Deep learning approach to obtain collaborative filtering neighborhoods

Jesús Bobadilla¹ · Ángel González-Prieto^{2,3} · Fernando Ortega¹ · Raúl Lara-Cabrera¹

Received: 7 April 2021 / Accepted: 30 August 2021 / Published online: 5 October 2021
© The Author(s) 2021

Abstract

In the context of recommender systems based on collaborative filtering (CF), obtaining accurate neighborhoods of the items of the datasets is relevant. Beyond particular individual recommendations, knowing these neighbors is fundamental for adding differentiating factors to recommendations, such as explainability, detecting shilling attacks, visualizing item relations, clustering, and providing reliabilities. This paper proposes a deep learning architecture to efficiently and accurately obtain CF neighborhoods. The proposed design makes use of a classification neural network to encode the dataset patterns of the items, followed by a generative process that obtains the neighborhood of each item by means of an iterative gradient localization algorithm. Experiments have been conducted using five popular open datasets and five representative baselines. The results show that the proposed method improves the quality of the neighborhoods compared to the *K*-Nearest Neighbors (KNN) algorithm for the five selected similarity measure baselines. The efficiency of the proposed method is also shown by comparing its computational requirements with that of KNN.

Keywords Gradient localization · Neighbors · Generative learning · KNN · Classification · Deep learning

1 Introduction

Obtaining collaborative filtering (CF) neighborhoods is important in the recommender systems (RSs) field, since it has several beyond prediction applications such as big data representation, get entity dependencies or to provide recommendation reliabilities. Neighborhoods have been usually obtained by applying the *K*-Nearest Neighbors (KNN) algorithm. This memory-based method is not accurate and efficient to make predictions and recommendations, compared to the model-based ones: mainly the matrix factorization (MF) and neuronal CF. Nevertheless, there are not

specific model-based methods to obtain CF neighborhoods, since the models do not require this intermediate step, as the KNN algorithm. The main contribution of the proposed approach is, precisely, to provide an original model-based deep architecture to obtain the neighborhood of each RS user or item.

The proposed model design uses both the gradient-based localization technique (GLT) and a deep learning RS architecture. The first one is a specific neural style transfer (NST) method borrowed from the image processing field, whereas the RS classification architecture is specifically designed for the CF field. By combining these concepts, we create deep gradient-based neighbors (DGN), a novel deep learning model able to catch the sparse CF distributions and to return accurate neighborhood sets. Roughly speaking, DGN operates as follows. First, we design a classification-based neural network (NN) containing as many output neurons as existing items in the CF dataset. After training this NN, we get the patterns of each item coded into the hidden layer of the NN. Then, for each active item, we run the GLT/NST algorithm to iteratively convert noisy values into suitable similarity values. Using these similarity

✉ Raúl Lara-Cabrera
raul.lara@upm.es

¹ Departamento de Sistemas Informáticos, ETSI Sistemas Informáticos, Universidad Politécnica de Madrid, C/ Alan Turing s/n, 28031 Madrid, Spain

² Departamento de Álgebra, Geometría y Topología, Universidad Complutense de Madrid, Plaza Ciencias 3, 28040 Madrid, Spain

³ Instituto de Ciencias Matemáticas (CSIC-UAM-UCM-UC3M), C/ Nicolás Cabrera, 13-15, 28049 Madrid, Spain

values, the neighborhood of each item can be computed as the most similar items.

The GLT/NST amounts to a minimization problem. Fixed an item j of the dataset, the GLT algorithm seeks to the input pattern for the trained NN that minimizes the classification error of predicting j . To carry out the optimization process, the GLT algorithm makes use of the gradient information of the previously learnt classification NN. Figure 1 shows the concept, where the lengths of the backward arrows identify the relevancy of each gradient. Higher values of the gradients lead to higher values of the neighbors pointed by these gradients (boxed input values). The one-hot encoding of the active item j (gray circle in the output) produces, by means of gradient optimization through backpropagation, subsequent refinements of the initial noisy values that converge to the optimal input pattern (iteration n in Fig. 1).

Furthermore, our proposed DGN method addresses the calculation of CF neighborhoods very efficiently. Once the classification NN model has learnt, the GLT/NST can perform the localization task by using the condensed information in the NN hidden layer. This number of hidden units is, in practice, much smaller than the number of users in a commercial CF dataset, leading to a more efficient computation than for KNN-based algorithms.

The paper is structured as follows. Section 2 provides the related work to this paper and Sect. 3 describes the proposed DGN method, both the associated NN and its input values (Sects. 3.1 and 3.2) and the applied gradient localization procedure (Sect. 3.3). For the sake of reproducibility, Sect. 3.4 provides precise description of the used implementation, and in Sect. 3.5, we discuss the time complexity of our proposal versus KNN. In Sect. 4, we describe the experiments conducted to test the idea (Sect. 4.1) as well as the obtained results (Sect. 4.2).

Fig. 1 The GLT/NST stage of the deep gradient-based neighbors (DGN) proposed method. Backward arrows represent gradient values

Finally, Sect. 5 summarizes the main conclusions of the paper and outlines future work lines.

2 Related work

Research in RSs has quickly evolved from their first content-based approaches [39] until the current NN models [42]. Along this road, the KNN algorithm had a prominent place. A substantial amount of research around KNN has been focused on the similarity measures [7] used to obtain the existing similarities among users or items. In this vein, user-based and item-based KNN approaches [22] have been implemented to make CF [7] predictions and recommendations. In the beginning of KNN-based research in CF, statistical metrics were chosen to feed the KNN algorithm: cosine, Pearson correlation, Jaccard, Euclidean, Mean Squared Differences (MSD), etc. [22]. Later, a set of specific metrics adapted to very sparse environments were designed such as Mean Squared Differences (MSD) [8] or PIP [1]. The set of specific metrics designed for CF allowed researchers to raise the accuracy of recommendations. It also made possible to afford new challenges such as explaining recommendations [3, 25], visualizing big data related information, detecting and avoiding shilling attacks [17, 38], issuing recommendations to groups of users [11], clustering [23, 24] and obtaining recommendation reliabilities [27]. It is remarkable the fact that the applications of the KNN algorithm in the CF field extend beyond the prediction and recommendation task; hence, it cannot be completely replaced by the current RS model-based approaches.

The classical approach in machine learning (ML) to get neighborhood sets is the use of some suitable similarity measure in the data domain and then to apply the KNN algorithm. However, this approach is not perfectly efficient neither accurate, since: (a) it does not properly scale, (b) similarity measures do not adequately work on very sparse scenarios such as the CF-based RS one, (c) it is not clear which of the many designed similarity measures is better to select for each dataset, (d) similarity measures are not adequate in cold start scenarios, and (e) the existing similarity measures do not capture the existing nonlinear relations in the CF information.

Only regarding predictions, the KNN memory-based approach has nearly been replaced by the MF [26, 29] model-based method. By using MF, we first need to train a model and then this model can be used to make predictions and recommendations. Once the model has been trained, recommendations can be made much faster than using the KNN algorithm, since they reduce to efficient matrix multiplication. Additionally, the MF accuracy [32] is also greater than the KNN accuracy. The main MF drawback is

that we must retrain periodically the model. Even though existing commercial RSs [16] make use of MF-based models, current research heads toward deep NN models [42]. Deep NN models can capture subtler complex non-linear relations existing in the CF RS datasets [21] and they also enable information fusion [14, 31] from CF, content-based [39], social [35, 40], context-aware [33] and demographic data [4]. Deep learning architectures are determined by the type of information that they manage: convolutional neural networks (CNNs) [36, 37], multilayer perceptrons (MLPs) [2, 13], autoencoders [41, 43], etc.

Most of the designed NN to address the RS task can be classified as either (1) neural collaborative filtering (NCF) [12–21] or (2) deep factorization machines (DeepFM) [19]. Usually, NCF models simultaneously process both users and items information. They provide two early stages to code both the user and the item latent vectors, transforming the sparse input data into dense information. Later, a MLP is sequentially added to make predictions. On the other hand, DeepFM architectures combine the NCF deep design with an auxiliary wide design to incorporate additional information, such as the demographic data.

Additionally, once the classification NN has learnt the underlying patterns and profiles, we shall use a gradient-based generative method to obtain the neighbors of each item. To accomplish this task, we take the idea of the GLT from the image processing field: GLT can find the most relevant areas in the CNN activation maps. A remarkable realization of GLT is the Grad-Cam [34] design, which improves the previously proposed Class Activation Map [44] method and aims to highlight the relevant areas in a NN when an image is entered in the network. We shall also apply in this work the NST technique [15], which captures image styles from the hidden layers of a CNN by making use of the GLT algorithm to iteratively find the input image that better fits with the styles of the map activation learnt by the NN. Specifically, first a random image is shown to the NN and feedforwarded, later a loss function is constructed to get the difference between the learnt style and the obtained activation maps, and finally the input image is gradient-based refined. Figure 2 shows an example where

Fig. 2 Neural style transfer (NST) example

an input picture is iteratively modified through NST until it gets the chosen NN base style.

3 The deep gradient-based neighbors (DGN) method

The proposed method, DGN, has two sequential stages: (a) neural learning of the item patterns and (b) gradient localization. The first stage aims to extract the patterns of the items by using a classification-based NN. The second stage finds the input item distribution that minimizes the NN loss function by processing successive iterations in a gradient descent minimization.

Figure 3 shows the big picture of the stages comprised in the proposed method: the first sequential stage (a) trains a classifier MLP to get the recommended item from a set of relevant versus non-relevant votes from a user to the RS items. The second sequential stage (b) makes use of the previously trained MLP to obtain the combination of input item values that minimizes the error on the output target item. To illustrate this idea, suppose that we want to obtain the neighbors of the second item in the RS (coloured in grey in Fig. 3b). A gradient descent localization process is run to get the input item values that minimize the error classification for the gray output item. After the optimization, the higher the absolute value of each input item, the higher the importance of this item in the target neighborhood.

To understand the rationale behind the method, we can think about an ideal scenario in which only three input items (left side of Fig. 3b) are able to produce a tiny classification error (right side of Fig. 3b). The other values of the input items are negligible. In this situation, we can conclude that there is a strong relation between the three input values and the chosen output value, so that they can be considered as neighbors. Of course, typically in real scenarios, the situation is more involved and the relevant items are not so clearly identified. It is precisely in this fuzzy context where the proposed model is more effective,

Fig. 3 Proposed method stages: **a** neural learning of the item patterns and **b** gradient localization

since its nonlinear nature is able to extract more subtleties of the hidden information.

The neural learning of the item patterns (Fig. 3a) has been taken from [6]. The proposed method in [6] shows an adequate performance by making item classification and its associated item recommendation. Our proposed method makes use of the item classification stage from [6], as input for the GLT.

3.1 Preprocessing of the recommendation matrix

In this section, we describe how to transform the sparse matrix of known votes, the usual input of CF RSs, into a dataset that can be used to train the aforementioned NN to provide accurate recommendations.

Roughly speaking, the core idea of this coding is to discard the item vectors as input neurons and to place them in the number of training samples instead. This procedure first appeared in [6] and Fig. 4 shows its architecture. The deep NN input and output layers have as many neurons as items can be voted by each user. Due to the chosen design, the NN classifier is trained using samples that contain the votes of users on all the items of the dataset. Specifically, this NN is fed with as many samples as relevant ratings are contained in the dataset.

To formalize precisely this idea, suppose that our system contains U users and I items, so that the known ratings are collected into a sparse matrix $R = (r_{u,i})$ of shape $U \times I$. If the u -th user voted the i -th item, then $r_{u,i} \in \mathcal{S}$ where $\mathcal{S}$ is the set of possible scores that can be assigned to an item. For simplicity, we shall suppose that $\mathcal{S}$ is a discrete subset of $\mathbb{R}$ (typically, integers from 1 to 5, or half-integers from 0 to 10). If the u -th user did not vote the i -th item, we set $r_{u,i} = \bullet$.

Now, we fix a threshold θ . Using this value, we discretize the matrix R into a dense matrix of relevant/irrelevant votes R^θ . This is a binary matrix whose (u, i) -entry is 1 if the u -th user rated the i -th with a score greater or equal to

θ and the entry is 0 otherwise. To be precise, the matrix $R^\theta = (r_{u,i}^\theta)$ is given by $r_{u,i}^\theta = 1$ if $r_{u,i} \neq \bullet$ and $r_{u,i} \geq \theta$ and $r_{u,i}^\theta = 0$ otherwise (i.e. if $r_{u,i} = \bullet$ or $r_{u,i} < \theta$).

Let us denote the u -th row of R^θ by

$$\mathbf{r}_u^\theta = (r_{u,1}^\theta, r_{u,2}^\theta, \dots, r_{u,I}^\theta)$$

This is a I -dimensional vector containing the relevance of the items for the u -th user. Also, set $\mathbf{e}_\ell$ for the ℓ -th vector of the canonical basis of $\mathbb{R}^I$ i.e., $\mathbf{e}_\ell$ is a I -dimensional vector whose components are all zero except the ℓ -th entry, which is one. Now, for each pair (u, i) with $r_{u,i}^\theta = 1$, we set

$$\mathbf{x}_{(u,i)} = \mathbf{r}_u^\theta - \mathbf{e}_i, \quad \mathbf{y}_{(u,i)} = \mathbf{e}_i.$$

In other words, $\mathbf{x}_{(u,i)}$ is the u -th row of R^θ where we have removed the 1 corresponding to the i -th item.

From these vectors, we construct the dataset to be fed to the NN as $\mathbf{X} = (\mathbf{x}_{(u,i)})$ and $\mathbf{Y} = (\mathbf{y}_{(u,i)})$, where the indices (u, i) run over the entries with $r_{u,i}^\theta = 1$. The collection $\mathbf{X}$ is used as the input data of the NN, which it is trained to predict the corresponding values of $\mathbf{Y}$. Beware that, unlike it is customary in ML, the vector $\mathbf{x}_{(u,i)}$ is a whole row of the input dataset and $\mathbf{y}_{(u,i)}$ is a whole row of the output dataset.

3.2 Running example

To explain the construction of the previous section, we provide here a running example. The rating matrix R of this toy dataset is shown in Table 1. It contains $U = 4$ users and $I = 6$ items. The users can rate an item with an element of $\mathcal{S} = \{1, 2, 3, 4, 5\}$ (and $\bullet$ for the absence of rating).

Suppose that we fix as a threshold of relevancy $\theta = 4$. In that case, the associated binary matrix of relevant/irrelevant items, R^θ , is shown in Table 2. Recall that the (u, i) entry in Table 2 is 1 if the corresponding entry in Table 1 is ≥ 4 and is 0 otherwise.

From this relevant/irrelevant matrix, we can create the samples of the final dataset. For this purpose, let us consider a nonzero entry of Table 2, say $(u, i) = (3, 4)$. The associated row to the u -th user is $\mathbf{r}_3^\theta = (0, 1, 0, 1, 0, 1)$. To

Fig. 4 Classification-based neural network for CF RS

Table 1 Ratings matrix R of the running dataset

Users (u)	Items (i)	1	2	3	3	5	6
1		•	4	•	5	•	1
2	2	•	•	4	•	•	•
3	•	•	5	•	5	2	4
4	4	•	•	•	•	•	5

Table 2 Relevant/irrelevant matrix R^θ of the running example

Users (u)	Items (i)					
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	1	0	0	0
3	0	1	0	1	0	1
4	1	0	0	0	0	1

get the associated input datum $\mathbf{x}_{(3,4)}$, we remove the 1 at the $i = 4$ position. The aim of the NN is precisely to learn this removed vote, so the output datum is precisely the $i = 4$ vector of the canonical basis. In this way, we have

$$\mathbf{x}_{(3,4)} = \mathbf{r}_3^\theta - \mathbf{e}_4 = (0, 1, 0, 0, 0, 1),$$

$$\mathbf{y}_{(3,4)} = \mathbf{e}_4 = (0, 0, 0, 1, 0, 0).$$

Proceeding analogously to the remaining samples, we obtain that the associated input and output data are the following.

$$\mathbf{x}_{(1,2)} = (0, 0, 0, 1, 0, 0), \quad \mathbf{y}_{(1,2)} = (0, 1, 0, 0, 0, 0),$$

$$\mathbf{x}_{(1,4)} = (0, 1, 0, 0, 0, 0), \quad \mathbf{y}_{(1,4)} = (0, 0, 0, 1, 0, 0),$$

$$\mathbf{x}_{(2,3)} = (0, 0, 0, 0, 0, 0), \quad \mathbf{y}_{(2,3)} = (0, 0, 1, 0, 0, 0),$$

$$\mathbf{x}_{(3,2)} = (0, 0, 0, 1, 0, 1), \quad \mathbf{y}_{(3,2)} = (0, 1, 0, 0, 0, 0),$$

$$\mathbf{x}_{(3,4)} = (0, 1, 0, 0, 0, 1), \quad \mathbf{y}_{(3,4)} = (0, 0, 0, 1, 0, 0),$$

$$\mathbf{x}_{(3,6)} = (0, 1, 0, 1, 0, 0), \quad \mathbf{y}_{(3,6)} = (0, 0, 0, 0, 0, 1),$$

$$\mathbf{x}_{(4,1)} = (0, 0, 0, 0, 0, 1), \quad \mathbf{y}_{(4,1)} = (1, 0, 0, 0, 0, 0),$$

$$\mathbf{x}_{(4,6)} = (1, 0, 0, 0, 0, 0), \quad \mathbf{y}_{(4,6)} = (0, 0, 0, 0, 0, 1).$$

These values can be collected into a single dataset, as shown in Table 3. Observe that, as we mentioned in Sect. 1, the NN to be trained has $I = 6$ input variables and also $I = 6$ output variables. This can be read from the shape of the dataset, which has $2I = 12$ columns.

3.3 Gradient localization in the CF context

The proposed DGN method borrows the gradient-based localization technique (GLT) from the image processing domain. In this context, a trained CNN for image classification may use a variety of image filters to process the input image to issue an accurate prediction. In this manner, it is usual to find filters able to detect several skins and patterns such as brick walls, elephant skin, and fish scales. As an example, Fig. 3 includes the 64 filters learnt by the VGG16 Block4_conv1 layer, where a variety of recognizable patterns are shown.

Each picture of a pattern in Fig. 5 has been obtained by means of an iterative gradient descent process where the initial image is just a randomized set of pixel values. From

Table 3 Final dataset of the running example

Input data (X)					
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	1	0	1
0	1	0	0	0	1
0	1	0	1	0	0
0	0	0	0	0	1
1	0	0	0	0	0
Output data (Y)					
0	1	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1
1	0	0	0	0	0
0	0	0	0	0	1

Fig. 5 Learnt patterns in the 64 filters of the VGG16 Block4_conv1 layer

the initial image, we obtain the resulting image at the end of the first iteration of the gradient descent algorithm. The resulting image is fed into the NN input to run a second iteration that produces a second resulting image. This iterative process is accomplished a fixed number of times,

Fig. 6 Noisy input image (left image) and gradient descent result (right image). Images in the middle have been obtained from iteration 1, 3 and 6. The resulting image has been obtained in the iteration 40. The results were obtained through the VGG16 model

or until the resulting image is very similar (less than a threshold) to the previous one. Figure 6 shows several images taken from different steps in the explained gradient descent algorithm.

Our goal in this work is to transfer this idea from image processing to the CF context. For this purpose, first we learn the patterns of each item by training a classification-based MLP. This MLP is trained with the dataset (X, Y) constructed following the method of Sect. 3.1. This training is conducted using the classical techniques of deep learning. As by-product of this training process, the MLP implements a nonlinear function

$$h: \mathbb{R}^I \rightarrow \mathbb{R}^I.$$

This function should be interpreted as follows. Given a user profile $\mathbf{x} \in \mathbb{R}^I$, the output $h(\mathbf{x})$ is an I -dimensional vector which encodes the learnt probability distribution each of the items of the dataset of being liked by the user profile $\mathbf{x}$.

Once this NN has been trained, in the second stage of the proposed DGN method, we use the gradient localization algorithm to obtain the items' pattern that minimizes the prescribed loss function. Each of these patterns is represented by an I -dimensional vector. Figure 3b shows the concept: for each output item (the item colored in gray in the figure), we can 'gradient-localize' to seek the input vector of item values that minimizes the output classification error. This vector of item values represents the expected neighborhood of the output item.

Let us mathematically formalize this idea. Suppose that we are searching the neighbors of the j -th item. We define the cost function

$$\mathcal{F}_j: \mathbb{R}^I \rightarrow \mathbb{R}_{\geq 0}, \quad \mathcal{F}_j(\mathbf{x}) = \frac{1}{2}(1 - h_j(\mathbf{x}))^2,$$

where h_j denotes the j -th component of the function $h = (h_1, h_2, \dots, h_I)$. In other words, $\mathcal{F}_j(\mathbf{x}) = 0$ if and only if $h_j(\mathbf{x}) = 1$, which means that $\mathbf{x} \in \mathbb{R}^I$ is a combination of items preferences (a 'profile' of likes) for which the NN considers that its favorite item should be j . In order to minimize $\mathcal{F}_j$, we use a standard gradient descent algorithm. For this purpose, observe that the gradient of $\mathcal{F}_j$ is given by

$$\nabla \mathcal{F}_j(\mathbf{x}) = -(1 - h_j(\mathbf{x})) \nabla h_j(\mathbf{x}).$$

Observe that the gradient $\nabla h_j(\mathbf{x})$ can be easily computed in terms of the internal weights of the NN by means of the usual backpropagation method. Therefore, the usual gradient descent method leads to the update rule

$$\mathbf{x} \leftarrow \mathbf{x} + \eta(1 - h_j(\mathbf{x})) \nabla h_j(\mathbf{x}).$$

Here, $\eta > 0$ is a hyper-parameter of the training process that corresponds to the step of the gradient descent. Normalization techniques can also be applied to regularize the

gradient, such that L^2 normalization (dividing the gradient by its L^2 norm). The initial guess for $\mathbf{x}$ can be taken as a random vector drawn from a uniform distribution, or simply as the zero vector.

As a result of this optimization, we get a preferred profile of items $\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_I^*) \in \mathbb{R}^I$. The absolute value of these entries should be understood as the similarity of each item with the j -th item, i.e., the similarity of the i -th item with the j -th item is $|x_i^*|$. In other words, it defines a similarity measure given by

$$\text{DeepSim}_j(i) = |x_i^*|.$$

Equivalently, it gives rise to a distance from j given by

$$d_j^{\text{DeepSim}}(i) = \frac{1}{|x_i^*|}.$$

Using this distance, we can consider the closed ball of radius $1/v$ around j , denoted by $B_j^{\text{DeepSim}}(v)$, with respect to the distance d_j^{DeepSim} . That is, an item $i \in B_j^{\text{DeepSim}}(v)$ if and only if $\text{DeepSim}_j(i) \geq v$. This ball can be seen as the 'neighborhood' of j with similarity threshold v .

3.4 Implementation of the gradient descent localization algorithm

The proposed DGN method has been implemented using Keras. Keras has been chosen since it offers rich and simple application programming interfaces (APIs), it can be used in an easy way and it is "the most used deep learning framework among top-5 winning teams on Kaggle" [9]. Additionally, Keras takes advantage of the TensorFlow's deployment capabilities. Alternative solutions to Keras are the NVIDIA Compute Unified Device Architecture (CUDA), DeepPy (the MIT-licensed deep learning framework), Deeplearning4j, ScikitLearn, Theano, TensorFlow, and the open source machine learning library Torch.

The main Keras instructions used to define our dense classification-based MLP are shown in Listing 1. As it can

```

model = Sequential()
model.add(Dense(200, input_dim=NUM_ITEMS,
                  activation='relu'))
model.add(Dense(NUM_ITEMS, activation="relu",
                  name='output_layer'))
model.compile(loss='mean_squared_error',
              optimizer='adam', metrics=['mae'])

```

Listing 1 Keras definition of the MLP in the DGN proposed method.

The gradient descent localization algorithm is a slightly more difficult piece of code, so we have provided our Python-based function in Listing 2. We assume a

previously trained MLP whose model is accessible through the `model` property. This MLP has a dense output layer whose name must be used as first parameter for any call to the `dgn` function. The second parameter of this function, `item`, is the number (j) of the item for which we want to compute its neighbors (count from 1 to I).

For this purpose, Line 2 in Listing 2 establishes the output whose loss function we want to minimize: as it can be seen, it is the `item` neuron of the *MLP* output layer. Line 3 defines the loss function to be minimized, as mentioned in Sect. 3.3. Line 5 makes the key process: it obtains the gradients with respect to the `item`. Line 6 performs a L^2 gradient normalization [10, Section 5.4.2]. Line 8 establishes the Keras backend function iteration that obtains losses and gradients from the inputs; this function is iteratively called in Line 11 to obtain the gradient descent values. The neighbor values are iteratively updated in Line 12 by using the vector input defined in Line 9.

```

def dgn(layer, item):
    output = model.get_layer(layer).output[:, item-1]
    loss = 1/2*(1. - output)**2

    gradient = K.gradients(loss, model.input)[0]
    gradient /= (K.sqrt(K.mean(K.square(gradient)))+1e-5)

    iteration = K.function([model.input], [loss, gradient])
    input_ = np.expand_dims(np.zeros(NUM_ITEMS), axis=0)
    for i in range(50):
        loss_value, gradient_value = iteration([input_])
        input_ += gradient_value * 0.1

    return input_[0]

```

Listing 2 Gradient descent localization function.

3.5 Computational complexity

The proposed DGN method efficiently obtains the neighborhood of each item in comparison with the KNN algorithm.

KNN has complexity $O(UI)$, with U being the number of CF users, and I being the number of CF items. In the KNN algorithm, as shown in Listing 3, the target item i must be compared to all the other items j in the dataset, and for each of those items, the rating of each user u must be evaluated.

```

sim_ij = None
for j in item:
    if j == i:
        # skip item
    for u in users:
        if u has rated i & j:
            # update sim_ij

```

On the other hand, the proposed method has complexity $O(IHG)$. Once the MLP has learnt, the neighborhood of the item i is computed using the code of the Listing 2: for each active user (I ratings) we just need to feedforward G generative iterations from an initially random item vector ($G = 50$ in our implementation). Conducting G iterations has complexity $O(IHG)$, where H is the number of neurons in the hidden layer of the MLP ($H = 200$ in our implementation according to Listing 1).

To compare $O(UI)$ and $O(IHG)$, it is necessary to determine the order of magnitude between U and HG . Simple CF datasets can contain a limited number of users, and both U and HG can be considered ‘similar’ in these specific cases. For example, in our experiments using MovieLens 1M dataset $U = 6040$ and $H \cdot G = 200 \cdot 50 = 10,000$. However, commercial CF datasets can contain tens of millions of users (such as Netflix, Spotify, and Amazon) and just thousands of items (movies, songs, etc.). For this reason, the KNN algorithm is not scalable for commercial CF datasets. Conversely, the proposed DGN method can efficiently afford the task; that is because HG is much smaller than U . Typically, H will range between hundreds and a few thousands. Indeed, a common rule-of-thumb in deep learning (DL) is to take H as half of the number of input/output neurons, so $H \sim I$ in our case. Finally, G will range between tens and a few hundreds and it is actually independent of the size of the dataset, so it does not compromise the scalability of the method.

4 Experimental analysis

In this section, we describe the experiments designed to compare the performance of the proposed DGN method with the KNN algorithm using several CF baselines: Mean Squared Differences MSD [5], Jaccard [5], Jaccard Mean Squared Differences (JMSD) [8], cosine [22] and Euclidean [22]. The selected RS datasets are as follows: MovieLens 100K [20], MovieLens 1M [20], FilmTrust [18], MyAnimeList[★] (a subset of the original dataset) [28] and Netflix[★] [30] (a subset of the original dataset). Table 4 shows the main features of these datasets.

Table 4 Main parameters of the datasets used in the experiments

Dataset	#Users	#Items	#Ratings	Scores	Sparsity
MovieLens 100 K	943	1682	99,831	1 to 5	93.71
MovieLens 1 M	6,040	3,706	911,031	1 to 5	95.94
FilmTrust	1,508	2,071	32,675	0.5 to 4.0	98.96
MyAnimeList*	19,179	2,692	548,967	1 to 10	98.94
Netflix*	23,012	1,750	535,421	1 to 5	98.68

4.1 Experimental design

The starting point of the experiments is to obtain the neighborhood of each item for each baseline and for the proposed method. Thus, a vector of size I (#Items in Table 4) containing the similarity of each item with respect to the rest of the items must be stored. This vector represents the fuzzy neighborhood of an item. During the experiments, we have computed this vector for each item of the selected datasets using the following similarity measures: MSD, Jaccard, JMSD, cosine and Euclidean, all of them combined with the usual KNN method, as well as using the proposed DGN method. Throughout this section, the relevant thresholds for the different datasets required by the proposed model have been fixed to $\theta = 4$ (MovieLens 100K, MovieLens 1M and Netflix*), $\theta = 3$ (FilmTrust) and $\theta = 7$ (MyAnimeList*).

To test the quality of the predicted neighbors, we have designed two quality measures inspired by the correlation between the highest voted items of each user and the

highest valued neighbors of each item. Figure 7 graphically illustrates this evaluation methodology basics. The u -th user voted that the j -th item is relevant. For this reason, we expect that the predicted neighbors of item j will also have relevant votes. In the example shown in Fig. 7, the user u voted as relevant the items 0, 1 and 5, as well as j . For this reason, it may be expected that 0, 1 and 5 are neighbors of the item j . For instance, item j could be the ‘Avatar’ film, and items 0, 1, and 5 could be ‘Star Wars,’ ‘Alien’ and ‘Blade Runner.’

Formally, fix a standardized similarity method m so that $m_j(i)$ is the similarity of the i -th item measured from the j -th item in the similarity measure m . We define the distance

$$d_j^m(i) = \frac{1}{|m_j(i)|}.$$

As in Sect. 3.3, let us denote by $B_j^m(v)$ be the closed ball of radius $1/v$ for d_j^m . Now, we set $\kappa_{m,i}^j(v) = m_j(i)$ if $i \in B_j^m(v)$ and $\kappa_{m,i}^j(v) = 0$ otherwise, i.e., $\kappa_{m,i}^j(v)$ is the similarity measure truncated to $B_j^m(v)$. These values can be collected in a vector

$$\mathbf{K}_m^j(v) = (\kappa_{m,1}^j(v), \kappa_{m,2}^j(v), \dots, \kappa_{m,I}^j(v)).$$

From the above considerations, we can design two quality measures: user quality measure and item quality measure. The user quality measure focuses on how good is $\mathbf{K}_m^j(v)$ as an approximation of the real ratings (the ‘correlation’ between $\mathbf{K}_m^j(v)$ and the actual ratings). To be precise, let us define the user quality measures as

$$\mathcal{U}_m^j(v)^\uparrow = |\{1 \leq u \leq U \mid \mathbf{K}_m^j(v) \cdot \mathbf{r}_u^{\text{den}} > 0\}|,$$

$$\mathcal{U}_m^j(v)^\downarrow = |\{1 \leq u \leq U \mid \mathbf{K}_m^j(v) \cdot \mathbf{r}_u^{\text{den}} \leq 0\}|,$$

where $\mathbf{r}_u^{\text{den}}$ is the row in the ratings matrix $\mathbf{R}$ corresponding to user u , turned into a dense vector by setting the i -th component of $\mathbf{r}_u$ to 0 if $r_{u,i} = \bullet$. Recall that $\mathbf{K}_m^j(v) \cdot \mathbf{r}_u^{\text{den}}$ denotes the usual dot product between $\mathbf{K}_m^j(v)$ and $\mathbf{r}_u^{\text{den}}$. In other words, $\mathcal{U}_m^j(v)^\uparrow$ (resp. $\mathcal{U}_m^j(v)^\downarrow$) counts the number of users for which the truncated similarities are positively (resp. non-positively) correlated, in the sense that the angle between $\mathbf{K}_m^j(v)$ and $\mathbf{r}_u^{\text{den}}$ is small (resp. large).

Fig. 7 Illustration of the testing methodology based on correlations developed for this paper

Additionally, if we focus on the items, we can consider the associated item quality measures

$$\mathcal{I}_m^j(v)^\uparrow = \begin{cases} 1 & \text{if } \mathcal{U}_m^j(v)^\uparrow > \mathcal{U}_m^j(v)^\downarrow \\ 0 & \text{if } \mathcal{U}_m^j(v)^\uparrow \leq \mathcal{U}_m^j(v)^\downarrow \end{cases},$$

$$\mathcal{I}_m^j(v)^\downarrow = 1 - \mathcal{I}_m^j(v)^\uparrow.$$

In this manner, $\mathcal{I}_m^j(v)^\uparrow = 1$ if the number of items for which the success (measured as number of users with positive ‘correlation’ with the neighborhood) is larger than the failure (measured as number of users with non-positive correlation with the neighborhood) and $\mathcal{I}_m^j(v)^\uparrow = 0$ otherwise.

Averaging these quantities and taking into account that $\mathcal{U}_m^j(v)^\uparrow + \mathcal{U}_m^j(v)^\downarrow = U$ and $\mathcal{I}_m^j(v)^\uparrow + \mathcal{I}_m^j(v)^\downarrow = 1$ for all $1 \leq j \leq I$, we get the quality measures

$$Q_m^{\text{user}}(v) = \frac{1}{I} \sum_{j=1}^I \frac{\mathcal{U}_m^j(v)^\uparrow}{\mathcal{U}_m^j(v)^\uparrow + \mathcal{U}_m^j(v)^\downarrow} = \frac{1}{UI} \sum_{j=1}^I \mathcal{U}_m^j(v)^\uparrow.$$

$$Q_m^{\text{item}}(v) = \frac{1}{I} \sum_{j=1}^I \frac{\mathcal{I}_m^j(v)^\uparrow}{\mathcal{I}_m^j(v)^\uparrow + \mathcal{I}_m^j(v)^\downarrow} = \frac{1}{I} \sum_{j=1}^I \mathcal{I}_m^j(v)^\uparrow.$$

4.2 Results of the experiments

With these quality measures, we conducted several experiments to evaluate the proposed method. Figures 8, 9 and 10 contain graphs showing the users and the item quality measure values. The blue line represents the proposed method, and the other graphs correspond to the baseline quality measures with KNN used to compute the neighborhoods.

In particular, Fig. 8 shows the MovieLens dataset results, both in their 1M and 100K versions. It can be seen that, as expected, the proposed deep DGN method provides higher quality results both for the user and the item quality measures. The results evidence that the obtained neighborhoods are more accurate when the proposed method is used, compared to the traditional KNN algorithm that is fed with five similarity measure baselines.

Beyond the quality comparative among the five baselines, which is not relevant for this paper, we can see an interesting behavior of the proposed method: it provides particularly good comparative results when the filter parameter $v \geq 0$ is low, which corresponds to a permissive filtering. Thus, the proposed method is particularly good when catching up the similarities of the complete set of existing items, whereas the traditional KNN similarity measures return their better results when a balanced K value is chosen. This is consistent with the evolution of the baselines shown in Fig. 8: they usually reach their better results when neighbors are filtered ($v = 1$ and $v = 2$

values). Conversely, the proposed method shows a downward trend when applied to the MovieLens dataset. Additionally, as expected, the model-based proposed method performs comparatively better when applied to the biggest dataset (MovieLens 1M).

Figure 9 shows the results obtained from the datasets FilmTrust (a) and MyAnimeList[★] (b). It is remarkable the high performance of the proposed DGN method when the user quality measure is applied, compared to all the baselines. This is a behavior more similar to the MovieLens 1M case than to the MovieLens 100K one. Indeed, the datasets MovieLens 1M, FilmTrust and MyAnimeList[★] contain a much larger number of items than MovieLens 100K and Netflix[★]. Thanks to this wide variability of the large datasets, the proposed DL method is able to capture the nonlinear relations between the dataset items better than the traditional KNN algorithm. In this sense, the proposed model shows a higher scalability than the KNN machine learning method, providing better comparative results when the number of items in the dataset grows.

This important feature is particularly noticeable when we use the user quality measure instead of the item quality one: the former tests the fine-grained user/item correlations, whereas the later loses this detailed information toward a more global comparison. Additionally, in the results of the DGN method, the FilmTrust and the MyAnimeList[★] datasets show a slightly increasing trend when a small filter ($v=1$) is applied. This is a reasonable behavior, pointing out that in complex scenarios (high number of items, for instance) our DGN method finds a better result by discarding those items that provide lower correlations (as the traditional KNN algorithm).

Our last testing dataset is Netflix[★], as shown in Fig. 10. This is a particularly difficult case, since it is a random excerpt of the large original Netflix dataset. For this reason, some of the general patterns and correlations presented in the source are missing in the excerpt. We can observe this fact by comparing the average user quality values obtained on this dataset (y axis) with the previous experiments, both for the proposed method and the baselines. This is also a particularly challenging environment for the proposed DGN method due to the low number of items extracted from the original dataset. The results depicted in Fig. 10 show that the proposed method maintains its better performance when no filters are used ($v = 0$). However, the KNN algorithm manages to return better results when neighbors are filtered ($v = 1$). It tells us that, for small datasets, the proposed method outperforms KNN when no filters are applied.

Finally, Fig. 11 shows the evolution of the users quality measure on the Netflix[★] dataset when we vary the number of neurons in the dense NN hidden layer. Results are shown

Fig. 8 MovieLens 1 M (a) and MovieLens 100 K (b) results. Higher values represent better results. The parameter v is a filter threshold to select the most promising neighbors. The higher the parameter v , the lower the number of chosen neighbors; the value $v = 0$ means not filtered

for number of neurons ranging from 50 to 300. As expected, by increasing the model complexity, we get improved quality results. In this case, setting more than 300 neurons does not increase significantly the obtained quality. It is remarkable that, when a reduced number of neighbors is chosen (say, $v = 2$ or $v = 3$), the NN model complexity is less significant when determining the obtained quality. When neighborhoods are used to predict or recommend within a CF process, the KNN algorithm selects the K most promising neighbors. In these scenarios, the proposed DGN method can offer an improved efficiency, in terms of computation time, compared to the traditional similarity measures based KNN method.

4.3 Discussion

To test the feasibility of the proposed DGN method, a complete set of experiments have been run. Five popular datasets and five suitable baselines based on state-of-the-art similarity measures have been used to perform the tests. The results show that the proposed method clearly outperforms the baselines. To detail this conclusion, we added a filtering parameter v . In this direction, our solution is consistently better than the baselines when we vary v .

It is worth mentioning that, in the range $v = 0$ (no filter is applied), the results of our DGN method are particularly promising, since they provide sound and consistent better performance than all baselines. This is a particularly hard regime, since providing accurate predictions of neighborhoods for all datasets demands a very fine analysis of the global behavior of the dataset. Predicting only a few similar neighbors might be easy, namely the most similar films to ‘Star Wars: A New hope’ are the other Star Wars prequels and sequels. However, providing an accurate global distance to the whole dataset requires to extract subtle nonlinear latent factors of the user consumption habits. This is a task in which our proposed method is particularly successful due to its intrinsic nonlinear nature.

Additionally, our model is also more efficient than KNN in terms of time complexity. The usual KNN has to carry out $O(IU)$ calculations, where U is the number of users and I the number of items. In contrast, the propose DGN method (with a fixed number of iterations for the gradient localization) only requires $O(IH)$ operations, where H is the number of hidden neurons in the NN. In real-world datasets, U is considerably larger than I , and H is usually taken to be of the order of I . Therefore, in datasets with a large number of users as the commercial ones, the DGN method

Fig. 9 FilmTrust (a) and MyAnimeList* (b) results. Higher values represent better results. The parameter v is a filter threshold to select the most promising neighbors. The higher the parameter v , the lower the number of chosen neighbors; the value $v = 0$ means not filtered

Fig. 10 Netflix* results. Higher values represent better results. The parameter v is a filter threshold to select the most promising neighbors. The higher the parameter v , the lower the number of chosen neighbors; the value $v = 0$ means not filtered

Fig. 11 Netflix* quality results obtained by setting different MLP models complexity. The parameter v is a filter threshold to select the most promising neighbors. The higher the parameter v , the lower the number of chosen neighbors; the value $v = 0$ means not filtered

is also much more efficient in time than KNN. The key point is that most of the complexity is hidden under the process of training of the underlying NN, which can be

carried out beforehand. In this way, DGN should be the chosen solution for applications with strong real-time requirements, like recommendations in streaming platforms.

5 Conclusions

Obtaining neighbors from CF data is an important goal. Beyond the classical prediction and recommendation tasks, there are significant applications that make use of

neighborhood sets, both of items and users, such as recommendation explainability, prevention of shilling attacks, computing prediction reliabilities, visualization of relation trees, and pre-clustering processes. For this purpose, the machine learning-based KNN method is, de facto, the industry standard by seeking to the nearest according to a pre-fixed distance. Despite its prevalence, KNN presents remarkable drawbacks, such as the lack of scalability, the weakness in sparse scenarios or cold start situations, and its inability to capture the existing nonlinear relations in the RS information.

To overcome these flaws, in this paper, we have proposed a generative-based deep learning method whose operation is completely different to the KNN algorithm and does not use any preexisting similarity measure. The proposed DL method learns the nonlinear relations among CF items by means of a classification NN. In this deep model, each item in the dataset is associated with both an input and an output neuron. In this way, the aim of the NN is to predict, given a ‘profile of relevant likes’ of a user, codified as a binary vector, an item that he/she would like, represented through one-hot encoding.

Once the NN has been trained, a generative process is conducted to obtain each item neighborhood via GLT. In some sense, through this method, the NN is able to highlight the most relevant elements that activate a given item. These relevant elements should be thus understood as the ‘neighbors’ of the item, since they share some latent features that capture nonlinear relations between the profiles of likes of users. This is the key point of this paper, since the GLT allows us to create a novel similarity measure of intrinsic deep nature. Using these similarities, this solution gives rise to the DGN method proposed in this paper.

As a continuation of this work, this paper outlines two relevant research lines: First, it would be interesting to apply the obtained neighborhoods to the beyond-prediction aforementioned areas such as recommendation, explainability, or visualization. This would allow us to get more insight in the nonlinear relations of the RS datasets. Finally, a promising future research line would be to test the feasibility of the proposed method in similar problems outside CF, where the information sources are usually dense and the amount of data to be extracted is much richer.

Acknowledgements This work was partially supported by *Ministerio de Ciencia e Innovación* of Spain under the project PID2019-106493RB-I00 (DL-CEMG) and the *Comunidad de Madrid* under *Convenio Plurianual* with the Universidad Politécnica de Madrid in the actuation line of *Programa de Excelencia para el Profesorado Universitario*. The second-named author acknowledges the hospitality of the Departamento de Matemáticas at Universidad Autónoma de Madrid, where part of this work was developed.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Ahn HJ (2008) A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Inf Sci* 178(1):37–51
2. Alashkar T, Jiang S, Wang S, Fu Y (2017) Examples-rules guided deep neural network for makeup recommendation. In: *Proceedings of the AAAI conference on artificial intelligence*, vol 31
3. Bilgic M, Mooney RJ (2005) Explaining recommendations: satisfaction versus promotion. In: *Beyond personalization workshop, IUI*, vol 5, p 153
4. Bobadilla J, González-Prieto Á, Ortega F, Lara-Cabrera R (2020) Deep learning feature selection to unhide demographic recommender systems factors. *Neural Comput Appl* 33:1–18
5. Bobadilla J, Hernando A, Ortega F, Bernal J (2011) A framework for collaborative filtering recommender systems. *Expert Syst Appl* 38(12):14609–14623
6. Bobadilla J, Ortega F, Gutiérrez A, Alonso S (2020) Classification-based deep neural network architecture for collaborative filtering recommender systems. *IJIMAI* 6(1):68–77. <https://doi.org/10.9781/ijimai.2020.02.006>
7. Bobadilla J, Ortega F, Hernando A, Gutiérrez A (2013) Recommender systems survey. *Knowl Based Syst* 46:109–132
8. Bobadilla J, Serradilla F, Bernal J (2010) A new collaborative filtering metric that improves the behavior of recommender systems. *Knowl Based Syst* 23(6):520–528
9. Chollet F et al (2015) Keras. <https://keras.io>
10. Chollet F et al (2018) Deep learning with python, vol 361. Manning, New York
11. Dara S, Chowdary CR, Kumar C (2020) A survey on group recommender systems. *J Intell Inf Syst* 54(2):271–295
12. Dziugaite GK, Roy DM (2015) Neural network matrix factorization. [arXiv:1511.06443](https://arxiv.org/abs/1511.06443)
13. Ebesu T, Fang Y (2017) Neural citation network for context-aware citation recommendation. In: *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pp 1093–1096
14. Feng C, Liang J, Song P, Wang Z (2020) A fusion collaborative filtering method for sparse data in recommender systems. *Inf Sci* 521:365–379

15. Gatys LA, Ecker AS, Bethge M (2016) Image style transfer using convolutional neural networks. In: 2016 IEEE conference on computer vision and pattern recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp 2414–2423. IEEE Computer Society. <https://doi.org/10.1109/CVPR.2016.265>
16. Gomez-Urbe CA, Hunt N (2015) The Netflix recommender system: algorithms, business value, and innovation. *ACM Trans Manag Inf Syst (TMIS)* 6(4):1–19
17. Gunes I, Kaleli C, Bilge A, Polat H (2014) Shilling attacks against recommender systems: a comprehensive survey. *Artif Intell Rev* 42(4):767–799
18. Guo G, Zhang J, Yorke-Smith N (2013) A novel bayesian similarity measure for recommender systems. In: Proceedings of the 23rd international joint conference on artificial intelligence (IJCAI), pp 2619–2625
19. Guo H, Tang R, Ye Y, Li Z, He X (2017) DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv:1703.04247*
20. Harper FM, Konstan JA (2015) The movielens datasets: history and context. *ACM Trans Interact Intell Syst* 5(4):1–19
21. He X, Liao L, Zhang H, Nie L, Hu X, Chua TS (2017) Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web, pp 173–182
22. Herlocker JL, Konstan JA, Terveen LG, Riedl JT (2004) Evaluating collaborative filtering recommender systems. *ACM Trans Inf Syst* 22(1):5–53
23. Logesh R, Subramaniaswamy V, Malathi D, Sivaramakrishnan N, Vijayakumar V (2020) Enhancing recommendation stability of collaborative filtering recommender system through bio-inspired clustering ensemble method. *Neural Comput Appl* 32(7):2141–2164
24. Logesh R, Subramaniaswamy V, Vijayakumar V, Gao XZ, Wang GG (2020) Hybrid bio-inspired user clustering for the generation of diversified recommendations. *Neural Comput Appl* 32(7):2487–2506
25. Misztal-Radecka J, Indurkha B (2020) Getting to know your neighbors (KYN). Explaining item similarity in nearest neighbors collaborative filtering recommendations. In: Adjunct publication of the 28th ACM conference on user modeling, adaptation and personalization, pp 59–64
26. Mnih A, Salakhutdinov RR (2008) Probabilistic matrix factorization. In: Advances in neural information processing systems, pp 1257–1264
27. Moradi P, Ahmadian S (2015) A reliability-based recommendation method to improve trust-aware recommender systems. *Expert Syst Appl* 42(21):7386–7398
28. MyAnimeList.net: MyAnimeList dataset. <https://www.kaggle.com/azathoth42/myanimelist> (2020). Accessed 18 May 2020
29. Ortega F, Lara-Cabrera R, González-Prieto Á, Bobadilla J (2021) Providing reliability in recommender systems through Bernoulli matrix factorization. *Inf Sci* 553:110–128
30. Ortega F, Zhu B, Bobadilla J, Hernando A (2018) CF4J: Collaborative filtering for Java. *Knowl Based Syst* 152:94–99
31. Pádúa FL, Lacerda A, Machado AC, Dalip DH et al (2019) Multimodal data fusion framework based on autoencoders for top-n recommender systems. *Appl Intell* 49(9):3267–3282
32. Portugal I, Alencar P, Cowan D (2018) The use of machine learning algorithms in recommender systems: a systematic review. *Expert Syst Appl* 97:205–227
33. Raza S, Ding C (2019) Progress in context-aware recommender systems—an overview. *Comput Sci Rev* 31:84–97
34. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D (2017) Grad-CAM: visual explanations from deep networks via gradient-based localization. In: 2017 IEEE international conference on computer vision (ICCV) pp 618–626. <https://doi.org/10.1109/ICCV.2017.74>
35. Tahmasebi H, Ravanmehr R, Mohamadrezai R (2020) Social movie recommender system based on deep autoencoder network using twitter data. *Neural Comput Appl* 33:1–17
36. Tang J, Wang K (2018) Personalized top-n sequential recommendation via convolutional sequence embedding. In: Proceedings of the eleventh ACM international conference on web search and data mining, pp 565–573
37. Tuan TX, Phuong TM (2017) 3D convolutional networks for session-based recommendation with content features. In: Proceedings of the eleventh ACM conference on recommender systems, pp 138–146
38. Turk AM, Bilge A (2019) Robustness analysis of multi-criteria collaborative filtering algorithms against shilling attacks. *Expert Syst Appl* 115:386–402
39. Wang D, Liang Y, Xu D, Feng X, Guan R (2018) A content-based recommender system for computer science publications. *Knowl Based Syst* 157:1–9
40. Yang X, Guo Y, Liu Y, Steck H (2014) A survey of collaborative filtering based social recommender systems. *Comput Commun* 41:1–10
41. Yi B, Shen X, Zhang Z, Shu J, Liu H: Expanded autoencoder recommendation framework and its application in movie recommendation. In: 2016 10th international conference on software, knowledge, information management and applications (SKIMA). IEEE, pp 298–303 (2016)
42. Zhang S, Yao L, Sun A, Tay Y (2019) Deep learning based recommender system: a survey and new perspectives. *ACM Comput Surv (CSUR)* 52(1):1–38
43. Zhang S, Yao L, Xu X (2017) Autosvd++ an efficient hybrid collaborative filtering model via contractive auto-encoders. In: Proceedings of the 40th international ACM SIGIR conference on Research and development in information retrieval, pp 957–960
44. Zho, B, Khosla A, Lapedriza A, Oliva A, Torralba A (2016) Learning deep features for discriminative localization. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 2921–2929

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.