applied sciences

MDPI

Article

Sparse Data Recommendation by Fusing Continuous Imputation Denoising Autoencoder and Neural Matrix Factorization

Xinyue Wan [1,*], Bofeng Zhang [1], Guobing Zou [1] and Furong Chang [1,2]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; bfzhang@shu.edu.cn (B.Z.); gbzou@shu.edu.cn (G.Z.); cfrkashger@shu.edu.cn (F.C.)
[2] School of Computer Science and Technology, Kashgar University, Kashgar 844006, China
* Correspondence: wxyshu@shu.edu.cn

Received: 12 December 2018; Accepted: 18 December 2018; Published: 24 December 2018

Abstract: In recent years, although deep neural networks have yielded immense success in solving various recognition and classification problems, the exploration of deep neural networks in recommender systems has received relatively less attention. Meanwhile, the inherent sparsity of data is still a challenging problem for deep neural networks. In this paper, firstly, we propose a new CIDAE (Continuous Imputation Denoising Autoencoder) model based on the Denoising Autoencoder to alleviate the problem of data sparsity. CIDAE performs regular continuous imputation on the missing parts of the original data and trains the imputed data as the desired output. Then, we optimize the existing advanced NeuMF (Neural Matrix Factorization) model, which combines matrix factorization and a multi-layer perceptron. By optimizing the training process of NeuMF, we improve the accuracy and robustness of NeuMF. Finally, this paper fuses CIDAE and optimized NeuMF with reference to the idea of ensemble learning. We name the fused model the I-NMF (Imputation-Neural Matrix Factorization) model. I-NMF can not only alleviate the problem of data sparsity, but also fully exploit the ability of deep neural networks to learn potential features. Our experimental results prove that I-NMF performs better than the state-of-the-art methods for the public MovieLens datasets.

Keywords: recommender systems; denoising autoencoder; deep neural networks; data imputation; matrix factorization; implicit feedback

1. Introduction

In the era of information explosion, big data exhibits a rich value and great potential, which brings transformative development to human society, but it also generates the serious "information overload" problem. Recommender systems are an effective way to alleviate the problem of "information overload", having been widely adopted by many online services, including E-commerce, online news, and social media sites [1]. Recommender systems can help determine which information to offer to individual consumers and allow online users to quickly find the personalized information that fits their needs [2]. Collaborative Filtering (CF) is a successful approach commonly used by many recommender systems [3]. CF [4,5] is based on the user's past interaction records (such as ratings) to simulate the user's preferences for the item. The scarcity of original interaction records has always been a difficult point for CF, which is the problem of data sparsity.

As the most popular approach among various CF techniques, Matrix Factorization (MF) [6,7], which projects users and items into a shared latent space, using a vector of latent features to represent a user or an item, has become a standard model for recommendation due to its scalability, simplicity, and flexibility [8,9]. Most previous research [3,7,10–15] on MF did not change the nature of linearity, such as integrating it

with neighbor-based models [7], combining it with topic models of item content [3], and extending it to factorization machines [15] for a generic modelling of features. Despite the effectiveness of MF for CF, such a linear model is insufficient to understand the complex and non-linear relationship between users and items [16]. Furthermore, the problem of data sparsity is difficult to solve in MF.

Of course, there has been some advanced research [1,2,17,18] on MF that essentially changed the linear structure of MF by combining the idea of deep learning. The combination of deep learning and MF mainly uses the implicit feedback data to learn the hidden vector of the user(item) by using the deep learning method, thereby predicting the user's preference for the item based on the hidden vector. Among existing models, an outstanding performance was shown for the NeuMF (Neural Matrix Factorization) model proposed in [1]. The NeuMF model combines matrix factorization and a multi-layer perceptron to achieve a good performance.

MF based on deep learning can be regarded as a kind of nonlinear generalization of the traditional hidden factor model. Its obvious advantage is the introduction of nonlinear feature transformations in the process of learning the hidden vector of the user(item), which has a better performance than traditional MF. However, this method still cannot improve the problem of data sparsity in MF.

In this paper, in order to alleviate the problem of data sparsity, we propose a new CIDAE (Continuous Imputation Denoising Autoencoder) model based on the Denoising Autoencoder (DAE). The CIDAE model performs regular continuous imputation for the missing parts of the original data and trains the imputed data as the desired output, which is different from the DAE algorithm that randomly adds noise to the input data. The advantage of CIDAE is that it alleviates the problem of data sparsity by using the idea of imputation.

Then, this paper optimizes the advanced NeuMF model based on the training process. The NeuMF model introduces the idea of deep neural networks into matrix factorization. The NeuMF model first proposes the GMF (Generalized Matrix Factorization) model and then combines the GMF model with the MLP (Multi-Layer Perceptron) model. We name the optimized NeuMF model the O-NeuMF-p model. Additionally, our experimental results prove that the accuracy of O-NeuMF-p is higher than that of NeuMF.

Finally, we fuse CIDAE and O-NeuMF-p with reference to the idea of ensemble learning. We name the fused model the I-NMF (Imputation-Neural Matrix Factorization) model, which combines the advantages of CIDAE and O-NeuMF-p. The I-NMF model can not only alleviate the problem of data sparsity, but also fully exploit the ability of deep neural networks to learn potential features, and I-NMF is also robust. Furthermore, our experimental results confirm that the performance of I-NMF is better than CIDAE, O-NeuMF-p, and current advanced recommendation algorithms for the public MovieLens (ML) [19] datasets.

In summary, our main contributions of this work are outlined as follows:

- We propose a new CIDAE model based on DAE. The advantage of CIDAE is that it alleviates the problem of data sparsity by using the idea of imputation.
- By optimizing the training process of the NeuMF model, we improve the accuracy and robustness of NeuMF. We name the optimized NeuMF model the O-NeuMF-p model.
- We fuse CIDAE and O-NeuMF-p with reference to the idea of ensemble learning. Additionally, we name the fused model the I-NMF model. The I-NMF model can not only alleviate the problem of data sparsity, but also fully exploit the ability of deep neural networks to learn potential features, and I-NMF is also robust.

This paper is organized as follows. In Section 2, we present the architecture and details of the CIDAE model. In Section 3, we briefly introduce the NeuMF model and explain in detail how we optimize its training process. In Section 4, we propose the idea of fusing CIDAE and O-NeuMF-p, and present the architecture and details of the I-NMF model. In Section 5, we evaluate all the models mentioned above using the public MovieLens datasets, and compare the results with current advanced recommendation algorithms. Concluding remarks with a discussion of some future work are in the final section.

2. CIDAE: Continuous Imputation Denoising Autoencoder

Our CIDAE consists of three parts: learning from the original data; continuous imputation for the missing parts of the original data; and learning from the imputed data. Figure 1 shows the structure of CIDAE.

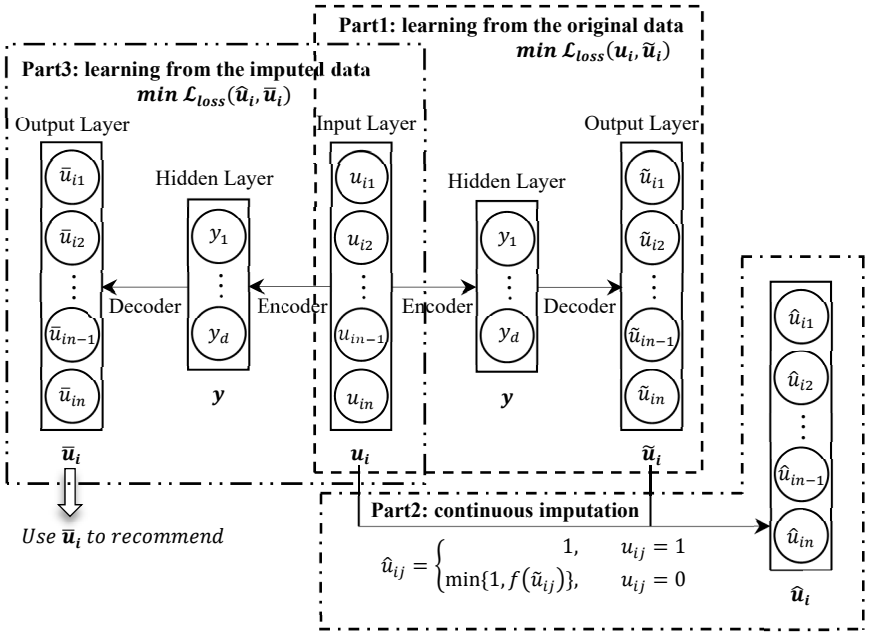


Figure 1. The structure of CIDAE.

2.1. Part 1: Learning from the Original Data

In the first part, there are many existing models [10–14] to choose from for learning the original data. In this paper, we chose the traditional autoencoder (AE) [20].

For each user u_i, some items in the original data are of interest to u_i, while the remaining items are the missing parts of the original data. The goal is to predict the items that are of interest to u_i from the missing parts of the original data.

In this part, labels of the items that the u_i is interested in are set to 1, and labels of the remaining items are set to 0. Similar to [16,21], labels for all items constitute the initial value of the user vector $\boldsymbol{u_i}$ ($u_{ij} = 0$ or 1).

According to the AE, we use $\boldsymbol{u_i}$ as the input and get the reconstructed $\boldsymbol{u_i}$ by encoding and decoding. The specific process is as follows:

Encoding: Map $\boldsymbol{u_i} \in \mathbb{R}^n$ to the d-dimensional hidden layer by encoding function f.

$$\boldsymbol{y} = f(\boldsymbol{u_i}) = \sigma(\boldsymbol{W}\boldsymbol{u_i} + \boldsymbol{b}) \tag{1}$$

where $\boldsymbol{W} \in \mathbb{R}^{d \times n}, \boldsymbol{b} \in \mathbb{R}^d$, $\boldsymbol{y} \in \mathbb{R}^d, n$ is the total number of items and σ is a non-linear mapping function, e.g., a sigmoid function.

Decoding: Map $\boldsymbol{y} \in \mathbb{R}^d$ to the n-dimensional space by decoding function g for reconstructing $\boldsymbol{u_i}$. Let $\widetilde{\boldsymbol{u_i}}$ denote the reconstructed vector of $\boldsymbol{u_i}$.

$$\widetilde{\boldsymbol{u_i}} = g(\boldsymbol{y}) = \sigma(\boldsymbol{W'}\boldsymbol{y} + \boldsymbol{b'}) \tag{2}$$

where $\boldsymbol{W'} \in \mathbb{R}^{n \times d}, \boldsymbol{b'} \in \mathbb{R}^{n}$, $\widetilde{\boldsymbol{u}}_{\boldsymbol{i}} \in \mathbb{R}^{n}$. We hope that $\widetilde{\boldsymbol{u}}_{\boldsymbol{i}}$ and $\boldsymbol{u_i}$ are as similar as possible, and the minimum objective function of AE is as follows:

$$\underset{\boldsymbol{W},\boldsymbol{W'},\boldsymbol{b},\boldsymbol{b'}}{argmin} \sum_{i=1}^{m} \mathcal{L}(\boldsymbol{u_i}, \widetilde{\boldsymbol{u}}_{\boldsymbol{i}}) + \frac{\lambda}{2}\left(||\boldsymbol{W}||_F^2 + ||\boldsymbol{W'}||_F^2 + ||\boldsymbol{b}||_F^2 + ||\boldsymbol{b'}||_F^2\right) \tag{3}$$

$$\mathcal{L}(\boldsymbol{u_i}, \widetilde{\boldsymbol{u}}_{\boldsymbol{i}}) = -\frac{1}{n}\sum_{j=1}^{n} u_{ij} log \widetilde{u}_{ij} + \left(1 - u_{ij}\right) log(1 - \widetilde{u}_{ij}) \tag{4}$$

where m is the total number of users and $\mathcal{L}$ is the loss function. Furthermore, we choose the cross entropy as the loss function. λ is a regularization parameter to prevent overfitting. Just like the traditional autoencoder, we optimize the objective function by using the gradient descent method.

2.2. Part 2: Continuous Imputation on the Missing Parts of the Original Data

In this part, CIDAE performs continuous imputation for the missing parts of the original data by using $\widetilde{\boldsymbol{u}}_{\boldsymbol{i}}$. Let $\hat{\boldsymbol{u}}_{\boldsymbol{i}} \in \mathbb{R}^{\boldsymbol{n}}$ denote the imputed vector of $\boldsymbol{u_i}$. For each $\hat{u}_{ij}$ in $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$, we can use the following formula to calculate:

$$\hat{u}_{ij} = \begin{cases} 1, \ u_{ij} = 1 \\ min\{1, f(\widetilde{u}_{ij})\}, \ u_{ij} = 0 \end{cases} \tag{5}$$

where $f(x)$ can be any continuous function, and it can be a linear or nonlinear function; for example, $f(x) = kx$ or $f(x) = ktanx$, where k is a hyper-parameter. Obviously, the larger the value of k, the larger the proportion of data that is imputed to 1. There are many choices for $f(x)$, which shows that CIDAE has great flexibility.

2.3. Part 3: Learning from the Imputed Data

The traditional Denoising Autoencoder (DAE) first adds noise to the input data and then reconstructs the noise-added data. CIDAE performs regular imputation of the output data, which is different from DAE.

The specific process is similar to the Part1, as we still choose $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ as the input and obtain the reconstructed $\boldsymbol{u_i}$ by encoding and decoding. In Part 3, let $\overline{\boldsymbol{u}}_{\boldsymbol{i}} \in \mathbb{R}^{\boldsymbol{n}}$ denote the reconstructed vector of $\boldsymbol{u_i}$. We want the value of $\overline{\boldsymbol{u}}_{\boldsymbol{i}}$ to be as similar as possible to $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$, not $\boldsymbol{u_i}$, which is different from the Part1. The minimum objective function of CIDAE is as follows:

$$\underset{\mathbf{W},\mathbf{W}^{'},\mathbf{b},\mathbf{b}^{'}}{\operatorname{argmin}} \sum_{i=1}^{m} \mathcal{L}(\hat{\boldsymbol{u}}_{\boldsymbol{i}}, \overline{\boldsymbol{u}}_{\boldsymbol{i}}) + \frac{\lambda}{2}(||\mathbf{W}||_F^2 + ||\mathbf{W}^{'}||_F^2 + ||\mathbf{b}||_F^2 + ||\mathbf{b}^{'}||_F^2) \tag{6}$$

$$\mathcal{L}(\hat{\boldsymbol{u}}_{\boldsymbol{i}}, \overline{\boldsymbol{u}}_{\boldsymbol{i}}) = -\frac{1}{\mathrm{n}}\sum_{\mathrm{j}=1}^{\mathrm{n}} \hat{u}_{ij} \log \overline{u}_{ij} + \left(1 - \hat{u}_{ij}\right) \log\left(1 - \overline{u}_{ij}\right) \tag{7}$$

Similar to [22], we introduce a hyper-parameter (α) in order to distinguish between the imputed data and the original data. We can distinguish the reconstruction error between the imputed data and the original data by using α as a weight. So, the final minimum objective function of CIDAE is as follows:

$$\underset{\mathbf{W},\mathbf{W}^{'},\mathbf{b},\mathbf{b}^{'}}{\operatorname{argmin}} \sum_{i=1}^{m} K(\hat{\boldsymbol{u}}_{\boldsymbol{i}}, \overline{\boldsymbol{u}}_{\boldsymbol{i}}) + \frac{\lambda}{2}\left(||\mathbf{W}||_F^2 + ||\mathbf{W}^{'}||_F^2 + ||\mathbf{b}||_F^2 + ||\mathbf{b}^{'}||_F^2\right) \tag{8}$$

$$\mathrm{K}(\hat{\boldsymbol{u}}_{\boldsymbol{i}}, \overline{\boldsymbol{u}}_{\boldsymbol{i}}) = \alpha\left(\sum_{\{j|u_{ij}=0\}} \mathcal{L}\left(\hat{u}_{ij}, \overline{u}_{ij}\right)\right) + \left(\sum_{\{j|u_{ij}=1\}} \mathcal{L}\left(\hat{u}_{ij}, \overline{u}_{ij}\right)\right) \tag{9}$$

In this way, CIDAE can adjust the loss weight of the imputed data through α. Just like the traditional denoising autoencoder, we optimize the objective function of the CIDAE by using the gradient descent method.

It is worth emphasizing that CIDAE trains the imputed data $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$ as the desired output. On the one hand, the imputation method can alleviate the sparseness of the original data. On the other hand, a well-founded imputation is more meaningful than random noise. In addition, the CIDAE model, like the DAE algorithm, can increase the robustness of the model by introducing disturbances.

3. Optimization of the NeuMF Model

In this section, we briefly introduce the NeuMF model [1], and then explain in detail how to optimize NeuMF from the training process. The NeuMF model consists of GMF (Generalized Matrix Factorization) and MLP (Multi-Layer Perceptron). Figure 2 shows the structure of NeuMF.

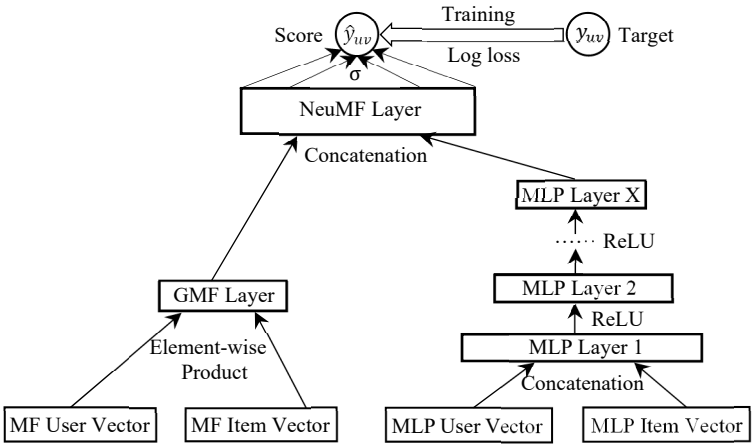


Figure 2. The structure of NeuMF [1].

3.1. GMF (Generalized Matrix Factorization)

GMF has four layers: input layer, embedding layer, inner layer, and prediction layer (output layer). The input to GMF is a sparse user ID and item ID. The embedding layer is a fully connected layer that can project sparse vectors into dense vectors. The vector obtained by the embedding layer can be regarded as the hidden vector of the user (item). The hidden vector of user u is denoted as $\boldsymbol{p}_{\boldsymbol{u}}$, and the hidden vector of the item v is denoted as $\boldsymbol{q}_{\boldsymbol{v}}$. The mapping formula of the inner layer is as follows:

$$\varphi \boldsymbol{p}_{\boldsymbol{u}}, \boldsymbol{q}_{\boldsymbol{v}} = \boldsymbol{p}_{\boldsymbol{u}} \odot \boldsymbol{q}_{\boldsymbol{v}} \tag{10}$$

where $\odot$ represents the element-wise product of vectors. The operation of the last prediction layer (output layer) is

$$\hat{y}_{uv} = a_{out}\left(\boldsymbol{h}^{\boldsymbol{T}}(\boldsymbol{p}_{\boldsymbol{u}} \odot \boldsymbol{q}_{\boldsymbol{v}})\right) \tag{11}$$

$$a_{out} = \sigma(x) = \frac{1}{(1+e^{-x})} \tag{12}$$

where a_{out} is the activation function. According to the literature [1], we adopt the sigmoid function $\sigma(x)$ as a_{out}. $\boldsymbol{h}$ is the weight of the prediction layer and is trained by the loss function.

3.2. MLP (Multi-Layer Perceptron)

MLP is designed to explore the interaction between users and items' latent features. Different from collaborative filtering of GMF, MLP is more flexible. MLP is not limited to the vector inner product,

and can deeply learn the potential interaction between the user's hidden vector $\boldsymbol{p}_u$ and the item's hidden vector $\boldsymbol{q}_v$. The specific process of MLP is as follows:

$$\begin{aligned} \boldsymbol{z}_1 &= \phi_1(\boldsymbol{p}_u, \boldsymbol{q}_v) = \begin{bmatrix} \boldsymbol{p}_u \\ \boldsymbol{q}_v \end{bmatrix} \\ \phi_2(\boldsymbol{z}_1) &= a_2\left(\boldsymbol{W}_2^T z_1 + \boldsymbol{b}_2\right) \\ \phi_L(\boldsymbol{z}_{L-1}) &= a_L\left(\boldsymbol{W}_L^T \boldsymbol{z}_{L-1} + \boldsymbol{b}_L\right) \\ \hat{y}_{uv} &= \sigma\left(\boldsymbol{h}^T \phi_L(\boldsymbol{z}_{L-1})\right) \end{aligned} \tag{13}$$

where $\boldsymbol{W}_x$ is the weight matrix of each layer, $\boldsymbol{b}_x$ is the bias vector of each layer, and a_x is the activation function of each layer. According to the literature [1,23], we adopt the Rectifier (ReLU) as a_x. The $\hat{y}_{uv}$ is the output of the last prediction layer and σ (sigmoid function) is the activation function of the prediction layer. $\boldsymbol{h}$ is the weight of the prediction layer and is trained by the loss function. MLP selects the four-layers ($L = 4$) tower structure, and halves the layer size for each successive higher layer. Together with the final prediction layer, MLP has a total of five layers.

3.3. The NeuMF Model

NeuMF is a comprehensive model that combines GMF with MLP. NeuMF combines the training of GMF and MLP by adding the "NeuMF layer". The specific combination process is as follows:

$$\begin{aligned} \phi^{GMF} &= \boldsymbol{p}_u^G \odot \boldsymbol{q}_v^G \\ \phi^{MLP} &= a_L\left(\boldsymbol{W}_L^T\left(a_{L-1}\left(\ldots a_2\left(\boldsymbol{W}_2^T\begin{bmatrix} \boldsymbol{p}_u^M \\ \boldsymbol{q}_v^M \end{bmatrix} + \boldsymbol{b}_2\right)\ldots\right)\right) + \boldsymbol{b}_L\right) \\ \hat{y}_{uv} &= \sigma\left(\boldsymbol{h}^T\begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}\right) \end{aligned} \tag{14}$$

where ϕ^{GMF} is the output result of the inner layer of GMF, and ϕ^{MLP} is the output of MLP after four layers. The "NeuMF layer" can be understood as a combination of GMF and MLP from the prediction layer. σ (sigmoid function) is the activation function of the "NeuMF layer". $\boldsymbol{h}$ is the weight of the "NeuMF layer" and is trained by the loss function.

3.4. The NeuMF-p Model

NeuMF-p(pre-training) refers to the model that initializes NeuMF. NeuMF is initialized by pre-training GMF and MLP. The experiment in paper [1] shows that NeuMF-p is better than NeuMF. The specific initialization process of NeuMF is described in the following.

Firstly, GMF and MLP are separately pre-trained. Pre-training at this time means that the GMF(MLP) model is first randomly initialized, and the model is then trained by the loss function until the model converges. Secondly, the parameters of the pre-trained GMF and the pre-trained MLP are the initialization parameters of NeuMF-p. Finally, h is the weight of the "NeuMF layer", and the initialized value is as follows:

$$\boldsymbol{h} \leftarrow \begin{bmatrix} \gamma \boldsymbol{h}^{\boldsymbol{GMF}} \\ (1-\gamma)\boldsymbol{h}^{\boldsymbol{MLP}} \end{bmatrix} \tag{15}$$

where $\boldsymbol{h}^{\boldsymbol{GMF}}$ is the weight of the prediction layer of GMF and $\boldsymbol{h}^{\boldsymbol{MLP}}$ is the weight of the prediction layer of MLP. The γ is a hyper-parameter and the value of γ is set to 0.5 [1].

Considering that the experiment [1] shows that NeuMF-p is better than NeuMF, this paper chooses to optimize the training process of NeuMF-p directly. Here is a brief introduction to the loss function of NeuMF-p and the original training process. The loss function of NeuMF-p is as follows:

$$\begin{aligned} L &= -\sum_{(u,v)\in y} \log \hat{y}_{uv} - \sum_{(u,v)\in y^-} \log(1-\hat{y}_{uv}) \\ &= -\sum_{(u,v)\in y\cup y^-} y_{uv} \log \hat{y}_{uv} + (1-y_{uv}) \log(1-\hat{y}_{uv}) \end{aligned} \tag{16}$$

This is the loss function of NeuMF-p, also known as the log loss or binary cross-entropy loss, which can be optimized by the SGD (Stochastic Gradient Descent) algorithm. The value of the target y_{uv} is 0 or 1, which can be regarded as a label. If $y_{uv} = 1$, it means that there is an interaction between user u and item v. If $y_{uv} = 0$, it means that there is no interaction between user u and item v. By using the sigmoid function as the activation function of the output layer, $\hat{y}_{uv}$ can be controlled to output in the range of $(0, 1)$, where y represents the set of observed interactions in dataset Y and y^- represents the set of counterexamples in dataset Y. The counterexample in the original paper [1] refers to the data of unobserved interactions in dataset Y.

3.5. The O-NeuMF-p Model

We name the optimized NeuMF-p model the O-NeuMF-p model. This paper optimizes the training process of NeuMF-p from the counterexample label. The original paper [1] records the label y_{uv} in the counterexample set y^- as 0, which we think is unreasonable. Recommender systems focus on the data that has no interaction between users and items. This means that counterexamples are the focus of the research. If the label of the counterexample is marked as 0 when training, it is not conducive to learning the interaction in y^-.

This paper refers to the idea of DAE, adding noise to the original data and reconstructing the data. We change the label in the counterexample set y^- from 0 to a random number. The specific change formula is as follows:

$$if\ (u,v) \in y^-,\ y_{uv} = random(0,r) \tag{17}$$

where r is a hyper-parameter and can be an arbitrary number between $[0, 1]$. If $r = 0$, then $y_{uv} = random(0,0) = 0$. In this case, it is back to the original situation, where the label y_{uv} in the counterexample set y^- is marked as 0. If $r = 1$, then $y_{uv} = random(0,1)$. This means that the label y_{uv} in y^- is a random number from 0 to 1.

The idea of adding random numbers can make the label y_{uv} in y^- no longer equal to 0, which is equivalent to adding a disturbance to the data of y^-. On the one hand, O-NeuMF-p is more robust than NeuMF-p. On the other hand, O-NeuMF-p is more focused on the learning of the y^- data, which is exactly what we want. In addition, our experimental results show that under the same dataset, O-NeuMF-p has a higher recommendation accuracy and quality than NeuMF-p. This confirms the advantage of introducing random numbers in O-NeuMF-p from the side.

4. I-NMF: Fusion of CIDAE and O-NeuMF-p

On the one hand, although O-NeuMF-p performs better than NeuMF-p, it still does not solve the problem of data sparsity. On the other hand, although CIDAE can alleviate the problem of data sparsity, CIDAE only utilizes the idea of a single-layer neural network(AE) and does not fully exploit the ability of deep neural networks to learn potential features. Therefore, this paper fuses CIDAE and O-NeuMF-p with reference to the idea of ensemble learning. We name the fused model the I-NMF (Imputation-Neural Matrix Factorization) model, which combines the advantages of CIDAE and O-NeuMF-p. The I-NMF model can not only alleviate the problem of data sparsity, but also fully exploit the ability of deep neural networks to learn potential features, and I-NMF is also robust. Figure 3 shows the structure of I-NMF.

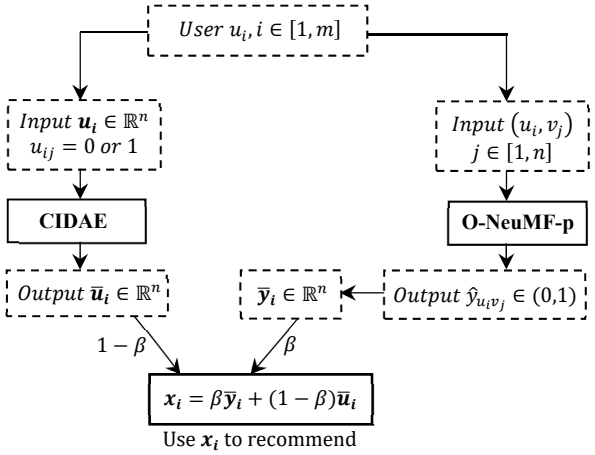


Figure 3. The structure of I-NMF.

It is worth noting that there was a paper [24] combining the AE algorithm and the MF algorithm. The paper [24] uses the AE's feature layer as the initial value of the hidden vector of MF, but the combined model is still linear, like MF. The idea of this paper is not the combination of the feature layer, but purely the combination of the prediction layer, which guarantees the integrity of CIDAE and O-NeuMF-p, so that both models can play their own advantages.

Specific Fusion Process

In data set $\mathcal{O}$, m is the total number of users, and n is the total number of items. Additionally, the data of $\mathcal{O}$ is divided into a training set $\mathcal{X}$ and test set $\mathcal{T}$.

On the one hand, the input of CIDAE trained with $\mathcal{X}$ is $\boldsymbol{u_i}(\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}, i \in [1, m])$, and the output is $\bar{\boldsymbol{u}}_{\boldsymbol{i}}$. On the other hand, the input of O-NeuMF-p trained with $\mathcal{X}$ is a group of user IDs and item IDs $((u_i, v_j)(i \in [1, m], j \in [1, n]))$, and the output is the predicted value $\hat{y}_{u_i v_j}(\hat{y}_{u_i v_j} \in (0, 1))$.

For each user u_i in $\mathcal{O}$, taking the top-N recommendation as an example, our goal is to use only the data in $\mathcal{X}$ and recommend the top-N items that may be of interest in the missing data. The accuracy and quality of the top-N recommendation is calculated by a comparison with the data in $\mathcal{T}$.

For each user u_i, firstly, we use the trained CIDAE to get the vector $\bar{\boldsymbol{u}}_{\boldsymbol{i}}(\bar{\boldsymbol{u}}_{\boldsymbol{i}} \in \mathbb{R}^n)$ of the user u_i. Then, we use the trained O-NeuMF-p to get n predicted values $\hat{y}_{u_i v_j}(j \in [1, n])$ of the user u_i. The vector consisting of these n predicted values is denoted as $\bar{\boldsymbol{y}}_{\boldsymbol{i}}(\bar{\boldsymbol{y}}_{\boldsymbol{i}} \in \mathbb{R}^n)$. In this way, we have the prediction vectors $\bar{\boldsymbol{u}}_{\boldsymbol{i}}$ and $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$ corresponding to these two models for each user u_i. The prediction vector for each user u_i in I-NMF is denoted as $\boldsymbol{x_i}(\boldsymbol{x_i} \in \mathbb{R}^{\boldsymbol{n}})$. The value of $\boldsymbol{x_i}$ is determined by the prediction vectors $\bar{\boldsymbol{u}}_{\boldsymbol{i}}$ and $\bar{\boldsymbol{y}}_{\boldsymbol{i}}$, and the specific formula is as follows:

$$\boldsymbol{x_i} = \beta \bar{\boldsymbol{y}}_{\boldsymbol{i}} + (1-\beta)\bar{\boldsymbol{u}}_{\boldsymbol{i}} \tag{18}$$

where β is a hyper-parameter, and $\beta \in [0, 1]$. The introduction of β can adjust the fusion ratio of CIDAE and O-NeuMF-p. When $\beta = 0$, CIDAE is used for recommendation alone. Conversely, when $\beta = 1$, O-NeuMF-p is used for recommendation alone.

We give the recommended top-N items based on the ranking of the value of $\boldsymbol{x_i}$ in the missing data of the training set $\mathcal{X}$. It is worth noting that the range of the value of $\boldsymbol{x_i}$ is not necessarily between 0 and 1, but this does not affect the resolution of the top-N recommendation. Since our goal is to recommend top-N items, we only need to care about the ranking of the value of $\boldsymbol{x_i}$.

Our experimental results show that the accuracy and quality of the top-N recommendation of I-NMF is higher than that of CIDAE and O-NeuMF-p, and is also better than some of the current advanced recommendation algorithms. This confirms from the side that I-NMF can combine the advantages of the two models, and I-NMF achieves good recommendation results by using the idea of ensemble learning.

5. Experiments

There are three sets of experiments in this paper. They are the experiment of CIDAE, the experiment of O-NeuMF-p, and the experiment of I-NMF. In this section, we answer the following questions through the three sets of experiments:

- Q1: Is CIDAE better than DAE and AE in the top-N recommendation?
- Q2: Is O-NeuMF-p better than NeuMF-p in terms of the accuracy and quality of the recommended results? What is the degree of optimization of NeuMF-p?
- Q3: Is I-NMF better than CIDAE and O-NeuMF-p in terms of the accuracy and quality of the recommended results?
- Q4: Is I-NMF better than some of the current advanced recommendation models?

5.1. Experimental Datasets

We evaluate all models using the public MovieLens (ML) [19] datasets. We specifically select the ML-100K dataset and the ML-1M dataset in MovieLens (ML). These two datasets are commonly used for evaluating the performance of recommender systems [25,26]. The specific information of the two datasets is shown in Table 1. It is worth emphasizing that our experiments rely very little on datasets. Specifically, we only use user IDs, item IDs, and ratings.

Table 1. Statistics of the evaluation datasets.

Datasets	Ratings	Items	Users	Sparsity
ML-100K	100,000	1682	943	93.70%
ML-1M	1,000,209	3706	6040	95.53%

Each rating in the two datasets is an integer from 1 to 5. Since this paper discusses the top-N recommendation, we only need to focus on whether the user is interested in the item. So, similar to [1,16], we convert the rating to 0 or 1, indicating whether the user has rated the item.

5.2. Performance Metrics

We use Hit Ratio (HR), Normalized Discounted Cumulative Gain (NDCG) [27], and Mean Average Precision (MAP) to evaluate the performance of models. The value of HR can directly reflect the accuracy of the ranking. The larger the HR value, the higher the accuracy of the ranking. NDCG can account for the position of the hit by assigning higher scores to hits at top ranks [1]. The larger the NDCG value, the higher the ranking quality [2]. Mean average precision calculates the mean of users' average precision. The larger the MAP value, the better the effect of the model.

For each model, we calculate HR@N (N = 5, 10, 15), NDCG@N (N = 5, 10, 15), and MAP@N (N = 5, 10, 15) for each test user and report the average score. It is worth noting that N can be any value. In our paper, the reason we set the value of N to 5, 10, and 15 is to have a span to show the recommended effect of our model under different values of N.

5.3. Experimental Method

To ensure the reliability of the experimental data, we introduce five-fold cross-validation. In five-fold cross-validation, the original sample is randomly partitioned into five equal sized subsamples. Of the five subsamples, a single subsample is retained as the validation data for testing the model, and the remaining four subsamples are used as training data. The cross-validation process is then repeated five times, with each of the five subsamples used exactly once as the validation data. The five results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

We use five-fold cross-validation for all algorithms (including comparison algorithms) on the ML-100K dataset and the ML-1M dataset. Therefore, the experimental results of each algorithm are the average of five-fold cross-validation. It is worth noting that all our models' parameter experiments are performed only in a certain fold cross-validation process to avoid overfitting.

In order to prove that our algorithm is statistically better than other algorithms, rather than the result of experimental fluctuations, we compare the relative increase rate of the experimental results with the standard deviation of the five-fold cross-validation of our algorithm. For a more intuitive expression, we record the increase rate of the A algorithm relative to the B algorithm as A%B (specific calculation method is $(A - B)/B$). Furthermore, we record the standard deviation of the five-fold cross-validation of the A algorithm as A-SD. If the relative increase rate (A%B) is greater than the standard deviation (A-SD), our algorithm (A) is statistically superior to another algorithm (B).

5.4. Performance Comparison

The models used for comparison are as follows:

- ItemPop: This model judges the popularity of the item by calculating the number of interactions, and then recommends items with high popularity. ItemPop is a non-personalized recommendation model [28].
- ItemKNN [4]: This model is a standard item-based collaborative filtering model. The item recommended by this model is similar to the item that the user once liked.
- BPR [28]: BPR is the state-of-the-art method for recommendation based on implicit feedback. This model optimizes the MF model based on Bayesian theory and gives personalized rankings. It is a highly competitive baseline for item recommendation.
- WRMF [12,29]: WRMF is a recommendation algorithm based on weight matrix, and it is an optimization algorithm of the MF model. The characteristic of this model is that it assigns different weights to the positive sample and the negative sample.
- CDAE [30]: It is a collaborative denoising autoencoder for collaborative filtering with implicit feedback. CDAE additionally plugs a user node to the input of autoencoders for reconstructing the user's ratings.
- ExpoMF [31]: Exposure MF (abbreviated as ExpoMF) is a new probabilistic approach that directly incorporates user exposure to items into collaborative filtering. The exposure is modeled as a latent variable and the model infers its value from data.

5.5. Experiment of the CIDAE Model

In this part, firstly, we compare the CIDAE model with the DAE algorithm and the AE algorithm on the ML-100K dataset and the ML-1M dataset. Then, we analyze the influence of parameters of CIDAE on experimental results using the ML-100K dataset.

Table 2 shows that the performance of the CIDAE model is better than the DAE algorithm and the AE algorithm on both datasets. Additionally, it can be found in Table 2 that CIDAE performs better on the ML-1M dataset. Compared with the AE algorithm, the advantage of CIDAE is that it can alleviate the problem of data sparsity by using the idea of imputation. Compared with the DAE algorithm, CIDAE does not add noise randomly, but performs regular continuous imputation on the output data. Therefore, the recommended ranking of CIDAE achieves a higher accuracy and better quality.

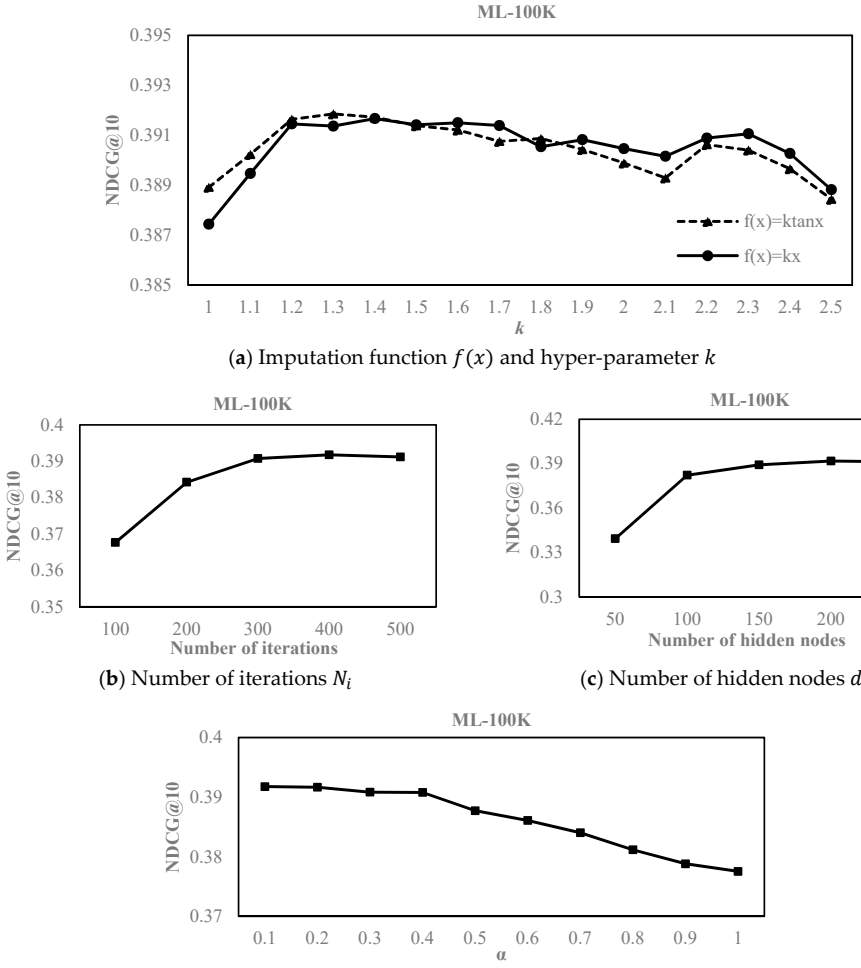
We also analyze the influence of parameters of CIDAE on experimental results on the ML-100K dataset. The specific parameters are as follows:

1. $f(x)$: Imputation function;
2. k: Hyper-parameter in imputation function $f(x)$;
3. N_i: Number of iterations;
4. d: Number of hidden nodes;
5. α: The loss weight of the imputed data.

Table 2. Performance comparison of CIDAE, DAE, and AE based on HR@N, NDCG@N, and MAP@N. (N = 5, 10, 15).

Datasets	Models	HR@5	HR@10	HR@15	NDCG@5	NDCG@10	NDCG@15	MAP@5	MAP@10	MAP@15
ML-100K	**CIDAE**	**0.421306**	**0.355360**	**0.314747**	**0.447365**	**0.392567**	**0.356907**	**0.339666**	**0.255331**	**0.210620**
	DAE	0.381883	0.326142	0.291124	0.402781	0.356766	0.326358	0.297374	0.223106	0.184123
	AE	0.386891	0.327394	0.292639	0.409233	0.360020	0.329322	0.300240	0.223058	0.183858
	CIDAE%AE	0.088952	0.085419	0.075549	0.093179	0.090405	0.083765	0.131316	0.144688	0.145561
	CIDAE-SD	0.007510	0.004360	0.003198	0.008413	0.005948	0.004636	0.006023	0.002972	0.002202
ML-1M	**CIDAE**	**0.402267**	**0.344528**	**0.309183**	**0.423420**	**0.375963**	**0.345131**	**0.322861**	**0.247092**	**0.206846**
	DAE	0.348828	0.304988	0.276929	0.364341	0.328476	0.304359	0.269107	0.207003	0.174379
	AE	0.350492	0.304143	0.275353	0.368488	0.330032	0.304874	0.270459	0.205960	0.172438
	CIDAE%AE	0.147721	0.132783	0.122857	0.149075	0.139169	0.132046	0.193753	0.199708	0.199540
	CIDAE-SD	0.001563	0.001495	0.001195	0.001480	0.001417	0.001243	0.001452	0.001454	0.001150

Figure 4 uses NDCG@10 as an indicator to show the performance of various parameters on CIDAE in the ML-100K dataset. Specifically, Figure 4a shows that different imputation functions $f(x)$ have different experimental results, but the difference is not obvious. So, we finally chose $f(x) = kx$ because its form is relatively simple. Conversely, the value of k can significantly affect the experimental results and we finally chose $f(x) = 1.4x$ because it works best. Figure 4b shows that when the number of iterations is 400, the accuracy of CIDAE is converged. Figure 4c shows that CIDAE achieves the highest accuracy when the number of hidden nodes is 200. So, we finally chose $N_i = 400$ and $d = 200$. Figure 4d shows that when $\alpha = 0.1$, CIDAE shows the highest accuracy. In addition, we can see that the accuracy is the worst when $\alpha = 1$. This confirms that it is meaningful to calculate the loss of the imputed data separately with the weight α.



(**a**) Imputation function $f(x)$ and hyper-parameter k

(**b**) Number of iterations N_i

(**c**) Number of hidden nodes d

(**d**) The loss weight of the imputed data α

Figure 4. Performance of various parameters in CIDAE.

5.6. Experiment of the O-NeuMF-p Model

In this part, firstly, we compare O-NeuMF-p with NeuMF-p on the ML-100K dataset and the ML-1M dataset. Then, we count the degree of optimization of NeuMF-p. Finally, we analyze the influence of the parameter of O-NeuMF-p on experimental results on both datasets.

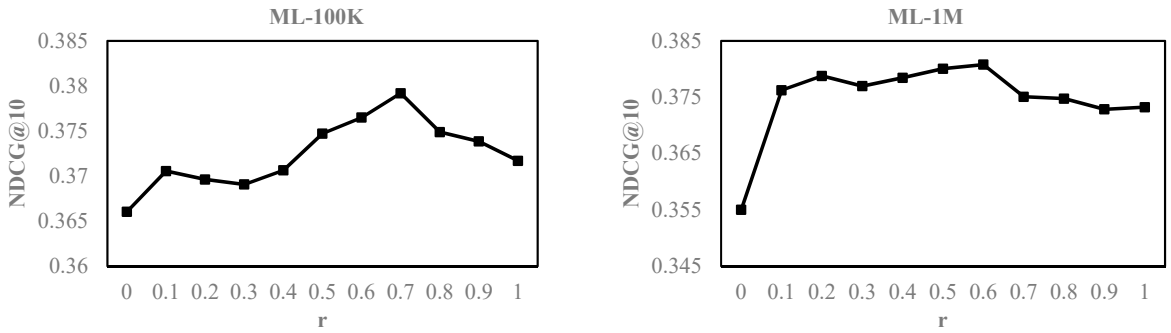
Table 3 shows that the performance of O-NeuMF-p is better than NeuMF-p on both datasets. Furthermore, it can be found in Table 3 that the optimization effect of NeuMF-p is more obvious on the ML-1M dataset. The O-NeuMF-p model is more focused on the learning of counterexamples than the NeuMF-p model, and is more robust. Additionally, the experimental results confirm that the idea of introducing random numbers into the label y_{uv} in y^- is beneficial and improves the accuracy of NeuMF-p.

Table 3. Performance comparison of O-NeuMF-p and NeuMF-p based on HR@N, NDCG@N, and MAP@N. (N = 5, 10, 15).

Datasets	Models	HR@5	HR@10	HR@15	NDCG@5	NDCG@10	NDCG@15	MAP@5	MAP@10	MAP@15
ML-100K	**O-NeuMF-p**	**0.403139**	**0.341947**	**0.304335**	**0.427511**	**0.375740**	**0.342528**	**0.324252**	**0.244377**	**0.202115**
	NeuMF-p	0.390153	0.332802	0.297135	0.412432	0.364942	0.333371	0.308308	0.233447	0.193468
	O-NeuMF-p %NeuMF-p	0.033284	0.027478	0.024231	0.036562	0.029586	0.027469	0.051713	0.046820	0.044698
	O-NeuMF-p -SD	0.006162	0.002480	0.002021	0.006155	0.003796	0.002892	0.007397	0.004844	0.003379
ML-1M	**O-NeuMF-p**	**0.401836**	**0.349859**	**0.315786**	**0.420715**	**0.377941**	**0.348670**	**0.321740**	**0.250935**	**0.212394**
	NeuMF-p	0.376824	0.332281	0.302629	0.392953	0.356452	0.331099	0.294546	0.230894	0.196529
	O-NeuMF-p %NeuMF-p	0.066374	0.052902	0.043474	0.070650	0.060286	0.053067	0.092326	0.086797	0.080725
	O-NeuMF-p-SD	0.005920	0.005609	0.005026	0.005747	0.005484	0.005080	0.005579	0.004942	0.004452

We also analyze the effect of the hyper-parameter r controlling the range of random numbers on the accuracy of O-NeuMF-p. It is worth noting that when $r = 0$, the O-NeuMF-p model is transformed into the NeuMF-p model.

Figure 5a shows that when $r = 0.7$, O-NeuMF-p achieves the highest accuracy on the ML-100K dataset. Figure 5b shows that when $r = 0.6$, O-NeuMF-p achieves the highest accuracy on the ML-1M dataset. Furthermore, it can be seen that on the ML-1M dataset, the value of r has little effect on the value of NDCG@10. It is worth emphasizing that for all values of r, the value of NDCG@10 of O-NeuMF-p is higher than that of NeuMF-p on the ML-100K dataset and the ML-1M dataset. This proves that O-NeuMF-p is more accurate than NeuMF-p on both datasets.



(**a**) On the ML-100K dataset (**b**) On the ML-1M dataset

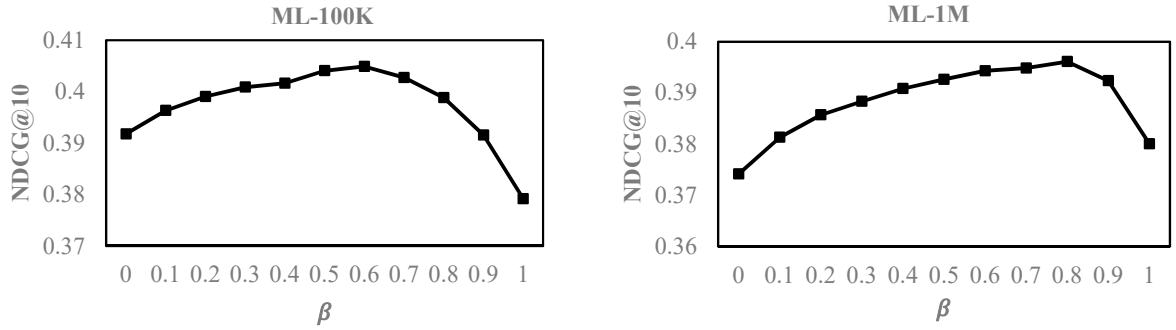
Figure 5. Performance of r based on NDCG@10.

5.7. Experiment of the I-NMF Model

In this part, firstly, we test the performance of I-NMF and compare it with CIDAE and O-NeuMF-p on the ML-100K dataset and the ML-1M dataset. Then, we analyze the influence of the parameter of I-NMF on the experimental results on both datasets. Finally, we compare I-NMF with current advanced recommendation algorithms on both datasets. The experimental results prove that our I-NMF model performs better. In addition, we also count the running time of I-NMF and comparison algorithms.

Table 4 shows that the performance of I-NMF is better than CIDAE and O-NeuMF-p on both datasets. On the one hand, the imputation ability of CIDAE can alleviate the sparseness of the original data. On the other hand, O-NeuMF-p uses the learning ability of deep neural networks to learn the interaction between users and items. Moreover, both CIDAE and O-NeuMF-p have a good robustness. It can be seen from the experimental results that CIDAE and O-NeuMF-p can complement each other to achieve a higher accuracy. Additionally, it is worth discussing that although CIDAE performs better on the ML-100K dataset, O-NeuMF-p performs better on the ML-1M dataset. This confirms the advantage of the powerful learning ability of deep neural networks for large datasets. In addition, this paper provides a new fusion idea; not only the combination of CIDAE and O-NeuMF-p, but also the combination of different types of recommendation algorithms.

We also analyze the effect of the hyper-parameter β controlling the fusion ratio on the accuracy of I-NMF. When $\beta = 0$, CIDAE is used for recommendation alone. Conversely, when $\beta = 1$, O-NeuMF-p is used for recommendation alone. Figure 6a shows that when $\beta = 0.6$, I-NMF achieves the highest accuracy on the ML-100K dataset. At this time, I-NMF is better than CIDAE and O-NeuMF-p. Figure 6b shows that when $\beta = 0.8$, I-NMF achieves the highest accuracy on the ML-1M dataset. It is worth emphasizing that for all values of β, I-NMF is better than CIDAE and O-NeuMF-p on the ML-1M dataset. This also confirms that CIDAE and O-NeuMF-p can complement each other very well.



(**a**) On the ML-100K dataset (**b**) On the ML-1M dataset

Figure 6. Performance of β based on NDCG@10.

Finally, we compare I-NMF with current advanced recommendation algorithms on both datasets. Table 5 shows that the performance of I-NMF is best on both datasets. On the ML-100K dataset, compared to the state-of-the-art method of ExpoMF, on average, I-NMF obtains 6.8%, 6.9%, and 9.1% relative improvements in HR, NDCG, and MAP metrics, respectively. On the ML-1M dataset, compared to the state-of-the-art method of ExpoMF, on average, I-NMF obtains 5.4%, 5.3%, and 6.0% relative improvements in HR, NDCG, and MAP metrics, respectively. These prove that the accuracy of our I-NMF model is very high.

In addition, we also count the running time (a fold cross-validation process) of I-NMF and comparison algorithms in Table 6. It is worth noting that our experiment is run under the CPU (Processor Intel Core i7-6700k, Memory 16GB), and if it can use the GPU to run, it may be faster.

Some parameters of our I-NMF model can affect the running time, such as: N_i (Number of iterations) and d (Number of hidden nodes) in CIDAE, the number of layers of MLP in O-NeuMF-p, and so on. But the hyper-parameters β (controlling the fusion ratio), r (controlling the range of random numbers), and α (the loss weight of the imputed data), and so on, do not affect the runtime. The training time in our model is relatively large. However, during prediction, the running time is negligible, where it only takes 18.2 s for the ML-100K dataset and 2 min for the ML-1M dataset. In addition to this, we can optimize the training time in the future.

Table 4. Performance comparison of I-NMF, CIDAE, and O-NeuMF-p based on HR@N, NDCG@N, and MAP@N. (N = 5, 10, 15).

Datasets	Models	HR@5	HR@10	HR@15	NDCG@5	NDCG@10	NDCG@15	MAP@5	MAP@10	MAP@15
ML-100K	**I-NMF**	**0.437091**	**0.368832**	**0.324451**	**0.463643**	**0.406837**	**0.368565**	**0.359102**	**0.271408**	**0.224247**
	CIDAE	0.421306	0.355360	0.314747	0.447365	0.392567	0.356907	0.339666	0.255331	0.210620
	O-NeuMF-p	0.403139	0.341947	0.304335	0.427511	0.375740	0.342528	0.324252	0.244377	0.202115
	I-NMF%CIDAE	0.037467	0.037911	0.030830	0.036388	0.036349	0.032663	0.057221	0.062963	0.064696
	I-NMF-SD	0.005536	0.002196	0.002866	0.005829	0.003351	0.003575	0.004329	0.001878	0.001698
ML-1M	**I-NMF**	**0.422925**	**0.365302**	**0.328502**	**0.442975**	**0.396000**	**0.364304**	**0.343087**	**0.267112**	**0.225634**
	CIDAE	0.402267	0.344528	0.309183	0.423420	0.375963	0.345131	0.322861	0.247092	0.206846
	O-NeuMF-p	0.401836	0.349859	0.315786	0.420715	0.377941	0.348670	0.321740	0.250935	0.212394
	I-NMF% O-NeuMF-p	0.052483	0.044140	0.040269	0.052909	0.047784	0.044839	0.066348	0.064468	0.062334
	I-NMF-SD	0.003211	0.003311	0.002990	0.003393	0.003400	0.003020	0.003694	0.003469	0.002841

Table 5. Performance comparison of I-NMF and advanced recommendation algorithms based on HR@N, NDCG@N, and MAP@N. (N = 5, 10, 15).

Datasets	Models	HR@5	HR@10	HR@15	NDCG@5	NDCG@10	NDCG@15	MAP@5	MAP@10	MAP@15
ML-100K	**I-NMF**	**0.437091**	**0.368832**	**0.324451**	**0.463643**	**0.406837**	**0.368565**	**0.359102**	**0.271408**	**0.224247**
	ItemPop	0.213072	0.189456	0.168726	0.222822	0.203229	0.186199	0.141576	0.104503	0.084945
	BPR	0.252880	0.217062	0.194386	0.265752	0.236420	0.216823	0.180987	0.131805	0.107027
	ItemKNN	0.353631	0.293864	0.256773	0.378404	0.328200	0.295540	0.279124	0.204424	0.165260
	WRMF	0.358073	0.306407	0.274969	0.379183	0.336027	0.308229	0.276228	0.207112	0.171358
	CDAE	0.381714	0.325527	0.290687	0.402744	0.356477	0.326134	0.297378	0.223183	0.184031
	ExpoMF	0.410737	0.343349	0.304449	0.434643	0.379345	0.345066	0.331008	0.248299	0.204915
	I-NMF% ExpoMF	0.064161	0.074218	0.065699	0.066722	0.072470	0.068098	0.084873	0.093068	0.094341
	I-NMF-SD	0.005536	0.002196	0.002866	0.005829	0.003351	0.003575	0.004329	0.001878	0.001698
ML-1M	**I-NMF**	**0.422925**	**0.365302**	**0.328502**	**0.442975**	**0.396000**	**0.364304**	**0.343087**	**0.267112**	**0.225634**
	ItemPop	0.210982	0.182530	0.167163	0.221308	0.197872	0.184095	0.152467	0.112520	0.092966
	BPR	0.290552	0.248252	0.222520	0.306210	0.271397	0.248911	0.221773	0.164860	0.135679
	ItemKNN	0.343951	0.287240	0.252039	0.365079	0.318339	0.287692	0.271287	0.201249	0.164336
	WRMF	0.386441	0.338752	0.308109	0.403729	0.364616	0.338220	0.299831	0.232759	0.197326
	CDAE	0.397328	0.340984	0.306816	0.419981	0.372899	0.342745	0.321560	0.247175	0.208099
	ExpoMF	0.401107	0.346231	0.312267	0.420421	0.375562	0.346169	0.324212	0.251804	0.212747
	I-NMF% ExpoMF	0.054395	0.055081	0.051991	0.053646	0.054420	0.052387	0.058218	0.060794	0.060574
	I-NMF-SD	0.003211	0.003311	0.002990	0.003393	0.003400	0.003020	0.003694	0.003469	0.002841

Table 6. Running time of I-NMF and comparison algorithms.

Datasets	Time	I-NMF	ItemPop	BPR	ItemKNN	WRMF	CDAE	ExpoMF
ML-100K	**Train Time**	2.4 min	1 s	11 min	5 s	5.5 s	1.3 min	2.2 min
	Test Time	18.2 s	1 s	2.6 s	36.7 s	2.5 s	1.4 min	2.7 s
ML-1M	**Train Time**	51.3 min	5 s	111 min	1.4 min	1.2 min	26.6 min	16.4 min
	Test Time	2 min	28 s	40 s	25 min	40 s	66 min	38 s

6. Conclusions and Future Work

Firstly, we propose a new CIDAE model based on DAE. The advantage of CIDAE is that it alleviates the problem of data sparsity by using the idea of imputation. Additionally, our experimental results show that CIDAE is more accurate than the AE algorithm and the DAE algorithm.

Then, we optimize the advanced NeuMF model based on the training process. O-NeuMF-p is more focused on the learning of counterexamples than NeuMF-p, and is more robust. Furthermore, the experimental results confirm that the idea of introducing random numbers into the label y_{uv} in y^- is beneficial and improves the accuracy of NeuMF-p.

Finally, this paper fuses CIDAE and O-NeuMF-p with reference to the idea of ensemble learning. I-NMF can not only alleviate the problem of data sparsity, but also fully exploit the ability of deep neural networks to learn potential features, and I-NMF is also robust. It can be seen from the experimental results that CIDAE and O-NeuMF-p can complement each other to achieve a higher accuracy. Moreover, our experimental results prove that I-NMF performs better than current advanced recommendation algorithms on both datasets.

It is worth emphasizing that this paper provides a new fusion idea; not only the combination of CIDAE and O-NeuMF-p, but also the combination of different types of recommendation algorithms.

In the future, there are two directions to extend our work. On the one hand, we can try more ways to combine different kinds of recommendation algorithms to achieve a better accuracy. On the other hand, we can try to incorporate auxiliary extra data into recommender systems, such as social relations, review text, type of items, personal information of users, and so on. We can further improve the accuracy of the model by adding extra valuable data.

Author Contributions: Conceptualization, X.W. and B.Z.; Data curation, X.W.; Formal analysis, X.W.; Funding acquisition, B.Z., G.Z., and F.C.; Investigation, X.W. and F.C.; Methodology, X.W.; Project administration, B.Z. and G.Z.; Resources, B.Z. and G.Z.; Software, X.W.; Supervision, B.Z.; Validation, X.W.; Visualization, X.W. and F.C.; Writing—original draft, X.W.; Writing—review & editing, X.W. and F.C.

Funding: This work was partially supported by the National Key Research and Development Program of China (No. 2017YFC0907505), National Natural Science Foundation of China (No. 61772128, 61303096), Shanghai Natural Science Foundation (No. 18ZR1414400, 17ZR1400200), Xinjiang Social Science Foundation (No. 2015BGL100), and Fundamental Research Funds for the Central Universities (No. 16D111208).

Acknowledgments: Thanks for experimental equipment support provided by Kanwei Zheng.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.-S. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.
2. Xue, H.-J.; Dai, X.-Y.; Zhang, J.; Huang, S.; Chen, J. Deep matrix factorization models for recommender systems. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 3203–3209.
3. Wang, H.; Wang, N.; Yeung, D.-Y. Collaborative Deep Learning for Recommender Systems. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, Australia, 10–13 August 2015; pp. 1235–1244.

4. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web, Hong Kong, China, 1–5 May 2001; pp. 285–295.
5. Zhang, H.; Shen, F.; Liu, W.; He, X.; Luan, H.; Chua, T.-S. Discrete Collaborative Filtering. In Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, 17–21 July 2016; pp. 325–334.
6. He, X.; Zhang, H.; Kan, M.-Y.; Chua, T.-S. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, 17–21 July 2016; pp. 549–558.
7. Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV, USA, 24–27 August 2008; pp. 426–434.
8. Billsus, D.; Pazzani, M.J. Learning Collaborative Information Filters. In Proceedings of the Fifteenth International Conference on Machine Learning, San Francisco, CA, USA, 24–27 July 1998; pp. 46–54.
9. Koren, Y.; Bell, R.; Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
10. Li, Y.; Hu, J.; Zhai, C.; Chen, Y. Improving one-class collaborative filtering by incorporating rich user information. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management, Toronto, ON, Canada, 26–30 October 2010; pp. 959–968.
11. Pan, R.; Scholz, M. Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 667–676.
12. Pan, R.; Zhou, Y.; Cao, B.; Liu, N.N.; Lukose, R.; Scholz, M.; Yang, Q. One-Class Collaborative Filtering. In Proceedings of the Eighth IEEE International Conference on Data Mining, Pisa, Italy, 15–19 December 2008; pp. 502–511.
13. Sindhwani, V.; Bucak, S.S.; Hu, J.; Mojsilovic, A. One-Class Matrix Completion with Low-Density Factorizations. In Proceedings of the 2010 IEEE International Conference on Data Mining, Sydney, Australia, 13–17 December 2010; pp. 1055–1060.
14. Yao, Y.; Tong, H.; Yan, G.; Xu, F.; Zhang, X.; Szymanski, B.K.; Lu, J. Dual-Regularized One-Class Collaborative Filtering. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, Shanghai, China, 3–7 November 2014; pp. 759–768.
15. Rendle, S. Factorization Machines. In Proceedings of the 2010 IEEE International Conference on Data Mining, Washington, DC, USA, 13–17 December 2010; pp. 995–1000.
16. Lee, J.-W.; Lee, J. IDAE: Imputation-boosted Denoising Autoencoder for Collaborative Filtering. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Singapore, 6–10 November 2017; pp. 2143–2146.
17. Bauer, J.; Nanopoulos, A. Recommender systems based on quantitative implicit customer feedback. *Decis. Support Syst.* **2014**, *68*, 77–88. [CrossRef]
18. Wu, H.; Zhang, Z.; Yue, K.; Zhang, B.; He, J.; Sun, L. Dual-regularized matrix factorization with deep neural networks for recommender systems. *Knowl. Based Syst.* **2018**, *145*, 46–58. [CrossRef]
19. Harper, F.M.; Konstan, J.A. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* **2015**, *5*, 1–19. [CrossRef]
20. Bengio, Y. Learning Deep Architectures for AI. *Found. Trends®Mach. Learn.* **2009**, *2*, 1–127. [CrossRef]
21. Sedhain, S.; Menon, A.K.; Sanner, S.; Xie, L. AutoRec: Autoencoders Meet Collaborative Filtering. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 111–112.
22. Strub, F.; Gaudel, R.; Mary, J. Hybrid Recommender System based on Autoencoders. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, Boston, MA, USA, 15 September 2016; pp. 11–16.
23. Glorot, X.; Bordes, A.; Bengio, Y. Deep Sparse Rectifier Neural Networks. In Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.

24. Dong, X.; Yu, L.; Wu, Z.; Sun, Y.; Yuan, L.; Zhang, F. A Hybrid Collaborative Filtering Model with Deep Structure for Recommender Systems. In Proceedings of the AAAI Conference on Artificial Intelligence (2017), San Francisco, CA, USA, 4–9 February 2017.
25. Wang, H.; Shi, X.; Yeung, D.-Y. Relational stacked denoising autoencoder for tag recommendation. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015; pp. 3052–3058.
26. Li, S.; Kawale, J.; Fu, Y. Deep Collaborative Filtering via Marginalized Denoising Auto-encoder. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 18–23 October 2015; pp. 811–820.
27. He, X.; Chen, T.; Kan, M.-Y.; Chen, X. TriRank: Review-aware Explainable Recommendation by Modeling Aspects. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 18–23 October 2015; pp. 1661–1670.
28. Rendle, S.; Freudenthaler, C.; Gantner, Z.; Schmidt-Thieme, L. BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, 18–21 June 2009; pp. 452–461.
29. Hu, Y.; Koren, Y.; Volinsky, C. Collaborative Filtering for Implicit Feedback Datasets. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 15–19 December 2008; pp. 263–272.
30. Wu, Y.; DuBois, C.; Zheng, A.X.; Ester, M. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, 22–25 February 2016; pp. 153–162.
31. Liang, D.; Charlin, L.; McInerney, J.; Blei, D.M. Modeling user exposure in recommendation. In Proceedings of the 25th International Conference on World Wide Web, Montréal, QC, Canada, 11–15 April 2016; pp. 951–961.