019. Privileged features distillation for e-commerce recommendations. *arXiv preprint arXiv:1907.05171* (2019).

[31] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 279–287.

[32] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*. 4133–4141.

[33] Bowen Yuan, Jui-Yang Hsia, Meng-Yuan Yang, Hong Zhu, Chih-Yao Chang, Zhenhua Dong, and Chih-Jen Lin. 2019. Improving ad click prediction by considering non-displayed events. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 329–338.

[34] Yuan Zhang, Xiaoran Xu, Hanning Zhou, and Yan Zhang. 2020. Distilling structured knowledge into embeddings for explainable and accurate recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 735–743.