International Journal of Computational Intelligence Systems (2022) 15:45
https://doi.org/10.1007/s44196-022-00102-8

RESEARCH ARTICLE

DeepRS: A Library of Recommendation Algorithms Based on Deep Learning

Hongwei Tao[1] · Xiaoxu Niu[1] · Lianyou Fu[1] · Shuze Yuan[1] · Xiao Wang[1] · Jiaxue Zhang[1] · Yinghui Hu[1]

Received: 9 February 2022 / Accepted: 27 June 2022

Abstract

In recent years, recommendation systems have become more complex with increasing research on user preferences. Recommendation algorithm based on deep learning has attracted a lot of attention from researchers in academia and industry, and many new algorithm models are proposed every year. Researchers often need to implement the proposed model to compare the results, which is a great challenge. Even if some papers provide source code, there are a variety of programming languages or deep learning frameworks, and it is not easy to compare the results in the different frameworks. In view of the lack of easily extensible deep learning-based recommendation algorithm libraries, based on the common analysis of deep learning algorithms in attention factorization machine (AFM), neural factorization machine (NFM), deep factorization machine (DeepFM) and deep cross-network (DCN), a recommendation algorithm library based on deep learning (DeepRS for short) is designed and implemented. It consists of three levels: framework level, abstract level and algorithm level. The framework level adopts the Tensorflow open source framework, which provides interfaces, such as automatic differentiation, tensor computing, GPU computing, and numerical optimization algorithms. The abstraction level uses the interface of the framework level to realize the embedding layer (EL), the full connection layer (FCL), the multi-layer perceptron layer (MLPL), the prediction layer (PL), the factorization machine layer (FML), the attention network layer (ANL), the cross-layer (CL) and the cross-network layer (CNL). The algorithm level implements the deep learning-based recommendation algorithms, such as AFM, NFM, DeepFM and DCN, on the basis of the abstraction level and the framework level. Experiments show that the proposed algorithm library has good scalability, ease of use and correctness.

Keywords Recommendation algorithm library · Deep learning · Tensorflow · Abstraction layer

✉ Hongwei Tao
hongweitao@zzuli.edu.cn

Xiaoxu Niu
541812030214@zzuli.edu.cn

Lianyou Fu
fly13233761631@163.com

Shuze Yuan
541507120151@zzuli.edu.cn

Xiao Wang
pandaxiaoxi@gmail.com

Jiaxue Zhang
541803010252@zzuli.edu.cn

Yinghui Hu
hyingh6@163.com

[1] College of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, China

Abbreviations

AFM	Attention factorization machine
NFM	Neural factorization machine
DeepFM	Deep factorization machine
DCN	Deep and cross networks
DeepRS	Recommendation algorithm library based on deep learning
EL	Embedding layer
FCL	Full connection layer
MLPL	Multi-layer perceptron layer
PL	Prediction layer
FML	Factorization machine layer
ANL	Attention network layer
CL	Cross layer
CNL	Cross-network layer
DNN	Deep neural networks
AUC	Area under curve
MLP	Multi-layer perceptron
LF	Latent Factor

Published online: 14 July 2022

1 Introduction

The rapid development of information technology, computer technology and sensor technology affects all walks of life and permeates every aspect of people's lives. Convenient social platforms and search engines not only provide convenience, but also generate huge amounts of information. Both consumers and content providers are paying more and more attention to how to extract target content from massive multi-source heterogeneous data. In this context, the recommendation systems are currently a successful solution [1]. Recommendation system is an information filtering tool, which processes the data generated by the user's daily behavior, analyzes the user's preferences for different entities or content with the obtained results, and recommends the relevant entities or content of interest according to the user's preferences [2]. In general, the recommendation system can filter a large amount of information effectively and recommend resources that meet users' needs. It has been widely used in many fields, such as commodity purchase and audio–visual recommendation [2, 3].

Recommendation algorithm is the core of recommendation system. The basic characteristics of traditional collaborative or content-based filtering algorithm models are artificial construction, inability to train end-to-end, poor recommendation quality, unable to deal with sparse data and cold start problems, as well as the failure to balance different evaluation indexes in recommendation quality. With the wide application of deep learning in computer vision, speech recognition and many other fields, academia and industry are racing to apply deep learning to a wider range of applications [4], because it can solve many complex problems, and provide good results, among which the recommendation system is one of its application fields. The introduction of deep learning into the recommendation system has greatly revolutionized the recommendation system architecture and brought more opportunities to recreate user experience to achieve higher customer satisfaction [5]. The recommendation system algorithm based on deep learning over- comes the limitations of the traditional algorithm model and realizes high-quality recommendation. It can not only automatically capture the nonlinear and non-trivial relationship between users and products, and encode more complex abstractions into higher-level data, but also find the complex relationship within the data itself from a large number of accessible data sources (such as pictures, texts, etc.) [6, 7].

However, the large number of recommendation algorithm models based on deep learning also poses great challenges for researchers and practitioners, because they need to reproduce the results of the existing models to evaluate the merits of the new algorithm models. Although some authors provide source code for reproducing, they use a variety of programming languages and deep learning frameworks, not to mention that most authors do not provide source code, which makes it difficult to understand and reproduce the model. Therefore, there is an urgent need for recommending algorithm libraries to solve the above problems. Most of the existing algorithm libraries are based on traditional recommendation algorithms, such as Mymedialite [8], Librec [9], etc. They regard the recommendation algorithm as a single whole. To make innovative modifications to the algorithm, researchers may need to re-implement the whole framework from scratch. Moreover, the models in these libraries cannot be trained end-to-end, and even some algorithm libraries are not written in Python, the first language of machine learning, so they cannot be well integrated into the existing recommendation services. At present, there is also an algorithm library based on deep learning—OpenRec [10], but it only provides the interfaces of the recommendation system model based on deep learning and have a few abstract components, so it does not have good scalability.

To deal with these challenges, this paper proposes a recommendation algorithm library based on deep learning—DeepRS. DeepRS is an extensible algorithm library consisting of a framework level, an abstract level, and an algorithm level. The framework level adopts the open source framework Tensorflow, which provides interfaces, such as automatic differential, tensor calculation, GPU calculation and numerical optimization algorithm. Abstract level uses the interfaces of the framework level to implement eight abstract components, and algorithm level realizes AFM [11], NFM [12], DeepFM [13], DCN [14] and other deep recommendation algorithms based on the abstract level and framework level. DeepRS enriches the recommendation algorithm toolbox, fills the gap in the recommendation algorithm library based on deep learning, makes it easier to reproduce the results of the recommendation model, and lowers the threshold of developing the recommendation model based on deep learning.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 introduces the architecture design and related algorithms of DeepRS, and describes in detail the design of framework layer, abstraction layer and algorithm layer in the algorithm package. Section 4 presents the experimental analysis of the rationality of the algorithm package, introduces various experimental environment settings and the training error and accuracy of NFM, AFM, DeepFM and DCN in these experimental environments. The comparative study is presented in Sect. 5. The paper is ended with the conclusion and future work in the last section.

2 Related Work

This section first introduces the traditional recommendation methods, then describes the deep learning-based recommendation methods, and finally gives the libraries related to the recommendation methods.

Traditional recommendation methods are mainly divided into three categories: collaborative filtering recommendation method, content-based recommendation method and hybrid recommendation method [2, 15]. Collaborative filtering recommendation method finds user preferences by mining users' historical behavior data, groups users based on different preferences, and recommends products with similar tastes [16]. This method is easy to use and simple. It can only calculate the similarity between users according to the historical scoring data of users. However, in many cases, it often encounters the problem of sparse matrix caused by insufficient scoring data and the cold start problem of new users without project scoring data [17, 18]. The content-based recommendation method is mainly based on the feature information between users and projects. The relationship between users will not affect the recommendation results, so there are no problems of cold start and sparse matrix [19, 20]. However, the recommendation results of this kind of methods are low in novelty, and they face the problem of feature extraction [21]. Hybrid recommendation method combines the characteristics of the first two traditional recommendation methods and can achieve good recommendation results, but it still faces some challenges and difficulties in processing multi-source heterogeneous auxiliary information, such as text and image [22, 23].

In recent years, deep learning has developed rapidly in the field of recommendation system. Recommendation methods based on deep learning technology can not only learn the potential feature representation of users or projects, but also learn the complex nonlinear interaction characteristics between users and projects [5–7]. They can deeply analyze user preferences, solve some problems in traditional recommendation methods, and better realize recommendation [24]. According to the neural network, the recommendation methods based on deep learning are mainly divided into four categories: the recommendation methods based on deep neural network, the recommendation methods based on convolutional neural network, the recommendation methods based on cyclic neural network and short-term memory neural network, and the recommendation methods based on graph neural network [15].

MyMediaLite is a recommendation library written in C# and runs on the.NET platform. It addresses two common scenarios in collaborative filtering: rating prediction and item prediction from positive-only implicit feedback [8]. LibRec is a Java library for recommender systems. It mainly identifies and implements the baselines that rarely use personalized information and traditional recommendation algorithms based on user–item interaction and context information [9]. The LibRec framework consists of three major components: generic interfaces, data structures and recommendation algorithms [9]. OpenRec is an open and modular Python framework for neural network-inspired recommendation algorithms. Each recommender is modeled as a computational graph [10]. It aims to simplify the process of extending and adapting the most advanced neural recommender to heterogeneous recommender scenarios. PREA is also implemented in Java. It mainly supports traditional recommendation algorithms, such as memory-based neighborhood algorithms, matrix factorization methods [25]. All performance critical codes of the library fastFM are written in C and wrapped in Cython. Fastfm supports random gradient descent and coordinate descent optimization routines as well as Markov chain Monte Carlo for Bayesian reasoning. Its solvers can be used for regression, classification and ranking problems [26]. Surprise is a Python library for recommender systems. It implements classical algorithms, such as the main similarity-based algorithms, and algorithms based on matrix factorization, such as singular value decomposition and non-negative matrix factorization [27]. Recommenderlab is a recommendation system library based on R language. It mainly focuses on collaborative filtering recommendation algorithm. The code of the recommendation system developed based on this framework will be very concise, but the flexibility of recommenderlab in practical application is slightly insufficient [28].

3 Recommendation Algorithm Library Design

The deep learning-based recommendation algorithm library DeepRS uses Python as the development language and PyPI as the third-party warehouse for development. Its

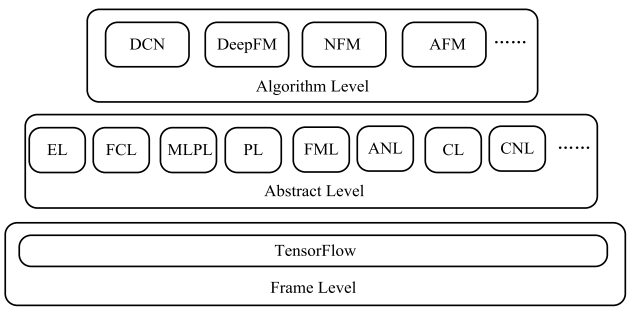

Fig. 1 Architecture diagram of DeepRS

architecture is shown in Fig. 1. It consists of three levels: framework level, abstraction level and algorithm level. The framework level is built on Tensorflow, so the recommendation system can easily take advantage of modern hardware such as GPU and expand to distribute computing environment. The framework level provides automatic differentiation, tensor calculation, GPU calculation, numerical optimization algorithm and other interfaces, which is the cornerstone of the DeepRS. The abstract level uses the interfaces in the framework level to implement the interfaces of EL, FCL, MLPL, PL, FML, ANL, CL and CNL. On the basis of the abstract level and framework level, the algorithm level realizes AFM, NFM, DeepFM, DCN and other deep learning-based recommendation algorithms.

3.1 Framework Level

Tensorflow is one of the most frequently used machine learning frameworks. It is an open source software library using data flow graph for numerical calculation. It can easily assign a single node to different computing devices to complete asynchronous parallel computing. It is very suitable for large-scale machine learning applications. It also allows deep neural network computing to be deployed to any number of CPUs and GPUs on servers and PCs using only one Tensorflow API. The framework level is the cornerstone of DeepRS and needs to implement various computing interfaces. DeepRS directly uses Tensorflow as the framework level to provide relevant inter- faces for upper-level services.

3.2 Abstract Level

The abstract level represents the reusable components in the recommendation algorithm. It includes embedding layer, full connection layer, multi-layer perceptron layer, prediction layer, factorization machine layer, attention network layer, cross-layer and cross-network layer.

3.2.1 Embedded Layer

The embedding layer is essentially a network layer that removes the activation functions in the full connection layer, but does not perform nonlinear function mapping on the output. Its function is to process the high-dimensional sparse feature vectors encoded by one-hot to ensure that the depth recommendation model can find the optimal value. Its representation is shown in Eq. (1).

$$x \mapsto x^T M \tag{1}$$

The parameters to be estimated is $M \in R^{m\times n}(n << m)$, where $x^{\mathrm{T}} \in R^{m\times 1}$ and $x^T M \in R^{1\times \mathrm{n}}$ represent the one-hot encoded vector of the category feature and the embedded vector after mapping of category feature respectively, m is the total number of category features, and n is the size of the embedded vector space.

3.2.2 Full Connection Layer

The full connection layer is essentially a nonlinear transformation with the output of affine function as the input. Its main role is to learn the representation of the input. Its expression is shown in Eq. (2).

$$a^{(l+1)} = f\left(W^{(l)}a^{(l)} + b^{(l)}\right) \tag{2}$$

The parameters to be estimated are $W^{(l)} \in R^{n\times m}, b^{(l)} \in R^{n\times 1}$. Among them, l represents the l-th fully connected layer, f is the activation function, $W^{(l)}$ and $b^{(l)}$ are referred to as the l -th layer weight matrix and bias, respectively, and $a^{(l)} \in R^{m\times 1}$ is the input of the l-th layer. m and n represent the number of neurons in layer l-th layer and the number of neurons in $(l + 1)$-th layer, respectively. Generally, the activation function f is a nonlinear function. If it is a linear function, no matter how many full connection layers are superimposed, the final learned function is still linear, so it is not desirable. The common activation functions consist of sigmoid function, ReLU function, tanh function, etc.

3.2.3 Multi-Layer Perceptron Layer

The multi-layer perceptron layer is essentially a network composed of multiple fully connected layers. Its main purpose is to abstractly represent complex inputs, such as video and images, in the learning machine. The expression of MLPL is shown in Eq. (3):

$$\begin{aligned} a^{(1)} &= x \\ a^{(2)} &= f\left(W^{(1)}a^{(1)} + b^{(1)}\right) \\ &\cdots\cdots \\ a^{(L-1)} &= f\left(W^{(L-2)}a^{(L-2)} + b^{(L-2)}\right) \\ y_{MLP}(x) &= f\left(W^{(L-1)}a^{(L-1)} + b^{(L-1)}\right) \end{aligned} \tag{3}$$

The parameters to be estimated are $W^{(l)}$ and $b^{(l)}$,$W^{(l)}$ and $b^{(l)}$ are called the weight matrix and bias of the l-th layer respectively.L is the depth of the multi-layer perceptron layer,f is the activation function, $a^{(l)}$ and $a^{(l+1)}$ represent the input and output of the l-th layer. The input of the multi-layer perceptron layer is x, the output is $y_{MLP}(x)$, and the output dimension is determined by the number of hidden units of the last layer in the MLPL.

3.2.4 Prediction Layer

The prediction layer essentially a function transformation of the input data of the last layer in the model, and the appropriate function transformation can make the model easier to learn. Its expression is shown in Eq. (4).

$$y_{predict}(x) = f(x + b) \tag{4}$$

Among them, $x, b \in R^n$ represents the input and bias parameters of the last layer in the deep learning model respectively, and f is the transformation function. Common transformation functions include softmax function, sigmoid function, linear function, and so on.

3.2.5 Factorization Machine Layer

The factorization machine layer is essentially a factorization machine whose main purpose is to automatically learn feature interaction functions in the recommendation system. In addition to the linear interaction between features, it can also use the inner product of potential vectors representing features to model paired interactive features. Its expression is shown in Eq. (5):

$$y(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \tag{5}$$

The parameter to be estimated in the formula is $w_0 \in R$, $w = (w_1, ..., w_n) \in R^n$, where w_i reflects the importance of first-order features, $x = (x_1, ..., x_n) \in R^n$ is the input. v_i and v_j denote separately the i-th row vector with k factors and the j-th row vector with k factors, and k is a hyperparameter that defines the dimension of the factor. $\langle v_i, v_j \rangle$ is the dot product of two vectors of length k, which is defined as follows:

$$\langle v_i, v_j \rangle = \sum_{f=1}^{k} v_{i,f} \times v_{j,f} \tag{6}$$

$\langle v_i, v_j \rangle$ is used to represent the importance of the second-order interaction features.

3.2.6 Attention Network Layer

The attention network layer is essentially a multi-layer perceptron, whose primary purpose is to identify the importance of different feature interactions in the recommendation system. Its expression is shown in Eq. (7).

$$a'_{i,j} = h^T \mathrm{Re}LU\big(W(v_i \otimes v_j) + b\big) \tag{7}$$

The parameters to be estimated are $W \in R^{t \times k}, b \in R^{t \times 1}, h \in R^{t \times 1}$, where, t represents the number of hidden units in the attention network layer, commonly known as attention factor, and k represents the length of input vector. $v_i \in R^{t \times 1}$ and $v_j \in R^{t \times 1}$ are separately the sample values of the i-th sample and the j-th sample, and $v_i \otimes v_j$ is the Hadamard product of v_i and v_j.

3.2.7 Cross-Layer

The crossing layer is essentially a residual layer that uses an identical mapping activation function, and its primary purpose is to learn crossing characteristics. Its expression is shown in Eq. (8).

$$x_{l+1} = f\big(x_0, x_l\big) = x_0 x_l^T w_l + b_l + x_l \tag{8}$$

where $x_0 \in R^d$ is the input vector of cross-layer, $x_l, x_{l+1} \in R^d$ are separately the input and output vectors of the l-th layer, $w_l, b_l \in R^d$ represents the weight and bias parameters of the l-th layer, respectively, and f is a $R^d \to R^d$ mapping function that fits residual $x_{l+1} - x_l$.

3.2.8 Cross-Network Layer

Cross-network layer is a network composed of multiple cross-layers. Its main purpose is to efficiently learn explicit cross features. Its expression is shown in Eq. (9):

$$\begin{aligned} x_1 &= f\big(x_0, x_0\big) = x_0 x_0^T w_0 + b_0 + x_0 \\ x_2 &= f\big(x_0, x_1\big) = x_0 x_1^T w_1 + b_1 + x_1 \\ &\cdots \\ x_{l+1} &= f\big(x_0, x_l\big) = x_0 x_l^T w_l + b_l + x_l \end{aligned} \tag{9}$$

where $x_0, x_1, ..., x_{l+1} \in R^d$, x_l is the input vector of the l-th cross-layer, and x_{l+1} is the output vector of the l-th cross-layer. $w_l, b_l \in R^d$ denotes the weight bias parameters of the l-th cross-layer, which are trainable.

3.3 Algorithm Level

This subsection introduces several implemented recommendation algorithms in DeepRS: AFM, NFM, DeepFM, and DCN. The network topology and formulas of these algorithms, as well as the regularization and optimization methods used are also described.

3.3.1 AFM Algorithm

The full name of AFM is attentional factorization machine, which is made up of embedding layer, attention network layer and prediction layer. AFM is a universal machine

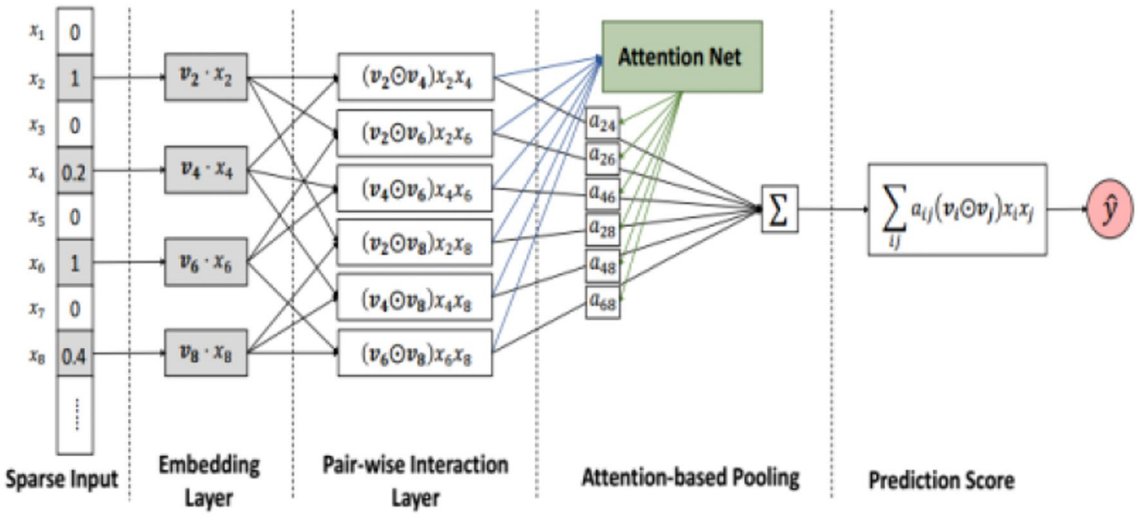

Fig. 2 Network topology of the AFM Algorithm

learner that learns any real-valued feature vectors. Its network topology is shown in Fig. 2.

The expression of AFM model is shown in Eq. (10).

$$y_{AFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + f(x) \tag{10}$$

$x = (x_1, ..., x_n) \in R^n$ is the input sparse vector. The $x_i = 0$ in the input vector means that the i-th feature does not exist in this sample. The first and second terms are the linear regression part, which are used to learn the weights between low-order features and the bias of modeling data. The third term is an attention layer-based pooling operation network, which mainly models second-order feature interactions and assigns different weights to the corresponding interactions, whose expression is Eq. (11).

$$f(x) = p^T \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{h^T ReLU\left(W\left(v_i \otimes v_j\right)x_i x_j + b\right)}{\sum_{(i,j)\in\{1,...,n\}^2} \exp\left(h^T ReLU\left(W\left(v_i \otimes v_j\right)x_i x_j + b\right)\right)} \left(v_i \otimes v_j\right)x_i x_j \tag{11}$$

After the sparse data of the input layer pass through the embedding layer, an embedding vector set $V_x = \{x_1 v_1, ..., x_n v_n\}$ is obtained, where v_i represents the embedding vector of the i-th feature. Since the input vector in the recommendation scene is very sparse, that is, most of the features are 0, the actual embedding vector set is very small, and only contains non-zero feature items, that is, $V_x = \{x_i v_i | x_i \neq 0 \wedge x_i \in x\}$. The parameters to be solved in this model are $\Theta = \{w_0, \{w_i, v_i\}, p, h, W, b\}$. AFM model can be applied to a variety of recommendation tasks, such as regression, classification and sorting tasks. For different tasks, the AFM model uses different loss functions. Log likelihood loss or hinge loss function is usually used in classification task, square loss or mean loss function is applied in regression task, and paired personalized ranking loss or contrast maximum spacing loss function is commonly used in sorting task [29]. The main focus here is on the regression task of explicitly feeding back of the real target value, and the $L2$ norm is used to prevent overfitting of the model, as shown in Eq. (12).

$$loss = \sum_{x\in X} \left(y_{AFM}(x) - y(x)\right)^2 + \lambda ||W||^2 \tag{12}$$

Among, X represents the training set, x represents an instance of the training set, W represents the weight matrix of the attention layer, and λ is used to control the intensity of regularization. The objective function is optimized using a random gradient descent algorithm, as shown in Eq. (13).

$$\theta \leftarrow \theta - \eta \times 2\left(y_{AFM}(x) - y(x)\right)\frac{dy_{AFM}(x)}{d\theta} - \eta\lambda\frac{d||W||^2}{d\theta} \tag{13}$$

where θ is the trainable model parameter, η is the learning rate, and λ is the regularization parameter. The core idea is to iteratively update parameters until the function converges. In the process of iteration, a training sample x is randomly selected each time to update in the direction of negative gradient of model parameters.

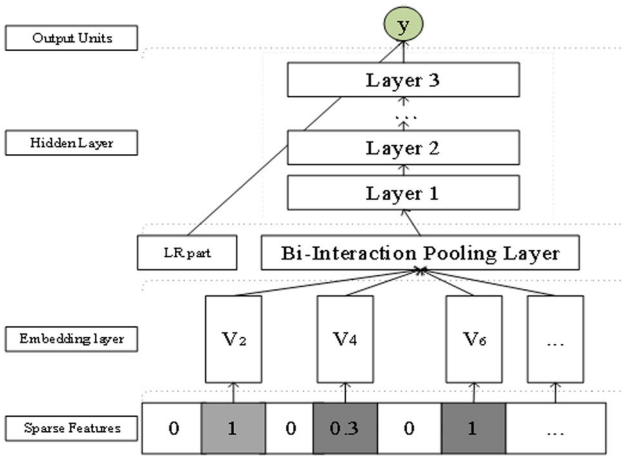

Fig. 3 Network topology of NFM Algorithm

3.3.2 NFM Algorithm

The full name of NFM is neural factorization machine, which is composed of embedding layer, multi-layer perceptron layer, full connection layer and prediction layer. Its main function is to learn the high-order interaction between sparse data in recommendation scenes. It is also a universal machine learner that can learn any real-valued eigenvectors. Its network topology is shown in Fig. 3.

The expression of the NFM model is as shown in Eq. (14).

$$y_{NFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + h^T \sigma_L\left(W_L\left(...\sigma_1\left(W_1 f_{BI}\left(V_x\right) + b_1\right)...\right) + b_L\right)$$
$$f_{BI}\left(V_x\right) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i v_i \otimes x_j v_j \tag{14}$$

Compare the expression of AFM algorithm, special emphasis should be placed on the third term, which is a multi-layer forward neural network of stacked multi-layer full connection layers. The third term is used to capture the relationships between higher-order features. In Eq. (14), L is the depth of multi-layer perceptron layer, and $W_l, b_l, \sigma_l (1 \le l \le L)$ represent the weight matrix, bias vector and activation function of l-th layer respectively. Vector h represents the weight of the prediction layer (the last layer). Parameters to be solved in this model are $\Theta = \left\{w_0, \left\{w_i, v_i\right\}, h, \left\{W_l, b_l\right\}\right\}$. Here we focus on the binary classification task recommended by implicit feedback. Therefore, the expression shown in Eq. (15) is optimized, and other tasks are processed in the same way.

$$loss = -\sum_{x \in X} \left(y(x) \log\left(y_{NFM}(x)\right) + (1 - y(x)) \log\left(\left(1 - y_{NFM}(x)\right)\right)\right) \tag{15}$$

In Eq. (15), X is the training set, x is an instance of the training set. The above objective function is also optimized by the random gradient descent, as shown in Eq. (16).

$$\theta \leftarrow \theta + \eta \times \left(\frac{f(x)}{y_{NFM}(x)} - \frac{1 - f(x)}{1 - y_{NFM}(x)}\right) \frac{dy_{NFM}(x)}{d\theta} \tag{16}$$

where θ is the trainable model parameter and η is the learning rate. The automatic differentiation interface provided by the framework layer can automatically calculate the gradient of the trainable parameters. Considering the problem of data sparse in the recommendation system based on deep learning, batch Adagrad algorithm is adopted as the optimizer rather than the naive SGD algorithm, because the learning speed of Adagrad algorithm can be adaptive in the training stage, resulting in its faster convergence speed. At the same time, dropout techniques are used in pooling operations to deal with over-fitting of models [30].

3.3.3 DeepFM Algorithm

DeepFM is a neural network based on factorization machine. It is composed of embedding layer, factorization machine layer, multi-layer perceptron layer, full connection layer and prediction layer. Its main function is to learn feature interactions and nonlinear representations between sparse data in the recommendation scenario. The DeepFM model consists of FM components and deep neural networks (DNN) components that share the same input. $y_{FM}(x)$ and $y_{DNN}(x)$ are the output of FM component and DNN component respectively. The FM component is an FM layer stacked on the embedding layer and input layer, which mainly models data bias, feature weight and second-order feature interaction. The DNN component is a multi-layer perceptron layer and full connection layer stacked

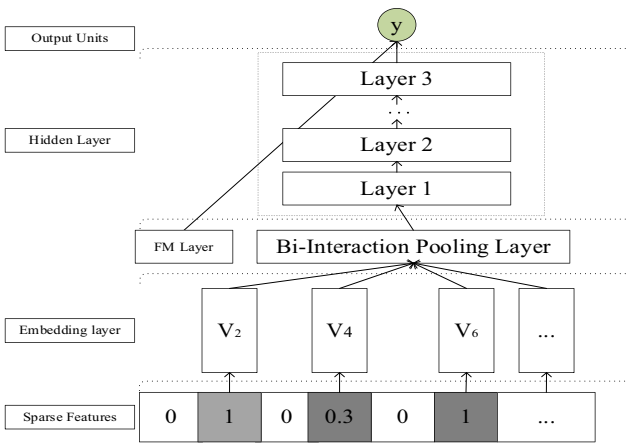

Fig. 4 Network topology of DeepFM Algorithm

on the embedded layer, which mainly models high-order feature interaction. Its network topology is shown in Fig. 4.

The expression of the DeepFM model is shown in Eq. (17):

$$y_{DeepFM}(x) = f\left(y_{FM}(x) + y_{DNN}(x)\right) \tag{17}$$

where f is a function used in the prediction layer. The sigmoid function is used in the dichotomization task and the linear function in the regression task.$x = \left(x_1, ..., x_n\right) \in R^n$ is the input vector. For the categorical features in the original data, it is directly encoded by one-hot, but if it is a continuous real-valued feature, it can be represented by discretization or directly by itself.

The expression of the FM component is given in Eq. (18).

$$y_{FM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle v_i \otimes v_j \right\rangle x_i x_j \tag{18}$$

vi represents the i-th embedding vector, which is the parameter learned through the embedding layer, and k is the factor dimension of the embedding vector. w_i reflects the importance of first-order features. See factorization machine layer in subsection 3.2.5 for details.

The expression for the DNN component is shown in Eq. (19).

$$\begin{aligned}
&a^{(0)} = \left[v_0 x_0, v_1 x_1, ..., v_i x_i, ..., v_n x_n\right] \left(x_0 \neq 0, x_i \neq 0, x_n \neq 0\right) \\
&a^{(1)} = \sigma(W^{(0)} a^{(0)} + b^{(0)}) \\
&... \\
&a^{(L-1)} = \sigma(W^{(L-2)} a^{(L-2)} + b^{(L-2)}) \\
&y_{MLP}(x) = \sigma(W^{(L-1)} a^{(L-1)} + b^{(L-1)}) \\
&y_{DNN}(x) = w y_{MLP}(x) + b
\end{aligned} \tag{19}$$

L represents the depth of multiple perceptron layers and σ is the activation function. Except for layer 0, and $a^{(l)}, W^{(l)}, b^{(l)}$ respectively represent the output, model weight and bias of the l -layer.$a^{(0)} \in R^{1 \times (m \times k)}$ is a row vector concatenated from the set of embedded vectors. The embedded vector set does not include items with a value of 0. $y_{MLP}(x)$ represents the output of the multilayer perceptron layer. Here, only the binary classification task using log likelihood loss function is described, as shown in Eq. (20).

$$loss = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i \log\left(y_{DeepFM}(x_i)\right) + \left(1 - y_i\right) \log\left(1 - y_{DeepFM}(x_i)\right)\right) \tag{20}$$

The random gradient descent is used for optimization, as shown in Eq. (21).

$$\theta \leftarrow \theta + \eta \times \left(\frac{y_i}{y_{DeepFM}(x_i)} - \frac{1 - y_i}{1 - y_{DeepFM}(x_i)} \right) \frac{dy_{DeepFM}(x_i)}{d\theta} \tag{21}$$

In Eq. (20), x_i and y_i respectively represent the values of the characteristics and target variables of the i-th sample in the training set,N indicate the number of samples of the training set. In Eq. (21), θ is the trainable model parameter, and η is the learning rate. The derivatives of other parameters in the model can be calculated automatically using the interface in the framework layer directly. In terms of preventing over-fitting, dropout technology is used in DNN and component $L2$ regularization in embedded layer. Use the early stopping strategy to choose the optimal number of iterations during the learning process [31].

3.3.4 DCN Algorithm

DCN is a deep crossover network. It consists of embedding layer, cross-network layer, multi-layer perceptron layer, full connection layer and prediction layer. Its main purpose is to learn the abstract representation of features and the feature interaction within the specified order. Its network topology is shown in Fig. 5.

The expression of the DCN model is the formula as Eq. (22).

$$\begin{aligned}
&x = [x_{dense}, x_{sparse}] \\
&x_1 = y_{embeeded}(x) \\
&x_{out} = \left[y_{CNL}(x_1), y_{MLP}(x_1)\right] \\
&y_{DCN}(x) = y_{predict}\left(y_{FCL}(x_{out})\right)
\end{aligned} \tag{22}$$

where $x_{dense} \in R^a$ represents the real-valued feature column vector, $x_{sparse} \in R^b$ represents the category feature column vector, $x \in R^{a+b}$ is the column vector spliced by

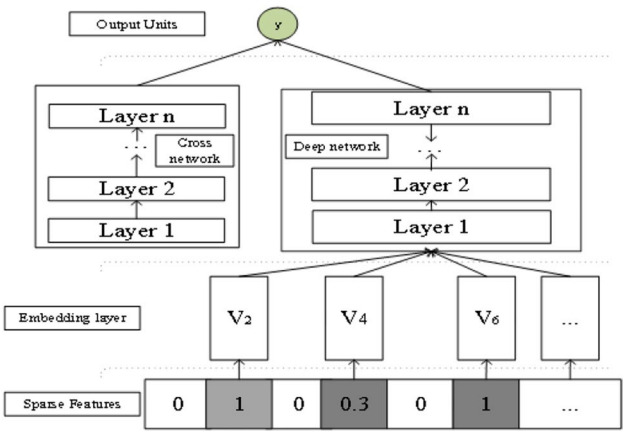

Fig. 5 Network topology of DCN Algorithm

Table 1 Default values of AFM, DFM, DeepFM and DCN algorithms

Algorithm name	Learning rate	Number of iterations	Batch size	Embedded vector size	Attention factor	Drop rate	Number of hidden units in multi-layer perceptron layer	Cross layer
AFM	0.1	200	256	8	8	0	–	–
NFM	0.1	200	256	256	–	0.3	(128,128)	–
DeepFM	0.1	200	256	8	–	0	(258,256)	–
DCN	0.1	200	256	8	–	–	(128,128,64)	3

the real-valued feature column vector and category feature column vector, $y_{embededd}, y_{CNL}, y_{MLP}, y_{FCL}, y_{predict}$ represents embedded layer component, cross-network layer component, multi-layer perceptron layer component, full connection layer component and prediction layer component respectively. x_1 is the output of embedded layer components, and x_{out} is column vectors joined together by the output of cross-network layer component and multi-layer perception layer component.

Here, only the binary classification task using the log likelihood loss function is described, as shown in Eq. (23). The random gradient descent algorithm is used to optimize the parameter value, as shown in Eq. (24).

$$loss = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\left(y_{DCN}(x_i)\right) + \left(1-y_i\right)\log\left(1-y_{DCN}(x_i)\right)\right) + \lambda\sum_{l}||W_l||^2 \tag{23}$$

$$\theta \leftarrow \theta + \eta \times \left(\frac{y_i}{y_{DCN}(x_i)} - \frac{1-y_i}{1-y_{DCN}(x_i)}\right)\frac{dy_{DCN}(x_i)}{d\theta} \tag{24}$$

In Eq. (23) and Eq. (24), x_i and y_i respectively represent the values of the characteristics and target variables of the i-th sample in the training set, N indicate the number of samples of the training set, λ is the regularization parameter of $L2$, W_l is the weight matrix of the l-th layer in the DCN model, and y_{DCN} is the output of Eq. (22), θ is a trainable model parameter, η is used to control the learning rate of the gradient descent method.

4 Experiments

DeepRS is developed in Windows 10, with VScode as the integrated development tool, Tensorflow as the framework level, Python as the development language and PyPI as the third-party warehouse. AFM, NFM, DeepFM and DCN algorithms are evaluated using the public dataset MovieLens [32]. This movie rating data set is widely used to evaluate collaborative filtering algorithms. It contains 20,000,263 ratings and 465,564 tag applications across 27,278 movies.

The dataset MovieLens is randomly divided into two parts: 80% for training and 20% for validation. The training set is used to learn the model, and the validation set is used

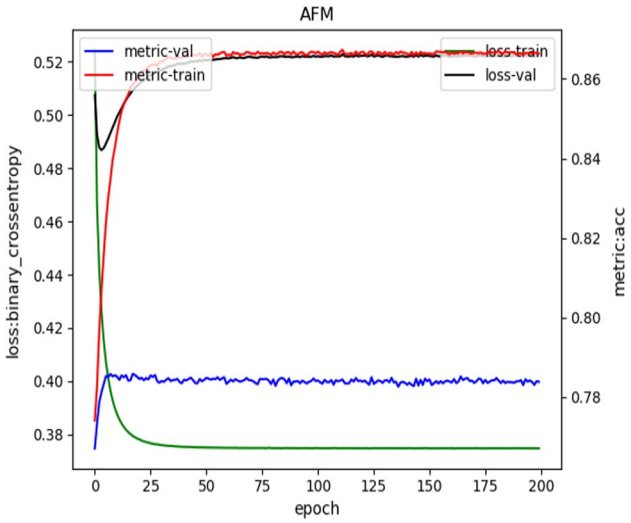

Fig. 6 AUC and Logloss of AFM algorithm

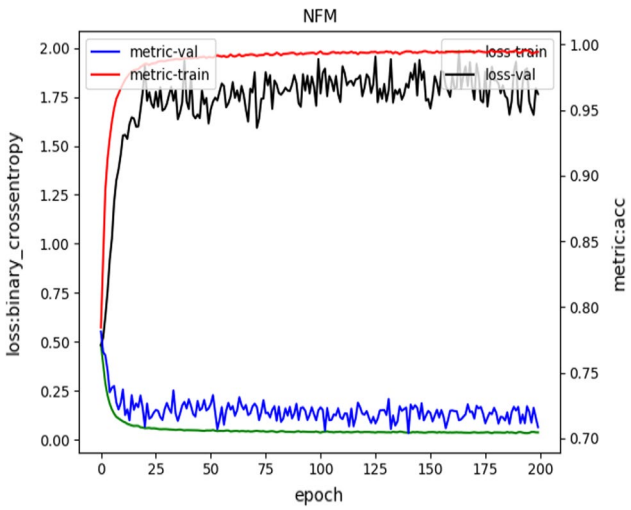

Fig. 7 AUC and Logloss of NFM algorithm

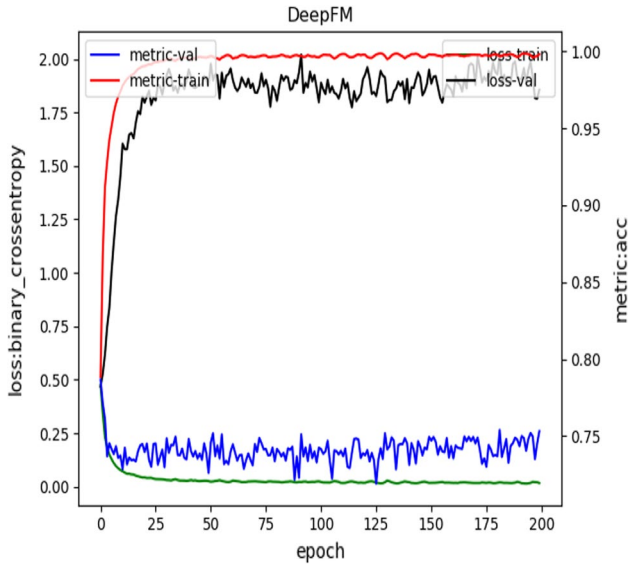

Fig. 8 AUC and Logloss of DeepFM algorithm

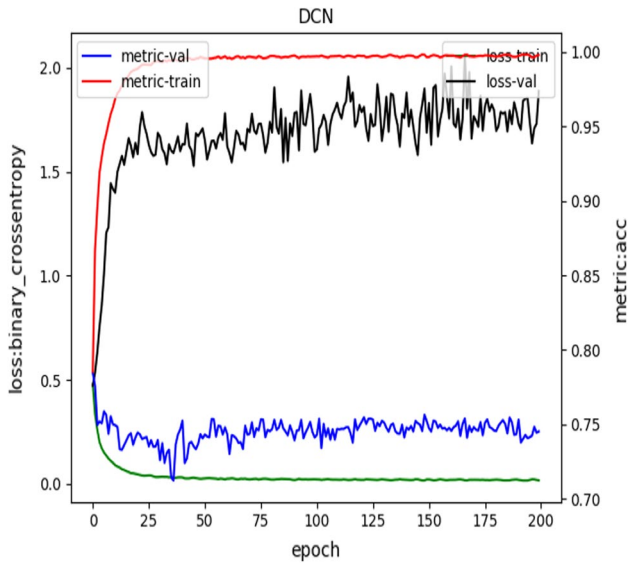

Fig. 9 AUC and Logloss of DCN algorithm

for tuning the hyper-parameters. Log loss and area under curve (AUC) are widely used for classification tasks with explicit feedback recommendations. Therefore, AUC is used as the evaluation index, and Log loss is used as the loss function. Table 1 gives the main parameter settings of each algorithm. The learning process of AFM, NFM, DeepFM, and DCN models is described in Fig. 6, Fig. 7, Fig. 8 and Fig. 9. Among them, the left ordinate axis is the cross entropy, the right ordinate axis is the accuracy rate, and the horizontal axis represents the number of iterations of the model. Metric-train and metric-val separately represent the accuracy of the model in the training set and validation set, and loss train and loss val are the cross entropy of the model in the training set and validation set respectively.

Figure 6 shows that with the increase of the number of iterations of AFM, the loss value of the training set gradually decreases to 0.36 and reaches a stable level, and the accuracy of the validation set gradually increases to around 0.785 and reaches a plateau, indicating that this model is neither over-fitting nor under-fitting. Figure 7 shows that the loss value of the training set gradually decreases with the increase of the number of iterations of NFM, and the accuracy of the validation set gradually decreases, indicating that the model is in overfitting. Figure 8 shows that with the increase of the number of iterations of DeepFM, the accuracy of the training set gradually increases to 1 and reaches a stable level, but the accuracy of the validation set gradually decreases to around 0.75, indicating that DeepFM is in overfitting. Figure 9 shows that as the number of iterations of DCN increases, the cross entropy of the training set gradually decreases to 0 and reaches a stable level, but the cross entropy of the validation set gradually increases to around 0.95, indicating that the model is in overfitting.

The average value of oscillation in a certain range after the training convergence of different algorithms in MovieLens data set is taken as the evaluation index. It can be seen from Table 2 that AFM algorithm has stronger performance ability and interpretability. The other three algorithms (NFM, DeepFM and DCN) all have overfitting problems, but in different practical applications, they can be solved by adjusting parameters, reducing the capacity of the model or adding more samples. In practical algorithm application, researchers or practitioners can directly call the encapsulated recommendation algorithm library based on deep learning to compare its prediction accuracy and loss value, so as to conveniently select the optimal recommendation algorithm, directly solve the core algorithm problems of recommendation system development and other applications.

Table 2 Performance comparison

Algorithm model	Average value of training results		Average value of validation results	
	AUC	Logloss	AUC	Logloss
AFM	0.868	0.378	0.785	0.520
NFM	0.990	0.100	0.725	1.750
DeepFM	0.998	0.009	0.740	1.825
DCN	0.999	0.005	0.750	1.170

Table 3 Comparing DeepRS to existing library for recommender systems

Library	Programming language	Backend	Implemented recommendation algorithms	Abstract level modularity
fastFM	C	×	Traditional algorithms	×
Mymedialite	C#	×	Traditional algorithms	×
Librec	Java	×	Traditional algorithms	×
PREA	Java	×	Traditional algorithms	×
Recommenderlab	R	×	Traditional algorithms	×
Surprise	Python	SciKits	Traditional algorithms	×
OpenRec	Python	Tensorflow	Algorithms based on deep learning	Multi-LayerPerceptron (MLP), Latent Factor (LF)
DeepRS	Python	Tensorflow	Algorithms based on deep learning	EL, FCL, MLPL, PL, FML, ANL, CL, CNL

5 Comparative Study

Existing recommendation algorithm libraries are limited in two aspects: lack of abstract level modularity support and lack of reliable backend support [10]. Furthermore, different recommendation algorithm libraries adopt different programming languages and implement different recommendation algorithms. Therefore, in the rest of this section, we compare DeepRS with Mymedialite [8], Librec [9], PREA [25], fastFM [26], Surprise [27], OpenRec [10], Recommenderlab [28] from four aspects: programming language, backend, implemented recommendation algorithm and abstraction level modularity. The comparison results are shown in the Table 3. It can be found from Table 3 that the programming languages used in the recommended algorithm libraries include C, C#, R, Java and Python, and only Surprise, OpenRec and DeepRS have reliable backend support. Most of the libraries are designed for traditional recommendation algorithms. OpenRec and DeepRS are designed for recommendation algorithms based on deep learning. However the abstract level modularity of OpenRec only contains two modules: MLP and LF, while DeepRS includes eight modules: EL, FCL, MLP, PL, FML, ANL, CL, and CNL. Therefore, DeepRS has better scalability than OpenRec.

6 Conclusion and Future Work

Nowadays, deep learning technology is widely used in the field of recommendation. The emergence of more and more recommendation algorithm models provides great convenience for the application and research of recommendation system. According to the specific nature of the recommendation data, selecting the appropriate recommendation algorithm can not only save time, but also improve the recommendation quality. Aiming at the sparsity and diversity of recommendation data, a deep learning-based recommendation algorithm library DeepRS is designed as a tool for recommendation task modeling, which is convenient for researchers and practitioners. Experiments also verify the ease of use, feasibility and accuracy of the algorithm package. Comparative study shows that DeepRS has more reusable components and is more scalable than existing recommendation algorithm packages. However, there is still a distance from the real scene, DeepRS only contains four recommendation algorithms based on deep learning. In future, DeepRS will be further improved through detailed API documents to make it more operable. At the same time, we are constantly adding new recommendation algorithms based on deep learning to DeepRS.

Acknowledgements The authors express great thanks to the financial support from the department of science and technology of Henan Province and the Zhengzhou University of Light Industry.

Author contributions All the authors equally contributed to this work.

Funding This work was financially supported by the Doctoral Research Fund of Zhengzhou University of Light Industry (2016BSJJ037); and the Science and Technology Project of Henan Province (202102210351,212102210076).

Data availability The dataset used in this research is openly accessible via: https://movielens.org.

Declarations

Conflict of interests The authors declare that they have no conflicts of interest.

References

1. Bobadilla, J., Ortega, F., Hernando, A., Gutirrez, A.: Recommender systems survey. Knowl. Syst. **46**(2013), 109–132 (2013)
2. Alhijawi, B., Kilani, Y.: The recommender system: a survey. Int. J. Adv. Intell. Parad. **15**(3), 229–251 (2020)
3. Yera, R., Martinez, L.: Fuzzy tools in recommender systems: A survey. Int. J. Comp. Intell. Syst. **10**(2017), 776–803 (2017)
4. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
5. Zhang, S.A., Yao, L.N., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Comp, Surv. **52**(1), 1–38 (2019)
6. Batmaz, Z., Yurekli, A., Bilge, A., Kaleli, C.: A review on deep learning for recommender systems: challenges and remedies. Art. Intell. Rev. **52**, 1–37 (2019)
7. Dau, A., Salim, N.: Recommendation system based on deep learning methods: a systematic review and new directions. Art. Intell. Rev. **53**, 2709–2748 (2020)
8. Gantner, Z., Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: MyMediaLite: A free recommender system library. In: Proceedings of the fifth ACM conference on Recommender systems. ACM, pp. 305–308 (2011)
9. Guo, G.B., Zhang, J., Sun, Z., Yorke-Smith, N.: LibRec: A Java library for recommender systems. In: Proceedings of the 23rd Conference on User Modelling, Adaptation and Personalization. Spinger, pp. 1–4 (2015)
10. Yang, L.Q., Bagdasaryan, E., Gruenstein, J., Hsieh, C.-K., Estrin, D.: Openrec: A modular framework for extensible and adaptable recommendation algorithms. IN: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. ACM, pp. 664–672 (2018)
11. Xiao, J., Ye, H., He, X.N., Zhang, H.W., Wu, F., Chua, T.-S.: Attentional factorization machines: learning the weight of feature interactions via attention networks. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. AAAI, pp. 3119–3125 (2017)
12. He, X.N., Chua, T.-S.: Neural factorization machines for sparse predictive analytics. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, pp. 40–48 (2017)
13. Guo, H.F., Tang, R.M., Ye, Y.M., Li, Z.G., He, X.Q.: DeepFM: A factorization-machine based neural network for CTR prediction. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. AAAI, pp. 2782–2788 (2017)
14. Wang, R.X., Fu, B., Fu, G., Wang, M.L.: Deep & cross network for Ad click predictions. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1–7 (2017)
15. Zhou, W.Z., Cao, D., Xu, Y.F., Liu, B.: A survey of recommendation systems. J. Hebei Univ. Sci. Techn. **41**(1), 76–87 (2020)
16. Su, X.Y., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Adv. Art. Int. **2009**, 1–19 (2009)
17. Zhao, J.Y., Zhuang, F.Z., Ao, X., et al.: Survey of collaborative filtering recommender systems. J. Cyb. Secur. **6**(5), 17–34 (2021)
18. Khojamli, H., Razmara, J.: Survey of similarity functions on neighborhood-based collaborative filtering. Expert Syst. Appl. **185**, 115482 (2021)
19. Pereira, N., Varma, S.: Survey on content based recommendation system. Int. J. Comp. Sci. Inf. Techn. **7**(1), 281–284 (2016)
20. Perez-Almaguer, Y., Yera, R., Alzahrani, A.A., Martnez, L.: Content-based group recom- mender systems: A general taxonomy and further improvements. Exp. Syst. With Appl. **184**, 115444 (2021)
21. Shu, J.B., Shen, X.X., Liu, H., Yi, B.L., Zhang, Z.L.: A content-based recommendation algorithm for learning resources. Mult. Syst. **24**(1), 163–173 (2018)
22. Qian, Y.F., Zhang, Y., Ma, X., Yu, H., Peng, L.M.: EARS: Emotionaware recommender system based on hybrid information fusion. Inf. Fus. **46**, 141–146 (2019)
23. Cano, E., Morisio, M.: Hybrid recommender systems: a systematic literature review. Int. Data An. **21**(6), 1487–1524 (2017)
24. Khan, Z.Y., Niu, Z.D., Sandiwarno, S., Prince, R.: Deep learning techniques for rating predi- ction: a survey of the state-of-the-art. Art. Int. Rev. **54**(1), 95–135 (2021)
25. Lee, J., Sun, M.X., Lebanon, G.: PREA: personalized recommendation algorithms toolkit. J. Mach. Learn. Res. **13**(1), 2699–2703 (2012)
26. Bayer, I.: fastFM: A library for factorization machines. J. Mach. Learn. Res. **17**(1), 6393–6397 (2016)
27. Hug, N.: Surprise: a python library for recommender systems. J. Op. So. Softw. **5**(52), 2174 (2020)
28. Hahsler, M.: recommenderlab: A framework for developing and testing recommendation algorithms. R package version 0.2–6, Tech. Rep. 1–40 (2015). Available: https://git-hub.com/mhahsler/recommenderlab
29. Zhou, Z.H.: Machine learing. Tsinghua University Press, Beijing (2016)
30. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
31. Wang, X., He, X.N., Cao, Y.X., Liu, M., Chua, T.-S.: KGAT: knowledge graph attention network for recommendation. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, pp. 950–958 (2019)
32. Harper, F.M., Konstan, J.A.: The movielens datasets: history and context. ACM Trans. Int. Int. Syst. **5**(4), 1–19 (2015)