Soft Computing
https://doi.org/10.1007/s00500-020-05162-6

METHODOLOGIES AND APPLICATION

An unsupervised detection method for shilling attacks based on deep learning and community detection

Yaojun Hao[1,2,3] · Fuzhi Zhang[1,2]

Abstract
In the detection methods for shilling attacks, the supervised methods require labeled samples to train the classifiers. Due to lack of the labeled sample profiles in real scenarios, the applicability of supervised detection method is restricted. For unsupervised methods, the prior knowledge is often required to guarantee the detection accuracy. To break the traditional limitations, we present an unsupervised method to detect various shilling profiles from reconstructed user–user graph based on deep learning and community detection. Firstly, we construct the user–user graph, whose edge weight is calculated by the similarity of user's behaviors. Secondly, the stacked denoising autoencoders are used to extract the robust graph features at different scales with different corruption rates. Based on the features at different scales, we generate multiple clustering results and reconstruct the user–user graph by evidence accumulation method. Thirdly, the community detection is carried out by using the persistence optimization algorithm. Extensive experiments on two datasets illustrate that our proposed method has better performance than some baseline detectors for detecting the simulated attacks and actual attacks.

Keywords Shilling attack · Unsupervised detection · Deep learning · Community detection

1 Introduction

Recommender systems based on collaborative filtering technology are extensively deployed on e-commerce sites to alleviate the information-overloading problems. However, for commercial advantage, attackers may register accounts and generate fake profiles in order to manipulate recommendation results. The fake profiles are usually referred to as attack profiles or shilling profiles (Gunes et al. 2014). The behaviors of injecting fake profiles are often called as shilling attacks, which seriously threaten the recommendation quality.

To ensure the recommendation quality, many researchers have taken efforts to study the detection methods for shilling attacks. In the existing detection methods, the supervised methods (Chirita et al. 2005; Burke et al. 2006; Williams et al. 2007; Yang et al. 2016b; Zhou et al. 2016; Dou et al. 2017; Tang and Tang 2011; Zhang et al. 2006; Zhang and Zhou 2014; Li et al. 2015) require labeled sample profiles to train the classifiers. Due to lack of labeled sample profiles in real scenarios, the applicability of such methods is restricted. The unsupervised methods (Bhaumik et al. 2011; Hurley et al. 2009; Mehta et al. 2007; Mehta and Nejdl 2009; Yang et al. 2016a; Gunes and Polat 2015; Zhang et al. 2015) usually need prior knowledge (e.g., the attack size, the candidate set of shilling profiles) to ensure the performance of detection (Zhang et al. 2018). Unfortunately, we hardly obtain such prior knowledge in practical applications.

To overcome the above limitations, we propose an unsupervised detector based on deep learning and community detection. Since shilling profiles are deliberately injected to manipulate the recommendation results by rating some target items, they are closely connected to each other and can be seen as the communities. Therefore, we construct the user–user graph according to their rating behaviors and discover the community-like structure by community detection

Communicated by V. Loia.

✉ Fuzhi Zhang
xjzfz@ysu.edu.cn

1 School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China

2 The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China

3 Department of Computer Science and Technology, Xinzhou Normal University, Xinzhou 034000, China

Published online: 14 July 2020

Table 1 The ordinary attack models

Attack model	I_S		I_F		i_t
	Items	Rating	Items	Rating	Rating
Random	Unused		Random selection	Mean of all items	$r_{\max}/r_{\min}$
Average	Unused		Random selection	Mean of each item	$r_{\max}/r_{\min}$
AoP	Unused		Top $x\%$ of most popular items	Mean of each item	$r_{\max}/r_{\min}$
User random shifting	Unused		Random selection	Mean of all items with random noise	$r_{\max}/r_{\min}$
User average shifting	Unused		Random selection	Mean of each item with random noise	$r_{\max}/r_{\min}$
Target random shifting	Unused		Random selection	Mean of all items	$(r_{\max}-1)/(r_{\min}+1)$
Target average shifting	Unused		Random selection	Mean of each item	$(r_{\max}-1)/(r_{\min}+1)$
Power item	Power item	Ratings fitting normal distribution with item mean and standard deviation			$r_{\max}/r_{\min}$
Power user	Items and ratings are sampled from those of the power users				$r_{\max}/r_{\min}$

method. For community detection, the classical modularity optimization and normalized cut (n-cut) methods can be cast as graph clustering (Cao et al. 2018). These methods can capture topology-based features, but are often limited to obtain the important information about the structure of the communities with complex connections in user–user graph (Cao et al. 2018). For the initial user–user graph, these community detection methods hardly provide any insight as to whether or not a sub-graph actually possesses community structure by the links of external behaviors (Chakraborty et al. 2014). Therefore, based on the initial user–user graph, we use the stacked denoising autoencoders (SDAEs) to extract deep-level and robust graph features from different scales. According to the nonlinear features extracted by SDAEs, we generate the base partitions by k-means algorithm and then reconstruct user–user graph by the evidence accumulation method. Based on the permanence optimization, the user profiles are effectively grouped.

The main contributions are summarized as follows:

(1) We propose an edge weight metric from the similarity of users' rating behaviors, which are calculated from the related information of the rating behaviors (e.g., co-rated items, filler ratio difference, and ratings' deviation). Moreover, we reconstruct the user–user graph by the evidence accumulation method based on deep-level and robust features.
(2) We present a method to extract the features of user–user graph by SDAEs from different scales.
(3) We conduct extensive experiments and focus on the superiority of the proposed method in comparison with the alternatives.

2 Background and related work

2.1 Shilling attack models

The items in a shilling profile can be divided into four sets, in which I_S and I_F denote the set of selected items and filler items, respectively; i_t denotes the selected target item; $I_\emptyset$ represents the unrated items (Gunes et al. 2014; Chirita et al. 2005; Burke et al. 2006). The ordinary attack models can be summarized in Table 1.

In particular, user random shifting attack and user average shifting attack are generated by user shifting strategy based on the random attack and average attack, respectively (Williams et al. 2007). Similarly, target random shifting attack and target average shifting attack are generated by target shifting strategy (Williams et al. 2007). To facilitate the representation, with equal proportion, we mix these four types attacks into a set and define them as shifting attack. More detailed information about various attack models can be found in Gunes et al. (2014), Chirita et al. (2005), Burke et al. (2006), Seminario and Wilson (2014), and Wilson and Seminario (2013).

2.2 Sparse denoising autoencoder

Autoencoder (AE) can be used to extract features from original input, whose structure usually includes the input layer, encoding layer, and decoding layer (Hinton et al. 2012; Bengio et al. 2013). Let $\mathbf{x}$ be the input to the AE, and obtain the encoding $\mathbf{h}$ as the hidden representation by the following equation:

$$\mathbf{h} = s_f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{p}) \tag{1}$$

where s_f, $\mathbf{W}^{(1)}$, and $\mathbf{p}$ denote the activation function, the weight matrix, and the bias vector of the encoding layer, respectively.

Similarly, the vector of decoding layer is used to reconstruct the input vector from the hidden representation by the following equation:

$$\hat{\mathbf{x}} = s_g(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{q}) \tag{2}$$

where $\hat{\mathbf{x}}$, s_g, $\mathbf{W}^{(2)}$, $\mathbf{q}$ denote the vector of decoding layer, the activation function, the weight matrix, and the bias vector of decoding layer, respectively.

Therefore, the feature extraction in AE can be described as a process for minimizing the error between input and output. Denoting parameters in an AE as $\theta = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{p}, \mathbf{q}\}$, the objective function can be described as:

$$J_{(\mathrm{AE})}(\theta) = \sum L(\mathbf{x}, s_g(s_f(\mathbf{x}))) \tag{3}$$

where $L()$ is a loss function in the input space.

When AE is constrained by condition requiring that most of the neurons are zero, the AE becomes a sparse autoencoder (SAE). In this paper, KL divergence is used to add sparsity restrictions for AE. Therefore, the reconstruction error of Eq. (3) can be expressed as:

$$J_{(\mathrm{SAE})}(\theta) = \sum L(\mathbf{x}, s_g(s_f(\mathbf{x}))) + \beta \sum_j \mathrm{KL}(\rho \| \hat{\rho}) \tag{4}$$

where β denotes the penalty factor; ρ is a sparse factor; $\hat{\rho}$ represents the average activation of hidden units; $\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$ is the KL divergence.

The denoising autoencoder (DAE) can be optimized to reconstruct input data from partial and random corruption (Vincent et al. 2008). When the original data $\mathbf{x}$ is corrupted and denoted as $\hat{\mathbf{x}}$, the reconstruction error of DAE can be expressed as:

$$J_{(\mathrm{DAE})}(\theta) = \sum L(\mathbf{x}, s_g(s_f(\tilde{\mathbf{x}}))). \tag{5}$$

Under the usual blank-out corruption noise, the input vector $\mathbf{x}$ is set to zero with probability $\alpha (\alpha \geq 0)$.

With sparse restriction, based on Eq. (3), the reconstruction error of the sparse denoising autoencoder (SDAE) can be expressed as:

$$J_{(\mathrm{SDAE})}(\theta) = \sum L(\mathbf{x}, s_g(s_f(\tilde{\mathbf{x}}))) + \beta \sum_j \mathrm{KL}(\rho \| \hat{\rho}_j). \tag{6}$$

The deep architecture for SDAEs is built by stacking SDAE to form a deep network.

2.3 Community detection

In general, community detection aims to find a network partition (Girvan and Newman 2002). Given a graph $G(V, E, \mathrm{SF})$, where V denotes the set of nodes, E denotes the set of edges, and SF is the function which determines the edge weight. The network partition can be described as $\mathrm{CS} = \{\mathrm{CS}_1, \mathrm{CS}_2, \ldots, \mathrm{CS}_z\}$, in which $V = \bigcup_{i=1}^{z} \mathrm{CS}_i$. When $\mathrm{CS}_i \cap \mathrm{CS}_j = \emptyset$, a partition of the nodes is acquired, which are called non-overlapping communities; otherwise, nodes are allowed to belong to more than one community, then overlapping communities are obtained.

Since we focus on the detection of shilling profiles and do not consider the overlapping community structure, we restrict our discussion to some works of non-overlapping community.

In the past decade, various methods have been proposed depending upon the modularity and modularity-based definitions. However, the modularity optimization methods may cause the resolution limitation problem or degeneracy problem (Chakraborty et al. 2017). The researchers have proposed several modifications of modularity to address the limits of modularity, such as modularity intensity (Sun et al. 2013), *Z*-Modularity (Miyauchi and Kawase 2016), permanence (Chakraborty et al. 2014), and so on. Besides the above methods, many different community detection methods have been proposed, including information theoretic approaches (Rosvall and Bergstrom 2008), label propagation (Raghavan et al. 2007; Xie et al. 2011), matrix blocking method (Chen and Saad 2012), and so on. Detailed surveys for above methods can be found in Chakraborty et al. (2017), Malliaros and Vazirgiannis (2013), and Pizzuti and Rombo (2014).

2.4 Related work

To defend recommendation systems, a number of supervised and unsupervised detection methods have been presented. In supervised methods, shilling profiles are usually separated by classification algorithms based on the extracted features. There are three major categories of detection features: ratings-based features, timestamp-based features, and item popularity-based features. Williams et al. (2007) used several ratings-based features and trained kNN, C4.5, and

SVM classifiers as detectors. These methods can effectively detect random, average, and bandwagon attacks. However, they are less effective under unknown types of attacks. Yang et al. (2016b) combined their features and the existed other features together, then used Re-scale AdaBoost as classification algorithm. Zhou et al. (2016) put forward SVM-TIA method, which firstly extracted several features and used SVM classifier to generate a candidate set of attackers. From this candidate set, the genuine ones are filtered out based on target item analysis. However, it cannot effectively improve the recall under low attack size. In Dou et al. (2017), from the view of shared latent factors, the user–item matrix and user–user matrix were decomposed. Hence, these shared latent factors can be seen as the detection features. However, it suffers from poor performance with low attack and filler sizes under standard attacks. In Tang and Tang (2011), according to the rating timestamps, span, frequency, and mount factors are calculated to detect the target items. Zhang et al. (2006) utilized the mean and entropy of time window to detect fake ratings. However, the long-term decentralized attacks are likely to evade the timestamp-based detectors. From the item popularity, Zhang and Zhou (2014) computed the detection features based on Hilbert–Huang transform method. Li et al. (2015) calculated the detection features based on item popularity distributions and used an improved ID3 decision tree as the classifier. Unfortunately, in these item popularity-based methods, some detection features are limited to the specific types of attacks and cannot be applicable to various attacks.

In unsupervised methods, the clustering algorithm was used to detect standard attacks with ratings-based features (Bhaumik et al. 2011). For AoP attack, the statistical detector based on Neyman–Pearson statistical method is presented in Hurley et al. (2009). Under various attacks, these two detection methods often suffer from poor precision. From the view of principal component analysis (PCA), a variable selection method was presented to detect standard attacks in Mehta et al. (2007), Mehta and Nejdl (2009). However, this method needs the prior knowledge of attack size and fails to detect AoP attack. Yang et al. (2016a) constructed the user–user unweighted graph according to users' co-rated items, and then analyzed users similarity to determine the attack profiles. In Gunes and Polat (2015), a hierarchical clustering algorithm was used to detect shilling profiles, which performed well under large size attacks but suffered from poor performance under small size attacks. In Zhang et al. (2015), the label propagation method was used to detect shilling attacks. However, it has to determine some attackers as the candidate set.

In this paper, the edge weight of user–user graph is calculated by the similarity of users' rating behaviors, which is different from the unweighted user–user graph in Yang et al. (2016a). Different from the unsupervised detectors in Mehta et al. (2007), Mehta and Nejdl (2009), and Zhang et al. (2015), there is no need for our method to acquire the information about the attacker size or candidate set of attackers. In our method, SDAEs is used to learn feature representations of user–user graph from different scales and the ensemble approach is utilized to reconstruct the user–user graph. Based on permanence optimization, shilling profiles are effectively grouped.

3 The proposed method

Figure 1 illustrates the framework of our unsupervised detection method, which can be divided into three stages: construction of user–user graph (stage 1), generation of clusters with SDAEs (stage 2), and detection (stage 3). At the first stage, we construct the user–user graph and estimate the edge weight by the similarity of rating behaviors. At the second stage, we use SDAEs to extract the features of user–user graph with different corruption rates, and then utilize k-means repeatedly with random cluster number for each SDAEs representation to generate base clusters. At the third stage, we integrate the different clustering results to reconstruct the user–user graph and use community detection method to identify the attackers.

3.1 Construction of user–user graph

Since the attackers inject a large number of shilling profiles to demote or promote a target item, the shilling profiles are naturally connected with each other through the co-rated items. Accordingly, we construct the user–user graph, in which the connections exist in the users with the co-rated items. However, some genuine profiles may also have the co-rated items with shilling profiles. Therefore, the simple connections are too shallow to reflect the differences between shilling and genuine profiles. In fact, edge weight plays a very important role in measuring the connection tightness between users. Thus, we use the weighted edges to further reflect the connection differences.

In recommendation systems, Pearson similarity is often used to measure the correlation between user ratings. However, in real scenarios, Pearson similarity is likely to be affected by the sparsity of user ratings. For example, in the Amazon food dataset,[1] when excluding the users with less than 3 ratings, more than 73.1% users have one and only one co-rated item with other users. Under this condition, Pearson coefficient hardly effectively calculates the similarity between two users with only one co-rated item.

In Yang et al. (2016a), when two users have more co-rated items than the threshold, they considered there was a link between two users. And then the unweighted user–user

[1] http://snap.stanford.edu/data/web-FineFoods.html.

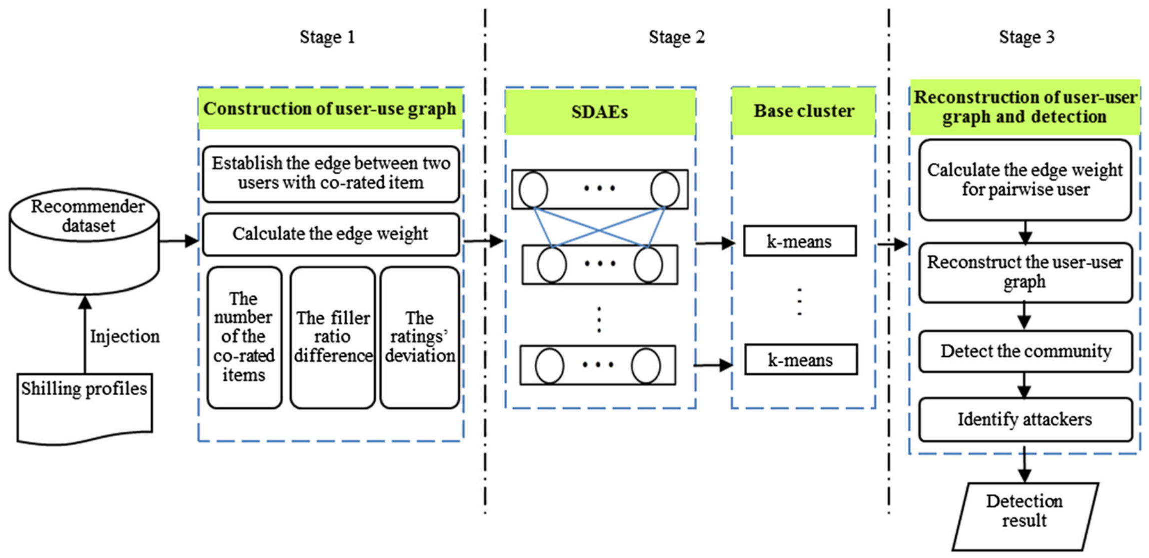

Stage 1
Stage 2
Stage 3
Recommender dataset
Injection
Shilling profiles
Construction of user-use graph
Establish the edge between two users with co-rated item
Calculate the edge weight
The number of the co-rated items
The filler ratio difference
The ratings' deviation
SDAEs
Base cluster
k-means
k-means
Reconstruction of user-user graph and detection
Calculate the edge weight for pairwise user
Reconstruct the user-user graph
Detect the community
Identify attackers
Detection result

Fig. 1 Framework of the proposed detection method

graph was constructed. The basic hypothesis in Yang et al. (2016a) is that there are more co-rated items in the attackers than those in genuine users (Yang et al. 2016a). However, the threshold is difficult to determine in practice.

To overcome the above limitations, we focus on the weighted edges to analyze the tightness between users. As shown in Table 1, the same attack model has the similar way to select filler items and has the similar way to give ratings for items. Therefore, from the attacker view, we comprehensively analyze the number of co-rated items, filler ratio difference, and ratings deviation to measure the similarity. And then the edge weight between users is estimated with the similarity.

Definition 1 Novelty of Item (NI). The NI is used to describe the extent of dissimilarity for two items (Zhang and Chen 2016), which can be calculated as:

$$\mathrm{NI}_i = \frac{1}{|U_i|} \sum_{u \in U_i, r_{u,i} \neq 0} \mathrm{NI}_{u,i} \tag{7}$$

where $r_{u,i}$ represents the rating value of user u to item i; U_i represents the set of users who give ratings to item i; $|\cdot|$represents the set cardinality; and $\mathrm{NI}_{u,i}$ represents the NI of item i for user u, which can be calculated as Castells et al. (2011):

$$\mathrm{NI}_{u,i} = \frac{1}{|I_u|} \sum_{j \in I_u, r_{u,j} \neq 0} (1 - \mathrm{sim}(i, j)) \tag{8}$$

where I_u represents the set of items rated by user u, $\mathrm{sim}(i, j)$ can be calculated by the cosine similarity.

In general, the item with greater novelty is less popular. Therefore, according to the Zipf's law, when items are sorted by the NI, the top 20% items are chosen as popular item set (PIS) and the others are chosen as novel item set (NIS).

Definition 2 Filler ratio difference. The filler ratio difference is used to describe the users' different degrees for the ratings in the PIS and NIS. The filler ratio difference between users u and v, $\Delta F(u, v)$, is calculated as:

$$\Delta F(u, v) = \left| \frac{|\mathrm{PIS}_u|}{|\mathrm{PIS}|} - \frac{|\mathrm{PIS}_v|}{|\mathrm{PIS}|} \right| + \left| \frac{|\mathrm{NIS}_u|}{|\mathrm{NIS}|} - \frac{|\mathrm{NIS}_v|}{|\mathrm{NIS}|} \right| \tag{9}$$

where PIS_u and PIS_v denote the subset of PIS rated by user u and user v, respectively. NIS_u and NIS_v denote the subset of NIS rated by user u and user v, respectively.

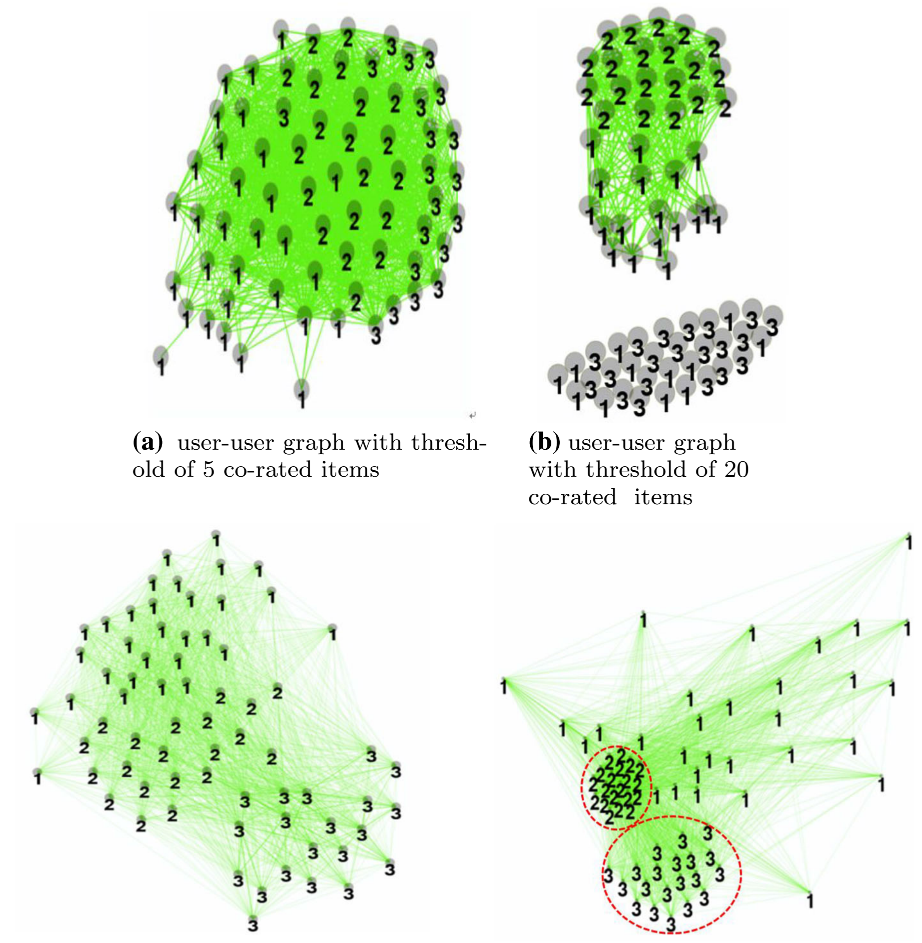
Definition 3 Similarity of users. The similarity between two users refers to the similar degree of rating behaviors for the users. The similarity between users u and v can be calculated as:

$$S(u, v) = \sum_{i=1}^{|I_u \bigcap I_v|} \frac{e^{-\lambda \cdot \Delta F(u,v)}}{|U_i|} \cdot \delta(r_{u,i}, r_{v,i}) \tag{10}$$

where U_i represents the set of users who rate item i. In Eq. (10), $\Delta F(u, v)$ is between 0 and 1, and $e^{-\Delta F(u,v)}$ is between e^{-1} and 1. To adjust the effect degree of filler ratio difference, λ is provided as a parameter to impact $\Delta F(u, v)$. The default value of λ is 1. $\delta(r_{u,i}, r_{v,i})$ is a symbolic function, which is formulated as:

$$\delta(r_{u,i}, r_{v,i}) = \begin{cases} 1, & \left| r_{u,i} - r_{v,i} \right| < \tau \\ 0, & \left| r_{u,i} - r_{v,i} \right| \geq \tau \end{cases} \tag{11}$$

Fig. 2 The user–user graph constructed by different methods



(a) user-user graph with threshold of 5 co-rated items

(b) user-user graph with threshold of 20 co-rated items

(c) user-user graph based on Pearson similarity

(d) user-user graph based on our proposed method

where τ is the threshold of ratings deviation for the same item between two users, that is, when the deviation exceeds τ, we consider that two ratings represent opposite preferences. In five-point rating scale, we set $\tau = 3$.

In Eq. (10), focusing on the complicated behaviors, the connections between attackers are closer than those between real users. Figure 2 shows the user–user graph constructed with different edge weights, in which 30 users with the label 1 are genuine users randomly selected from the Netflix dataset, 20 users with the label 2 are AoP attackers with filler size 5%, and 20 users with the label 3 are average attackers with filler size 3%. In Fig. 2a and b, the user–user graph is constructed according to Yang et al. (2016a), in which the thresholds are set to 5 and 20, respectively. In Fig. 2c, the edge weights between users are calculated by Pearson similarity. In Fig. 2d, the edge weights are calculated by the proposed method.

As shown in Fig. 2d, the connections between the attackers are relatively closer than those of real users, and the attackers are inclined to cluster together. In Fig. 2a, the connections between attackers are confused by those of genuine users. Due to the large threshold in Fig. 2b, although some attackers can be clustered together, there are more isolated nodes including genuine users and attackers. In Fig. 2c, the different connections between attackers and real users cannot be fully reflected. Therefore, the connections between attackers in Fig. 2d are tighter than those in other figures.

Based on the above analysis, let DR denote user ratings. $\mathbf{M}$ is the adjacency matrix of weighted user–user graph. Function $indexu(cur_i)$ returns the user ID corresponding to cursor cur_i in the set of users. The $rate(DR, temp_i, i)$ returns the ratings for the item i with the user *tempi*. The generation of adjacency matrix for user–user graph can be described in Algorithm 1.

There are two parts in Algorithm 1. The first part (lines 1–9) counts the number of user ratings based on different popular item sets. The second part (lines 10–25) calculates the similarity between users. In particular, the items are selected when the number of ratings is greater than 1 (line 11). For each of these items, the similarity between users is calculated according to Eq. (10) (lines 12–17), and then the user similarity reflected by different items is accumulated (line 18).

Algorithm 1 Construction the weighted user–user graph.

```
Input: $DR$;
Output: M;
1: if (PIS and NIS have not be updated) then
2:     Calculate $NI_i$ according to Eq. (7)
3:     Generate the items sequence $SQ$ by $NI_i$ on ascending order
4:     $PIS = \left\{i_{sq} | sq = 1 \ldots \lceil |I| \times 0.2 \rceil\right\}$
5:     $NIS = \left\{i_{sq} | sq = \lceil |I| \times 0.2 \rceil + 1 \ldots |I|\right\}$
6: end if
7: for each $u \in U$ do
8:     Calculate $|U_i|$ , $|PIS_u|$,and $|NIS_u|$
9: end for
10: for each $i \in I$ do
11:     if $|U_i| > 1$ then
12:         for $curi = 1$ to $|U_i| - 1$ do
13:             for $curj = curi + 1$ to $|U_i|$ do
14:                 tempi=indexu(curi)
15:                 tempj=indexu(curj)
16:                 if abs(rate(DR,tempi,i)-rate(DR,tempj,i))$< 3$ then
17:                     Calculate the filler ratio difference $\Delta F$ according to
                        Eq. (9)
18:                     M(tempi,tempj)=M(tempi,tempj)+$\frac{e^{-\lambda \cdot \Delta F}}{|U_i|}$
19:                     M(tempj,tempi)=M(tempi,tempj)
20:                 end if
21:             end for
22:         end for
23:     end if
24: end for
25: return M
```

After all the items are traversed, the adjacency matrix **M** is finally returned (line 25).

In the first part, the time complexity of calculating the item novelty and dividing the popular item set (lines 2–5) is at most $O(|U| \times |I|^3 + |I|^2)$. When the number of each item's ratings is counted in advance and their mean can be expected to spi, the time complexity of this process can be denoted as $O(|U| \times spi^3 + |I| \times \log_2 |I|)$. In fact, the ratings in recommendation systems are very sparse and spi is far less than $|I|$. Moreover, this process can be executed off line. The time complexity for calculating the number of users' ratings in PIS and NIS (line 8) is at most $O(|U| \times |I|)$. Given the condensed representation of rating records with sparsity spk, the time complexity for calculating the number of users' ratings in PIS and NIS (line 8) can be denoted as $O(|U| \times |I| \times spk)$.

In the second part, the time complexity to compute users similarity from each item is at most $O(|U|^2)$. After completing the outer loop, the time complexity of this part is at most $O(|I| \times |U|^2)$.

In summary, the time complexity of this algorithm is at most $O(|U| \times |I|^3 + |I|^2 + |U| \times |I| + |I| \times |U|^2)$, and can be decreased to $O(|U| \times spi^3 + |I| \times \log_2 |I| + |U| \times |I| \times spk + |I| \times |U|^2)$ In recommendation system, since the user–item rating matrix is very sparse, $spi \ll |I|$, $spk \ll 1$. Thus, the time complexity of Algorithm 1 can be described as $O(|U| \times spi^3 + |I| \times |U|^2)$.

3.2 Generation of different clustering results

In the constructed user–user graph, based on the rating behaviors, attackers are inclined to cluster together. Since the number of clusters is unknown, we can use community detection method to detect the attackers. In practice, the attackers always attempt to become the genuine users' neighbors by providing fake ratings, so the relationships between attackers and genuine users are often confused. To discover the deep and stable contacts, we randomly drop out the edges with certain probability (i.e., the adjacency matrix is corrupted with blank-out noise), and extract the robust features by SDAEs. Then the k-means algorithm is utilized to cluster users based on the features extracted from user–user graph.

To discover the deep and stable contacts in user–user graph, we extract the robust features by SDAEs and impose different corruption rates during the learning process. In the structure of SDAEs, the nodes number in the first layer is set to the number of users, and the nodes number in the other layers is set to $\{2048, 512, 128, 32\}$, respectively. Therefore, each column in adjacency matrix of user–user graph can be seen as an input vector for SDAEs. Moreover, in SDAEs, the input is trained to reconstruct the data from the corrupted version. Under the usual blank-out corruption noise, the corrupted version of input vector can be seen as a representation that drops out the edges between user and user with certain probability. During the learning process, the noise has significant effects on the final representation. In general, at low noise levels, the finer features of the data can be extracted by the network (Chen et al. 2014). However, at high noise levels, global, coarse-grained features of the data can be extracted by the network (Chen et al. 2014). Therefore, we use different levels of noise to corrupt the adjacent matrix **M** of user–user graph G, and can obtain different representations $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(L)}$ by SDAEs under L corruption rates.

Based on the representation of user–user graph, we use k-means algorithm to cluster complex user nodes. As a simple and efficient clustering algorithm, k-means does not need complex parameter setting. However, k-means is not a stable method. Thus, cluster ensemble method is used to combine multiple base partitions into a robust, stable, and accurate partition (Zhong et al. 2015). A suitable ensemble of the base partitions can discover more hidden information of the cluster structure and lead to improvement of the quality of the final clustering (Zhong et al. 2015). When base partitions are generated by k-means with fixed or random number of clusters, evidence accumulations is an effective framework which can convert the base partitions into a co-association matrix (Zhong et al. 2015; Fred and Jain 2005). For the representation under a certain corruption rate in SDAEs, we repeatedly select random cluster number to generate base partitions or clusters.

As far as the number of clusters k is concerned, when the value of k is too large, the clusters will have good homogeneity (Zhong et al. 2015) and the consistency of each member in the cluster is better, but some global structural information is likely to be ignored. Also, large values may produce an over-fragmentation of the data (in the limit, each pattern forming its own cluster) (Fred and Jain 2005). If the k value is too small, the composition of the cluster will become complex and the cluster may be mixed with members who should not be clustered into the same group (Fred and Jain 2005).

In this paper, due to the lack of priori knowledge for community number, the k-means with random cluster number k is used to generate the base partitions, where the class number k is between 20 and $\sqrt{m}$ (m represents the number of users). Therefore, for each representation of user–user graph, k-means is repeated multiple times with random k to generate base partitions or clusterers.

3.3 Reconstruction of user–user graph and detection

In Fred and Jain (2005), the evidence accumulation method produces a mapping of the clustering results into a new similarity measure between patterns. In a base clustering result, the relationship between users u, v is reflected in that they are in the same class or not in the same class, that is, $RS_{u,v}$, the relationship between them can be expressed as:

$$RS_{u,v} = \begin{cases} 1, & C_u - C_v = 0 \\ 0, & C_u - C_v \neq 0 \end{cases} \tag{12}$$

where C_u and C_v represent the class labels of users u and v, respectively.

In order to integrate t base clustering results over n representations, the ensemble relationship between users, $ERS_{u,v}$ can be calculated as:

$$ERS_{u,v} = \frac{\sum_1^{nt} RS_{u,v}}{nt} \tag{13}$$

where n represents the number of representations by SDAEs, t represents the repetitions number of k-means algorithm under each representations.

Based on the integrated user relationship *ERS*, the user–user graph can be reconstructed as new graph, denoted as RG. In RG, we use the persistence optimization algorithm to detect the community. In Chakraborty et al. (2014), the traditional modularity was improved with persistence, which alleviated the resolution limitation and degradation problem in community detection. The persistence measures the degree of internal connections in current community and the degree of attraction to neighbors in other communities. Moreover, it paid more attention to ring-like communities. The persistence

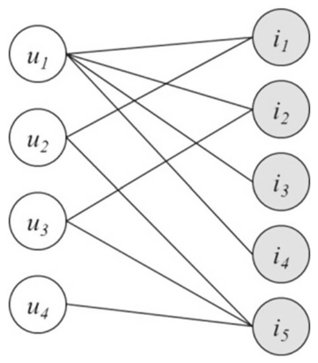


Fig. 3 User–item bipartite graph

of a vertex v can be calculated as Chakraborty et al. (2014):

$$Perm(v) = \left[\frac{I(v)}{E_{\max}(v)} \cdot \frac{1}{d(v)}\right] - [1 - c_{in}(v)] \tag{14}$$

where $I(v)$ denotes the number of internal connections of vertex v; $E_{\max}(v)$ denotes the maximum number of connections of vertex v to a single external community; $d(v)$ denotes the degree of vertex v; $c_{in}(v)$ means the internal clustering factor of vertex v. With the defined permanence, the heuristic optimization algorithm can be used to detect the community (Chakraborty et al. 2014).

In user–user graph, since the internal connections of shilling profiles are tighter than those of genuine ones, they can be fully detected as some communities. In community detection results, the communities of shilling profiles need to be further determined by their attributes. Because the shilling profiles are usually injected to promote (or demote) the target items, there are dense connections to the target items in user–item bipartite graph of attackers communities. By contrast, in genuine profiles communities, most of connections are random and sparse in user–item bipartite graph. Therefore, the maximum filling rate of the item in the community, mf, is extracted as an indicator for the community of shilling profiles, which can be calculated as:

$$mf_{commu} = \frac{\max(dbi(i))}{|commu|} \tag{15}$$

where $dbi(i)$ is the degree of item i in the community $commu$; $|commu|$ denotes the number of users in the community.

The mf indicator is demonstrated by a toy example in Fig. 3.

In Fig. 3, u_1, u_2, u_3, u_4 are user profiles in the community, and i_1, i_2, i_3, i_4, i_5 are the items. Since i_5 has maximum number of links, the mf of the community is 3/4 = 0.75.

Based on the above analysis, let **M** represent the adjacency matrix of the user–user graph, *SDAEopts* denote the parameters of the SDAEs, n represent the number of corruption versions, t represent the repetitions number of k-means algorithm under each representations, **C** represent the label matrix for users according to Eq. (12), **RGM** represent the adjacency matrix for the reconstructed user–user graph according to Eq. (13), mf represent the maximum filling rate of the item in the community according to Eq. (15). Thus, the shilling attack detection can be detailed in Algorithm 2.

Algorithm 2 Shilling attack detection.

```
Input: $\mathbf{M}$, $SDAEopts, t, n$;
Output: $fuser$;
1: $fuser = \emptyset$
2: $nuser = size(\mathbf{M})$
3: $DN = SDAESetup(SDAEopts)$
4: for $i = 1$ to $n$ do
5:     $M_i = Corrupting(\mathbf{M}, noi(i))$
6:     $SDAE = SDAETrain(DN, M_i, SDAEopts)$
7:     $tot = 1$
8:     $H(i) = getHfea(SDAE)$
9:     for $j = 1$ to $t$ do
10:        $k = random(20, \sqrt{nuser})$
11:        $C(tot, :) = kmeans(H(i), k)$
12:        $tot = tot + 1$
13:    end for
14: end for
15: for $z = 1$ to $n * t$ do
16:    for $i = 1$ to $nuser - 1$ do
17:        for $j = i + 1$ to $nuser$ do
18:            if $C(z, i) == C(z, j)$ then
19:                $\mathbf{RGM}(i, j) = \mathbf{RGM}(i, j) + \frac{1}{n*t}$
20:                $\mathbf{RGM}(j, i) = \mathbf{RGM}(i, j)$
21:            end if
22:        end for
23:    end for
24: end for
25: $Community = DetectCom(\mathbf{RGM})$
26: $cs = maxindex(Community)$
27: for $i = 1$ to $cs$ do
28:    $s(i) = comsize(Community, i)$
29:    $mf(i) = comfea(Community, i, s(i))$
30:    if $mf(i) > 0.75$ and $s(i) > 1$ then
31:        $com = member(Community, i)$
32:        $fuser = fuser \cup com$
33:    end if
34: end for
35: return $fuser$
```

There are three parts in Algorithm 2. The first part (lines 1–14) is to generate the base clusters. In particular, the function $SDAESetup(SDAEopts)$ is used to generate SDAEs according to the parameter setting, the function $noi(i)$ is used to return the corruption rate for SDAE, the function $Corrupting(\mathbf{M}, noi(i))$ is used to corrupt the **M** with noise $noi(i)$, the function $SDAETrain(DN, \mathbf{M}, SDAEopts)$ is used to train the SDAEs, and the function $getHfea(SDAE)$ returns the hidden layer in SDAEs.

The second part (lines 15–24) mainly uses evidence accumulation method to reconstruct the user–user graph based on base clustering results.

The third part (lines 25–35) mainly detects community based on persistence and determines the attacker communities with maximum filling rate of the item. The function $maxindex(Community)$ returns the number of communities, the function $comsize(Community, i, s(i))$ returns the number of community size, the function $comfea(Community, i)$ returns the mf of the community, and the function $member(Community, i)$ returns all members of the community.

In the first part, the training of SDAEs (lines 1–8) can be completed within the polynomial time complexity, whose time complexity can be expressed as $O(y)$ (Arora et al. 2014). The time complexity in k-means clustering algorithm is $O(|U| \times k \times st \times fd)$, where fd is the features' dimension, st is the iteration number. Therefore, the time complexity of generating the base clustering results (lines 4–14) is $O(n \times y + t \times |U| \times k \times st \times fd)$. In the second part, according to the accumulation method, the time complexity of generating co-association matrix is $O(n \times t \times |U|^2)$. In the third part, the time complexity in community detection algorithm is $O(|U| \times |edge|)$, where $|edge|$ denotes the number of edge in the user–user graph. The time complexity for computing community size (line 28) is $O(|U|)$. The time complexity of computing the mf indicator according to Eq. (15) (line 29) is $O(|commu| \times |I|)$, where $|commu|$ and $|I|$ represent the community size and the number of items, respectively. Therefore, the time complexity of the third part is $O(|U| \times |edge|) + O(cs \times (|U| + |commu| \times |I|))$, where cs denote the number of communities.

Overall, the time complexity of Algorithm 2 can be expressed as $O(n \times y + t \times |U| \times k \times st \times fd) + O(n \times t \times |U|^2) + O(|U| \times |edge|) + O(cs \times (|U| + |commu| \times |I|))$, where n, t, k, st, fd, cs, and $|commu|$ are much smaller than $|U|$ or $|I|$. Thus, the time complexity of Algorithm 2 can be described as $O(n \times t \times |U|^2)$.

4 Experimental evaluation

4.1 Experimental data and setting

In this paper, the following two datasets are used for the evaluation of our detection method.

(1) *Netflix dataset*[2] In the Netflix dataset, 542,182 ratings are randomly selected from January 6, 2000, to December 31, 2005, as experimental samples, in which 4000 movies are rated by 5000 users.

[2] This dataset was constructed to support participants in the Netflix prize (http://netflixprize.com); we downloaded it when it was available online.

(2) *Amazon dataset* Amazon dataset is crawled from Amazon.cn by Xu et al. (2013), in which 645,072 users have written 1,205,125 reviews on 136,785 items. In this dataset, 5055 users are labeled (Xie et al. 2011; Xu et al. 2013).

Since the Netflix dataset was published for the Netflix prize and used to train the recommendation algorithms, the published user profiles should be processed by Netflix Corp. in advance and they can be assumed as genuine ones. In the Netflix dataset, we generate the shilling profiles based on the attack models in Table 1, in which push attacks are carried out and the target item is randomly selected. Therefore, the Netflix experimental dataset includes two parts, shilling profiles generated from attack models and genuine profiles. Similarly, due to 5055 labeled reviewers, the Amazon experimental dataset also includes two parts, shilling profiles and genuine ones.

The different deep architectures for SDAEs were discussed in Lv et al. (2015). According to Lv et al. (2015), the number of hidden layers should be about three and it should be neither too small nor too large. Therefore, in Algorithm 2, the number of nodes in the first layer is set to the same number of users, and the number of nodes in the other layers is set to $\{2048, 512, 128, 32\}$, respectively. Following the suggestion in Geras and Sutton (2014), the noise level α is set to $\{0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$, respectively. According to the experimental results in Jiang et al. (2015), the sparse parameter $\rho = 0.05$, and the sparse penalty $\beta = 0.9$. For the consideration of computational complexity, we only cluster the users in the low dimension features of the last hidden layers. In our experiments, the final evaluation results are reported from the mean of 20 times experiments. The experiments are implemented on the PC with Intel i7-5500U CPU and 8GB memory.

4.2 Evaluation metrics

Since the labels have been provided in the experimental datasets, like the metrics in Mehta et al. (2007), Mehta and Nejdl (2009), and Yang et al. (2016a), the precision and recall metrics have been used to evaluate the performance of the detectors. Recall and precision metrics can be calculated as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{16}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{17}$$

where TP represents the number of shilling profiles which are correctly classified, FN represents the number of shilling profiles which are misclassified as genuine ones, and FP represents the number of genuine profiles which are misclassified as shilling ones.

4.3 Comparison of detection performance for nine methods on the Netflix dataset

In this section, we will compare our method DCEDM (ensemble detection method based on deep learning and community detection) with the eight baseline methods, including four supervised methods and four unsupervised methods. In supervised methods, we divided the genuine profiles in the Netflix dataset as training and test sets according to fivefold cross-validation method. In training set, we, respectively, inject random, average, 30% AoP, shifting, power item, and power user attacks with $\{1\%, 3\%, 5\%\}$ filler sizes. To balance the portion of genuine profiles and shilling profiles in training set, we set the attack size is 4% for each attack models. That is, under 6 attack models with 3 kinds of filler sizes, the ratio between shilling profiles and genuine profiles is $6 \times 3 \times 4\% = 72\%$. In test set, we, respectively, inject each type of attacks in Table 1 with $\{1\%, 3\%, 5\%\}$ filler sizes and $\{3\%, 5\%, 10\%, 12\%\}$ attack sizes. The generated shilling profiles are injected into test set, respectively. Therefore, there are $3 \times 4 \times 6 = 72$ test sets for each training set. To ensure that comparisons are made on the same order of magnitude, the test sets in the supervised methods are directly taken as test sets in unsupervised methods. Because there is no training set in unsupervised methods, the training set in the supervised methods is abandoned in unsupervised methods.

(1) *PCA-VarSelect* A classical unsupervised detection approach based on principal component analysis (PCA), which requires the prior knowledge of the number of attackers (Mehta et al. 2007; Mehta and Nejdl 2009). In PCA-VarSelect, we assume that we have known the number of attackers in advance.
(2) *GM-TIA* An unsupervised detector based on graph mining, which includes two stages. In the first stage, according to the co-rated items, the unweighted user–user graph is constructed and then the suspicious profiles are locked based on the similarity of random paths. In the second stage, the shilling profiles are detected from the suspicious profiles based on the target item analysis (Yang et al. 2016a).
(3) *SCD* An unsupervised detection approach based on the user–user graph proposed in Yang et al. (2016a), which directly uses the persistence optimization algorithm Chakraborty et al. (2014) to detect the community and determines the shilling profiles according to the mf indicator proposed in this paper.
(4) *UD-HMM* An unsupervised method for shilling attack detection based on hidden Markov model and hierar-

chical clustering in Zhang et al. (2018). Firstly, hidden Markov model is used to model user's history rating behaviors and user's suspicious degree is calculated. Then, the hierarchical clustering is used to group users according to user's suspicious degree.

(5) *C4.5-13* A supervised ensemble detector based on 13 user features proposed in Chirita et al. (2005), which uses C4.5 for classification.

(6) *RAdaBoost* A supervised ensemble detector based on 18 user features (Yang et al. 2016b). In the experiment, the number of iterations and the number of weak classifiers are selected via fivefold cross-validation on the training dataset.

(7) *SDAEs-PCA* A supervised ensemble detector based on the features extracted from multiple views, in which the features are automatically extracted by SDAEs and then combined based on PCA(principal component analysis) (Hao et al. 2019). In the experiment, the number of weak classifiers is determined via fivefold cross-validation on the training dataset.

(8) *SVM-TIA* A supervised detector based on SVM and target item analysis, which first extracted 7 rating-based features and used SVM classifier to generate a candidate set of attackers, and then filtered out the genuine ones from the candidate set based on target item analysis (Zhou et al. 2016)

Table 2 shows the precision of nine detection methods under six types of attacks with different filler sizes and attack sizes. Obviously, the proposed DCEDM has the best precision on 72 test sets.

From the comparison of precision in Table 2, in unsupervised methods (PCA-VarSelect, GM-TIA, SCD, UD-HMM, DCEDM), SCD has the lowest precision under random, average, and shifting attacks. PCA-VarSelect can effectively detect random, average, shifting attacks. However, when detecting power user and power item attacks, the precision of PCA-VarSelect decreases dramatically. This indicates that the shilling profiles generated by power user and power item attack models may evade the detection based on the principal component analysis. Although GM-TIA and UD-HMM remain the relative high precision under various attacks, their precision is never higher than that of DCEDM. In supervised methods (C4.5-13, RAdaBoost, SDAEs-PCA, SVM-TIA), C4.5-13, RAdaBoost, and SVM-TIA use the detection features extracted from ratings and have low precision under power item and power user attacks. This may because the extracted detection features in C4.5-13, RAdaBoost, and SVM-TIA can not be fully applied in these two types of attacks. In these three methods, RAdaBoost and SVM-TIA have higher precision than C4.5-13. For RAdaBoost, this is because RAdaBoost extracts more features from user profiles and uses powerful ensemble method for classification. For SVM-TIA, this is because it relies on target item analysis method to further separate the genuine profiles from shilling profiles. Due to the features extracted from multiple views, SDAEs-PCA has higher precision than C4.5-13, RAdaBoost, and SVM-TIA. Compared the above methods, DCEDM has the best precision under various attacks.

Table 3 shows the recall of nine detection approaches under six types of attacks with different filler sizes and attack sizes.

As shown in Table 3, PCA-VarSelect and SVM-TIA have the relative low recall in nine detection methods. For PCA-VarSelect, this indicates that most of principal components of genuine profiles are lower than those of shilling profiles and no longer effective for separating the shilling profiles from genuine ones. For SVM-TIA, this is because it uses only 6 rating-based features and SVM cannot be strong enough to fully discover the shilling profiles. Due to the advantages of multiple view features in SDAEs-PCA, it can effectively detect various attacks. For UD-HMM, its recall always maintains at 1 under random, average, shifting attacks. However, when detecting AoP, power user, and power item attacks, the recall of UD-HMM decreases. This indicates that the hidden Markov model can detect the attacks with random filler items, but cannot effectively identify the attacks with filler items reflected some preferences. Also, it can be seen that the methods based on the user–user graph (GM-TIA, SCD, DCEDM) always maintain the relative high level of recall under various attacks. The possible reason is that the connection between shilling profiles cannot be easily forged by attackers. By the further processing for user–user graph, the recall of DCEDM is better than that of GM-TIA and SCD. Therefore, under various shilling attacks, DCEDM has better recall metric than other methods.

4.4 Experiment on the Amazon dataset

To evaluate the performance of the DCEDM in real-world dataset, some experiments are conducted on the Amazon review dataset in this section. In supervised methods, we divided the genuine profiles as training set and test set according to fivefold cross-validation method. In unsupervised methods, the test set in supervised methods is used to conduct experiments, and the training set in the supervised methods is abandoned.

In the practical environment, precision and recall are often in contradiction with each other. Therefore, we utilize $F1$-measure to further evaluate the nine detectors.

$$F1-\text{measure} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}. \tag{18}$$

Figure 4 shows the recall, precision, and $F1$-measure of the nine detectors.

Table 2 Precision of nine methods with six attack types at various filler sizes across various attack sizes on the Netflix dataset

Filler size	1%				3%				5%			
Attack size	3%	5%	10%	12%	3%	5%	10%	12%	3%	5%	10%	12%
Random attack												
PCA-VarSelect	0.607	0.645	0.798	0.824	0.560	0.676	0.808	0.835	0.557	0.647	0.824	0.836
GM-TIA	0.761	0.789	0.794	0.837	0.760	0.799	0.817	0.846	0.793	0.821	0.848	0.864
SCD	0.262	0.329	0.298	0.286	0.294	0.290	0.321	0.350	0.118	0.183	0.315	0.350
UD-HMM	0.858	0.866	0.900	0.937	0.947	0.957	0.965	0.956	0.978	0.951	0.975	0.987
C4.5-13	0.212	0.331	0.496	0.514	0.266	0.352	0.525	0.572	0.315	0.361	0.533	0.593
RAdaBoost	0.502	0.564	0.686	0.768	0.535	0.614	0.714	0.828	0.589	0.603	0.779	0.845
SDAEs-PCA	0.744	0.782	0.824	0.884	0.750	0.829	0.873	0.936	0.806	0.799	0.878	0.925
SVM-TIA	0.493	0.529	0.562	0.615	0.544	0.595	0.693	0.751	0.569	0.582	0.730	0.797
DCEDM	0.882	0.901	0.917	0.956	0.985	0.977	0.983	0.940	0.946	0.955	0.967	0.984
Average attack												
PCA-VarSelect	0.568	0.644	0.781	0.821	0.541	0.653	0.777	0.821	0.491	0.633	0.782	0.814
GM-TIA	0.728	0.743	0.787	0.794	0.752	0.749	0.786	0.794	0.760	0.753	0.793	0.819
SCD	0.165	0.181	0.213	0.259	0.228	0.280	0.312	0.351	0.228	0.294	0.314	0.358
UD-HMM	0.876	0.861	0.917	0.934	0.929	0.918	0.941	0.961	0.939	0.959	0.960	0.975
C4.5-13	0.252	0.348	0.486	0.501	0.266	0.399	0.532	0.560	0.292	0.399	0.563	0.587
RAdaBoost	0.491	0.514	0.655	0.803	0.523	0.516	0.701	0.818	0.458	0.694	0.737	0.777
SDAEs-PCA	0.719	0.823	0.858	0.897	0.702	0.819	0.872	0.911	0.788	0.830	0.888	0.925
SVM-TIA	0.519	0.535	0.551	0.608	0.564	0.588	0.784	0.795	0.590	0.613	0.720	0.783
DCEDM	0.892	0.900	0.924	0.941	0.919	0.907	0.947	0.958	0.923	0.958	0.968	0.972
AoP attack												
PCA-VarSelect	0.121	0.368	0.557	0.558	0.055	0.151	0.324	0.350	0.007	0.004	0.086	0.162
GM-TIA	0.657	0.684	0.728	0.742	0.660	0.687	0.754	0.748	0.721	0.738	0.757	0.778
SCD	0.658	0.198	0.227	0.247	0.229	0.285	0.292	0.331	0.223	0.270	0.297	0.330
UD-HMM	0.821	0.837	0.853	0.846	0.854	0.868	0.910	0.912	0.891	0.919	0.937	0.938
C4.5-13	0.236	0.330	0.414	0.445	0.160	0.171	0.299	0.524	0.203	0.268	0.323	0.491
RAdaBoost	0.262	0.343	0.603	0.631	0.272	0.332	0.625	0.730	0.399	0.557	0.654	0.735
SDAEs-PCA	0.716	0.751	0.831	0.894	0.727	0.815	0.889	0.913	0.756	0.795	0.880	0.937
SVM-TIA	0.217	0.200	0.369	0.407	0.227	0.281	0.399	0.419	0.294	0.368	0.453	0.598
DCEDM	0.854	0.846	0.882	0.895	0.896	0.875	0.912	0.940	0.942	0.965	0.961	0.951
Shifting attack												
PCA-VarSelect	0.572	0.635	0.775	0.797	0.541	0.650	0.786	0.822	0.536	0.641	0.780	0.824
GM-TIA	0.744	0.736	0.753	0.772	0.738	0.735	0.716	0.774	0.722	0.755	0.772	0.795
SCD	0.188	0.209	0.226	0.225	0.229	0.281	0.309	0.356	0.224	0.295	0.318	0.359
UD-HMM	0.804	0.816	0.841	0.905	0.912	0.922	0.949	0.954	0.922	0.932	0.956	0.962
C4.5-13	0.255	0.357	0.492	0.501	0.259	0.358	0.524	0.543	0.292	0.346	0.522	0.592
RAdaBoost	0.413	0.505	0.625	0.762	0.485	0.653	0.723	0.811	0.509	0.595	0.732	0.835
SDAEs-PCA	0.765	0.806	0.826	0.886	0.784	0.849	0.876	0.926	0.795	0.803	0.896	0.934
SVM-TIA	0.518	0.535	0.590	0.642	0.531	0.614	0.758	0.776	0.527	0.598	0.689	0.811
DCEDM	0.828	0.834	0.856	0.918	0.948	0.869	0.938	0.957	0.897	0.941	0.971	0.994
Power user attack												
PCA-VarSelect	0.019	0.023	0.025	0.046	0.003	0.040	0.028	0.043	0.002	0.002	0.002	0.025
GM-TIA	0.692	0.733	0.758	0.787	0.695	0.731	0.801	0.824	0.750	0.771	0.814	0.834
SCD	0.222	0.233	0.308	0.267	0.256	0.192	0.311	0.353	0.127	0.185	0.313	0.353
UD-HMM	0.805	0.817	0.823	0.841	0.824	0.838	0.828	0.870	0.835	0.888	0.917	0.932
C4.5-13	0.100	0.113	0.142	0.200	0.117	0.137	0.170	0.213	0.129	0.157	0.218	0.234

Table 2 continued

Filler size	1%				3%				5%			
Attack size	3%	5%	10%	12%	3%	5%	10%	12%	3%	5%	10%	12%
RAdaBoost	0.158	0.170	0.222	0.287	0.155	0.164	0.223	0.287	0.150	0.169	0.229	0.286
SDAEs-PCA	0.738	0.786	0.813	0.861	0.725	0.813	0.864	0.881	0.722	0.798	0.897	0.920
SVM-TIA	0.297	0.328	0.382	0.416	0.384	0.424	0.491	0.554	0.454	0.495	0.558	0.635
DCEDM	0.854	0.882	0.896	0.906	0.826	0.861	0.840	0.921	0.870	0.900	0.948	0.958
Power item attack												
PCA-VarSelect	0.003	0.006	0.008	0.005	0.008	0.002	0.003	0.002	0.001	0.004	0.008	0.003
GM-TIA	0.708	0.745	0.793	0.784	0.719	0.738	0.775	0.814	0.723	0.751	0.775	0.818
SCD	0.189	0.173	0.303	0.169	0.125	0.188	0.319	0.332	0.125	0.186	0.321	0.297
UD-HMM	0.819	0.834	0.866	0.882	0.838	0.853	0.885	0.911	0.846	0.851	0.876	0.922
C4.5-13	0.052	0.075	0.100	0.122	0.077	0.073	0.111	0.145	0.070	0.067	0.106	0.165
RAdaBoost	0.079	0.063	0.118	0.226	0.086	0.096	0.149	0.339	0.060	0.070	0.127	0.248
SDAEs-PCA	0.721	0.751	0.793	0.850	0.731	0.771	0.850	0.864	0.773	0.824	0.878	0.892
SVM-TIA	0.170	0.174	0.229	0.290	0.185	0.191	0.273	0.320	0.371	0.404	0.439	0.477
DCEDM	0.883	0.906	0.919	0.941	0.887	0.926	0.941	0.949	0.891	0.920	0.935	0.967

As shown in Fig. 4, the precision, recall, and $F1$-measure of SDAEs-PCA and DCEDM are significantly higher than those of other seven methods, respectively. For overall performance, by taking advantages of machine learning methods and multiple view information, SDAEs-PCA has a slight advantage than DCEDM. PCA-VarSelect has the lowest precision, recall, and $F1$-measure. This may be because the genuine users and attackers artificially and purposefully select filler items to rate in practice. Thus, their principal components are close to each other and hardly to be used to detect attacks. Due to the more decisions involved, the ensemble methods (SDAEs-PCA, RAdaBoost, and DCEDM) can obtain some advantages in recall and $F1$-measure. It is noteworthy that GM-TIA and SVM-TIA maintain relative high precision and $F1$-measure. This indicates that target item analysis is a powerful method in practice. Although C4.5-13 and SCD have different advantages in precision and recall, their overall performance is not as good as DCEDM. Therefore, DCEDM has better performance than some baseline methods and is a promising methods in real-world dataset.

4.5 Statistical significance between DCEDM and other detectors

To verify whether or not there are significant differences between DCEDM and other eight detectors in the recall, precision, and $F1$-measure metrics, we use the Wilcoxon rank-sum test (Perolat et al. 2015) based on the above experimental results. In this paper, the null hypothesis is assumed that DCEDM is as good as other detectors for recall, precision, and $F1$-measure. For the Netflix dataset, we take the precision, recall, and $F1$-measure metrics under different filler sizes in the Wilcoxon rank-sum test. For the Amazon dataset, we take the precision, recall, and $F1$-measure metrics of 20 times in the Wilcoxon rank-sum test. Based on the experiment results on the Netflix and Amazon datasets, the p values and test results are listed in Tables 4 and 5, in which the significance level is set to 0.05.

As shown in Table 4, all the p values of DCEDM vs. other detectors for precision and $F1$-measure on the Netflix dataset are less than 0.05, and all the test results are 1. These results mean we should reject the null hypothesis at the significance level of 0.05. Therefore, there are significant statistical differences between DCEDM and other detectors for precision and $F1$-measure metrics. Based on the previous comparisons, we can safely draw a conclusion that DCEDM has better precision and $F1$-measure than other eight detectors under six types of attacks on the Netflix dataset.

In Table 4, it can also be noticed that p values of DCEDM vs. SDAEs-PCA for recall on the Netflix dataset are greater than 0.05, and the test results are 0. These indicate that the statistical differences between DCEDM and SDAEs-PCA for recall are not significant. This is consistent with the conclusion that the recall of DCEDM is higher than or equal to that of other detectors.

As shown in Table 5, all the p values of DCEDM vs. other detectors on the Amazon dataset are less than 0.05, and all the test results are 1. These results show that the statistical differences between DCEDM and other detectors are significant for recall, precision, and $F1$-measure with multiple running results. Based on the previous comparisons of the recall, precision, and $F1$-meature on the Amazon dataset, the statistical differences are consistent with the conclusion that DCEDM has better performance than some baseline detection methods.

Table 3 Recall of nine methods with six attack types at various filler sizes across various attack sizes on the Netflix dataset

Filler size	1%				3%				5%			
Attack size	3%	5%	10%	12%	3%	5%	10%	12%	3%	5%	10%	12%
Random attack												
PCA-VarSelect	0.60	0.64	0.80	0.82	0.55	0.67	0.81	0.83	0.55	0.64	0.82	0.83
GM-TIA	0.91	0.96	0.95	0.97	0.92	0.95	0.96	0.97	0.93	0.92	0.96	0.98
SCD	0.84	0.82	0.82	0.81	0.92	0.92	0.91	0.96	0.95	0.96	0.97	0.95
UD-HMM	1	1	1	1	1	1	1	1	1	1	1	1
C4.5-13	0.89	0.91	1	1	1	1	1	1	1	1	1	1
RAdaBoost	0.95	1	1	1	0.97	1	1	1	1	0.98	1	1
SDAEs-PCA	1	1	1	1	1	1	1	1	1	1	1	1
SVM-TIA	0.63	0.65	0.68	0.77	0.70	0.79	0.81	0.90	0.80	0.83	0.85	0.89
DCEDM	1	1	1	1	1	1	1	1	1	1	1	1
Average attack												
PCA-VarSelect	0.57	0.64	0.78	0.81	0.53	0.65	0.78	0.82	0.48	0.63	0.78	0.81
GM-TIA	0.87	0.92	0.94	0.94	0.89	0.95	0.95	0.95	0.92	0.98	0.98	1
SCD	0.85	0.88	0.91	0.93	0.86	0.92	0.93	0.93	0.92	0.95	0.95	0.97
UD-HMM	1	1	1	1	1	1	1	1	1	1	1	1
C4.5-13	0.98	0.99	1	1	0.97	1	1	1	1	0.95	0.95	1
RAdaBoost	0.93	0.96	1	1	1	1	1	1	1	1	1	1
SDAEs-PCA	1	1	1	1	1	1	1	1	1	1	1	1
SVM-TIA	0.62	0.65	0.70	0.79	0.71	0.82	0.84	0.90	0.80	0.82	0.89	0.91
DCEDM	1	1	1	1	1	1	1	1	1	1	1	1
AoP attack												
PCA-VarSelect	0.12	0.36	0.55	0.55	0.05	0.15	0.32	0.35	0.02	0.03	0.09	0.16
GM-TIA	0.90	0.95	0.95	0.96	0.91	0.95	0.96	0.95	0.89	0.91	0.92	0.96
UD-HMM	0.88	0.89	0.92	0.95	0.89	0.92	0.95	0.97	0.89	0.93	0.95	0.98
SCD	0.82	0.83	0.85	0.88	0.92	0.93	0.91	0.94	0.95	0.92	0.90	0.87
C4.5-13	0.95	0.97	0.90	0.90	1	0.99	0.98	0.97	1	1	1	0.99
RAdaBoost	0.89	0.87	0.92	1	0.98	0.93	0.97	0.90	0.90	0.89	0.93	0.88
SDAEs-PCA	1	1	1	1	1	1	1	1	1	1	1	1
SVM-TIA	0.46	0.49	0.53	0.61	0.52	0.58	0.68	0.74	0.60	0.59	0.68	0.71
DCEDM	1	1	1	1	1	1	1	1	1	1	1	1
Shifting attack												
PCA-VarSelect	0.57	0.63	0.77	0.79	0.53	0.65	0.79	0.82	0.53	0.64	0.78	0.82
GM-TIA	0.89	0.90	0.92	0.95	0.92	0.93	0.95	0.97	0.92	0.95	0.98	0.96
SCD	0.86	0.88	0.90	0.90	0.88	0.88	0.92	0.92	0.95	0.97	0.97	0.98
UD-HMM	1	1	1	1	1	1	1	1	1	1	1	1
C4.5-13	0.98	0.98	1	1	0.97	1	1	0.99	0.95	0.96	0.98	1
RAdaBoost	0.93	0.98	1	1	1	0.96	0.95	1	0.94	1	1	0.98
SDAEs-PCA	1	1	1	1	1	1	1	1	1	1	1	1
SVM-TIA	0.71	0.73	0.77	0.79	0.76	0.79	0.81	0.84	0.78	0.80	0.87	0.91
DCEDM	1	1	1	1	1	1	1	1	1	1	1	1
Power user attack												
PCA-VarSelect	0.01	0.02	0.02	0.04	0.01	0.03	0.02	0.04	0.02	0.02	0.03	0.03
GM-TIA	0.94	0.93	0.98	0.96	0.89	0.93	0.90	0.98	0.92	0.89	0.95	0.92
SCD	0.93	0.92	1	0.94	0.96	1	1	1	1	1	1	1
UD-HMM	0.85	0.86	0.88	0.89	0.87	0.89	0.92	0.94	0.88	0.91	0.93	0.95
C4.5-13	0.22	0.25	0.27	0.31	0.24	0.26	0.29	0.35	0.23	0.27	0.30	0.36

Table 3 continued

Filler size	1%				3%				5%			
Attack size	3%	5%	10%	12%	3%	5%	10%	12%	3%	5%	10%	12%
RAdaBoost	0.21	0.23	0.29	0.35	0.23	0.25	0.31	0.35	0.25	0.28	0.33	0.37
SDAEs-PCA	0.96	0.97	0.99	1	1	1	1	1	1	1	1	1
SVM-TIA	0.12	0.15	0.18	0.20	0.14	0.16	0.19	0.21	0.16	0.19	0.23	0.25
DCEDM	0.98	0.98	1	1	1	1	1	1	1	1	1	1
Power item attack												
PCA-VarSelect	0.02	0.02	0.03	0.05	0.02	0.02	0.03	0.03	0.04	0.04	0.06	0.07
GM-TIA	0.92	0.94	0.95	0.97	0.92	0.94	0.96	0.95	0.94	0.96	0.96	0.97
SCD	0.92	1	1	0.93	1	1	1	0.92	1	1	1	0.95
UD-HMM	0.86	0.87	0.88	0.88	0.87	0.89	0.91	0.93	0.89	0.90	0.92	0.94
C4.5-13	0.88	0.91	0.94	0.95	0.87	0.89	0.93	0.96	0.92	0.95	1	1
RAdaBoost	0.95	0.96	0.97	0.98	0.94	0.96	0.97	0.90	0.96	0.97	0.99	1
SDAEs-PCA	0.98	0.98	0.99	1	1	1	1	1	1	1	1	1
SVM-TIA	0.23	0.29	0.35	0.41	0.52	0.59	0.61	0.65	0.55	0.62	0.67	0.69
DCEDM	0.98	1	1	1	1	1	1	1	1	1	1	1

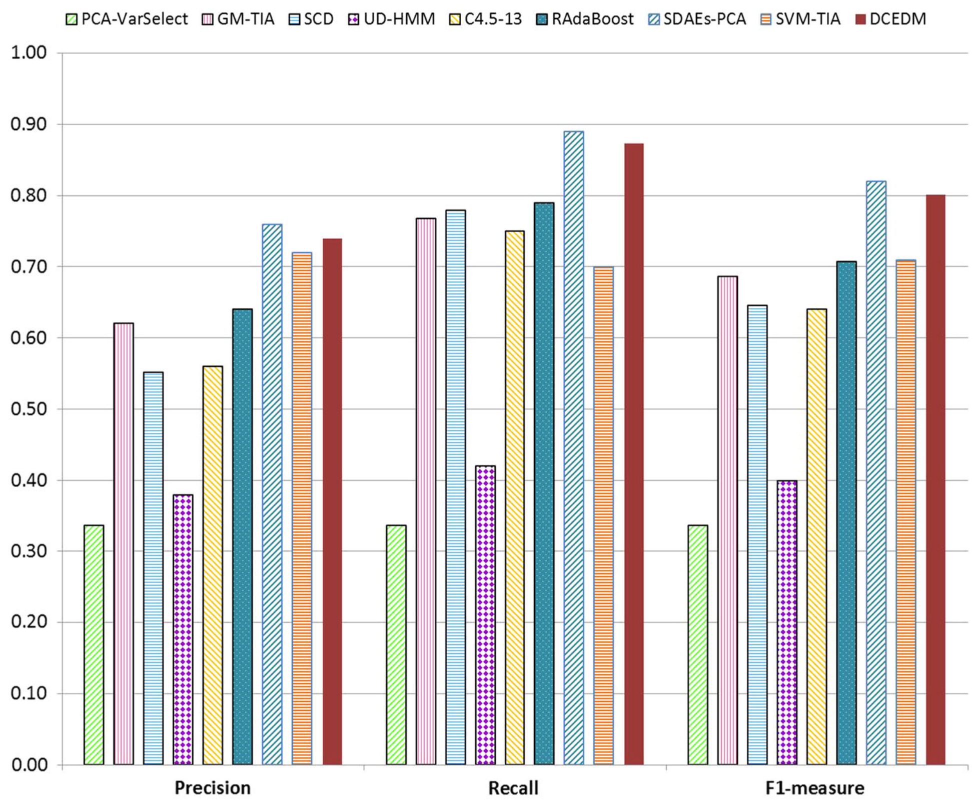


Fig. 4 Precision, recall, and $F1$-measure of nine detector methods on the Amazon dataset

5 Conclusion and future work

Recommender systems based on collaborative filtering can effectively alleviate information overload problem, but it is vulnerable to shilling attacks. Due to lack of the labeled samples, the supervised detection methods are often limited in practices. To effectively detect various shilling attacks, we construct the user–user graph with the edge weight of rating behaviors similarity. To mining the features lied in the user relations, we use SDAEs to extract the deep and robust graph features from different scales and reconstruct the user–user graph. With community detection method, we determine the community of shilling profiles. The experimental results on

Table 4 The p values and test results of DC-EDM vs. other detectors for recall, precision, and $F1$-measure on the Netflix dataset

Method	Filler size (%)	Precision		Recall		$F1$-measure	
		p values	Test results	p values	Test results	p values	Test results
DCEDM vs. PCA-VarSelect	1	3.064E−09	1	4.431E−10	1	3.06E−09	1
	3	3.473E−09	1	2.297E−10	1	3.06E−09	1
	5	3.064E−09	1	2.321E−10	1	3.06E−09	1
DCEDM vs. GM-TIA	1	3.473E−09	1	1.327E−08	1	3.06E−09	1
	3	3.935E−09	1	2.076E−10	1	3.06E−09	1
	5	3.064E−09	1	8.114E−10	1	3.06E−09	1
DCEDM vs. SCD	1	3.064E−09	1	3.588E−08	1	3.06E−09	1
	3	3.064E−09	1	3.028E−07	1	3.06E−09	1
	5	3.064E−09	1	8.669E−07	1	3.06E−09	1
DCEDM vs. UD-HMM	1	3.204E−04	1	8.769E−04	1	0.000552764	1
	3	4.661E−02	1	9.935E−05	1	0.038239796	1
	5	2.665E−02	1	9.958E−05	1	0.025271162	1
DCEDM vs. C4.5-13	1	3.064E−09	1	7.454E−06	1	3.06E−09	1
	3	3.064E−09	1	2.517E−06	1	3.06E−09	1
	5	3.064E−09	1	9.950E−05	1	3.06E−09	1
DCEDM vs. RAdaBoost	1	3.064E−09	1	5.815E−05	1	3.06E−09	1
	3	3.473E−09	1	6.631E−06	1	3.47E−09	1
	5	3.064E−09	1	1.697E−05	1	3.06E−09	1
DCEDM vs. SDAEs-PCA	1	3.324E−06	1	1.874E−01	0	1.18E−05	1
	3	1.714E−05	1	3.379E−01	0	1.71E−05	1
	5	7.085E−07	1	3.379E−01	0	7.08E−07	1
DCEDM vs. SVM-TIA	1	3.064E−09	1	4.484E−10	1	3.06E−09	1
	3	3.064E−09	1	2.330E−10	1	3.06E−09	1
	5	3.064E−09	1	2.327E−10	1	3.06E−09	1

Table 5 The p values and test results of DCEDM vs. other detectors for recall, precision, and $F1$-measure on the Amazon dataset

Method	Precision		Recall		$F1$-measure	
	p values	Test results	p values	Test results	p values	Test results
DCEDM vs. PCA-VarSelect	1.56E−10	1	1.67E−10	1	2.88E−10	1
DCEDM vs. GM-TIA	1.38E−07	1	0.033189112	1	2.89E−10	1
DCEDM vs. SCD	1.60E−10	1	1.64E−10	1	2.92E−10	1
DCEDM vs. UD-HMM	1.43E−09	1	1.36E−09	1	3.01E−10	1
DCEDM vs. C4.5-13	1.60E−06	1	1.62E−10	1	2.50E−10	1
DCEDM vs. RAdaBoost	1.46E−10	1	1.46E−08	1	3.94E−09	1
DCEDM vs. SDAEs-PCA	0.033872434	1	1.62E−10	1	3.65E−09	1
DCEDM vs. SVM-TIA	1.57E−08	1	1.54E−10	1	3.02E−10	1

the Netflix and Amazon datasets demonstrate that DCEDM outperforms some baseline detectors.

Since the deep learning method and clustering method take a certain amount of running time, in the next work, we will further optimize the proposed method to improve the efficiency. In addition, we will implement our method on the big data processing platform to further improve the function.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Nos. 61772452, 61379116), the Scientific and Technology Innovation Program of Higher Education Institutions in Shanxi, China (No. 2019L0847), the Youth Project of Humanities and Social Sciences Financed by Ministry of Education, China (No. 20YJC630034), and the Natural Science Foundation of Hebei Province, China (No. F2015203046).

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interest.

References

Arora S, Bhaskara A, Ge R, Ma T (2014) Provable bounds for learning some deep representations. In: International conference on machine learning, pp 584–592

Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828

Bhaumik R, Mobasher B, Burke R (2011) A clustering approach to unsupervised attack detection in collaborative recommender systems. In: Proceedings of the international conference on data mining (DMIN). Citeseer, p 1

Burke R, Mobasher B, Williams C, Bhaumik R (2006) Classification features for attack detection in collaborative recommender systems. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 542–547

Cao J, Jin D, Yang L, Dang J (2018) Incorporating network structure with node contents for community detection on large networks using deep learning. Neurocomputing 297:71–81

Castells P, Vargas S, Wang J (2011) Novelty and diversity metrics for recommender systems: choice, discovery and relevance. In: Proceedings of international workshop on diversity in document retrieval (DDR), pp 1–8

Chakraborty T, Srinivasan S, Ganguly N, Mukherjee A, Bhowmick S (2014) On the permanence of vertices in network communities. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 1396–1405

Chakraborty T, Dalmia A, Mukherjee A, Ganguly N (2017) Metrics for community analysis: a survey. ACM Comput Surv 50(4):1–37

Chen J, Saad Y (2012) Dense subgraph extraction with application to community detection. IEEE Trans Knowl Data Eng 24(7):1216–1230

Chen M, Weinberger K, Sha F, Bengio Y (2014) Marginalized denoising auto-encoders for nonlinear representations. In: International conference on machine learning, pp 1476–1484

Chirita P-A, Nejdl W, Zamfir C (2005) Preventing shilling attacks in online recommender systems. In: Proceedings of the 7th annual ACM international workshop on Web information and data management. ACM, pp 67–74

Dou T, Yu J, Xiong Q, Gao M, Song Y, Fang Q (2017) Collaborative shilling detection bridging factorization and user embedding. In: International conference on collaborative computing: networking, applications and worksharing. Springer, pp 459–469

Fred ALN, Jain AK (2005) Combining multiple clusterings using evidence accumulation. IEEE Trans Pattern Anal Mach Intell 27(6):835–850

Geras KJ, Sutton C (2014) Scheduled denoising autoencoders. arXiv:1406.3269

Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826

Gunes I, Polat H (2015) Hierarchical clustering-based shilling attack detection in private environments. In: Proceedings of the 3rd international symposium on digital forensics and security, pp 1–7

Gunes I, Kaleli C, Bilge A, Polat H (2014) Shilling attacks against recommender systems: a comprehensive survey. Artif Intell Rev 42(4):767–799

Hao Y, Zhang F, Wang J, Zhao Q, Cao J (2019) Detecting shilling attacks with automatic features from multiple views. Secur Commun Netw 2019:6523183:1–6523183:13

Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag 29(6):82–97

Hurley N, Cheng Z, Zhang M (2009) Statistical attack detection. In: Proceedings of the third ACM conference on recommender systems. ACM, pp 149–156

Jiang N, Rong W, Peng B, Nie Y, Xiong Z (2015) An empirical analysis of different sparse penalties for autoencoder in unsupervised feature learning. In: 2015 international joint conference on neural networks (IJCNN). IEEE, pp 1–8

Li W, Gao M, Li H, Xiong Q, Wen J, Ling B (2015) A shilling attack detection algorithm based on popularity degree features. Acta Autom Sin 41(9):1563–1576

Lv Y, Duan Y, Kang W, Li Z, Wang F-Y (2015) Traffic flow prediction with big data: a deep learning approach. IEEE Trans Intell Transp Syst 16(2):865–873

Malliaros FD, Vazirgiannis M (2013) Clustering and community detection in directed networks: a survey. Phys Rep 533(4):95–142

Mehta B, Nejdl W (2009) Unsupervised strategies for shilling detection and robust collaborative filtering. User Model User Adapt Interact 19(1–2):65–97

Mehta B, Hofmann T, Fankhauser P (2007) Lies and propaganda: detecting spam users in collaborative filtering. In: Proceedings of the 12th international conference on intelligent user interfaces. ACM, pp 14–21

Miyauchi A, Kawase Y (2016) Z-score-based modularity for community detection in networks. PLoS ONE 11(1):e0147805

Perolat J, Couso I, Loquin K (2015) Generalizing the wilcoxon rank-sum test for interval data. Int J Approx Reason 56:108–121

Pizzuti C, Rombo SE (2014) Algorithms and tools for protein–protein interaction networks clustering, with a special focus on population-based stochastic methods. Bioinformatics 30(10):1343–1352

Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106

Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105(4):1118–1123

Seminario CE, Wilson DC (2014) Attacking item-based recommender systems with power items. In: Proceedings of the 8th ACM conference on recommender systems. ACM, pp 57–64

Sun PG, Gao L, Yang Y (2013) Maximizing modularity intensity for community partition and evolution. Inf Sci 236:83–92

Tang T, Tang Y (2011) An effective recommender attack detection method based on time sfm factors. In: 2011 IEEE 3rd international conference on communication software and networks. IEEE, pp 78–81

Vincent P, Larochelle H, Bengio Y, Manzagol P-A (2008) Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on machine learning. ACM, pp 1096–1103

Williams CA, Mobasher B, Burke R (2007) Defending recommender systems: detection of profile injection attacks. Serv Oriented Comput Appl 1(3):157–170

Wilson DC, Seminario CE (2013) When power users attack: assessing impacts in collaborative recommender systems. In: Proceedings of the 7th ACM conference on Recommender systems. ACM, pp 427–430

Xie J, Szymanski BK, Liu X (2011) SLPA: uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: 2011 IEEE 11th international conference on data mining workshops. IEEE, pp 344–349

Xu C, Zhang J, Chang K, Long C (2013) Uncovering collusive spammers in chinese review websites. In: Proceedings of the 22nd ACM international conference on conference on information & knowledge management. ACM, pp 979–988

Yang Z, Cai Z, Guan X (2016a) Estimating user behavior toward detecting anomalous ratings in rating systems. Knowl Based Syst 111:144–158

Yang Z, Lin X, Cai Z, Zongben X (2016b) Re-scale adaboost for attack detection in collaborative filtering recommender systems. Knowl Based Syst 100:74–88

Zhang F, Chen H (2016) An ensemble method for detecting shilling attacks based on ordered item sequences. Secur Commun Netw 9(7):680–696

Zhang F, Zhou Q (2014) HHT-SVM: an online method for detecting profile injection attacks in collaborative recommender systems. Knowl Based Syst 65:96–105

Zhang S, Chakrabarti A, Ford J, Makedon F (2006) Attack detection in time series for recommender systems. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 809–814

Zhang Y, Tan Y, Zhang M, Liu Y, Tat-Seng C, Ma S (2015) Catch the black sheep: unified framework for shilling attack detection based on fraudulent action propagation. In: Proceedings of the 24th international conference on artificial intelligence, IJCAI'15. AAAI Press, pp 2408–2414

Zhang F, Zhang Z, Zhang P, Wang S (2018) UD-HMM: an unsupervised method for shilling attack detection based on hidden markov model and hierarchical clustering. Knowl Based Syst 148:146–166

Zhong C, Yue X, Zhang Z, Lei J (2015) A clustering ensemble: two-level-refined co-association matrix with path-based transformation. Pattern Recognit 48:2699–2709

Zhou W, Wen J, Xiong Q, Gao M, Zeng J (2016) SVM-TIA a shilling attack detection method based on svm and target item analysis in recommender systems. Neurocomputing 210:197–205