AutoDenoise: Automatic Data Instance Denoising for Recommendations

Weilin Lin
City University of Hong Kong
mlmr.lin@gmail.com

Xiangyu Zhao*
City University of Hong Kong
xianzhao@cityu.edu.hk

Yejing Wang
City University of Hong Kong
yejing.wang@my.cityu.edu.hk

Yuanshao Zhu
City University of Hong Kong
Southern University of Science and Technology
zhuys2019@mail.sustech.edu.cn

Wanyu Wang
City University of Hong Kong
wanyuwang4-c@my.cityu.edu.hk

ABSTRACT

Historical user-item interaction datasets are essential in training modern recommender systems for predicting user preferences. However, the arbitrary user behaviors in most recommendation scenarios lead to a large volume of noisy data instances being recorded, which cannot fully represent their true interests. While a large number of denoising studies are emerging in the recommender system community, all of them suffer from highly dynamic data distributions. In this paper, we propose a Deep Reinforcement Learning (DRL) based framework, AutoDenoise, with an Instance Denoising Policy Network, for denoising data instances with an instance selection manner in deep recommender systems. To be specific, AutoDenoise serves as an agent in DRL to adaptively select noise-free and predictive data instances, which can then be utilized directly in training representative recommendation models. In addition, we design an alternate two-phase optimization strategy to train and validate the AutoDenoise properly. In the searching phase, we aim to train the policy network with the capacity of instance denoising; in the validation phase, we find out and evaluate the denoised subset of data instances selected by the trained policy network, so as to validate its denoising ability. We conduct extensive experiments to validate the effectiveness of AutoDenoise combined with multiple representative recommender system models.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Instance Denoising, Recommender System, Reinforcement learning

*Xiangyu Zhao is the corresponding author.

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583339

ACM Reference Format:
Weilin Lin, Xiangyu Zhao, Yejing Wang, Yuanshao Zhu, and Wanyu Wang. 2023. AutoDenoise: Automatic Data Instance Denoising for Recommendations. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3543507.3583339

1 INTRODUCTION

Recommender System (RS) is an essential technology in the era of information explosion, which can deliver a favorable experience for users and considerable economic benefits for companies [8, 13, 16]. As a data-driven technique, RS models user preference based on previous data instances (i.e., interaction logs) [25], thereby substantially improving the quality of online service platforms (e.g., e-commerce sites and streaming video applications) [5, 27]. In General, RS is modeled based on recorded data instances from historical user-item interactions. However, recording user behavior with the above approach inevitably raises challenges for modeling RS. First, the existing recommendation dataset collects all historical data instances indiscriminately, while some of them may be noisy instances that do not reflect the real intention of users [12]. For example, in *click-through rate* (CTR) prediction [3, 9, 17, 28, 31, 43], some click actions may come from user curiosity or mistake. Furthermore, the noisy instances have almost the same traits as noise-free instances [14], which renders a non-trivial task for human experts to separate them in the recording.

To address the above challenges, denoising training data is attracting growing attention from researchers. Recent studies have shown that by using denoising techniques in recommender systems, models can be trained in a more efficient manner with better performance within comparable computational cost [7, 11, 32, 37]. Typically, denoising methods involve a "searching" behavior to figure out noise and a "deciding" behavior to execute denoising actions. According to the methods of denoising, existing efforts can be roughly categorized into two groups: selection-based and reweighting-based methods. For selection-based methods [6, 35], they aim to train a selection network to discard noisy instances, thereby feeding the model with more representative data. For reweighting-based methods [29, 30], they tend to lower the contributions of the noisy instances by assigning lower weights throughout the model training. In practice, they consider the loss values as the indicator to discriminate noise from noise-free instances, i.e., high values imply noisy data instances during training.

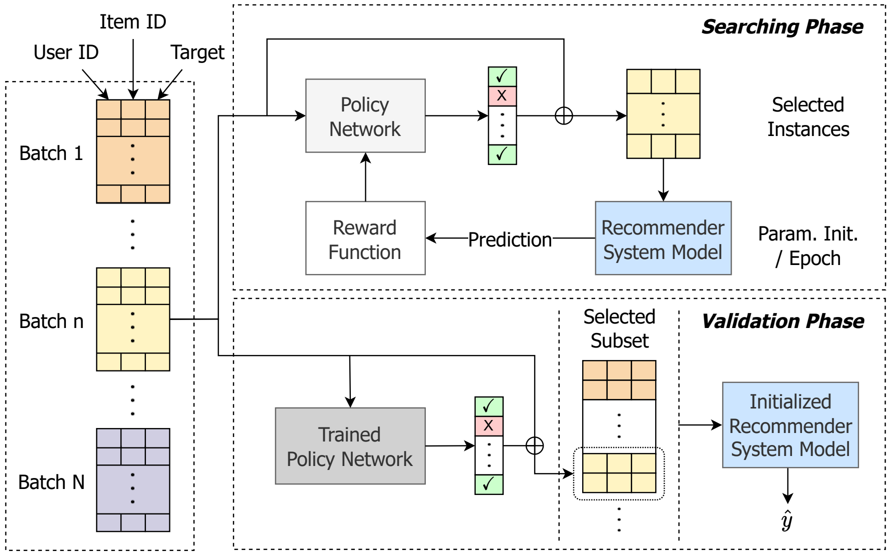

Figure 1: Overview of the proposed framework.

However, the above methods suffer from the following limitations. First, selection-based methods rely heavily on sampling distributions for decision-making [36], while the complex and dynamic data distribution in the contemporary online environment tends to limit their performance. In other words, the more dynamic environment is, the more biased selection may occur. Second, the reweighting-based methods influencing the model training processes lack transferability. Existing reweighting-based methods require specific configurations in the case of a given model or recommendation task, which is time-consuming and challenging to transfer to other settings. Therefore, a promising instance denoising scheme should be robust and easily-transfer.

To bridge this research gap, we propose a Deep Reinforcement Learning (DRL) based instance denoising framework, which can model various complex data distributions of real-world datasets and filter a noise-free subset without noise. We propose this framework based on three main motivations: (i) The reinforcement learning methods are empirically proven to be effective in the optimum-searching problems [18, 19, 38–42], which can also have the potential to distinguish the noisy data instances effectively. (ii) Policy network in DRL is used to select instances from a collection of mini-batch, allowing it to capture fine-grained patterns from various data distributions. Such design exactly facilitates mitigating the biased selection problem of selection-based methods. (iii) Separately training the prediction model and the policy network for the denoising and the prediction processes, owns natural transferability with the noise-free data subset from the denoising policy network.

Nevertheless, applying the DRL-based approach to instance denoising will encounter the following challenges. On the one hand, the data distribution varies dramatically among mini-batches, rendering it challenging to design an appropriate reward function to optimize the policy network. What's more, since the policy network and RS model need to be optimized simultaneously in this setting, it is challenging to train them properly with the same data batch. To address the above challenges, we propose a DRL-based framework, AutoDenoise, with an instance denoising policy network. AutoDenoise scores each instance of a mini-batch by two probabilities for "select/deselect" actions with the policy network, and evaluates the performance of the sampled instances as "select" with the RS model. Meanwhile, we design an instance-level reward function based on the results of the RS model. Specifically, this reward function compares a data instance's current loss with its losses in previous searching epochs, for learning to distinguish the noisy data instances in different distributions. Moreover, we propose an alternate two-phase optimization strategy, i.e., the searching phase and the validation phase, to properly train and validate the policy network. After that, we train a randomly initialized RS model on the selected data subset from scratch to evaluate the effectiveness of the new data subset. The main contributions of this work can be summarized as follows: (i) We propose a DRL-based instance denoising framework, which adaptively filters out noises in each input mini-batch of data instances. To the best of our knowledge, this is a pioneering effort in instance denoising for CTR prediction; (ii) We design a novel two-phase training process to effectively train and validate the policy network, as well as generate a transferable noise-free data subset; (iii) We validate the effectiveness of AutoDenoise on three public benchmark datasets and prove the transferability of the denoised datasets.

2 FRAMEWORK

In this section, we will introduce the overview, the proposed two phases, and the optimization methods of AutoDenoise framework.

2.1 Overview

As illustrated in Figure 1, the whole framework consists of two running phases, i.e., the searching phase and the validation phase.

Searching Phase. In this phase, DRL is adopted to automatically select instances with sequential optimization for the model and policy network. Specifically, the policy network serves as a DRL agent, which performs "select" and "deselect" actions with corresponding

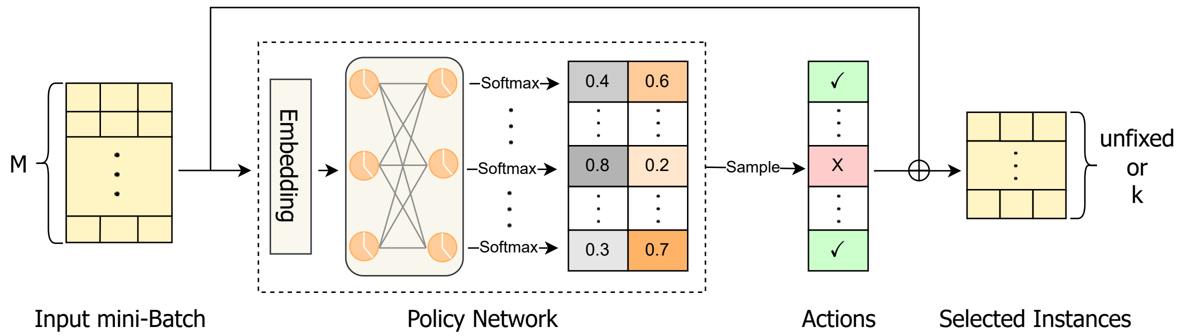

Figure 2: The selection procedure of the policy network.

probabilities, thereby removing noisy instances. Then, the selected noise-free instances are fed into an RS model, and the reward can be calculated by the previous and current losses of the RS model, which are used to optimize the policy network.

Validation Phase. Unlike the searching phase, this phase aims to evaluate the performance of the policy network. In detail, after finishing the searching phase of each epoch, i.e., one iteration on the entire training set, the process switches to the validation phase. In the validation phase, the policy network predicts the entire training set in batches and selects the top-k instances with the highest probability of "select" action, which can be considered as noise-free data. Finally, an initialized RS model is trained with a noise-free subset until convergence, for which the final test performance is presented as the policy network denoising capability.

2.2 Searching Phase

In this section, we focus on introducing core elements of policy networks and DRL construction (e.g., environment, state, rewards, etc.). As an instance denoising task for applying DRL, we consider each mini-batch as the state of the input policy network and then execute the DRL actions ("select" or "deselect") based on the outcome probabilities. DRL interacts with the environment by feeding selected instances into the RS model and calculates the corresponding rewards, aiming to maximize the reward of the policy network. Next, we will dive into the details of these components.

Environment. RS serves as the environment in DRL, which receives a mini-batch of selected instances and outputs the predictions. To fairly evaluate and optimize the policy network, we initialize its parameters at the beginning of each training epoch to make the output predictions comparable with previous ones.

State. The state is the input mini-batch, which consists of a bunch of data instances. Suppose that we have M instances for a mini-batch and N batches in total, the whole training set $\boldsymbol{X} = \left\{\boldsymbol{X}_1^n, ..., \boldsymbol{X}_M^n\right\}_{n=1}^N$, where $\boldsymbol{X}_m^n$ represent the m^{th} instance of the n^{th} mini-batch in the training epoch. To compare the prediction results at the instance level for every training epoch, we fix their sequence order. In other words, all $\boldsymbol{X}_m^n$ have an identical position throughout the training process. Since the state is the input mini-batch, the n^{th} mini-batch can be expressed in the general form, i.e., $s_n = [\boldsymbol{X}_1^n \cdots \boldsymbol{X}_m^n \cdots \boldsymbol{X}_M^n]$. The function of the policy network is to score each instance in s_n with probability and determine the "select/deselect" action.

Policy and Action. The policy and action can be considered as the "searching" and "deciding" behaviors in the instances of denoising, respectively. Therefore, we propose a policy network to search for the noise-free instances based on the state s_n and execute the action a_m^n for each instance in the mini-batch s_n. The whole selection process is illustrated in Figure 2. To be specific, given a batch of data $s_n = [\boldsymbol{X}_1^n \cdots \boldsymbol{X}_m^n \cdots \boldsymbol{X}_M^n]$, the policy network first transforms each instance $\boldsymbol{X}_m^n$ to a dense vector $\boldsymbol{E}_m^n$ through the embedding lookup operation,

$$\boldsymbol{E}_m^n = \boldsymbol{\mathcal{A}} \boldsymbol{X}_m^n \tag{1}$$

where $\boldsymbol{\mathcal{A}}$ consists of the learnable weight matrixes for all feature fields[1]. $\boldsymbol{E}_m^n$ denotes the concatenated feature embedding vector corresponding to $\boldsymbol{X}_m^n$. Subsequently, $\boldsymbol{E}_m^n$ is fed to the Multilayer perceptron (MLP) with the nonlinear activation to obtain a fine-grained feature representation [23]. Suppose that the MLP has $\{L + 1\}$ layers in total, the above operation of l^{th} ($l \in [1, L]$) layer can be formulated as:

$$\boldsymbol{h}_l = \sigma\left(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l\right) \tag{2}$$

where $\boldsymbol{h}_l$ is the output of the l^{th} layer. $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ are the learnable weight matrix and bias vector for the corresponding layer, respectively, and $\sigma(\cdot)$ denotes the nonlinear activation function. In general, we use *ReLu* as the activation function for hidden layers. Then, we apply the $Softmax$ function to the output layer, so as to retrieve the probability of "select" and "deselect" actions for every instance. In this setting, the action a_m^n can be sampled from the output probabilities. To reminder, $\boldsymbol{h}_0 = \boldsymbol{E}_m^n$ represent the first fully-connected layer and the last layer $\boldsymbol{h}_L$ is the output layer.

Reward. To optimize the policy network in the desired way, i.e., learning to distinguish the noise instances, the reward design should be associated with the performance of the noise instance identification model. Following this idea, we define the reward according to the prediction error of the RS model, i.e., the value of loss L comparing the prediction results and the corresponding ground-truth labels. In order to avoid optimization conflicts among data instances, we consider each instance separately in the training search phase. Specifically, we create a matrix $\boldsymbol{LMat} \in \mathbb{R}^{C \times MN}$ to store the loss values of previous and current searching epochs for each instance, where C and MN are the numbers of stored epochs and the number of loss values. Given the loss L_{mn} (m^{th} instance of

[1]A feature field is a group of feature values belonging to the same category, e.g., the feature field "gender" comprises two values, "female" and "male".

the n^{th} batch) for the current epoch, the reward R_{mn} is defined as the difference between the corresponding instance's averaged loss values in the past C epochs and the loss in the current epoch:

$$R_{mn} = \frac{1}{C}\sum_{c=1}^{C} L_{mn}^{c} - L_{mn} \tag{3}$$

where $R_{mn} > 0$ means that the data instance's loss in the current epoch is smaller than its averaged loss values in the past C epochs, indicating that the policy network conducts better action on the instance in the current epoch, and vice versa. For fair comparison and improvement incentive, we initialize the parameters of the RS model at the beginning of each training epoch and overwrite the loss values of the earliest stored epoch with the latest one. Under this setting, the reward can be viewed as the prediction improvement. Since the parameters of the policy network are different, the reward can reflect the optimization results and be used to adjust them. Considering the action-sampling behaviors of policy network would introduce randomness, which cannot represent its actual selection ability, we choose to use the average value of L_{mn}^{c} in Equation (3) to make the current reward R_{mn} more reliable.

2.3 Validation Phase

As mentioned in the training phase, the RS model is initialized in each training epoch and usually not converged. Therefore, we design a validation phase to fully evaluate the performance of the policy network and collect the best noise-free subset. To implement this, we split the validation phase into noise-free subset selection and RS model training. For subset selection, we iterate the entire training set and select the data instance with k highest confidence in each mini-batch, i.e., the top-k selection strategy. After obtaining the selected subset, we initialize the parameters of the RS model and train it until convergence, which is alignment with the typical recommendation system training setup. Finally, with the same test and validation sets as in the searching phase, we can evaluate the quality of the selected subsets and the denoising ability of the trained policy network[2]. Only the policy network and noise-free subset with the best testing performance can be adopted.

Individual Selection & Top-k Selection. Compared with the individual selection used in the searching phase, we adopt top-k selection in the validation phase, i.e., we select the k prediction instances with the highest confidence by a trained policy network for each mini-batch. The motivation for this scheme is individual selection provides a "local" view for policy network optimization, while the top-k can offer a "global" view enabling the selection of a more representative noise-free subset. From a policy training perspective, a "local" view at the instance level enables the policy network to learn the effectiveness of each instance and filter out noisy instances. Nevertheless, the "local" view may not be the best choice for decision-making, because the contribution of weakly-selected instances may be minor compared to strongly-selected instances during model training, which may affect the model that uses the average of the loss values for optimization. In this sense, weakly-selected instances can also be considered as a kind of noise, requiring a "global" constraint from the top-k selection strategy.

[2]In practice, we set a total of 50 training epochs in the searching and validation phases to train and evaluate the policy network.

Algorithm 1 *Opt_I*: Optimization Algorithm for Searching Phase

Input: N mini-Batches of data $\boldsymbol{X} = \left\{X_1^n, ..., X_M^n\right\}_{n=1}^{N}$, policy network $\boldsymbol{\Theta}$, initial RS model $\boldsymbol{W}$, loss matrix $\boldsymbol{LMat}$
Output: trained policy network $\boldsymbol{\Theta}^*$ and full loss vector $\boldsymbol{LVec}^*$

Create empty $\boldsymbol{LVec}$
Initialize $\boldsymbol{W}$
for all $n \in [1, N]$ **do**
  $s_n \leftarrow [X_1^n \cdots X_m^n \cdots X_M^n]$
  Sample $a_1 \sim \Theta(a \mid s_n)$
  Select instances $\boldsymbol{X}_{slct1}$
  Estimate $\mathcal{L}_{model}(\boldsymbol{W})$ by Equation (4)
  Estimate R_n by Equation (3) with $\boldsymbol{LMat}$
  Update $\boldsymbol{\Theta}$ by Equation (8)
  Sample $a_2 \sim \Theta(a \mid s)$
  Select instances $\boldsymbol{X}_{slct2}$
  Estimate $\mathcal{L}_{model}(\boldsymbol{W})$ by Equation (4)
  Append $\mathcal{L}_{model}(\boldsymbol{W})$ to $\boldsymbol{LVec}$
  Update $\boldsymbol{W}$ by Equation (5)
end for
$\boldsymbol{\Theta}^* \leftarrow \boldsymbol{\Theta}$

2.4 Optimization Method

With the working procedures mentioned above, we will further detail the optimization methods for updating both the RS model and the policy network in this section.

2.4.1 Recommender System Model. Since RS models in the searching phase and validation phase are identical, we introduce model parameters and the optimization method together. We denote the parameters of the RS model as $\boldsymbol{W}$. The CTR prediction could be regarded as a binary classification task under a supervised manner, where the input instances contain both features and the label. Thus, we apply *binary-cross-entropy loss* function $\mathcal{L}_{model}$ to optimize $\boldsymbol{W}$:

$$\mathcal{L}_{model}(\boldsymbol{W}) = -\frac{1}{M}\sum_{m=1}^{M}\left[y_m \log \hat{y}_m + (1 - y_m)\log(1 - \hat{y}_m)\right] \tag{4}$$

where y_m and $\hat{y}_m$ are the ground truth label and the probability predicted by the RS model for the m^{th} instance in a mini-batch, respectively. As we described in Section 2.2, the loss value can be treated as the performance of the input selected instances in the searching phase. By minimizing the Equation (4), we obtain well-trained $\boldsymbol{W}$ as follows, where α is learning rate:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} \mathcal{L}_{model}(\boldsymbol{W}) \tag{5}$$

2.4.2 Policy Network. As it is a reinforcement learning framework, the optimal policy network can be parameterized by $\boldsymbol{\Theta}$. Given the reward function, i.e., Equation (3), the objective function required optimization is formulated as:

$$\max_{\Theta} J(\Theta) = \mathbb{E}_{a \sim \Theta(a|s)}\left[R(a \mid s)\right] \tag{6}$$

where $a \sim \Theta(a \mid s)$ means an action a is sampled via the policy $\Theta(a \mid s)$ in state s. To maximize the above function, we apply a policy gradient algorithm, named REINFORCE [33], and adopt Monte-Carlo sampling to simplify the estimation of the gradient

Algorithm 2 *Opt_II*: Optimization Algorithm for Validation Phase

Input: N mini-Batches of data $\boldsymbol{X} = \left\{\boldsymbol{X}_1^n, ..., \boldsymbol{X}_M^n\right\}_{n=1}^N$, trained policy network Θ^*, initial RS model $\boldsymbol{W}$
Output: Selected data subset $\boldsymbol{X}_{sub}$, well-trained RS model $\boldsymbol{W}^*$

1: Create empty $\boldsymbol{X}_{sub}$
2: Initialize $\boldsymbol{W}$
3: **for all** $n \in [1, N]$ **do**
4: $\quad s_n \leftarrow [\boldsymbol{X}_1^n \cdots \boldsymbol{X}_m^n \cdots \boldsymbol{X}_M^n]$
5: $\quad$ Sample $a_3 \sim \Theta^*(a \mid s_n)$
6: $\quad$ Select instances $\boldsymbol{X}_{slct3}$
7: $\quad$ Append $\boldsymbol{X}_{slct3}$ to $\boldsymbol{X}_{sub}$
8: **end for**
9: **while** RS model $\boldsymbol{W}$ not converged **do**
10: $\quad$ Sample a batch of data from $\boldsymbol{X}_{sub}$
11: $\quad$ Update $\boldsymbol{W}$ by Equation (5)
12: **end while**

$\nabla_{\Theta} J(\Theta)$, which can be expressed as:

$$\begin{aligned}
\nabla_{\Theta} J(\Theta) &= \sum_a R(a \mid s) \nabla \Theta(a \mid s) \\
&= \sum_a R(a \mid s) \Theta(a \mid s) \nabla \log \Theta(a \mid s) \\
&= \mathbb{E}_{a \sim \Theta(a|s)} [R(a \mid s) \nabla \log \Theta(a \mid s)] \\
&\approx \frac{1}{T} \sum_{t=1}^{T} R(a \mid s) \nabla \log \Theta(a \mid s)
\end{aligned} \tag{7}$$

where T is the sample number, and we set $T = 1$ here to improve the computational efficiency. The parameters Θ can be updated as:

$$\Theta \leftarrow \Theta + \beta \nabla_{\Theta} J(\Theta) \tag{8}$$

where β is the learning rate.

2.4.3 Optimization Algorithm for Searching Phase. Based on the optimization methods for the RS model and the policy network mentioned above, we can conduct the two-phase training to optimize and evaluate the proposed framework. The searching phase is illustrated in Algorithm 1. In this phase, we iterate the whole training set and alternatively optimize Θ and $\boldsymbol{W}$ to obtain a well-performed policy. For a fair comparison of the reward, we initialize the RS model $\boldsymbol{W}$ at the beginning of this phase (line 2). During the training epoch, we first sample a mini-batch of data instances as the state s_n for the policy network (line 4). Next, we sample actions a_1 for every data instance in the batch (line 5) and collect them as a selective batch $\boldsymbol{X}_{slct1}$ (line 6), which will be fed into the RS model and gain the corresponding loss values (line 7). With this loss value and the saved ones from previous epochs[3], i.e., in $\boldsymbol{LMat}$, we can calculate the reward (line 8) and update our policy network (line 9). Similarly, the optimization of the RS model (line 10-14) follows a similar procedure as in line 5 to line 9. However, we sample new actions a_2 according to the updated Θ (line 10), and update $\boldsymbol{W}$ (line 14) rather than Θ as line 9. The updated policy network in the final mini-batch will be returned as a well-trained policy network (line 16). What's more, to record loss values for future calculation of reward baseline, we create an empty loss vector $\boldsymbol{LVec}$ (line 1) and record the test loss value (line 13) during the iteration.

[3]For the first C epochs in searching phase, where previous C losses are now available, we design a wart-up train method in Section 2.4.5.

Algorithm 3 Overall Optimization Algorithm of AutoDenoise

Input: N mini-Batches of data $\boldsymbol{X} = \left\{\boldsymbol{X}_1^n, ..., \boldsymbol{X}_M^n\right\}_{n=1}^N$, Warm-up epoch C, Training epoch T, empty loss matrix $\boldsymbol{LMat} = [\boldsymbol{LVec}_1, ..., \boldsymbol{LVec}_C]$, initial RS model $\boldsymbol{W}$ and policy network Θ
Output: well-trained policy network parameters Θ^* and the selected data subset $\boldsymbol{X}_{sub}$

1: **for** $i = 1; i <= C; i++$ **do**
2: $\quad$ Initialize $\boldsymbol{W}$
3: $\quad$ **while** RS model $\boldsymbol{W}$ not converged **do**
4: $\quad\quad$ Sample a batch of data $[\boldsymbol{X}_1^n \cdots \boldsymbol{X}_m^n \cdots \boldsymbol{X}_M^n]$
5: $\quad\quad$ Estimate $\mathcal{L}_{model}(\boldsymbol{W})$ by Equation (4)
6: $\quad\quad$ Append $\mathcal{L}_{model}(\boldsymbol{W})$ to $\boldsymbol{LVec}_i$
7: $\quad\quad$ Update $\boldsymbol{W}$ by Equation (5)
8: $\quad$ **end while**
9: **end for**
10: **for** $t = 1; t <= T; t++$ **do**
11: $\quad \Theta^*, \boldsymbol{LVec}^* \leftarrow Opt\_I(\boldsymbol{X}, \Theta, \boldsymbol{W}, \boldsymbol{LMat})$
12: $\quad \boldsymbol{LVec}_{t\%C} \leftarrow \boldsymbol{LVec}^*$
13: $\quad \boldsymbol{X}_{sub}, \boldsymbol{W}^* \leftarrow Opt\_II(\boldsymbol{X}, \Theta^*, \boldsymbol{W})$
14: $\quad$ Test $\boldsymbol{W}^*$
15: $\quad$ **if** $\boldsymbol{W}^*$ perform the best **then**
16: $\quad\quad$ Save $\boldsymbol{X}_{sub}$ and Θ^*
17: $\quad$ **end if**
18: **end for**

Specifically, the policy network Θ from the previous optimization step cannot represent the optimized performance of the current batch of data. We utilize every mini-batch of data twice in this phase to sequentially optimize Θ and $\boldsymbol{W}$ (line 9 and 14), thus avoiding disturbing the returning loss and the optimizing environment.

2.4.4 Optimization Algorithm for Validation Phase. In Algorithm 2, we demonstrate the whole procedure of the validation phase, including the selection process (line 3-8) and the full-training process of the RS model (line 9-12). In this phase, we aim to evaluate the true performance of the trained policy network Θ^* by training the RS model to converge on the selected subset. Specifically, we first create an empty set $\boldsymbol{X}_{sub}$ (line 1) and initialize $\boldsymbol{W}$ (line 2). Then, we iterate the whole training set for subset selection with Θ^*. Similar to the procedure in the searching phase, every mini-batch will follow the sampling (line 5) and selection (line 6) processes. Then the obtained noise-free data batch $\boldsymbol{X}_{slct3}$ is appended to $\boldsymbol{X}_{sub}$ (line 7). After that, we will train the initialized $\boldsymbol{W}$ to converge with $\boldsymbol{X}_{sub}$ (line 9-12). And we consider the performance of the well-trained $\boldsymbol{W}^*$ as our validation result.

2.4.5 Overall Optimization Algorithm. With the optimization algorithms for the searching phase and validation phase, we will further detail the overall optimization algorithm in this subsection. The whole algorithm is depicted in Algorithm 3.

Warm-up Train. To build the loss matrix $\boldsymbol{LMat}$ for reward calculation, we conduct a Warm-up Train before the searching phase. Specifically, only the RS model will be trained under the

Table 1: Overall performance with different backbone recommendation models.

Backbone Model	Metrics	FM		Wide & Deep		DCN		IPNN		DeepFM	
		w/o	**w**	**w/o**	**w**	**w/o**	**w**	**w/o**	**w**	**w/o**	**w**
MovieLens1M	AUC ↑	0.8080	**0.8095***	0.7957	**0.8022***	0.8092	**0.8115***	0.8023	**0.8063***	0.8048	**0.8097***
	Logloss ↓	0.5334	**0.5292***	0.5400	**0.5333***	0.5235	**0.5219***	0.5433	**0.5363***	0.5306	**0.5225***
KuaiRec-Small	AUC ↑	0.8654	**0.8657**	0.8641	**0.8642**	0.8641	**0.8649**	0.8630	**0.8635**	0.8653	**0.8659**
	Logloss ↓	0.4183	**0.4178**	0.4204	**0.4201**	0.4202	**0.4192***	0.4228	**0.4217***	0.4184	**0.4172***
Netflix	AUC ↑	0.6523	**0.6541***	0.6575	**0.6620***	0.6711	**0.6761***	0.6682	**0.6734***	0.6604	**0.6683***
	Logloss ↓	0.7835	**0.7803***	0.9350	**0.9115***	0.7742	**0.7434***	0.7760	**0.7542***	0.8926	**0.8800***

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the **w/o** version.

Table 2: Overall performance comparison with baselines. (Backbone model: DeepFM)

Dataset	Metrics	**w/o**	Drop3	BDIS	LSBo	LSSm	T-CE	R-CE	AutoDenoise
MovieLens1M	AUC ↑	0.8048	0.7701	0.6995	0.7659	0.7752	0.7814	0.7817	**0.8097***
	Logloss ↓	0.5306	0.5776	0.7996	0.5851	0.5841	1.7886	1.8263	**0.5225***
KuaiRec-Small	AUC ↑	0.8653	0.8485	0.8018	0.8439	0.8566	0.8140	0.8189	**0.8659***
	Logloss ↓	0.4184	0.4472	0.6694	0.4501	0.4931	1.7509	1.8955	**0.4172***
Netflix	AUC ↑	0.6604	0.6444	0.5726	0.6355	0.6471	0.6785	0.6832	0.6683
	Logloss ↓	0.8926	1.3277	1.1192	1.4491	1.0330	2.5780	2.5801	**0.8800***

"*" indicates the statistically significant improvements over the best baseline. ↑: higher is better; ↓: lower is better.

same settings as in the searching phase, i.e., being initialized at the beginning of every epoch and using the same loss function, denoted as $\mathcal{L}_{model}$. After iterating C epochs, we can obtain a $\{C \times MN\}$ matrix and start the searching phase with it.

The whole optimizing procedure consists of a warm-up train (line 1-9) and two training phases within the testing process (line 10-18). In the warm-up train, we first initialize the parameters of RS model $\boldsymbol{W}$ and train it to converge for C epochs (line 1-9), where we sample a mini-batch (line 4) for model optimization (line 5) and save the corresponding loss values (line 6) for the reward calculation. After that, we follow Algorithm 1 (line 11) and Algorithm 2 (line 13) to train and evaluate the policy network. With the $\boldsymbol{W}^*$ optimized by the selected subset, we can finally evaluate its performance (line 14) and save the optimal policy and the subset (line 15-17). The saved $\boldsymbol{X}_{sub}$ after the total T epochs is considered to be the noise-free subset. In the practical inference stage, we replace it with the original training set for model training.

3 EXPERIMENT

In this section, we conduct extensive experiments on three public datasets to investigate the following research questions:

- **RQ1:** How does the proposed framework, AutoDenoise, perform compared with the training with noisy data and baselines?
- **RQ2:** Could we directly transfer the data subset selected by AutoDenoise based on a recommendation model to train other models?
- **RQ3:** What is the impact of core components in AutoDenoise?
- **RQ4:** How do hyper-parameters influence AutoDenoise?
- **RQ5:** Are instances dropped by AutoDenoise really noises?

In the subsequent sections, we first describe the public data sets and evaluation metrics adopted in this paper. Then, we briefly review selected baselines and explain experimental results to answer corresponding research questions.

3.1 Experimental Settings

3.1.1 Datasets. We evaluate the model performance of AutoDenoise and baselines on three public datasets with different densities, **MovieLens1M**[4] (4.47%), **KuaiRec-Small**[5] (99.62%), and **Netflix**[6] (0.02%). For all datasets, we randomly select 80% as the training set, 10% as the validation set, and the remaining 10% as a test set.

3.1.2 Evaluation Metrics. To fairly evaluate recommendation performance, we select the AUC (Area Under the ROC Curve) scores and Logloss (logarithm of the loss value) as metrics, which are widely used for click-through prediction tasks [3, 9]. In practice, higher AUC scores or lower logloss values at the 0.001-level indicate a significant improvement [3].

3.1.3 Implementation Details. The implementation of AutoDenoise is based on a PyTorch-based public library[7], which involves sixteen state-of-the-art RS models. We develop the policy network of AutoDenoise as an individual class so that it can easily incorporate with different backbone RS models, like in Table 2.

3.2 Overall Performance (RQ1)

This section evaluates the effectiveness of AutoDenoise. As in Table 1, we compare the performances of recommendation models trained on a noise-free dataset denoised by AutoDenoise (**w**), against models trained on the original dataset with noise (**w/o**), and then evaluate them on the identical test dataset. We choose various advanced backbone recommendation models, including FM [24], Wide & Deep [3], DCN [28], and DeepFM [9]. In addition, we also compare the denoising quality of AutoDenoise with

[4]https://grouplens.org/datasets/movielens/1m/
[5]https://github.com/chongminggao/KuaiRec
[6]https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data
[7]https://github.com/rixwew/pytorch-fm

Table 3: Transferability test.

Recommendation Model	Methods	MovieLens1M		KuaiRec-Small		Netflix	
		AUC ↑	Logloss ↓	AUC ↑	Logloss ↓	AUC ↑	Logloss ↓
FM	Normal	0.8080	0.5334	0.8654	0.4183	0.6523	0.7835
	AutoDenoise	**0.8082**	**0.5274***	**0.8657**	**0.4179**	**0.6532***	**0.7783***
Wide & Deep	Normal	0.7957	0.5400	0.8641	0.4204	0.6575	0.9350
	AutoDenoise	**0.8001***	**0.5363***	**0.8645**	**0.4202**	**0.6587***	**0.9212***
DCN	Normal	0.8092	0.5235	0.8641	0.4202	0.6711	0.7742
	AutoDenoise	**0.8111***	**0.5230**	**0.8648**	**0.4193**	**0.6762***	**0.7476***
IPNN	Normal	0.8023	0.5433	0.8630	0.4228	0.6682	0.7760
	AutoDenoise	**0.8029**	**0.5368***	**0.8644***	**0.4206***	**0.6723***	**0.7588***

"*" indicates the statistically significant improvements over the best baseline. ↑: higher is better; ↓: lower is better.

state-of-the-art instance selection methods (Drop3 [34], BDIS [2], LSBo [15], LSSm [15]) and instance denoising methods (T_CE [29], R_CE [29])[8] in Table 2. We could conclude that:

- In light of the result in Table 1, integrating AutoDenoise can boost the performance for all backbone models on all datasets. Since KuaiRec-Small is far denser than MovieLens1M and Netflix (Data density: 99.62% $\gg$ 4.47% > 0.02%), the less remarkable improvement may come from the dominant contribution of clean instances, while noisy instances contribute less to user preference modeling, reasoning that the dense user-item interactions can accurately show the whole picture of a user. On the contrary, for sparse datasets MovieLens1M and Netflix, the noise-free subset from AutoDenoise can improve more in performance.
- In Table 2, all denoising methods fail to enhance the performance of DeepFM, which may be attributed to two reasons. (i) For instance selection methods (Drop3 [34], BDIS [2], LSBo [15], LSSm [15]), they tend to compress the dataset, resulting in discarding excessive samples and missing critical information. (ii) For denoising models like T-CE and R-CE [29], they improve the AUC score on a highly sparse dataset (Netflix, 0.02%), yet are unable to provide benefit on a dense dataset. The reason is that T-CE and R-CE highly depend on the negative sampling strategy, which samples unobserved user-item interactions. However, for the highly-dense dataset as KuaiRec-Small (99.62%), it would be hard to sample effective unobserved interactions. On the contrary, with the excellent exploring ability of the DRL structure, AutoDenoise is compatible with all datasets of diverse densities.

3.3 Transferability Test (RQ2)

In practice, recommender system models are updated frequently, causing it time-consuming and costly to train a denoising framework for each model. Therefore, we examine the transferability of AutoDenoise in this section.

Specifically, based on DeepFM, after obtaining the noise-free dataset selected by AutoDenoise, we directly use it to train other backbone recommendation models (e.g., FM [24], Wide & Deep [3], and DCN [28]) and investigate their performances. Results are presented in Table 3, where 'Normal' means training on the original data with noise, i.e., the '**w/o**' in Table 1. We can observe that all backbone models' performances are enhanced by training on the noise-free dataset selected by AutoDenoise. These experiments validate that AutoDenoise's output has a strong transferability and the potential to be utilized in commercial recommender systems.

[8]These two baselines could be applied to general denoising tasks. We do not test the performance of other denoising models since the different task settings as discussed in Section 3.1.

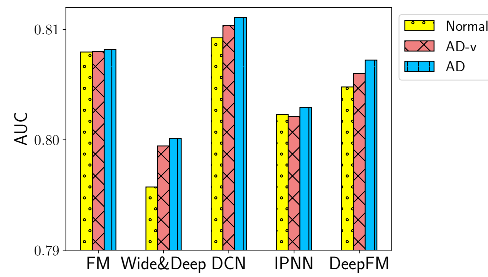

Figure 3: Ablation study on MovieLens1M.

3.4 Ablation Study (RQ3)

We investigate the core components of AutoDenoise through a range of experiments in this section. As we discussed in Section 2.3, a vital part of AutoDenoise is how to generate denoising results in the validation stage, i.e., adopting individual selection or top-k selection. Therefore, we design a variant of AutoDenoise, named **AutoDenoise-v**, to figure out the impact of different selection strategies. The only difference compared to origin AutoDenoise is that AutoDenoise-v applies individual selection in policy validation while AutoDenoise adopts top-k selection.

We test their AUC scores on MovieLens1M, and the results are illustrated in Figure 3. It can be observed that AutoDenoise (AD) outperforms AutoDenoise-v (AD-v) on all models, and AutoDenoise-v cannot generate noise-free datasets to improve IPNN performance. The reason is that AutoDenoise-v focuses on individual scores while neglecting the global situation. Besides, AutoDenoise could capture batch-wise global information to generate a robust policy by adopting a top-k selection strategy, which compares scores between the same batch of data. This experiment validates the rationality of our model design.

3.5 Parameter Analysis (RQ4)

In this section, we conduct experiments on MovieLens1M with DeepFM to investigate the sensitivity of two critical hyperparameters: (i) the number of warm-up training epochs C; (ii) the number of selected samples K. The results are visualized in Figure 4, where two Y-axes are used to scale the AUC scores and logloss values, respectively. The X-axis in Figure 4 (a) represents the number of

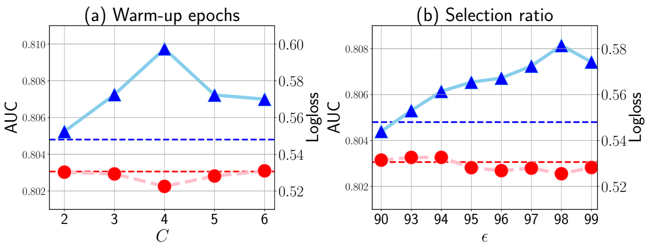

Figure 4: Parameter analysis on MovieLens1M.

warm-up epochs C, and X-axis in Figure 4 (b) is the selected ratio $\epsilon = K/M$. For both figures, blue lines illustrate the AUC score, and red lines are the Logloss value. In addition, we also mark the 'Normal' DeepFM performance as a dashed line for visual comparison (AUC 0.8048, Logloss 0.5306). We can find that:

- With the changing hyperparameters, AutoDenoise is superior to the Normal in most cases ($2 \leq C \leq 6$, $95\% \leq \epsilon \leq 99\%$), which can fully demonstrate its robustness. We achieve the best performance with the default setting (i.e., $C^* = 4$, $\epsilon^* = 98\%$).
- From Figure 4 (a), the parameter C directly influence the reward computation in Equation (3). For $C < C^*$, AutoDenoise suffers from the randomness of too few warm-up epochs, i.e., the first term in Equation (3), resulting in performance degradation. However, a large warm-up epoch leads to a smooth average value, and the loss value of every batch, i.e., the last term in Equation (3), have a more significant impact on the training process, leading to an unstable reward and impairing the denoising accuracy.
- As illustrated in Figure 4 (b), it is crucial to make appropriate discards of instances. On one hand, dropping too many instances results in inferior performance with missing valuable information. On the other hand, if we keep excessive instances, the model inevitably introduces noises and fails to get the best performance.

3.6 Case Study (RQ5)

We illustrate the effectiveness of AutoDenoise with examples in this section. Specifically, we select a user (ID: 5051) from MovieLens1M and visualize part of the training set in Table 4, all test data instances in Table 5. We present titles and genres of movies for the training set, together with the ground-truth click signals. For test data, we additionally exhibit the prediction result of DeepFM [9] trained on original noisy data (Normal) and noise-free data (AutoDenoise).

In Table 4, AutoDenoise detects the noise **The Wizard of Oz** with a positive click signal. This movie is a children's and musical movie, while all the other movies the users prefer are mainly action, crime, and war movies. By dropping this noise, AutoDenoise contributes to correcting the prediction of DeepFM on **Wonderland** (from 0.46 to 0.57, in Table 5), which is a documentary movie about war that the user usually enjoy. This case study also demonstrates the potential interpretability of AutoDenoise.

4 RELATED WORK

In this section, we will briefly introduce the related works to our framework, i.e., instance selection and denoising.

Instance Selection. Instance selection plays an important role in selecting the most predictive data instances to scale down the training set without performance degradation of predictive model [20]. It can be divided into two groups, i.e., wrapper methods [1, 4, 10, 34], and filter methods [21, 22, 26], whose selection criterion is based on the model performance and a selection function, respectively. CNN [10] is one of the earliest proposed methods in the field of instance selection. It keeps the instances near the class border and drops the internal ones to generate an effective data subset. DROP [34], a series of representative wrapper methods, relies on the instance associates, i.e., the k nearest neighbors on k-NN, to conduct the selection manner. LSSm and LSBo [15] are methods based on local set [1], i.e., the largest hypersphere that contains cases in the same class. LSSm defines and considers usefulness and harmfulness as measures to decide whether to remove an instance, while LSBo relies on the class borders to make the decision. In contrast to previous instance selection methods, AutoDenoise uses a policy network to automatically distinguish noisy instances rather than a fixed selection rule, which can better discover the inner relationship in a highly dynamic data distribution.

Denoising. The selection manner also works well in the fields of denoising, which can be categorized into selection-based method [6, 7, 32, 35], along with the reweighting-based methods [29, 30]. The WBPR [7] assigns higher selecting probabilities to the missing interactions of the popular items since it considers them to be the real negative instances. IR [32] finds out the negative interactions and changes their label to revise the unreliable behaviors from users. R-CE and T-CE [29] consider that noisy instances would have high loss value, thus they dynamically assign lower weight to the high-loss instances as well as truncate the ones with weight lower than the threshold in two strategies, respectively. However, the above models either suffer from selection bias or insufficient transferability, which are solved by AutoDenoise with its DRL framework.

Table 4: Training dataset for user 5051.

Title	Genres	Click
The Patriot	Action|Drama|War	1
Raiders of the Lost Ark	Action|Adventure	1
The Wizard of Oz	**Children's|Musical**	**1**
...	...	...

Table 5: Test dataset for user 5051.

Title	Genres	Click	Normal/AutoDenoise
The Extra-Terrestrial	Drama|Fantasy	1	0.77/0.82
Lawrence of Arabia	Adventure|War	1	0.94/0.93
Wonderland	**Documentary|War**	**1**	**0.46/0.57**

5 CONCLUSION

We propose a DRL-based instance denoising method, AutoDenoise, to improve the recommendation performance of RS models by adaptively selecting the noise-free instances in every mini-batch of data and training the RS model with the selected noise-free subset. We design a two-phase optimization strategy to properly train and evaluate the proposed framework. The extensive experiments validate the effectiveness and compatibility of AutoDenoise and its selected noise-free data subset. Further experiments prove the transferability of the noise-free subset, i.e., the noise-free subset selected with one RS model can transfer well to other state-of-the-art backbone RS models with significant performance improvement.

REFERENCES

[1] Henry Brighton and Chris Mellish. 2002. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery* 6, 2 (2002), 153–172.

[2] Qingqiang Chen, Fuyuan Cao, Ying Xing, and Jiye Liang. 2022. Instance Selection: A Bayesian Decision Theory Perspective. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 6, 6287–6294. https://doi.org/10.1609/aaai.v36i6.20578

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[4] Chien-Hsing Chou, Bo-Han Kuo, and Fu Chang. 2006. The generalized condensed nearest neighbor rule as a data reduction method. In *18th international conference on pattern recognition (ICPR'06)*, Vol. 2. IEEE, 556–559.

[5] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*. 293–296.

[6] Jingtao Ding, Guanghui Yu, Xiangnan He, Fuli Feng, Yong Li, and Depeng Jin. 2019. Sampler design for bayesian personalized ranking by leveraging view data. *IEEE transactions on knowledge and data engineering* 33, 2 (2019), 667–681.

[7] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2012. Personalized ranking for non-uniformly sampled items. In *Proceedings of KDD Cup 2011*. PMLR, 231–247.

[8] Zhabiz Gharibshah, Xingquan Zhu, Arthur Hainline, and Michael Conway. 2020. Deep learning for user interest and response prediction in online display advertising. *Data Science and Engineering* 5, 1 (2020), 12–26.

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proc. of IJCAI*.

[10] Peter Hart. 1968. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory* 14, 3 (1968), 515–516.

[11] Kaixi Hu, Lin Li, Qing Xie, Jianquan Liu, and Xiaohui Tao. 2021. What is Next when Sequential Prediction Meets Implicitly Hard Interaction?. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 710–719.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*. Ieee, 263–272.

[13] PN Vijaya Kumar and V Raghunatha Reddy. 2014. A survey on recommender systems (RSS) and its applications. *International Journal of Innovative Research in Computer and Communication Engineering* 2, 8 (2014), 5254–5260.

[14] Dongha Lee, SeongKu Kang, Hyunjun Ju, Chanyoung Park, and Hwanjo Yu. 2021. Bootstrapping user and item representations for one-class collaborative filtering. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 317–326.

[15] Enrique Leyva, Antonio González, and Raúl Pérez. 2015. Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. *Pattern Recognition* 48, 4 (2015), 1523–1537.

[16] Tzu-Heng Lin, Chen Gao, and Yong Li. 2019. Cross: Cross-platform recommendation for social e-commerce. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 515–524.

[17] Weilin Lin, Xiangyu Zhao, Yejing Wang, Tong Xu, and Xian Wu. 2022. AdaFS: Adaptive Feature Selection in Deep Recommender System. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3309–3317.

[18] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated embedding size search in deep recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2307–2316.

[19] Kunpeng Liu, Yanjie Fu, Pengfei Wang, Le Wu, Rui Bo, and Xiaolin Li. 2019. Automating feature subspace exploration via multi-agent reinforcement learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 207–215.

[20] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Martínez-Trinidad, and Josef Kittler. 2010. A review of instance selection methods. *Artificial Intelligence Review* 34, 2 (2010), 133–143.

[21] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, and J Fco Martínez-Trinidad. 2008. Prototype selection via prototype relevance. In *Iberoamerican congress on pattern recognition*. Springer, 153–160.

[22] Thanapant Raicharoen and Chidchanok Lursinsap. 2005. A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm. *Pattern recognition letters* 26, 10 (2005), 1554–1567.

[23] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. 2016. Multilayer perceptron: Architecture optimization and training. (2016).

[24] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining*. IEEE, 995–1000.

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

[26] José C Riquelme, Jesús S Aguilar-Ruiz, and Miguel Toro. 2003. Finding representative patterns with ordered projections. *pattern recognition* 36, 4 (2003), 1009–1018.

[27] Jian Wang and Yi Zhang. 2013. Opportunity model for e-commerce recommendation: right product; right time. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 303–312.

[28] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[29] Wenjie Wang, Fuli Feng, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2021. Denoising implicit feedback for recommendation. In *Proceedings of the 14th ACM international conference on web search and data mining*. 373–381.

[30] Yu Wang, Xin Xin, Zaiqiao Meng, Joemon M Jose, Fuli Feng, and Xiangnan He. 2022. Learning Robust Recommenders through Cross-Model Agreement. In *Proceedings of the ACM Web Conference 2022*. 2015–2025.

[31] Yejing Wang, Xiangyu Zhao, Tong Xu, and Xian Wu. 2022. Autofield: Automating feature selection in deep recommender systems. In *Proceedings of the ACM Web Conference 2022*. 1977–1986.

[32] Zitai Wang, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. 2021. Implicit Feedbacks are Not Always Favorable: Iterative Relabeled One-Class Collaborative Filtering against Noisy Interactions. In *Proceedings of the 29th ACM International Conference on Multimedia*. 3070–3078.

[33] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[34] D Randall Wilson and Tony R Martinez. 2000. Reduction techniques for instance-based learning algorithms. *Machine learning* 38, 3 (2000), 257–286.

[35] Wenhui Yu and Zheng Qin. 2020. Sampler design for implicit feedback data by noisy-label robust learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 861–870.

[36] Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M Jose. 2018. fBGD: Learning embeddings from positive unlabeled data with BGD. (2018).

[37] Chi Zhang, Yantong Du, Xiangyu Zhao, Qilong Han, Rui Chen, and Li Li. 2022. Hierarchical Item Inconsistency Signal Learning for Sequence Denoising in Sequential Recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2508–2518.

[38] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. 2021. DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 750–758.

[39] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: a survey. *ACM SIGWEB Newsletter* Spring (2019), 1–15.

[40] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Recommender Systems Conference*. ACM, 95–103.

[41] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1040–1048.

[42] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Dawei Yin, Yihong Zhao, and Jiliang Tang. 2017. Deep Reinforcement Learning for List-wise Recommendations. *arXiv preprint arXiv:1801.00209* (2017).

[43] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1059–1068.