Neural Computing and Applications (2021) 33:14167–14177
https://doi.org/10.1007/s00521-021-06065-9

ORIGINAL ARTICLE

Deep autoencoders for feature learning with embeddings for recommendations: a novel recommender system solution

Kiran Rama[1] · Pradeep Kumar[1] · Bharat Bhasker[1,2]

Received: 12 July 2020 / Accepted: 19 April 2021 / Published online: 6 June 2021

Abstract
We propose "Deep Autoencoders for Feature Learning in Recommender Systems," a novel discriminative model based on the incorporation of features from autoencoders in combination with embeddings into a deep neural network to predict ratings in recommender systems. The work has two major motivations. The first is to engineer features for recommender systems in a domain-agnostic way using autoencoders. The second is to develop a method that sets a benchmark for predictive accuracy. In our proposed solution, we build a user autoencoder and item autoencoder that extract latent features for the users and items, respectively. The additional features engineered are the latent features for the users and items, and these come from the bottleneck activations of the autoencoder. Our method of feature engineering is domain agnostic, as the inner-most activations would differ for domains without any additional effort required on part of the modeler. Next, we then use the activations of the inner-most layers of the autoencoders as features in a subsequent deep neural network to predict the ratings along-with user and item embeddings. Our method incorporates additional linear and nonlinear latent features from the autoencoders to improve predictive accuracy. This is different from the existing approaches that use autoencoders as full-fledged recommender systems or use autoencoders to generate features for a subsequent supervised learning algorithm or without using embeddings. We demonstrate the out performance of our solution on four different datasets of varying sizes and sparsity, namely MovieLens 100 K, MovieLens 1 M, FilmTrust and BookCrossing datasets, with strong experimental results. We have compared our DAFERec method against mDA-CF, TrustSVD, SVD variants, BiasedMF, ItemKNN and I-AutoRec methods. The results demonstrate that our proposed solution beats the benchmarks and is a highly flexible model that works on different datasets solving different business problems like book recommendations, movie recommendations and trust.

Keywords Autoencoders · Recommender systems · Feature engineering · BookCrossing · User autoencoder · Item autoencoder

1 Introduction

In the world of rapid advancements in technology and Internet adoption, an increasing number of devices and exponentially growing data, recommender systems are becoming increasingly important to enable users to tackle the increasing information overload in order to make the right choices. Recommender Systems form a very useful mechanism to improve the quality and efficacy of decisions by users. Increased number of products and services, increased competition for the customer's wallet share and mind share, focus on personalized products, services and communications to the customers have been the focus of almost all companies. There are various definitions of

✉ Kiran Rama
efpm04013@iiml.ac.in

Pradeep Kumar
pradeepkumar@iiml.ac.in

Bharat Bhasker
bhasker@iimraipur.ac.in

[1] Department of Information Technology and Systems, Indian Institute of Management Lucknow, IIM Road, Prabandh Nagar, Mubarakpur, Lucknow, Uttar Pradesh 226013, India

[2] Indian Institute of Management Raipur, Naya Raipur, India

recommender systems (RS) in the literature and we will not attempt to go through them but make a broad statement that they provide a rating to a user-item pair. A popular categorization of recommender system techniques is into the Content-based, Collaborative Filtering based and Hybrid techniques. The differences between these methods have been covered in literature in detail [1], and we will not go into the semantics in this paper. Deep learning methods have seen increasing adoption due to the advancements in Graphical processing unit hardware that speeds up neural network training, advancements in software algorithms and the availability of automatic differentiation enabling software like PyTorch [2], MXNet [3] and TensorFlow [4]. Deep learning methods have seen huge success in structured dataset RSes like the DNNRec [5] method.

We now describe the motivation of the study. Hybrid recommender systems that combine collaborative filtering (that only looks at user-item interactions) and content-based methods (that look at user and item features) are popular as they address the cold-start problem. However, they require feature engineering that is domain-specific. Manual Feature Engineering requires domain expertise and is not scalable across domains and requires maintenance over time. The criticism of CF methods is that while they capture latent user and item features that are linear in nature, they fail to capture subtle factors. In light of the above, the first motivation for this study is to engineer subtle nonlinear features automatically in a domain-agnostic way. A novel deep learning method called autoencoders is a self -supervised method that creates a nonlinear high-dimensional representation in the innermost layer and have multiple hidden layers of lower dimensions in between. In other words, they attempt to learn an identity function with a compressed set of the input layer in between and the network is trained using gradient descent like in other deep neural network methods like in DNNRec, the loss here being the squared loss. The innermost layer of the autoencoder, also referred to as the bottleneck layer, has been used to engineer features as it represents a lower-dimensional representation of the input features [6]. The second motivation of this study is to leverage the features learned for the users and items by autoencoders in a deep learning solution for rating prediction and thereby address the gap in CF methods that do not learn nonlinear latent factors.

We describe a novel solution called DAFERec (Deep Autoencoders for Feature Learning for Recommendations) that uses the inner-most layer activations of a deep autoencoder as features in a Deep Neural Network for recommendations. Our key contributions are as follows. First is that we use the activations of the inner-most bottleneck layer of the autoencoder as features in a deep neural network (DNN) based recommender system in combination in combination with embeddings. Second is that we demonstrate that this combination enables learning of sophisticated features that have nonlinearity from the user autoencoder and the item autoencoder, in addition to the user and item embeddings. We explain that the outperformance is achieved as our method can learn nonlinear latent representations of features from the autoencoder in the DNN. Third contribution is that this approach of incorporating nonlinear latent features leads to a powerful feature map that beats the comparative techniques results in CF. We demonstrate the outperformance of DAFERec on multiple datasets like BookCrossing, MovieLens 100 k, MovieLens 1 M, and FilmTrust datasets. Our model beats the previous best results reported for predictive accuracy on these datasets in literature. The scope of our study is for the RS case where the goal is to predict numerical ratings and not for the case where the output is a polarity score or for the case where the output is a ranking of items for a user.

We organize the paper into five sections. Section 1 introduces the study and explains the motivation for this study from a business and technical perspective and also highlights the key contributions of the work. Section 2 looks at work related to the solution proposed in this paper and also looks at the comparative methods in this field. The third section describes our DAFERec solution and algorithmic aspects with a mathematical foundation. Section 4 contains the details of the experimental setup and evaluation metrics. The fourth section has a results discussion and. The final section highlights the opportunities for further research.

2 Past work

We look at related work in literature to our solution and describe how our work is different from or extends each of these works, in this section.

CF methods have been hugely successful in recommender systems with matrix factorization (MF) approach heavily employed to find latent factors of users and items. Several methods have been used for MF, including Singular Value Decomposition (SVD), variants of SVD like SVD++ and other methods like Non-Negative Matrix Factorization (NMF) [7], Bayesian form of Probabilistic Matrix Factorization (BPMF) [8], etc. Latent Factor methods specifically combine user and item latent factors into the same latent factor space, making them directly comparable and explain ratings by characterizing them on these factors. Singular Vector decomposition (SVD) captures linear relationships between users and items in a compressed form [9]. Addition of bias term to SVD, that is independent of the interactions results in SVD++ .

SVD++ is a very powerful method and has one of the best RMSEs on the ML-1M dataset. TrustSVD extends SVD++ with trust information [10] using both the explicit and implicit influence of user-item ratings to generate predictions. Deep Learning methods have been successfully applied DNNRec [5], RBMs [11], NCFs [12], DCNs [13], etc. It was shown that wide linear models [14] a combination of wide and deep networks allows the memorization of sparse feature interactions and generalization as well. CNN and RNNs have been used to extract features about users and items to solve the cold-start problem. Examples of CNNs include image feature extraction [15], speech feature extraction [16], text feature extractions [17], etc. For textual data, RNNs have been employed to learn the language model and then use these features learned in the modeling process, with variants like LSTM, GRU, QRNN etc. [18].These methods have been used in a family of models called Collaborative Sequencing Models [19]. Audio features have been engineered using LSTM-based networks and also using CNNs [20].

Autoencoders have been employed as stand-alone recommender systems both at a user level and at an item level. [6, 21, 22] are good examples of the application of autoencoders. Autoencoders have been stacked to form powerful ensembles. An example of such a novel technique is called AutoRec [23]. The I-AutoRec method [24] has the best benchmark RMSE on the ML-1M dataset. There have been variations of stacked autoencoder approaches that add Gaussian noise to make them generalize better [25]. This helps discover more robust features versus simply learning the identity function. Deep collaborative filtering (DCF) [26] leverages side information for outperformance with the treatment of dense and categorical features into a powerful network. This method uses a user autoencoder and an item autoencoder to learn a compressed representation of the user and item features, respectively. These are used alongside the latent features learned from matrix factorization of the rating matrix. Side information integration [27] has been accomplished by extending the construction of embeddings by neural networks to a collaborative filtering setting. COFILS, Collaborative Filtering to Supervised Learning [28] uses the values of the compressed representation of users and items resulting from CF into a subsequent model as features, to further achieve outperformance. A-COFILS [29] extended COFILS to use autoencoders to generate user and item features but in a way that is similar to the DCF technique that was explained earlier. Combining matrix factorization methods and autoencoders [30] has been used as a way to learn effective latent representations using deep learning. It models the mappings between the latent factors used in CF and the latent layers in deep models. This has the best-reported benchmarks on the BookCrossing and ML 100 K datasets.

The existing approaches have several drawbacks. Stand-alone autoencoder based recommender systems suffer from poor predictive accuracy. That is the reason they have not seen adoption to the levels of collaborative filtering methods. Deep collaborative filtering is able to generate high predictive accuracy but the features engineered from the side information tend to be domain-specific. COFILS approach does not make use of embeddings and therefore misses out on rich representations that can be learned by these embeddings. Our approach has several differences to prior approaches like [23–28]. Instead of using AutoRec for rating prediction, we take the activations of the bottleneck layers for the user autoencoder and the item autoencoder and use them as features in a subsequent deep neural network along-with embedding. Our method differs from the DCF approach as we do not make use of any side information and the user and item autoencoders in our model are based on the user and item identifiers only. That said and done, our approach is scalable enough to take advantage of side information and is domain agnostic in the sense that no domain-specific information are engineered using the side information. The semantic similarity of COFILS with our approach is that our method uses latent features that are learned by the autoencoders as features in a particular instance of supervised learning, namely a deep neural network. The differences with COFILS are that our approach is based on an autoencoder that can learn non-linear latent features, and we also make use of embeddings. Further we use a DNN which was not used in COFILS. Further, these features were combined using a random forest technique in A-COFILS and not using a DNN, like in the proposed DAFERec solution.

As we see, autoencoders have been employed as stand-alone recommender systems, and the activations from the inner-most layer have been used as features in non-recommender system applications. Based on our research, we believe this is the first time that the activations from the innermost layer of the user and item autoencoders have been used as features in a DNN solution in combination with embeddings for recommender systems to achieve outperformance. While autoencoders have been employed in recommender systems in a stand-alone way, there is no study to the best of our knowledge that uses the features learned by the autoencoders in a recommender system.

3 Proposed solution DAFERec (deep neural network with autoencoder features and embeddings for RS)

We present a framework for rating prediction by a recommender system. The solution has two major parts, namely extraction of features using an autoencoder and prediction in a DNN incorporating these features. The extraction of features is achieved by training a user autoencoder and an item autoencoder for recommendations. Prediction involves combining the activations of the bottleneck layer of the user autoencoder and item autoencoder along-with the embeddings for the user and item identifiers into a deep neural network. Table 1 has the symbol list and their corresponding meanings.

3.1 Feature extraction using item autoencoder and user autoencoder

The first step in our solution is to build an autoencoder to predict the ratings. The first step in the solution is to create a rating matrix from the list of $\langle \text{user}, \text{item}, \text{rating} \rangle$ tuples. The pivot operation is standard in several programming languages, and we will not go into the intricacies of the data preparation. The explanations below are for the item autoencoder. The same holds true for the user autoencoder except that the rating matrix will be the user-item matrix instead of the item-user matrix that is used for the item autoencoder. As in Table 1, $R \in \mathbb{R}^{m*n}$ is an observed user-item rating matrix, $U = \{U_1, U_2, U_3.....U_m\}$ is a list of users, $I = \{I_1, I_2, I_3.....I_n\}$ is a list of items, $r_U^{(u)} = (R_{u1}, R_{u2}, ...R_{un}) \in \mathbb{R}^n$ and $r_I^{(i)} = (R_{i1}, R_{i2}, ...R_{im}) \in \mathbb{R}^m$ refer to the individual rows of the user-item rating matrix and the item-rating matrix respectively. Given that we refer to R as the rating matrix, the simple transpose operation of R, denoted by, R^T is the item-rating matrix. Clearly, the user autoencoder has the user identifiers as inputs and the item identifiers as features, while the item autoencoder has the item identifiers as inputs and the user identifiers as features. The inputs to the user autoencoder and the item autoencoder are denoted by E_U and E_I respectively and shown in Eqs. (1) and (2) below.

$$E_U = \begin{pmatrix} r_U^{(1)} \\ r^{(2)}U \\ \dots \\ r_U^{(m)} \end{pmatrix} \text{is the input to the user autoencoder} \quad (1)$$

And

$$E_I = \begin{pmatrix} r_I^{(1)} \\ r_I^{(2)} \\ \dots \\ r_I^{(m)} \end{pmatrix} \text{is the input to the item autoencoder} \quad (2)$$

Table 1 Notations and meanings

Symbol	Meaning
U, I;$r_{i,j,t}$	Number of users and items; rating to an item j by user i at time t
$R \in \mathbb{R}^{m*n}$	User-item rating matrix
$r_u^{(u)}$	Row of ratings to items by user u
R_{uy}	Rating to item y by user u
$r_i^{(i)}$	Row of ratings given by users to item i
R_{ix}	Rating to item x by a user i
W_H, b_H	Weight matrix and bias of the affine portion of the layer
h_i	Hidden nodes in the i th layer
$\text{Re}LU(x)$	Keeps the positive values as-is and transforms the non-0 values to 0
p	Dropout probability
$\text{concat}(v_1, v_2)$	Combine vectors v_1 and v_2 into a single vector
n_batch	Number of batches
$\sigma(x)$	$\sigma(x) = \frac{1}{1+e^{-x}}$
x_t	One-hot vector for the levels in the categorical variable
$V * x_t$	Embedding for the categorical variable
K_U	User factors
K_I	Item factors
n_emb	Number of embeddings
$f(r;\theta)$	Reconstruction of input $r \in \mathbb{R}^d$

The item-based autoencoder takes as input the values of $r_U^{(i)}$ for each user and attempts to recreate the values at the output, after passing these values through a low-dimensional latent space. The item autoencoder architecture is shown in Fig. 1.

The user autoencoder architecture is very similar to only input differences, as explained earlier and is shown in Fig. 2. The autoencoder learns the identify function through various order feature interactions that are combined in its innermost layers. This can be formally expressed as Eq. (3), where $f(r;\theta)$ is the reconstruction of the input $r \in \mathbb{R}^d$

$$\min \sum_{r \in S} ||(r - f(r,\theta))||^2 \tag{3}$$

The equations for the user autoencoder are given by Eqs. (4) to (6). As we see, the encoder E follows an affine transformation R_U with a nonlinear sigmoid transformation in (4), which is followed by a lower-dimensional affine and nonlinear sigmoid transformation in (5). The decoder is the exact mirror image of the encoder in the opposite direction, meaning the number of hidden nodes increases, as we see in Fig. 2. The parameters θ are learned through back-propagation. To ensure efficient autoencoder training, a batch normalization step is added after each hidden layer. We have not explicitly shown the regularization term in the autoencoder equation, as the same is achieved by a combination of the weight decay parameter in the Adam Optimizer and dropouts in the different layers of the encoder and the decoder as shown in Fig. 2. The weight decay and the dropouts are used to regularize the weights so that the approach generalizes.

$$E = \sigma\left(W_2.\left(\sigma\left(W_1.R^I + b_1\right)\right) + b_2\right) \tag{4}$$

$$D = \sigma(W_3.E + b_3) \tag{5}$$

$$f(r;\theta) = W_4.D + b_4 \tag{6}$$

For the item autoencoder, the equations are given in (7) to (9), and they are similar to the equations in (4) to (6).

$$E = \sigma\left(W_2.\left(\sigma\left(W_1.R^U + b_1\right)\right) + b_2\right) \tag{7}$$

$$D = \sigma(W_3.E + b_3) \tag{8}$$

$$f(r;\theta) = W_4.D + b_4 \tag{9}$$

3.2 DNN integrating user and item embeddings with user and item autoencoder features

In Step 2 of the solution, the activations of the bottleneck layers of the user autoencoder and item autoencoder are used as features in a Deep Neural Network (DNN) as shown in the solution flow diagram in Fig. 4. The autoencoder features are used in combination with embeddings for the users and items, all together as inputs into a DNN. Figure 3 contains the overall architecture of Step 2 of the solution. Embeddings are an alternate representation of the one-hot encoding or dummy variable coding in traditional machine learning. They are dense lookup matrices that return a dense array for each identifier instead of a single one-hot encoded vector. They are able to learn a richer representation of the identifier in multiple dimensions. The weights of these embeddings are learned like any other features, similar to the autoencoder derived features in our solution.

Figure 4 presents the overall solution architecture.

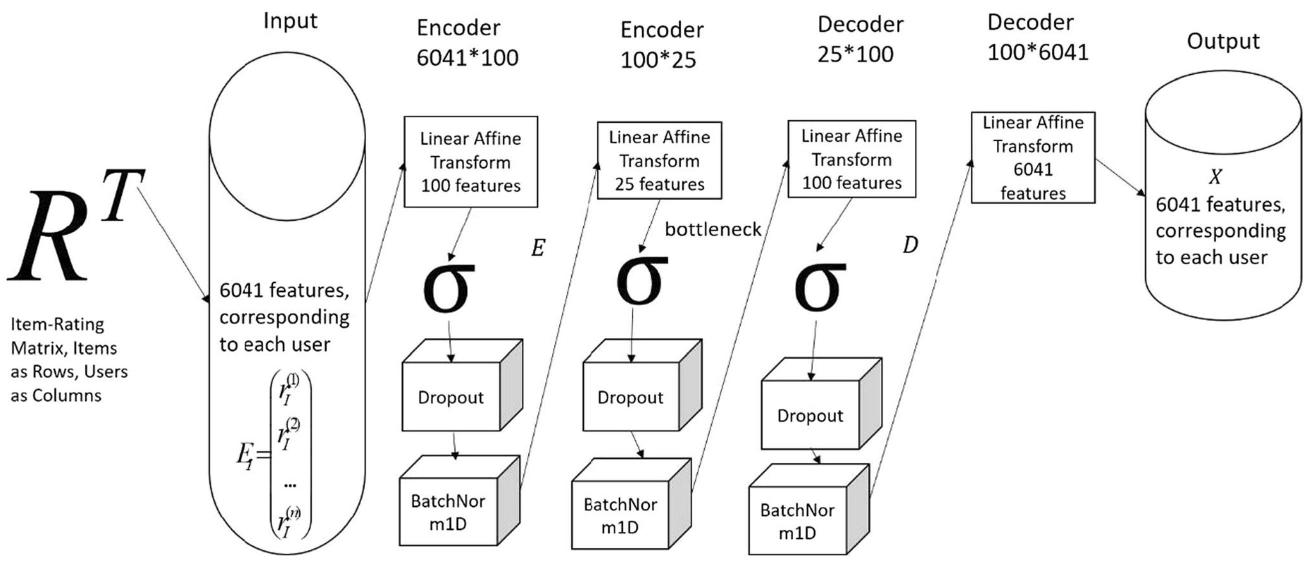
R^T
Item-Rating Matrix, Items as Rows, Users as Columns
Input
6041 features, corresponding to each user
$E_I = (r_I^{(1)}, r_I^{(2)}, \dots, r_I^{(n)})$
Encoder 6041*100
Linear Affine Transform 100 features
σ
Dropout
BatchNorm1D
E
Encoder 100*25
Linear Affine Transform 25 features
bottleneck
Decoder 25*100
Linear Affine Transform 100 features
D
Decoder 100*6041
Linear Affine Transform 6041 features
Output
X
6041 features, corresponding to each user

Fig. 1 Item autoencoder architecture—Example for ML1M dataset (Step 1)

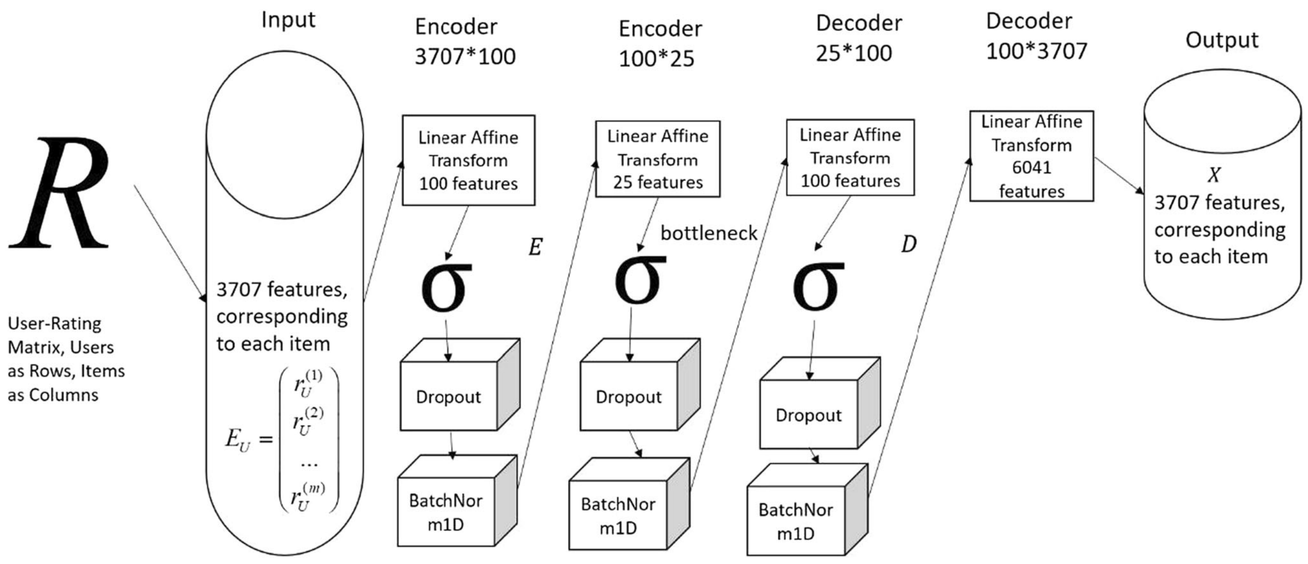
R
User-Rating Matrix, Users as Rows, Items as Columns
Input
3707 features, corresponding to each item
$E_U = \begin{pmatrix} r_U^{(1)} \\ r_U^{(2)} \\ \dots \\ r_U^{(m)} \end{pmatrix}$
Encoder 3707*100
Linear Affine Transform 100 features
σ
Dropout
BatchNorm1D
E
Encoder 100*25
Linear Affine Transform 25 features
bottleneck
Decoder 25*100
Linear Affine Transform 100 features
D
Decoder 100*3707
Linear Affine Transform 6041 features
Output
X
3707 features, corresponding to each item

Fig. 2 User autoencoder architecture for ML-1M dataset. Similar to item autoencoder

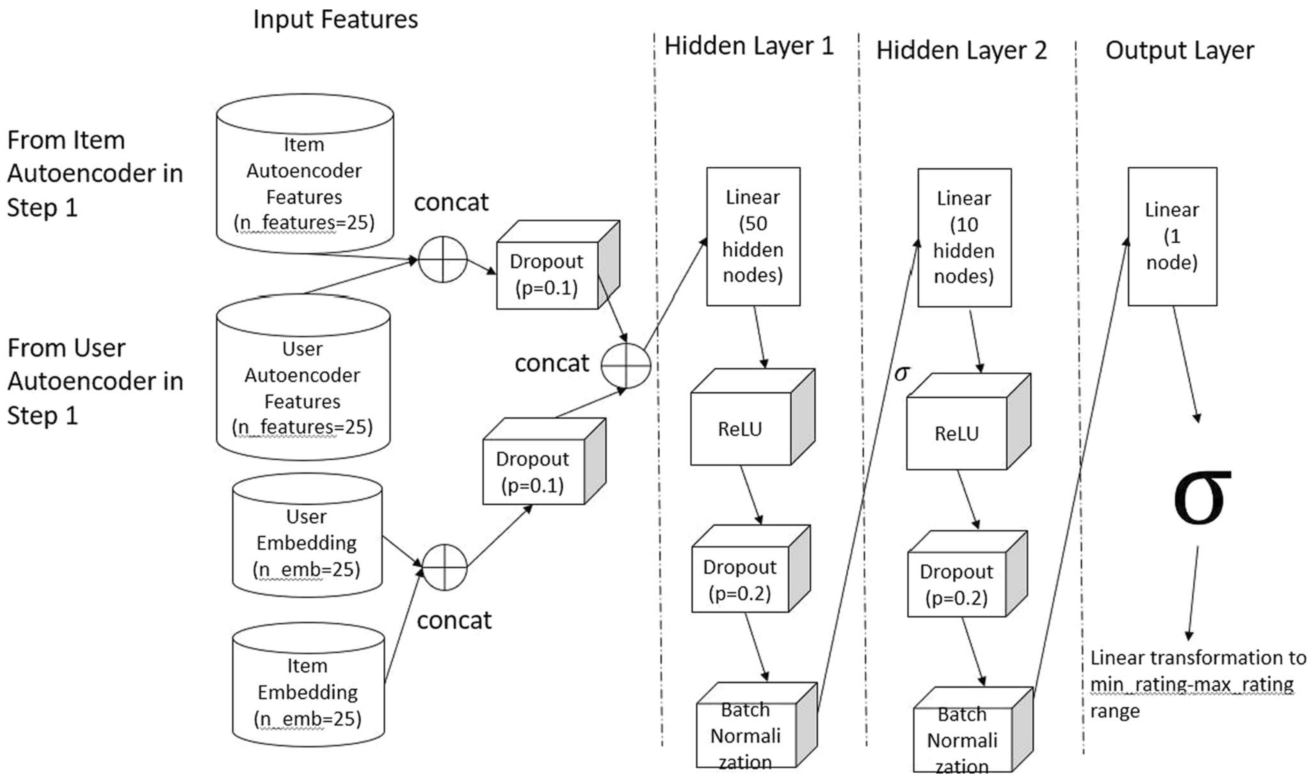
Input Features
From Item Autoencoder in Step 1
Item Autoencoder Features (n_features=25)
From User Autoencoder in Step 1
User Autoencoder Features (n_features=25)
User Embedding (n_emb=25)
Item Embedding (n_emb=25)
concat
Dropout (p=0.1)
concat
Dropout (p=0.1)
concat
Hidden Layer 1
Linear (50 hidden nodes)
ReLU
Dropout (p=0.2)
Batch Normalization
σ
Hidden Layer 2
Linear (10 hidden nodes)
ReLU
Dropout (p=0.2)
Batch Normalization
Output Layer
Linear (1 node)
σ
Linear transformation to min_rating-max_rating range

Fig. 3 DAFERec architecture: proposed solution

Our proposed solution has two hidden layers that are basically an affine followed by a *ReLU* transformation. Dropouts enable regularization and help in a generalized solution. The batch normalization ensures that similar to the inputs being on the same scale, which helps in faster convergence, the intermediate layers also have the same benefit. While our proposed solution has encompassed two hidden layers, an arbitrary number of hidden layers might be added to create a deeper neural network. Figure 4 depicts the overall flow of the solution.

The outputs of each of the hidden layers in the DNN are shown in Eqs. (10) to (12) below. As we see, each of the hidden layers perform an affine and nonlinear *ReLU* activation in consecutive order. The output of the innermost

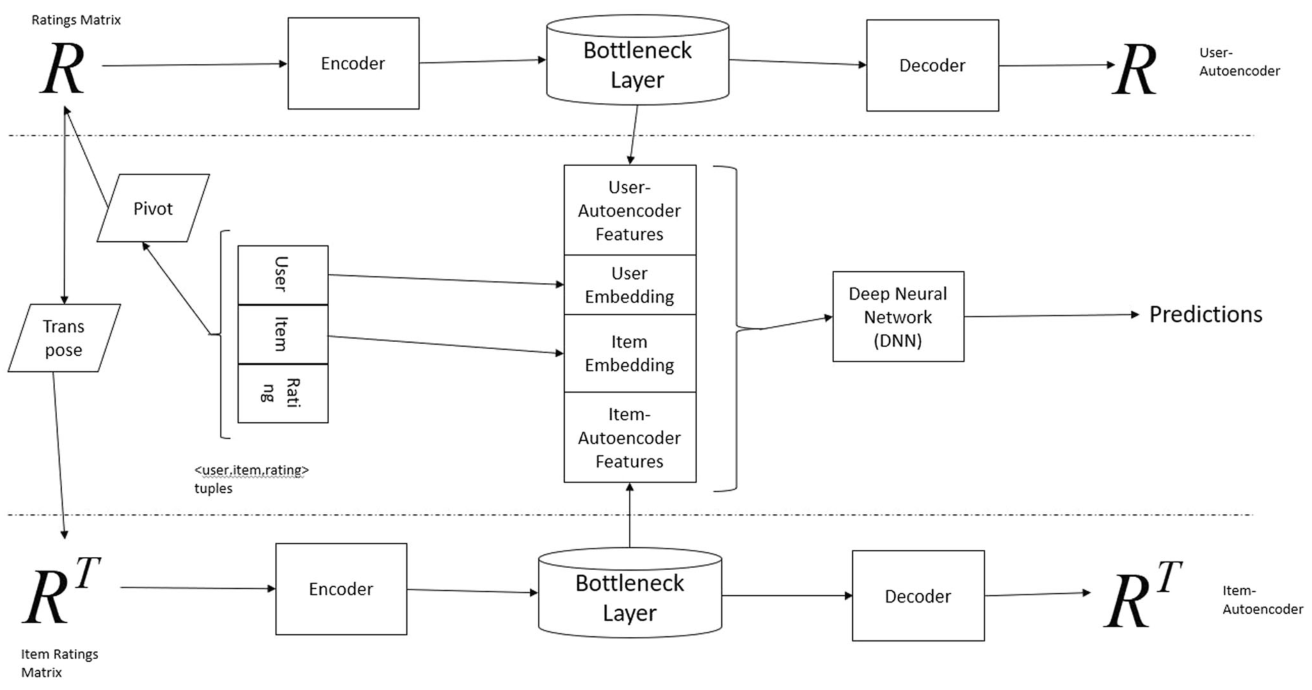

Fig. 4 DAFERec overall flow: proposed solution

layer is passed through a sigmoid activation function, as shown in Eq. (13). The resulting output, which is a real number between 0 and 1, is then transformed to the rating scale, as shown in (14). (15) shows the entire list of operations performed on the input

$$X_1 = \text{dropout}_1(\text{Re}LU(W_1 * X_0 + b_1)) \quad (10)$$

$$X_2 = \text{dropout}_2(\text{Re}LU(W_2 * X_1 + b_2)) \quad (11)$$

$$X_3 = \text{dropout}_3(\text{Re}LU(W_3 * X_2 + b_3)) \quad (12)$$

$$\sigma_{\text{output}} = \sigma(X_3) \quad (13)$$

$$\text{output} = \min_{\text{rating}} + \sigma_{\text{output}} * \left(\max_{\text{rating}} - \min_{\text{rating}}\right) \quad (14)$$

$$\sigma_{\text{output}} = \sigma(\text{dropout}_3(\text{Re}LU(W_3 * \text{dropout}_2 (\text{Re}LU * (W_2 * (\text{dropout}_1(\text{Re}LU(W_1 * X_0 + b_1))) + b_2)) + b_3)) \quad (15)$$

Considering a motivational example of ML 100 K dataset, that has 100,836 ratings from 6000 users for 4000 items. The first step of DAFEREC will create an user autoencoder that has 4000 features at the input and the same is attempted to be replicated at the output. The second step of DAFEREC will create an item autoencoder that has 6000 features at the input and the same are attempted to be replicated at the output. The inner-most layers of the user autoencoder and item autoencoder have 25 features each, say, for example. The third step of DAFEREC uses a Deep Neural Network with the 25 user autoencoder features combined with the 25 item autoencoder features, along-with the user and item embeddings, thus learning a very rich representation from incorporating multiple nonlinearities.

4 Experimental setup, results and discussion

This section presents the dataset, holdout dataset creation methodology, criteria for evaluation and describes how the comparative methods were implemented.

4.1 Datasets used

In this paper, we considered MovieLens data. The benchmark MovieLens 100 K (ML100k) and MovieLens 20 M (ML 0M) datasets are used for experimentation. Of the different MovieLens datasets, the ML20M dataset and ML100K datasets are employed for this study. ML20M dataset has 20 million ratings while the ML100k dataset has 100,000 ratings. These datasets do not include any demographic data, include tagging features, and map the MovieLens movies to IMDB and YouTube. ML20M dataset is employed as it is the suggested dataset for research. ML100K dataset has a smaller set of users but is similar in format and structure to the ML20M dataset and is the dataset that is recommended for research and development. Both these datasets are provided by GroupLens. In addition to movie recommendation datasets, we also considered book recommendations. The BookCrossing [31] dataset was utilized for book recommendation that consists of book ratings provided by users. It is a relatively large dataset as we see in Table 2 with over a million ratings. In addition to the above three datasets, we also considered the FilmTrust dataset [32] that also contains side information. The characteristics of these datasets are in Table 2.

4.2 Side information

The use of side information in algorithm construction has been employed in several studies, and they also present a way of addressing the cold-start issue. The experiments were performed without using side information. Side information was not considered as these are not available for all the datasets, and the goal is to achieve the state-of-the-art considering just the $\langle \text{user}, \text{item}, \text{rating} \rangle$ tuples. Further, most of the benchmark papers on these datasets have state-of-the-art scores for the datasets without side information. So, while DAFERec has the capability to incorporate side information, the same was not attempted in this paper. Additional information available about the user and item profiles was not considered for reasons already stated. The BookCrossing dataset contains attributes of books like author, title, ISBN, publisher and also attributes of authors like title, gender, age, location, etc. Incorporating side information into DAFERec is actually very straightforward and just involves stacking the features from the side information as input into the first hidden layer in Fig. 3. Incorporating these would further improve the results of DAFERec, but a comparison against the state-of-the-art in the benchmark papers would not be possible. TrustSVD is not comparable to our proposed solution as it uses side information about the users and items. We include a reference to this method as the best-reported RMSE on the FilmTrust dataset is using this method. Only for the purposes of comparison, our solution has been tested with side information for the FilmTrust cases to ensure an apple-to-apple comparison.

4.3 Evaluation metrics

To be able to benchmark the solution against existing papers, multiple evaluation criteria were considered like the L1-error (also called MAE, that penalizes the absolute deviation), the L2-error (also called MSE, that penalizes larger errors more severely) and a penalization that is in between like the RMSE. Equations (15) to (17) contain the forms of RMSE, MSE and MAE respectively.

$$\text{RMSE} = \sqrt{\sum_{i,j}^{m,n} T_{ij}\left(r_{ij}^{\text{actual}} - r_{ij}^{\text{predicted}}\right)^2} \tag{16}$$

$$\text{MSE} = \sum_{i,i}^{m,n} T_{ij}\left(r_{ij}^{\text{actual}} - r_{ij}^{\text{predicted}}\right)^2 \tag{17}$$

$$\text{MAE} = \sum_{i,j}^{m,n} T_{ij}\left|\left(r_{ij}^{\text{actual}} - r_{ij}^{\text{predicted}}\right)\right| \tag{18}$$

4.4 Comparison with existing approaches

Other than these, the best scores reported by recommender system packages like LibRec, LensKit, MyMediaLite, and Surprise were also considered.The same cross-validation methodology that was used in the benchmark papers and software systems was used in our study. The set of benchmark papers considered in our study are in Table 3.

5 Results and discussion

Before we discuss the results, we present our testing process and validation approach.

5.1 Testing process and validation approach

To be able to compare our solution against the reported benchmarks on the six datasets, it is important to use the same evaluation metric that was used in the papers that reported the benchmark results. Since different papers used different metrics like MSE, RMSE, and MAE, we used the same metrics to evaluate algorithm performance. The existing approaches test environment is similar to the benchmark reference papers in Table 3.

5.2 Results

Here, we discuss the results that are presented. We see that the proposed solution in the paper, DAFERec, outperforms on all the benchmark datasets. The proposed solution is benchmarked against some key papers in the area and the software tools that are listed in Table 3. As already mentioned, to ensure comparability, the validation and training split for the datasets were chosen to be the same as that in the benchmark papers. The tenfold cross-validation technique that we used ensures the reliability of the obtained results.The MSE, RMSE, and MAE metrics for the different methods are tabulated in Table 4. We have made our implementation available for reproducing the results.[1] The DAFEREC hyper-parameters and architecture for each of the datasets is optimized on the respective datasets using the holdout dataset.

[1] https://github.com/efpm04013/experiment3.

Table 2 Datasets used for experimentation

Dataset	Users	Items	Ratings	Sparsity	Scale
BookCrossing	278,858	271,379	1,149,780	99.99%	[0–10]
ML100K	610	9742	100,836	93.7%	[1–5]
ML1M	6000	4000	1,000,000	95.8%	[1–5]
FilmTrust	1508	2071	35,497	98.86%	[1–5]

Table 4 Results of running DAFERec on the different datasets

SI#	Dataset	RMSE	MAE
1	BookCrossing	1.6160	1.24
2	FilmTrust	0.7905	0.609
3	FilmTrust (with side information)	0.781	0.601
4	MovieLens 100 K	0.8545	0.6679
5	MovieLens 1 M	0.8268	0.643

5.3 Performance delta against existing methods

Table 5 shows the extent of outperformance of DAFERec against the comparative studies.

We see that DAFERec beats the benchmarks reported in the literature for the four datasets of varying sizes. We have demonstrated the outperformance of DAFERec on different datasets. Based on the results, we see that the outperformance is on datasets of varying sizes and sparsity. Note that the outperformance on the BookCrossing dataset is rather stark, and the reason for the same is that there are not many benchmarks available for the BookCrossing recommendation dataset. The explanation of the outperformance is that it is due to the nonlinear latent features about users and items that are captured by the user autoencoder and the item autoencoder, respectively.

6 Conclusion and future research

Improving the predictive accuracy of recommender systems has very useful management implications as it limits the choice overload resulting in better choices for users and more profits for businesses. We propose a novel solution called DAFERec (Deep Autoencoders for Feature Learning with Embeddings for Recommendations). DAFERec combines the inner-most activations of the autoencoders as features into a DNN together with the embeddings. The higher order innermost nonlinear latent features from the user autoencoder and the item autoencoder are employed to enable the DNN to learn a compressed higher order representation. The usage of latent variables in recommender systems is to exploit the underlying user preferences or item characteristics. This work proposes a novel approach to feature engineering in recommender systems using features from the bottleneck layers of the autoencoder in combination with embeddings in a DNN to generate recommendations. This method learns linear and nonlinear latent variables from both the autoencoder features as well as the embeddings, resulting in a rich feature set that is exploited by the DNN for improved system performance. Our approach provides a useful way to learn effective linear and nonlinear latent factors to improve recommender system performance. We also propose a systematic and easy way to integrate auxiliary information as additional features into our architecture and demonstrate the outperformance on the FilmTrust dataset, where the existing state-of-the-art TrustSVD method used side information. Details about the type of side information used is in Sect. 4.2.

There are several opportunities for further research. One opportunity could be to understand the additional value that side information about users and items could bring to the predictive accuracy of the recommender systems. A study could incorporate all the side information from each of the datasets to publish the state-of-the-art possible with such addition. No such study exists, and most of the studies are concentrated on the MovieLens, Epinions, and Netflix datasets. Part of this study could also focus on the benefits

Table 3 Comparative benchmarks on the considered datasets

Referenced data	Reference paper	Side information included?	RMSE	MAE
Book crossing	mDA-CF [33]	N	2.251	NA
FilmTrust	TrustSVD [34]	Y	0.789	0.609
	BiasedMF[34]	N	0.802	0.616
MovieLens 100 K	mDA-CF [26]	N	0.8852	0.68
	ItemKNN [35]	N	0.899	0.703
MovieLens 1 M	I-AutoRec [24]	N	0.831	NA
	mDA-CF [33]	N	0.8416	NA
	SVD+ + [36]	N	0.851	0.668

Table 5 Performance Delta of DAFERec vs. comparative studies

Dataset	Reference Paper	Side Information included?	RMSE	MAE
BookCrossing	mDA-CF [33]	N	28%	NA
FilmTrust	TrustSVD [34]	Y	1%	1.3%
	BiasedMF [34]	N	1.4%	1.1%
MovieLens 100 K	mDA-CF [26]	N	3.5%	1.8%
	ItemKNN [35]	N	4.9%	5%
MovieLens 1 M	I-AutoRec [24]	N	0.5%	NA
	mDA-CF [33]	N	1.8%	NA
	SVD + + [36]	N	2.8%	4.1%

of side information for users and items with many ratings versus the ones with few ratings so that the utility of the auxiliary information in several cases is comparable. The incremental value from a denoising autoencoder could also be incorporated. A denoising autoencoder corrupts information randomly in the input to help increase the generalization and prevent the learning of just an identity function. Such a study could help identify if there is incremental value and the extent of the incremental value of denoising autoencoders. One of the limitations of the study is that autoencoders are slow to train compared to other linear matrix factorization methods. A part of future work could also be to optimize the computation of our algorithm for training time and prediction time considerations.

Funding Not applicable.

Data availability Publicly datasets used with links below.Book-Crossing: http://www2.informatik.uni-freiburg.de/~cziegler/BX/BX-CSV-Dump.zip, MovieLens100K: http://files.grouplens.org/datasets/movielens/ml-latest-small.zip, MovieLens1M: http://files.grouplens.org/datasets/movielens/ml-1m.zip, FilmTrust: https://www.librec.net/datasets/filmtrust.zip.

Code availability Code made available for reproducible research at https://github.com/efpm04013/experiment3.

Declarations

Conflict of interests Not Applicable.

References

1. Aggarwal CC (2016) An introduction to recommender systems. RecomSyst. https://doi.org/10.1007/978-3-319-29659-3_1
2. Ketkar N (2017) Introduction to pytorch. Deep learning with python. Apress, Berkeley, pp 195–208. https://doi.org/10.1007/978-1-4842-2766-4_12
3. Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang, Z. (2015) MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. https://doi.org/10.1145/2532637
4. Google (2018) Tensorflow. 2018
5. Kiran R, Kumar P, Bhasker B (2020) DNNRec: a novel deep learning based hybrid recommender system. Expert SystAppl. https://doi.org/10.1016/j.eswa.2019.113054
6. Lee W, Song K, Moon IC (2017) Augmented variational autoencoders for collaborative filtering with auxiliary information. In: Proceedings of the 2017 ACM on conference on information and knowledge management - CIKM '17. pp 1139–1148. https://doi.org/10.1145/3132847.3132972
7. Langville A (2005) Nonnegative matrix factorization. Matrix 25(6):1336–1353. https://doi.org/10.1109/TPAMI.2011.18
8. Salakhutdinov R, Mnih A (2008) Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In: Proceedings of the 25th international conference on machine learning - ICML '08. https://doi.org/10.1145/1390156.1390267
9. Sarwar BM., Karypis G, Konstan JA, Riedl JT (2000) Application of dimensionality reduction in recommender system - a case study. Architecture. https://doi.org/10.1138/744
10. Guo G, Zhang J, Yorke-Smith N (2015) TrustSVD: collaborative filtering with both the explicit and implicit influence of user trust and of item ratings. In: Proceedings of the national conference on artificial intelligence.
11. Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In: Proceedings of the 24th international conference on Machine learning - ICML '07. pp 791–798. https://doi.org/10.1145/1273496.1273596
12. He X, Liao L, Zhang H, Nie L, Hu X, Chua T-S (2017) (NCF) Neural collaborative filtering. https://doi.org/10.1145/3038912.3052569
13. Catherine R, Cohen W (2017) Transnets. In: Proceedings of the eleventh ACM conference on recommender systems - RecSys '17. pp 288–296. https://doi.org/10.1145/3109859.3109878
14. Cheng HT, Ispir M, Anil R, Haque Z, Hong L, Jain V, et al. (2016) Wide & deep learning for recommender systems. In: Proceedings of the 1st workshop on deep learning for recommender systems - DLRS 2016. pp 7–10. https://doi.org/10.1145/2988450.2988454
15. van den Oord A, Dieleman S, Schrauwen B (2013) Deep content-based music recommendation. Electron InfSyst Depart (ELIS). https://doi.org/10.1109/MMUL.2011.34.van
16. Liang D (2014) Content-aware collaborative music recommendation using pre-trained neural networks. In: ISMIR 2015: Proceedings of the 16th international society for music information retrieval conference. pp 295–301
17. Tan J, Wan X, Xiao J (2016) A neural network approach to quote recommendation in writings. In: Proceedings of the 25th ACM international on conference on information and knowledge management - CIKM '16. pp 65–74. https://doi.org/10.1145/2983323.2983788
18. Bansal T, Belanger D, McCallum A (2016) Ask the GRU. In Proceedings of the 10th ACM conference on recommender

systems - RecSys ’16. pp 107–114. https://doi.org/10.1145/2959100.2959180

19. Ko Y-J, Maystre L, Grossglauser M, Durrant RJ, Kim K-E (2016) Collaborative recurrent neural networks for dynamic recommender systems. JMLR 63:366–381. https://doi.org/10.1109/ICDE.2016.7498326
20. Balakrishnan A (2014) Deepplaylist: using recurrent neural networks to predict song similarity. In: Stanfort University, pp 1–7. Retrieved from https://cs224d.stanford.edu/reports/BalakrishnanDixit.pdf
21. Wu Y, DuBois C, Zheng AX, Ester M (2016) Collaborative denoising auto-encoders for top-N recommender systems. In: Proceedings of the ninth ACM international conference on web search and data mining - WSDM ’16. pp 153–162. https://doi.org/10.1145/2835776.2835837
22. Li X, She J (2017) Collaborative variational autoencoder for recommender systems. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining - KDD ’17. pp 305–314. https://doi.org/10.1145/3097983.3098077
23. Sedhain S, Menon AK, Sanner S, Xie L (2015) Autorec: autoencoders meet collaborative filtering. In: WWW 2015 companion: proceedings of the 24th international conference on world wide web. pp 111–112. https://doi.org/10.1145/2740908.2742726
24. Sedhain S, Menon AK, Sanner S, Xie L (2015) Autorec: autoencoders meet collaborative filtering. In: 24th international conference on world wide web. https://doi.org/10.1145/2740908.2742726
25. Vincent P, Larochelle H, Bengio Y, Manzagol PA (2008) Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on machine learning - ICML ’08. https://doi.org/10.1145/1390156.1390294
26. Li S, Kawale J, Fu Y (2015) Deep collaborative filtering via marginalized denoising auto-encoder. In: Proceedings of the 24th ACM international on conference on information and knowledge management - CIKM ’15. pp 811–820. https://doi.org/10.1145/2806416.2806527
27. Strub F, Gaudel R, Mary J (2016) Hybrid recommender system based on autoencoders. In: proceedings of the 1st workshop on deep learning for recommender systems - DLRS 2016. pp 11–16. https://doi.org/10.1145/2988450.2988456
28. Braida F, Mello CE, Pasinato MB, Zimbrão G (2015) Transforming collaborative filtering into supervised learning. Expert SystAppl. https://doi.org/10.1016/j.eswa.2015.01.023
29. Barbieri J, Alvim LGM, Braida F, Zimbrão G (2017) Autoencoders and recommender systems: COFILS approach. Expert SystAppl 89:81–90. https://doi.org/10.1016/j.eswa.2017.07.030
30. Strub F, Mary J, Strub F, Mary J, Filtering C, Autoencoders D, Strub F (2015) Collaborative filtering with stacked denoising autoencoders and sparse inputs. In: NIPS workshop on machine learning for ecommerce, December 2015
31. Ziegler CN, McNee SM, Konstan JA, Lausen G (2005) Improving recommendation lists through topic diversification. https://doi.org/10.1145/1060745.1060754
32. Guo G, Zhang J, Yorke-Smith N (2013) A novel bayesian similarity measure for recommender systems. In: IJCAI international joint conference on artificial intelligence
33. Li S, Kawale J, Fu Y (2015) Deep collaborative filtering via marginalized denoising auto-encoder. https://doi.org/10.1145/2806416.2806527
34. LibRec - Examples & Comparison with Other Recommendation Libraries (n.d.). Retrieved 24 Nov 2018, from https://www.librec.net/release/v1.3/example.html
35. LibRec - Examples & comparison with other recommendation libraries. (n.d.). Retrieved 27 July 2019, from https://www.librec.net/release/v1.3/example.html
36. MyMediaLite: Example Experiments. (n.d.). Retrieved 24 Nov 2018, from http://www.mymedialite.net/examples/datasets.html