Copyright © 2019 Tech Science Press CMC, vol.59, no.1, pp.331-344, 2019

DPIF: A Framework for Distinguishing Unintentional Quality Problems From Potential Shilling Attacks

Mohan Li[1], Yanbin Sun[1, *], Shen Su[1], Zhihong Tian[1], Yuhang Wang[1, *] and Xianzhi Wang[2]

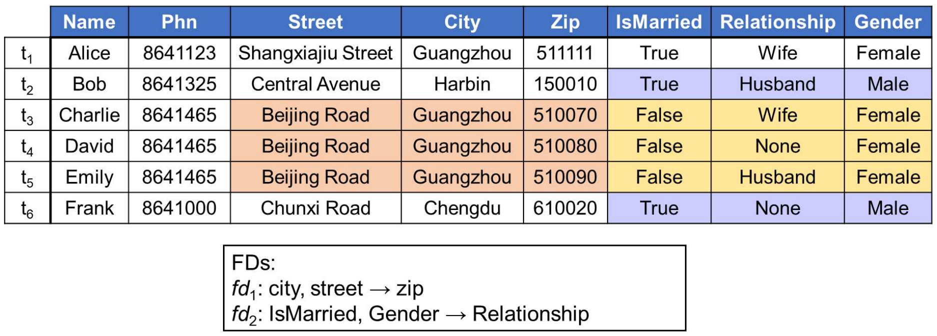
Abstract: Maliciously manufactured user profiles are often generated in batch for shilling attacks. These profiles may bring in a lot of quality problems but not worthy to be repaired. Since repairing data always be expensive, we need to scrutinize the data and pick out the data that really deserves to be repaired. In this paper, we focus on how to distinguish the unintentional data quality problems from the batch generated fake users for shilling attacks. A two-steps framework named DPIF is proposed for the distinguishment. Based on the framework, the metrics of homology and suspicious degree are proposed. The homology can be used to represent both the similarities of text and the data quality problems contained by different profiles. The suspicious degree can be used to identify potential attacks. The experiments on real-life data verified that the proposed framework and the corresponding metrics are effective.

Keywords: Data quality, shilling attacks, functional dependency, suspicious degree, homology.

1 Introduction

Low quality data can severely impact on the usability of data, and may lead to huge losses [Eckerson (2002)]. A lot of recent work has been proposed to evaluate the data quality and a lot of data cleaning rules are proposed [Fan and Geerts (2012); Fan, Geerts, Jia et al. (2008); Cao, Fan and Yu (2013); Fan, Geerts and Wijsen (2012); Chu, Ilyas and Papotti (2013a); Interlandi and Tang (2015); Fan, Li, Ma et al. (2012); Li, Li, Cheng et al. (2018); Li and Li (2016)]. However, most of these rules are adept at detecting errors, but not good at fixing errors. For example, functional dependencies $fd_1 : city, street \rightarrow zip$ and $fd_2 : IsMarried, Gender \rightarrow Relationship$ can be used to clean the data shown by Fig. 1. A functional dependency ϕ is violated if two tuples t_i and t_j are same in the attributes in the left-hand side, but different in the attribute in the right-hand side. By using the FDs, we will be of the opinion that there may exist errors in the colored grids since these grids violate the dependencies. However, it is difficult to know how to further locate and fix the

[1] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, 510006, China.
[2] School of Software, University of Technology Sydney, Sydney, Australia.
[*] Corresponding Authors: Yanbin Sun. Email: sunyanbin@gzhu.edu.cn;
Yuhang Wang. Email: apple125110@gmail.com.



	Name	Phn	Street	City	Zip	IsMarried	Relationship	Gender
t_1	Alice	8641123	Shangxiajiu Street	Guangzhou	511111	True	Wife	Female
t_2	Bob	8641325	Central Avenue	Harbin	150010	True	Husband	Male
t_3	Charlie	8641465	Beijing Road	Guangzhou	510070	False	Wife	Female
t_4	David	8641465	Beijing Road	Guangzhou	510080	False	None	Female
t_5	Emily	8641465	Beijing Road	Guangzhou	510090	False	Husband	Female
t_6	Frank	8641000	Chunxi Road	Chengdu	610020	True	None	Male

FDs:
fd_1: city, street → zip
fd_2: IsMarried, Gender → Relationship

Figure 1: The dataset containing user profiles

errors. For instance, t_2 and t_6 violate fd_2, but modify each of the attributes $IsMarried$, $Gender$ and $Relationship$ can make fd_2 not violated. Repair the low quality data is much more complex than locate the data quality problems. Since unreasonable repair is not advisable, we may pay a lot of effort or even need human power, such as domain experts or crowdsourcing to find the proper repairs [Garcia-Ulloa, Xiong and Sunderam (2017); Zheng, Li and Cheng (2016); Zhang, Chen, Tong et al. (2015); Chu, Morcos, Ilyas et al. (2015)].

Repairing low quality data is very expensive, thus we need to scrutinize the data and pick out the data that really deserves to be repaired. One type of data that is not worth repairing are the maliciously manufactured fake user profiles that ready to be used for shilling attacks. The following example illustrates the situation.

Example 1.1. Consider the data shown by Fig. 1. By using fd_1 and fd_2, we will find that (1) t_2 and t_6 violates fd_2, (2) t_3, t_4 and t_5 violate both of the two dependencies. However, if we carefully check t_3, t_4 and t_5, we will find that the three tuples are very similar in the text. The similarity exists even in the attributes with different values and the violations of dependencies. It is normal to have the correct values similar, but it is suspicious if the wrong value is similar. A possible case is that someone is trying to automatically generating some fake user profiles for shilling attack.

Shilling attacks [Gunes, Kaleli, Bilge et al. (2014)] aim to change the predictions of a recommender system by creating a set of fake users and the corresponding ratings. In modern recommender systems, a user is always asked to enter some personal information when creating a new account. Since the value fields may be specified, a labor-saving way is to use similar settings on these information when creating fake users. Moreover, fake users are often created in batches and the creators may not be familiar with the semantic of different attributes. Thus these fake profiles tend to make similar mistakes. In contrast, unintentional data quality problems are often sporadic and do not get strong group similarities.

Based on the above ideas, this paper proposes a two-steps framework, named Dirty Profile Identify Framework (DPIF for short), which aims to distinguish the unintentional data quality problems from potential shilling attacks. First a set of candidates which may be erroneous are detected by data quality rules, and then clustering is utilized to find potential shills. The contributions of this paper are as follows.

1. We provide a two-steps framework for distinguishing the unintentional data quality problems from potential shilling attacks.
2. We propose the metrics of homology and suspicious degree. The homology is defined to represent both the text similarity and the error similarities of different profiles. The suspicious degree is also defined to identify potential attacks.
3. The experiments on real-life data are carried out to verify the effectiveness of the proposed framework and metrics.

The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 provides an overview of the framework. Section 4 studies the process of finding candidate dirty tuples. Section 5 gives the methods for distinguishing quality problems and potential attacks. Section 6 shows the experimental study. Section 7 concludes the paper.

2 Related works

2.1 Data quality

There are currently a lot of works on data quality evaluation [Fan, Geerts, Jia et al. (2008); Fan, Geerts and Wijsen (2012); Chu, Ilyas and Papotti (2013a); Interlandi and Tang (2015); Fan and Geerts (2012); Ilyas, Chu et al. (2015); Chu, Ilyas and Papotti (2013b); Rammelaere, Geerts and Goethals (2017); Cao, Fan and Yu (2013); Kruse and Naumann (2018); Li, Li, Cheng et al. (2018); Li and Li (2016); Garcia-Ulloa, Xiong and Sunderam (2017); Li, Sun, Jiang et al. (2018)]. A lot of data quality rules, such as functional dependency, conditional functional dependency, denial constraint and editing rule, are used to detect data quality problems in the given datasets. The rules can be roughly classified into two categories.

The first category includes functional dependency, conditional functional dependency, denial constraints and some other similar dependencies [Fan, Geerts, Jia et al. (2008); Cao, Fan and Yu (2013); Fan, Geerts and Wijsen (2012); Chu, Ilyas and Papotti (2013a)]. These dependencies are good at finding quality problems but not good at fixing quality problems. The reason is that these rules usually indicate which combinations of values in the data are not correct, but do not indicate how to pinpoint and fix the errors.

The second category of dependencies tries to add information related to data repair in the rules [Interlandi and Tang (2015); Fan, Li, Ma et al. (2012)]. These rules can effectively fix errors in the data, but they are usually defined on small subsets of the data, which makes them to be limited in the application scenario and expensive to obtain.

Some crowdsourced methods are also proposed to overcome the difficulties in data repair [Chu, Morcos, Ilyas et al. (2015); Zhang, Chen, Tong et al. (2015); Garcia-Ulloa, Xiong and Sunderam (2017)]. These repair methods provide accurate repair results, but manpower repairs are often expensive and inefficient, so we must check before the repair operation to ensure that the data is worth repairing.

2.2 Shilling attacks

With the widespread use of big data analytics, security issues derived from big data are also receiving attention [Liu, Peng and Wang (2018); Wang, Liu, Qiu et al. (2018); Wang, Tian, Zhang et al. (2018); Tian, Cui, An et al. (2018); Chen, Tian, Cui et al. (2018); Zhang, Wang, Wang et al. (2018); Tian, Su, Shi et al. (2018); Sun, Li, Su et al. (2018)]. Shilling attack is a kind of attack on big data which negatively impacts the accuracy of a recommender system. The attackers try to inject dirty profiles into the data set, and injected dirty profiles may severely reduce the data quality. Existing techniques, including supervised and unsupervised methods, are mainly focus on how to identify the user's abnormal rating behavior [Gunes, Kaleli, Bilge et al. (2014)]. The current research focus on shilling attacks is mainly on how to manipulate the rating to influence the accuracy of the recommendation system. The types of the attacks include random attack, average attack, bandwagon attack [Mobasher, Burke, Bhaumik et al. (2007)], and average-noise injecting attack, average-target shift attack [Williams, Mobasher, Burke et al. (2006)], etc.

The methods of detecting shilling attack can be roughly classified into supervised and unsupervised methods. Supervised methods [Wu, Wu, Cao et al. (2012); Yang, Xu, Cai et al. (2016); Zhang and Zhou (2014)] require a well-labeled training data set. These data are often hard to get in real-life applications, because even if for humans, it is difficult to judge whether a score record is a malicious attack. Furthermore, supervised methods also fall short in the scenario that the type of attack is not known in advance. Unsupervised methods [Zhang, Zhang, Zhang et al. (2018); Bryan, O'Mahony and Cunningham (2008); Bhaumik, Mobasher and Burke (2011); Yang, Cai and Guan (2016)] can overcome some of the shortcomings of supervised methods. These methods are mainly based on clustering user rating behaviors to discover shilling attacks.

The existing works basically pay attention to observing the abnormality of the user's rating behavior, but ignores the abnormality of the user's personal information (so-called user profile). Dependencies between different attributes of a user profile are often readily available, and there are many existing works that can detect anomalous values on dependent attributes. In order to inject fake ratings, attackers often need to create a lot of fake user profile to register accounts. When creating these accounts, an attacker may ignore attribute dependencies and cause many different types of data quality problems in user profiles. Observing the homology of these data quality issues can effectively help us detect abnormal accounts and discover potential shilling attacks.

3 Overview of DPIF

Let R be a user profile table defined on a set of attributes $attr(R)$. Each tuple $t \in R$ corresponds to the information of a user. $t[A]$ is the values of t on attributes $A \subseteq attr(R)$. For

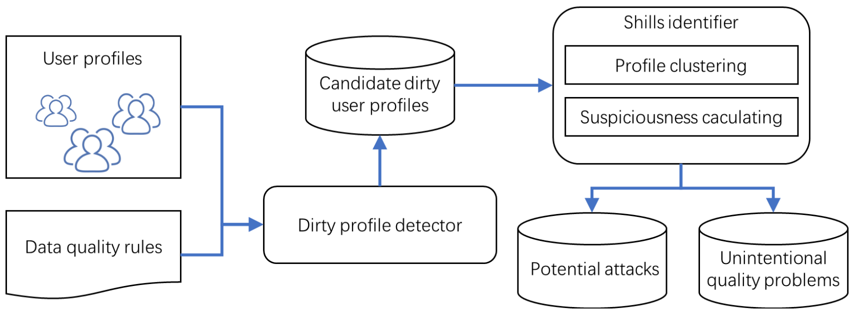


Figure 2: A overview of DPIF

example, for the tuple t_3 in Fig. 1, $t_3[city,\ street,\ zip]$ is $(BeijingRoad,\ Guangzhou,\ 510070)$. A data quality rule set Φ is defined on $attr(R)$, and can be used to detect the candidate dirty user profiles. The main idea two-steps workflow of DPIF is as follows, and the main com-ponents are shown by Fig. 2.

1. Dirty profile detector uses Φ to detect possible errors in R. The possible erroneous data are collected and organized by the provenance, i.e., by the tuples they come from (named candidate dirty user profiles) and the rules they violate.

2. Candidate dirty user profiles are given to the shills identifier. Shills identifier clusters the dirty user profiles based on provenance, and calculates the suspicious degree of each user profile. Based on the suspicious degree, the dirty user profiles are distinguished, as either potential shilling attack or unintentional quality problems.

For example, if we use DPIF to process the table shown by Fig. 1, the candidate dirty user profiles are $\{t_2, t_3, t_4, t_5, t_6\}$ since the colored grids are detected as the violation of fd_1 and fd_2. Then, shills identifier may identify that $\{t_3, t_4, t_5\}$ are potential attacks since they are similar in both text and errors. $\{t_2\}$ and $\{t_6\}$ are identified as unintentional quality problems because they have no suspicious analogues. Technical details of dirty profile detector and shills identifier are discussed in Section 4 and Section 5.

4 Detecting candidate dirty profiles

For the ease of discussion, we choose function dependencies (FD) as data quality rules, but it is easy to verify that other types of data quality rules discussed in Section 2 can also apply to the dirty profile detector.

A FD is a constraint between two sets of attributes in a relation. Given two attribute sets X and Y in relation $attr(R)$, a FD $\phi : X \rightarrow Y$ means that each X value in R is associated with precisely one Y value in R. X is called the left-hand side of $X \rightarrow Y$, denoted by $lhs(\phi)$, and Y is called the right-hand side of ϕ, denoted by $rhs(\phi)$. For example, $fd_1 :\ city,\ street \rightarrow zip$ means that $city$ and $street$ can uniquely determine the value of zip. Therefore, if two records t_i and t_j are the same in $city$ and $street$, but different in

zip, we can detect an error in $t_i[city, street, zip]$ and $t_j[city, street, zip]$. We choose FD to be the quality rule, because FDs are defined on the entire dataset rather than the subset of data. Thus high recall can be obtained with a small number of rules.

Only a pair of tuples can violate a FD, thus the process of this step is as follows. For each FD $\phi \in \Phi$, scan R to find how many pairs of tuples violate ϕ. Concretely, $\forall t_i, t_j \in R$, if $t_i[lhs(\phi)] = t_j[lhs(\phi)]$ but $t_i[rhs(\phi)] \neq t_j[rhs(\phi)]$, we say that t_i, t_j violate ϕ. Then, $t_i[lhs(\phi) \cup rhs(\phi)]$ and $t_j[lhs(\phi) \cup rhs(\phi)]$ are marked as potential dirty area, and $t_i,\ t_j$ are marked as candidate dirty profiles.

The time complexity is $O(|\Phi||R|^2)$. Some other existing techniques can also be employed to speed up this step. Since the optimization of dirty data detection is beyond the scope of this paper, we choose the easiest implementation.

4.1 Organization of the intermediate results

To make the subsequent process effective, we need to carefully organize the intermediate results. Three kinds of lists need to be maintained, i.e., $candi(R)$, $vio(t)$ and $vio(t_i, t_j)$.

- $candi(R)$ stores all the candidate dirty profiles.
- $vio(t)$ stores all the FDs that are violate by t, i.e., for each t_i, t_j violate ϕ, ϕ is in both $vio(t_i)$ and $vio(t_j)$.
- $vio(t_i, t_j)$ stores all the FDs that are jointly violate by t_i and t_j, i.e. for each t_i, t_j violate ϕ, ϕ is in $vio(t_i, t_j)$.

Example 4.1. Let the dataset and FDs in Fig. 1 be R and Φ, respectively. Then, $candi(R)$= $\{t_2, t_3, t_4, t_5, t_6\}$, for t_2, t_3 and t_6, we have $vio(t_2) = \{fd_2\}$, $vio(t_3) = \{fd_1, fd_2\}$, $vio(t_2, t_3) = \emptyset$, $vio(t_2, t_6) = \{fd_2\}$.

Please note that $vio(t_i, t_j)$ is not necessarily equal to $vio(t_i) \cap vio(t_j)$ since t_i and t_j can violate the same FD joinly with other tuples. Actually, $vio(t_i, t_j) \subseteq vio(t_i) \cap vio(t_j)$. These lists will be given to shills identifier and make the identifying process more effectively.

5 Distinguishing unintentional quality problems and potential attacks

The intermediate results are given to the shills identifier, and the aim of the shills identifier is to divide $candi(R)$ into two subsets, containing unintentional quality problems and potential attacks respectively. The main idea of the division is that it is suspicious if two profiles have similar errors. Therefore, this step can be further refining into two steps.

1. We propose a definition of the *homology* as the similarity of both the text and the errors contained by two different profiles, and cluster the candidate dirty profiles based on homology.

2. Profiles for attacks are usually generated in batches, thus larger clusters clusters are more likely to be potential attacks. Therefore, we can define the suspicious degree based on the size of the clusters.

The following subsections provide details of the two steps.

5.1 Clustering the candidate dirty profiles

First we provide the definition of homology to reflect both the text similarity and the similarities of the "errors" contained by different profiles. Homology is the similarity in trait from shared ancestry. High homology of two profiles means that they are very similar in text and the errors, thus they may be generated in batch and are considered as suspicious attacks. The definition is as follows.

Definition 5.1 (profile homology)**.** The homology of t_i and t_j is denoted by $H(t_i, t_j)$, and can be computed by Formula (1).

$$H(t_i, t_j) = \alpha \times textSim(t_i, t_j) + (1 - \alpha) \times vioSim(t_i, t_j) \tag{1}$$

where α is the normalization parameter, $textSim(t_i, t_j)$ is the text similarity of t_i and t_j, $vioSim(t_i, t_j)$ is the similarity of the violations of t_i and t_j, that is,

$$vioSim(t_i, t_j) = \frac{|vio(t_i, t_j)|}{|vio(t_i) \cup vio(t_j)|} \tag{2}$$

Please note that the definition of $vioSim(t_i, t_j)$ is different from the Jaccard similarity of two sets, because $vio(t_i, t_j)$ is not necessarily equal to $vio(t_i) \cap vio(t_j)$. However, when the data has many quality problems, $vio(t_i, t_j)$ might be much lower than $vio(t_i)$ and $vio(t_j)$, which makes $vioSim$ close to 0. In this case, we can use $vio(t_i) \cap vio(t_j)$ instead of $vio(t_i, t_j)$, to avoid excessive low homology. The $textSim(t_i, t_j)$ can be calculated by any reasonable similarities. For example, normalized absolute error can be used to measure the numerical attributes, normalized edit distance or cosine distance can be used to measure the string attributes, etc.

A cluster is a set of profiles, thus the definition of the homology of clusters can be given based on the definition of profile homology.

Definition 5.2 (cluster homology). The homology of a cluster c is the minimum homology of the profiles in c, that is,

$$H(c) = \min_{t,t' \in c} H(t, t') \tag{3}$$

The homology of two clusters c_i and c_j is the minimum homology of the profiles in $c_i \cup c_j$, that is,

$$H(c_i, c_j) = \min_{t,t' \in c_i \cup c_j} H(t, t') \tag{4}$$

Based on the homology, a single-linkage agglomerative clustering can be used to cluster profiles that are similar in both error and text. Given a threshold θ of homology, the agglomerative clustering aims to generate a set of clusters $C = \{c_1, c_2, ..., c_l\}$ which satisfies three conditions, i.e., (1) $H(c_i) \geq \theta$ for each $c_i \in C$, (2) $c_i \cap c_j = \emptyset$ for each $c_i \neq c_j$, and (3) $H(c_i, c_j) < \theta$ for each $c_i \neq c_j$.

The procedure of the agglomerative clustering is shown by Algorithm 5.1. The clustering process starts from the bottom layer and merges the two clusters with the highest homology value each time. In Algorithm 5.1, Line 1 to 3 initiate the clusters in C, each cluster contains exactly one element in $candi(R)$. Please note that profiles not in $candi(R)$ do not need to be clustered, since there are no errors in these profiles. In the loop from Line 4 to 9, $tmpc1$ and $tmpc2$ point to the two clusters with the highest homology. The union of the two clusters $tmpc1 \cup tmpc2$ is added to C, while $tmpc1$ and $tmpc2$ are removed from C.

Algorithm 5.1 The single-linkage agglomerative clustering of candidate dirty profiles

```
Input: $candi(R), \theta$
Output: $C = \{c_1, c_2, ..., c_l\}$ which satisfies the three conditions
 1: for each $t_i \in candi(R)$ do
 2:     $c_i \leftarrow \{t_i\}$
 3: end for
 4: while $\exists c, c' \in C$ s.t. $H(c, c') \geq \theta$ do
 5:     $tmpc1, tmpc2 \leftarrow \arg\max_{c,c' \in C} H(c, c')$
 6:     $newclu = tmpc1 \cup tmpc2$
 7:     $C \leftarrow C \setminus \{tmpc1, tmpc2\}$
 8:     $C \leftarrow C \cup \{newclu\}$
 9: end while
10: return $C$
```

Example 5.1. For the dataset shown by Fig. 1, we define the $textSim(t_i, t_j)$ as the percents of same values in t_i and t_j, $\alpha = 0.5$, and $\theta = 0.8$. The matrix in Fig. 3 shows the homology of profiles. The grey grids shows the homology higher than θ. If we use Algorithm 5.1 to cluster the profiles, we will get $C = \{\{t_2\}, \{t_3, t_4, t_5\}, \{t_6\}\}$.

5.2 Calculating the suspicious degree

As we discussed before, profiles for attacks are usually generated in batches, thus larger clusters or tighter clusters in clustering results are more likely to be potential attacks. Therefore, larger clusters are more likely to be suspicious. We provide a metric of suspicious degree based on the size of the cluster.

Definition 5.3 (suspicious degree)**.** The suspicious degree of a cluster is the ratio of its size

	t_2	t_3	t_4	t_5	t_6
t_2	1	0	0	0.06	0.5
t_3	0	1	0.81	0.81	0
t_4	0	0.81	1	0.81	0.06
t_5	0.06	0.81	0.81	1	0
t_6	0.5	0	0.06	0	1

Figure 3: The homology matrix

to the size of the largest cluster, that is,

$$susp(c) = \frac{|c|}{\max\limits_{c' \in C} \{|c'|\}}. \tag{5}$$

The suspicious degree of a profile t is the suspicious degree of the cluster it located in, that is, $susp(t) = susp(c^{(t)})$, where $c^{(t)}$ is the cluster containing t.

Based on the results of Example 5.1, the suspicious degree of the clusters can be calculated, that is, $susp(\{t_3, t_4, t_5\}) = 3/3 = 1$, $susp(\{t_2\}) = susp(\{t_6\}) = 1/3 \approx 0.33$.

All clusters can be sorted according to the suspicious degree, so that we can check from top to bottom which profiles might be potential attacks. We can set a parameter k, the top k clusters with the highest suspicious degree are considered potential attacks, while the remaining clusters are considered unintentional quality problems.

6 Experimental results

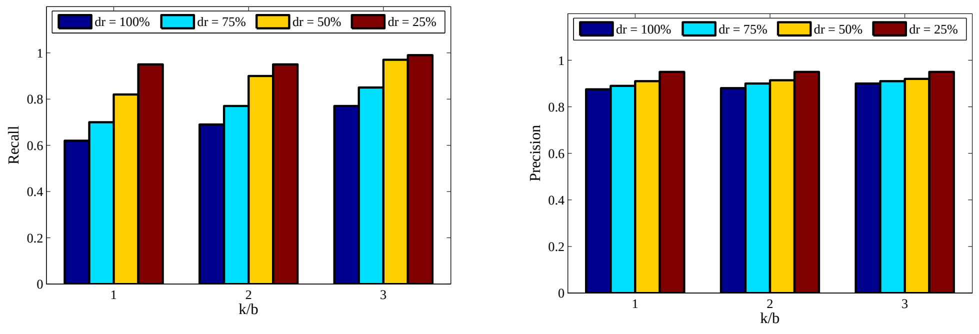
We conduct the experiments on real-life data set. The codes are written in Python and run on a machine with i7 1.80 GHz Intel CPU and 8 GB of RAM. The dataset which was used in the experiments is a UCI dataset[1] consisting of restaurant data with consumer ratings.

We use the DPIF proposed in this paper to distinguish the unintentional data quality problems from potential shilling attacks. We calculate the suspiciousness of each cluster and check if the attack occurs in clusters of top-k suspicious degree. In this process, there are four parameters that are very important, namely (p1) the methods of generating dirty profiles, (p2) the size of k when selecting top-k clusters, (p3) the threshold of the homology θ, and (p4) the selection of the FD set. We tested the recall and precision for different cases of the four parameters. In all experiments, there are 10% unintentional quality problems in the data. The measure of effectiveness is recall and precision, which are defined as follows.

$$recall = \frac{tp}{tp + fn}, \quad precision = \frac{tp}{tp + fp}$$

where tp, fp, fn are the number of true positive attacks, false positive attacks and false negative attacks.

[1] https://archive.ics.uci.edu/ml/datasets



(a) (b)

Figure 4: Recall and precision when varying k and dr

6.1 Varying dirty profiles and k

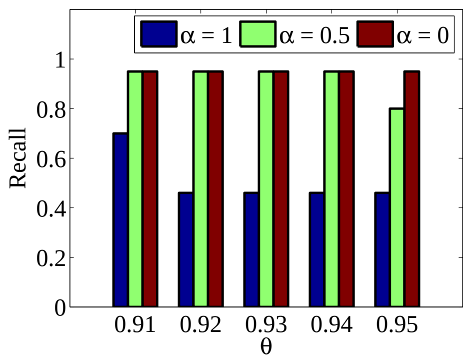
We put (p1) and (p2) together for the experiment, because the two parameters are related. There are numerous ways to generate dirty profiles. We choose to change some values in the normal profile to generate dirty profiles in batches. More concretely, we use one normal profile and random modify the profile in the range of dr (i.e., dr is the range that the modification allowed to be appear) to get a set of fake profiles. The profiles in the same set tend to be more similar to each other, and the set is called a *batch*. The number of batches is denoted by b. Ideally, each cluster in the result of DPIF is corresponding to a batch. Thus, to find all the potential attacks, k should be no less than b. Therefore, in the experiment, we vary k/b(i.e., $k = b$, $k = 2b$, etc) and dr. The recall and precision are shown by Fig. 4.

We find that increase of dr has a more obvious impact on and will also slightly affect precision. This is because that a larger value of dr means higher randomness when generating fake profiles, thus the quality issues that appear in the profile are more different. In other words, $vioSam$ will be lower, which decreases homology. dr also impact precision. However, the normal profiles are not very similar to each other, thus they have low homology which makes them hard to be clustered together. Therefore, dr has less impact on precision.

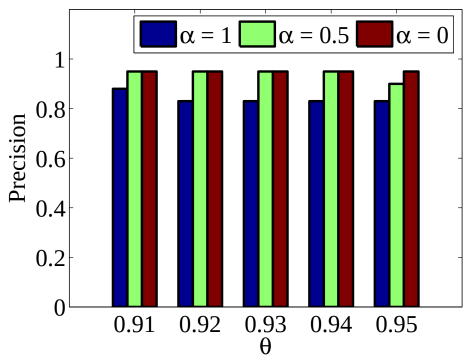
The different value of k also impacts the recall and precision. Since we only check top-k clusters, the higher k is, the higher recall and precision we get. When $k = b$ and $dr = 50\%$, the recall is only 0.82, when k is increased to $3b$, the recall is increased to 0.97. k does not affect precision much. This is because most normal profiles are in the small clusters, which are not likely to appear in top-k suspicious clusters.

6.2 Varying the homology threshold θ

As is discussed in Section 5, the higher the value of θ, the less likely the dirty profiles are clustered. Since we only check the top-k clusters, the larger θ, the lower the recall. The precision is less affected by θ. We test three different cases of α, i.e., $\alpha = 1$, $\alpha = 0.5$



(a)



(b)

Figure 5: Recall and precision when varying θ

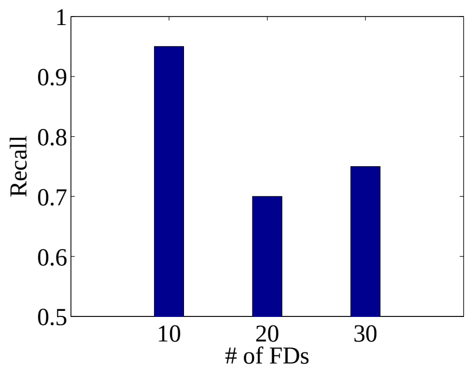
and $\alpha = 0$. The value of α is actually the weight of $textSim$ and $vioSim$. $\alpha = 0$ means that we only consider $vioSim$. $\alpha = 0.5$ means that we only consider $textSim$. $\alpha = 1$ means that we only consider $textSim$. The experimental result is shown by Fig. 5. In the experiments, dr is fixed to be 25%.

It can be observed from the result that when $\alpha = 1$ the recall and precision are obviously lower than the case that $\alpha = 0.5$ and $\alpha = 0$. This is because that in our scenario, even if the fake profiles are inserted in batch, these fake profiles are somewhat different from each other. Using high similarity threshold (greater than 0.9 in our experiments) is hard to distinguish normal quality problems from batch produced fake profiles. In experimental result shown by Fig. 4, the results of $\alpha = 0$ are better than $\alpha = 0.5$. However, it does not mean that lower α is always better. For example, as we have discussed, the increase of dr will lead to the decrease of $vioSam$. In that case, higher α will be better.

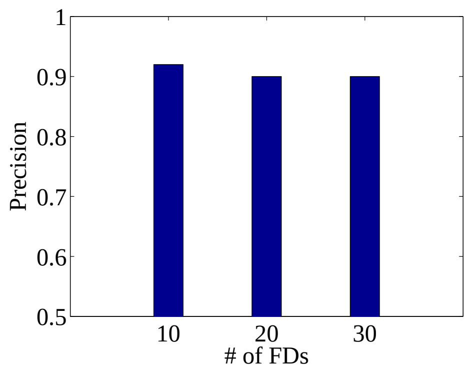
6.3 Varying the FD set

We test three FD sets, consisting of 10, 20 and 30 different FDs, respectively. The results of the recall and precision are shown by Fig. 6. It can be observed from the results that the number of FDs cannot decide the effectiveness. That is, more FDs does not mean higher recall and precision. For example, in our experiment the set consisting of 10 FDs achieved the highest recall and precision. The recall the set consists of 20 FDs is higher than the set consisting 30 FDs, but for precision, it is the opposite.

We further examined the performance of each FD throughout the process and found that in general the FDs with shorter left-hand side (i.e., has more attributes in the left-hand side) tend to be more effective than the ones with a longer left-hand side. This is because the left-hand side is more easy to satisfy.



(a)



(b)

Figure 6: Recall and precision when varying the FD set

7 Conclusions

This paper proposes DPIF, which aims to distinguish the unintentional data quality problems from potential shilling attacks. The process can be divided into two steps. First a set of candidates which may be erroneous is detected by data quality rules, and then clustering is used to find potential shills. In future work, we will explore different data quality rules, and will study the methods of automatically learning the parameters.

Acknowledgement: The work is supported by the National Natural Science Foundation of China (Nos. 61702220, 61702223, 61871140, 61572153, 61572492, U1636215) and the National Key Research and Development Plan (Grant Nos. 2018YEB1004003, 2018YF-B0803504).

References

Bhaumik, R.; Mobasher, B.; Burke, R. (2011): A clustering approach to unsupervised attack detection in collaborative recommender systems. *Proceedings of the International Conference on Data Mining*, pp. 181-187.

Bryan, K.; O'Mahony, M.; Cunningham, P. (2008): Unsupervised retrieval of attack profiles in collaborative recommender systems. *Proceedings of the 2008 ACM conference on Recommender systems*, pp. 155–162.

Cao, Y.; Fan, W.; Yu, W. (2013): Determining the relative accuracy of attributes. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 565-576.

Chen, J.; Tian, Z.; Cui, X.; Yin, L.; Wang, X. (2018): Trust architecture and reputation evaluation for internet of things. *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-9.

Chu, X.; Ilyas, I. F.; Papotti, P. (2013): Discovering denial constraints. *PVLDB*, vol. 6, no. 13, pp. 1498-1509.

Chu, X.; Ilyas, I. F.; Papotti, P. (2013): Holistic data cleaning: Putting violations into context. *IEEE 29th International Conference on Data Engineering*, pp. 458-469.

Chu, X.; Morcos, J.; Ilyas, I. F.; Ouzzani, M.; Papotti, P. et al. (2015): Katara: a data cleaning system powered by knowledge bases and crowdsourcing. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1247-1261.

Eckerson, W. (2002): Data warehousing special report: data quality and the bottom line. *Applications Development Trends*, vol. 1, no. 1, pp. 1-9.

Fan, W.; Geerts, F. (2012): Foundations of data quality management. *Synthesis Lectures on Data Management*, vol. 4, no. 5, pp. 1-217.

Fan, W.; Geerts, F.; Jia, X.; Kementsietsidis, A. (2008): Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems*, vol. 33, no. 2, pp. 1-48.

Fan, W.; Geerts, F.; Wijsen, J. (2012): Determining the currency of data. *ACM Transactions on Database Systems*, vol. 37, no. 4, pp. 25.

Fan, W.; Li, J.; Ma, S.; Tang, N.; Yu, W. (2012): Towards certain fixes with editing rules and master data. *VLDB Journal*, vol. 21, no. 2, pp. 213-238.

Garcia-Ulloa, D. A.; Xiong, L.; Sunderam, V. (2017): Truth discovery for spatio-temporal events from crowdsourced data. *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1562-1573.

Gunes, I.; Kaleli, C.; Bilge, A.; Polat, H. (2014): Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review*, vol. 42, no. 4, pp. 767-799.

Ilyas, I. F.; Chu, X. (2015): Trends in cleaning relational data: consistency and deduplication. *Foundations and Trends® in Databases*, vol. 5, no. 4, pp. 281-393.

Interlandi, M.; Tang, N. (2015): Proof positive and negative in data cleaning. *IEEE 31st International Conference on Data Engineering*, pp. 18-29.

Kruse, S.; Naumann, F. (2018): Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 759-772.

Li, M.; Li, J. (2016): A minimized-rule based approach for improving data currency. *Journal of Combinatorial Optimization*, vol. 32, no. 3, pp. 812-841.

Li, M.; Li, J.; Cheng, S.; Sun, Y. (2018): Uncertain rule based method for determining data currency. *IEICE Transactions on Information and Systems*, vol. 101, no. 10, pp. 2447-2457.

Li, M.; Sun, Y.; Jiang, Y.; Tian, Z. (2018): Answering the min-cost quality-aware query on multi-sources in sensor-cloud systems. *Sensors*, vol. 18, no. 12, pp. 4486.

Liu, Y.; Peng, H.; Wang, J. (2018): Verifiable diversity ranking search over encrypted outsourced data. *Computers, Materials & Continua*, vol. 55, no. 1, pp. 37.

Mobasher, B.; Burke, R.; Bhaumik, R.; Williams, C. (2007): Toward trustworthy recommender systems: an analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, vol. 7, no. 4, pp. 23.

Rammelaere, J.; Geerts, F.; Goethals, B. (2017): Cleaning data with forbidden itemsets. *IEEE 33rd International Conference on Data Engineering*, pp. 897-908.

Sun, Y.; Li, M.; Su, S.; Tian, Z.; Shi, W. et al. (2018): Secure data sharing framework via hierarchical greedy embedding in darknets. *Mobile Networks and Applications*.

Tian, Z.; Cui, Y.; An, L.; Su, S.; Yin, X. et al. (2018): A real-time correlation of host-level events in cyber range service for smart campus. *IEEE Access*, vol. 6, pp. 35355-35364.

Tian, Z.; Su, S.; Shi, W.; Yu, X.; Du, X. et al. (2018): A data-driven model for future internet route decision modeling. *Future Generation Computer Systems*.

Wang, Y.; Tian, Z.; Zhang, H.; Su, S.; Shi, W. (2018): A privacy preserving scheme for nearest neighbor query. *Sensors*, vol. 18, no. 8, pp. 2440.

Wang, Z.; Liu, C.; Qiu, J.; Tian, Z.; Cui, X. et al. (2018): Automatically traceback rdp-based targeted ransomware attacks. *Wireless Communications and Mobile Computing*, vol. 2018.

Williams, C.; Mobasher, B.; Burke, R.; Sandvig, J.; Bhaumik, R. (2006): Detection of obfuscated attacks in collaborative recommender systems. *Proceedings of the ECAI'06 Workshop on Recommender Systems*, vol. 94.

Wu, Z.; Wu, J.; Cao, J.; Tao, D. (2012): Hysad: a semi-supervised hybrid shilling attack detector for trustworthy product recommendation. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 985-993.

Yang, Z.; Cai, Z.; Guan, X. (2016): Estimating user behavior toward detecting anomalous ratings in rating systems. *Knowledge-Based Systems*, vol. 111, pp. 144-158.

Yang, Z.; Xu, L.; Cai, Z.; Xu, Z. (2016): Re-scale adaboost for attack detection in collaborative filtering recommender systems. *Knowledge-Based Systems*, vol. 100, pp. 74-88.

Zhang, C. J.; Chen, L.; Tong, Y.; Liu, Z. (2015): Cleaning uncertain data with a noisy crowd. *IEEE 31st International Conference on Data Engineering*, pp. 6-17.

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. *Knowledge-Based Systems*, vol. 148, pp. 146-166.

Zhang, F.; Zhou, Q. (2014): Hht-svm: an online method for detecting profile injection attacks in collaborative recommender systems. *Knowledge-Based Systems*, vol. 65, pp. 96-105.

Zhang, G.; Wang, T.; Wang, G.; Liu, A.; Jia, W. (2018): Detection of hidden data attacks combined fog computing and trust evaluation method in sensor-cloud system. *Concurrency and Computation: Practice and Experience*.

Zheng, Y.; Li, G.; Cheng, R. (2016): Docs: a domain-aware crowdsourcing system using knowledge bases. *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 361-372.