



Outlier detection from multiple data sources

Yang Ma^a, Xujun Zhao^a, Chaowei Zhang^b, Jifu Zhang^{a,*}, Xiao Qin^b

^a Taiyuan University of Science and Technology (TYUST), Taiyuan, Shanxi 030024, China

^b Department of Computer Science and Software Engineering, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347, United States

ARTICLE INFO

Article history:

Received 28 February 2020

Received in revised form 20 September 2021

Accepted 21 September 2021

Available online 23 September 2021

Keywords:

Multi-source

kNN-join

Outlier-join

Locality sensitive hashing

ABSTRACT

The preparation of a dataset by merging multiple data sources using the data fusion method may lead to the loss of vital information from each multi-source dataset and a certain amount of correlative information among the multiple data sources. Based on the extensive analysis of “the unique characteristics” of multi-source outliers, we propose multi-source outlier detection techniques to reliably identify outliers in multiple datasets. Several real-world examples are considered to classify multi-source outliers into three types (*Type I–III*) depending on the correlation among datasets. We design a baseline algorithm, which is an intuitive solution, and an optimal algorithm known as multiple-data-sources oriented outlier detection (*MOD*) to obtain high-score outliers. In addition, we build the *MOD+* method to speed up the outlier detection process. A new density metric combining kNN and RNN is introduced to evaluate the deviation degrees of multi-source outliers. The new outlier are applied to develop three *outlier-join* operators. *MOD* and *MOD+* are adept at (1) mining outlier information from each one of multi-source datasets and (2) sensing correlative outlier information among these multiple datasets. We implement and evaluate the three outlier detection algorithms by using synthetic and real-world datasets. The experimental results demonstrate that the proposed methods are promising and practical in the context of detecting outliers from multi-source datasets.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Outlier detection is aimed at searching a dataset to discover data objects that do not conform to expected behaviors. Mining outliers has been extensively employed in the fields of credit card fraud detection [1], astronomical spectral analysis [2,3], and intrusion detection [4], among others. Most existing algorithms [5,6], which detect outliers from a single dataset, exhibit severe performance degradation when the data are residing in multiple datasets. Moreover, the outliers detected from multiple datasets offer more insights than those identified from a single dataset. It is not uncommon that normal data objects are misclassified as outliers. Ideally, outliers should be manually verified by professionals with domain expertise. However, the process to manually verify outliers identified from multiple sources is time consuming. In the worst case scenario, outliers verification may be challenging for even domain experts due to the lack of distinctive features. Although data fusion can merge multi-source datasets into a single dataset, the merged dataset may not include certain vital information from each one of multi-source datasets and some correlative information among these multi-source datasets (see Sections 1.1 and 5.6 for details).

* Corresponding author.

E-mail address: jifuzh@sina.com (J. Zhang).

A growing number of big data applications necessitate the detection of outliers from multiple data sources [7]. To address this application requirement, we propose a novel outlier detection operator called *outlier-join*, which discovers outliers from a union of multiple interdependent data sources that have mutually influential relationships. Unlike conventional outlier detection operators, the proposed *outlier-join* operator validates the behaviors of outliers detected from dataset R in another dataset S . We demonstrate that *outlier-join* is a productive data mining primitive facilitating the implementation of knowledge discovery applications. We design a baseline algorithm for implementing the *outlier-join* computation. Moreover, we speed up the performance of the baseline algorithm by developing an optimal *outlier-join* algorithm.

1.1. Outlier examples from multiple datasets

We consider a bank that issues credit cards to a large number of customers. The bank relies on past transaction data to predict customer default payments using data mining methods. From the perspective of risk management, it is necessary to mine outliers from massive amounts of credit card data. This paper introduces the *outlier-join* operator that offers insightful knowledge for discovering outliers from a union of multiple credit card client datasets. For example, the *outlier-join* can determine whether an outlier detected from undeveloped-area credit data is also an outlier in developed-area credit data. The application of *outlier-join* to the two credit datasets can enable the empirical analysis of developed/undeveloped area differential in terms of default payments.

With the advent of big data, the number of data sources and volumes of data have rapidly increased. Data fusion has become a leading technology to perform data mining from multi-source datasets, with the data fusion problems being resolved by the integration of multi-source information prior to further processing. Unfortunately, data fusion cannot detect multi-source outliers owing to the following two reasons. First, the creation of a dataset by data fusion may lead to the loss of key characteristics of the original single dataset. Second, in modern distributed databases and storage systems, most large datasets (e.g., large tables) are achieved across multiple servers or sites. In this context, it is impractical and unnecessary to leverage data fusion integrating multiple large datasets to construct a gigantic dataset.

In what follows, we argue that data fusion cannot adequately address the issue of multi-source outlier detection. We offer evidence through a concrete example, where the *outlier-join* reported in this paper is applied to census scenario. In our example, R and S are two census datasets, which are acquired from city C , towns A and B in city C .

In the first case, the 2010 and 2015 census data of town A are used as datasets R and S , respectively. According to the 2010 census dataset (i.e., R), stay-at-home children (children who are temporarily cared for by their grandparents in daily life) in town A represent outliers. On the contrary, stay-at-home children in town A are detected as normal data according to the 2015 census dataset (i.e., S). Although the amount of data of town A in 2015 may be higher than that in 2010, we are more concerned about the prominent increase in the number of stay-at-home children and the ratio between the number of stay-at-home children and a total population of town A caused by the increase in the number of outgoing workers. Specifically, with the rapid development of China's urban economy, an increasing number of young rural couples work in certain regions, temporarily without their children in the same region. The results illustrate that the number of migrant workers in town A significantly increased between 2010 and 2015, suggesting that stay-at-home children represented a notable social problem in town A . If the 2010 and 2015 census datasets (i.e., R and S) are integrated using data fusion technology, the newly created dataset may not include critical and sensitive information regarding datasets R and S , even though this information may considerably influence the multi-source outlier detection. Hence, in this example, the data fusion technique fails in addressing the problem in the first case multi-source outlier detection.

In the second case, census data for town A and city C in 2015 are used as datasets R and S , respectively, with $R \subset S$. The stay-at-home children in city C are treated as outliers; however, these children represent normal data objects in town A . Obviously, the problem of stay-at-home children is well mitigated in city C except for a few towns like town A . If the census data in town A and city C are fused before detecting multi-source outliers, the data characteristics of town A are expected to overlap with those of city C . In this case, multi-source outliers cannot be detected correctly after data fusion.

In the third case, census data for town A and B in 2015 are used as datasets R and S , respectively, with $R \cap S = \emptyset$. The stay-at-home children in town A are outliers; however, the stay-at-home children in town A are considered as normal data in town B . Please note that the data objects of stay-at-home children in town A do not belong to dataset S . We add the data objects to dataset S before detecting outliers from dataset S . In this scenario, the governor of town A can likely better manage the social problem of stay-at-home children than the governor of town B . If datasets R and S are fused, the problems of stay-at-home children in towns A and B cannot be compared.

According to these examples, it is necessary to develop an effective outlier detection technique to discover useful information from multiple data sources to facilitate decision-makings.

1.2. Contributions

The examples elaborated in Section 1.1 motivate us to design a multi-source outlier detection operator called *outlier-join* to improve outliers' trustworthiness. The contributions of this study can be summarized as follows.

Contribution 1. We introduce the novel concept of multi-source outliers, where outliers are detected from multiple data sources; in this case, the outlier data can be cross-certified.

Contribution 2. We analyze the significance of multi-source outliers by using concrete examples; we classify these outliers into three categories (i.e., *Type I*, *Type II* and *Type III*).

Contribution 3. We design a baseline algorithm for detecting multi-source outliers; moreover, we propose an optimal algorithm based on *kNN-join* to accelerate the execution of the *Baseline* algorithm.

Contribution 4. Incorporating *k* Nearest Neighbors (*kNN* for short) and Reverse Nearest Neighbors (*RNN* for short), we propose a measurement approach to distinguishing outliers. We apply the measurement approach to design a multi-source outliers operator called *outlier-join*.

Contribution 5. Using high-dimensional datasets, we conduct extensive experiments to investigate the effectiveness of the *outlier-join* algorithm.

As a further discussion (see Section 6), we examine a special multi-source outlier type, which is different from the three outlier categories addressed in this study.

1.3. Roadmap

The rest of the paper is organized as follows. Section 2 describes the background knowledge on *kNN-join* operation and *kNN*-based outlier detection. Section 3 analyses outlier detection in multi-source. Section 4 presents an outlier measure based on *kNN* and *RNN*, and apply the measure to design a multi-source outlier operator *outlier-join*. Section 5 describes experimental setups and results. Section 6 future discusses another type of multi-source outliers. Section 7 summarizes the prior work related to multi-source and outlier detection. Finally, Section 8 concludes this paper.

2. Preliminaries

In this section, we first present the *kNN-join* method, followed by an introduction of the *kNN*-based outlier detection method. Table 1 lists the major symbols and notations used throughout this paper.

2.1. *kNN-join*

We consider two datasets R and S in space $\mathbb{R}^d$, where each object (e.g., $r \in R$ and $s \in S$) is represented as a d -dimensional object. We measure The similarity distance between objects $r \in R$ and $s \in S$ using their Euclidean distance $dis(r, s)$. the operation $knn(r, S)$ [8] returns a set of k nearest neighbors or *kNN* of point r from set S .

Given object $r \in R$, the *kNN-join* operation $R \bowtie S$ of datasets R and S returns a combination set of each object r 's *kNN* set. Thus, we express $R \bowtie S$ using operation $knn(r, S)$ as follows.

$$R \bowtie S = \{(r, knn(r, S)) \mid \text{for all } r \in R\}. \quad (1)$$

2.2. *kNN*-based outlier detection

Given an input dataset with N points, parameters n and k , a point q is an outlier if there exist no more than $n - 1$ other points q' such that $D^k(q') > D^k(q)$, where $D^k(q)$ is a summation of the distances between query point q and its k nearest neighbors. In other words, if we rank points in a non-increasing order of their $D^k(q)$ distance, the top n points in this ranking are identified as outliers. We employ any of the L_p metrics [9] like the L_1 ("Manhattan") or L_2 ("Euclidean") metrics to gauge the distance between a pair of points. This outlier detection method is referred to as *Top-n* proposed by Jin et al. [9].

For a large dataset, it is a highly time consuming to find outlier points according to $D^k(q') > D^k(q)$. Hence, we perform a search in a small sample dataset, which an extremely small part of a large dataset. Moreover, we obtain a special point q such that only $n - 1$ points q' satisfy the condition $D^k(q') > D^k(q)$. Let $\varepsilon = D^k(q)$, parameter ε represents the deviation degree,

Table 1
Symbol and notation.

Symbol	Notation	Symbol	Notation
$knn(r, S)$	the <i>kNN</i> of r	$\mathbb{R}^d$	d -dimensional space
R	dataset	$dis(r, s)$	the distance of r and s
S	dataset	ε_R	deviation degree in R
r	an object in R	ε_S	deviation degree in S
s	an object in S	$ R $	object number in R
q	a query point in $\mathbb{R}^d$	$ S $	object number in S
ε	deviation degree	$D^k(q)$	distance sum of q and its <i>kNN</i>
$R_{outlier}$	outlier set in R	RS	intersection of R and S
$S_{outlier}$	outlier set in S	$N_k(q)$	number of <i>RNN</i>

which is a criterion for detecting outliers in an original large dataset. Specifically, a point q' in the original dataset is an outlier if distance $D^k(q')$ is larger than threshold ε (i.e., $D^k(q') > \varepsilon$).

3. Problem analysis

3.1. Problem statement

Outliers are defined as a small set of data objects that are considerably different from the other data objects according to certain measures. More often than not, the outliers detected from a single data source exhibit local features. Outliers that are incorrectly extracted cannot accurately reflect the characteristics of deviations; such outliers inevitably lead to ill decision-making. If outliers are detected from multiple sources, the cross-certification of outliers can provide users with insightful information. For example, in the case of credit card fraud detection, outliers can truly capture users' fraudulent behaviors if the outliers detected from bank transactions can be cross-certified by those obtained from other banks.

To justify our motivations, we consider four examples illustrated in Fig. 1. Fig. 1(a) shows traditional outliers detected from a single data source, where objects O_1 , O_2 , and O_3 are marked as outliers. Note that O_1 , O_2 , and O_3 are far from the other objects (i.e., red and blue dots). From the perspective of the relationship between two datasets, we consider three examples of different types of multi-source outliers, as shown in Fig. 1(b)–(d).

In the first example, as shown in Fig. 1(b), outliers are identified from the two intersected datasets R and S , where O_1 and O_3 are outliers with respect to both R and S . Unlike O_1 and O_3 , O_2 is an outlier from the perspective of R ; O_2 , which is close to the other objects in S , is treated as a normal data. Outlier O_3 is more trustworthy than O_2 owing to the outliers' cross-certification. O_2 is a suspicious outlier, which should be further verified against the other datasets. Intuitively, such trustworthy outliers are more beneficial than suspicious outliers in various applications such as fraud detection, loan-application processing, intrusion detection, fault diagnosis, and the like.

In the second example demonstrated in Fig. 1(c), outliers O_1 , O_2 , and O_3 are identified from dataset R , which is a subset of S . O_1 is a normal object; O_2 and O_3 are outliers in dataset S . Specifically, O_2 and O_3 are outliers from the perspective of datasets R and S ; O_1 is an outlier in R . Hence, O_1 is a local outlier; O_2 and O_3 are envisioned as both local and global outliers. Obviously, O_2 and O_3 have more distinct deviation characteristics than O_1 , which indicates that O_2 and O_3 are more trustworthy than O_1 . Such outliers can be widely applied in network performance monitoring, fault diagnosis, satellite image analysis, and other similar fields.

In example 3 depicted in Fig. 1(d), datasets R and S are independent of one other (i.e., two disjoint sets), and no shared object exists in R and S . Nevertheless, datasets R and S share a structure (i.e., R and S have the same attribute dimensions), even though the data in the two sets may have been acquired in different time periods. O_1 , O_2 , and O_3 are outliers identified from dataset R . We add O_1 , O_2 , and O_3 to dataset S and measure their deviations in dataset S . Assuming that O_2 has a small deviation and O_1 and O_3 have a large deviation in S , we may confirm that O_1 and O_3 are also outliers in S . Using the same method, outliers O'_1 , O'_2 , and O'_3 detected from S may be validated by dataset R . Fig. 1(d) shows that O'_1 and O'_3 are outliers in R . In summary, outliers set $\{O_1, O_3, O'_1, O'_3\}$ is more trustworthy than outliers set $\{O_2, O'_2\}$. Such outliers can be used to realize incremental outlier detection in a few application fields, including but not limit to crime detection in electronic commerce, disaster weather monitoring, and time-series monitoring.

3.2. Problem formulation

Considering the examples illustrated in Fig. 1, we classify multi-source outliers into the following three types.

Type I: We consider two datasets R and S , with $RS = \{R \cap S\} \neq \emptyset$, $R \not\subseteq S$, and $S \not\subseteq R$. If o is an outlier in R and S ($o \in RS$), o is a high-score outlier that passes the cross-certification; if o is an outlier only in R , o is defined as a low-score outlier that fails in the cross-certification. A low-score outlier may correspond to noise data. The objective is to identify high-score outliers.

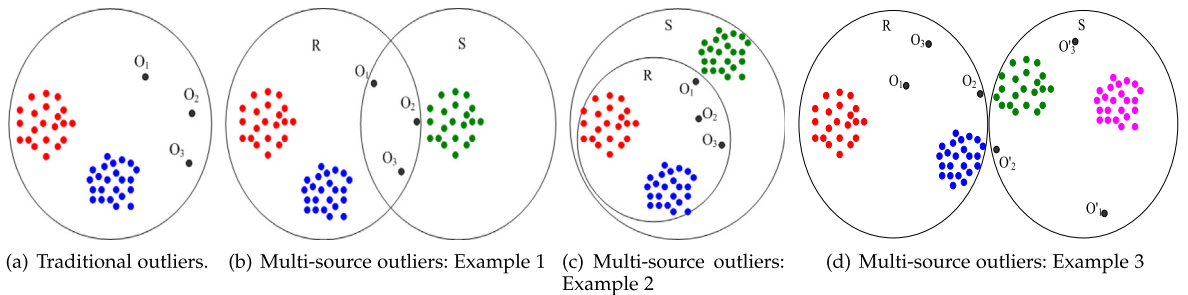


Fig. 1. Three simple examples of multi-source outliers.

Type II: We consider a case involving two datasets R and S , with $R \subset S$. If object o is an outlier in both R and S , o is treated as a high-score outlier that is a local and global outlier. If o is an outlier only in R , o is a low-score outlier that is only a local outlier.

Type III: Finally, we consider a case involving two datasets R and S , with $R \cap S = \emptyset$. If outlier o in R (or S) can be validated as an outlier in another dataset S (or R), o is defined as a high-score outlier. If o is an outlier in only R (or S), o will be referred to as a low-score outlier.

To search the multi-source outliers of *Type I* or *Type II*, the most intuitive strategy consists of the following two steps. First, outliers sets R_{outlier} and S_{outlier} are detected from R and S , respectively, by using the k NN-based outlier detection method. Second, we perform an intersection operation between the R_{outlier} and S_{outlier} , followed by computation of the intersection set RS_{outlier} (i.e., $RS_{\text{outlier}} = \{R_{\text{outlier}} \cap S_{\text{outlier}}\}$) containing high-score outliers.

In *Type III*, outlier sets R_{outlier} and S_{outlier} are derived from datasets R and S , respectively. The two outlier sets are validated by S and R ; the verified outlier set is regarded as a high-score outliers set.

3.3. An initial design and its analysis

3.3.1. A baseline solution

We delineate a baseline algorithm to detect high-score outliers of *Type I* or *Type II*. Given two input datasets R and S , the algorithm aims to identify outliers residing in both R and S . We use the k -nearest-neighbour scheme (a.k.a., k NN) governed by two thresholds (i.e., ε_R and ε_S) to discover outliers in the input datasets. ε_R and ε_S can be obtained by processing a sample dataset using the k NN method.

Algorithm 1. The baseline algorithm: *Baseline*

Input: $R, S, k, \varepsilon_R, \varepsilon_S$

Output: RS_{outlier} ;

```

1: for ( $i = 0; i < |R|; i++$ ) do//for each object  $o_i \in R$ 
2:   for ( $j = 1; j < |R|; j++$ ) do// $|R|$  is the number of objects in  $R$ 
3:      $D_i \leftarrow \text{distance}(i, j)$ ;//Save distance between  $o_i$  and  $o_j$  in  $D_i$ 
4:   end for
5:    $d_{\min} \leftarrow \text{sum}(o_i, D_i)$ ;//select  $k$ NN from list  $D_i$ , and calculate sum between  $o_i$  and its  $k$ NN
6:   if ( $d_{\min} > \varepsilon_R$ ) then// $o_i$  is an outlier if  $d_{\min}$  exceeds  $\varepsilon_R$ 
7:      $R_{\text{outlier}} \leftarrow \text{object } o_i$ ;
8:   end if
9: end for
10: for ( $i = 0; i < |S|; i++$ ) do//for each object  $o_i \in S$ 
11:   for ( $j = 1; j < |S|; j++$ ) do// $|S|$  is the number of objects in  $S$ 
12:      $D_i \leftarrow \text{distance}(i, j)$ ;//Save distance between  $o_i$  and  $o_j$  in  $D_i$ 
13:   end for
14:    $d_{\min} \leftarrow \text{sum}(o_i, D_i)$ ;//select  $k$ NN from list  $D_i$ , and calculate sum between  $o_i$  and its  $k$ NN
15:   if ( $d_{\min} > \varepsilon_S$ ) then// $o_i$  is an outlier if  $d_{\min}$  exceeds  $\varepsilon_S$ 
16:      $S_{\text{outlier}} \leftarrow \text{object } o_i$ ;
17:   end if
18: end for
19:  $RS_{\text{outlier}} \leftarrow R_{\text{outlier}} \cap S_{\text{outlier}}$ ;
20: output( $RS_{\text{outlier}}$ );

```

Algorithm 1 is the pseudocode of the *Baseline* solution, which is comprised of two key phases (see Lines 1–9 for phase 1 and Lines 10–18 for phase 2). The two phases determine outliers R_{outlier} and S_{outlier} in datasets R and S , respectively. The algorithm performs the calculation by omitting the intersection (i.e., RS_{outlier}) of the outlier sets R_{outlier} and S_{outlier} .

Now we briefly outline the steps that lead to outlier set R_{outlier} in Phase 1; phase 2 follows a similar process to yield its outlier set S_{outlier} . D_i represents the list of distances between object o_i and all the other objects in dataset R . In Lines 1–4, the distance between object o_i and a neighbouring object are repeatedly computed. All the distances pertaining to object o_i are stored in list D_i . Line 5 selects the k minimum distance values from list D_i , and calculates the sum of the distances. This distance sum is defined as $d_{\min}$. If $d_{\min}$ is larger than the given threshold ε_R , object o_i is envisioned as an outlier in R . In this case, Line 7 stores o_i in outlier set R_{outlier} .

Time Complexity of the Baseline Solution. The time complexity of Algorithm 1 is $O((|R|^2 + |S|^2) \times o(d))$, where $|R|$ and $|S|$ denote the numbers of objects in dataset R and S , respectively. $o(d)$ is the time complexity of distance calculation between two points (i.e., the complexity of $\text{distance}(i, j)$). Different distance measurement methods correspond to different time complexity. For example, the time complexity of the Euclidean distance calculation is n^2 , where n is the number of attributes.

Choosing a distance measurement method largely depends on application domains. For example, to measure the similarity between two sets, we apply the Jaccard distance. To compute the distance between two sequences, we adopt the Hamming distance.

3.3.2. The MOD algorithm

The time complexity of the above baseline algorithm indicates that the processing time is proportional to the sizes of R and S ; in other words, the search for high-score outliers is highly time-consuming when the amount of data is large. Considering the large time complexity of Algorithm 1, we optimize the baseline algorithm; the optimized version of Algorithm 1 is termed multiple-data-source oriented outlier detection (MOD).

The MOD scheme consists of the following three steps, and its pseudocode is presented as Algorithm 2.

- **Step 1.** We obtain an intersection set between original sets R and S (i.e., $RS = \{R \cap S\}$, see Line 1).
- **Step 2.** After separately performing two k NN-join operations $RS \bowtie R$ and $RS \bowtie S$, we deploy the k NN-based outlier detection method to get two outliers sets $R_{outlier}$ and $S_{outlier}$ from $RS \bowtie R$ and $RS \bowtie S$, respectively (see Lines 2–19).
- **Step 3.** Multi-source outliers are detected; these outliers are same items from $R_{outlier}$ and $S_{outlier}$ (see Line 20).

Algorithm 2. Optimal method: MOD

Input: $R, S, k, \varepsilon_R, \varepsilon_S$
Output: $RS_{outlier}$;
1: $RS \leftarrow R \cap S$; //select same objects from R and S
2: **for** ($i = 0$; $i < |RS|$; $i++$) **do** //for each object $o_i \in RS$
3: **for** ($j = 1$; $j < |R|$; $j++$) **do** //
4: $D_i \leftarrow \text{distance}(i, j)$; //Save distance between o_i and o_j in D_i
5: **end for**
6: $d_{min} \leftarrow \text{sum}(o_i, D_i)$; //select k NN from list D_i , and calculate sum between o_i and its k NN
7: **if** ($d_{min} > \varepsilon_R$) **then** // o_i is an outlier if d_{min} exceeds ε_R
8: $R_{outlier} \leftarrow \text{object } o_i$;
9: **end if**
10: **end for**
11: **for** ($i = 0$; $i < |RS|$; $i++$) **do** // $|RS|$ is the number of objects in RS
12: **for** ($j = 1$; $j < |S|$; $j++$) **do** //
13: $D_i \leftarrow \text{distance}(i, j)$;
14: **end for**
15: $d_{min} \leftarrow \text{sum}(o_i, D_i)$; //select k NN from list D_i , and calculate sum between o_i and its k NN
16: **if** ($d_{min} > \varepsilon_S$) **then** // o_i is an outlier if d_{min} exceeds ε_S
17: $S_{outlier} \leftarrow \text{object } o_i$;
18: **end if**
19: **end for**
20: $RS_{outlier} \leftarrow R_{outlier} \cap S_{outlier}$;
21: **output**($RS_{outlier}$);

Time Complexity of MOD. We show that the time complexity of Algorithm 2, which is an optimal algorithm (i.e., MOD), is $O((|RS| \times |R| + |RS| \times |S|) \times o(d)) = O(|RS| \times (|R| + |S|) \times o(d))$, where $o(d)$ is the time complexity of the distance calculation between two points (i.e., the complexity of $\text{distance}(i, j)$). $|RS|$, $|R|$ and $|S|$ denote the numbers of objects in datasets $R \cap S$, R and S , respectively. Importantly, when $|RS|$ is small, the time complexity of the MOD scheme is significantly smaller than that of the Baseline algorithm (see Algorithm 1). In the worst case scenario where $|RS|$ is approximate to $|R|$ and $|S|$, the time complexity of MOD is downgraded to that of the Baseline algorithm.

4. Handling outlier-join

In Section 4.1, we describe the method of evaluating outliers using k NN and RNN. Section 4.2 presents the implementation of outlier-join operators detecting three types of multi-source outliers discussed in Section 3.2.

4.1. Outlier measurement using k NN and RNN

In Algorithms 1 and 2, we use the Top- n method to measure and detect multi-source outliers. The Top- n approach relies on the distance from each data point to its k Nearest Neighbors(k NN). It is worth noting that k NN's measures and queries are

the foundation of the *Top-n* method. *kNN*'s queries only focus on the influence of nearest neighbors on query point q , and the impacts of query point q on the other objects are ignored. A reverse nearest neighbors query or *RNN* returns data objects that have query objects in the set of their k nearest neighbors. *RNN* represents a complementary problem to search the k nearest neighbors or *kNN* of a query object. The goal of a reverse k nearest neighbors query is to clarify the influence of a query object on an entire dataset. It is arguably true that seamlessly integrating *kNN* and *RNN* to seek outliers can likely effectively yield authentic outliers.

Given a finite multidimensional dataset $R \subset \mathbb{R}^d$ and a query point $q \in \mathbb{R}^d$, the reverse nearest neighbors (a.k.a., *RNN*) of query point q are defined by employing operation $knn(r, R)$ that returns the k nearest neighbors of data object r in dataset R . Thus, we have

$$RNN(q, R) = \{r | q \in knn(r, R), \text{ for all } r \in R\}. \quad (2)$$

In the process of detecting local outliers, it is prudent to calculate the deviation degree of each object. Combining the *kNN* and *RNN* approaches, we introduce a new density metric – local density of q (see Definition 1 below) – as a measurement to distinguish outliers.

Definition 1 (*local density of q*). Given a query point q and dataset $R \subset \mathbb{R}^d$, the density of q is defined as: $den(q, R) = \frac{SUM(q) \times \alpha + 1}{N_k(q) \times (1 - \alpha) + 1}$, where $SUM(q) = \sum_{i=1}^k dis(q, r_i)$ returns a summation of the distances between query point q and its k nearest neighbors. $N_k(q)$ is the reverse nearest neighbors count of q within dataset R , and α is a weight applied to adjust the proportion between the *kNN* and *RNN*.

Since outliers, by definition, are the absolute minority in a dataset, the local outliers in the *Top-n* method [9] are discovered by searching the number of n points with the largest *LOF* scores. Similarly, the n points with the largest local density are regarded as local outliers in our approach. After computing the local density of each query point, all the density values are sorted in descending order, where n points with the highest density are elected as outliers.

According to Definition 1, the local density measures are only affected by the k nearest neighbors when parameter α is configured to 1. Specifically, our method gracefully downgrades to the traditional *Top-n* algorithm [9] if parameter α equals to 1 (i.e., $\alpha = 1$). When parameter α is set to 0, local density is manipulated by the number of reverse nearest neighbors. In this case, our method is reshaped to an *RNN*-based algorithm, which is a simplified version of the well-known *AntiHub* algorithm [10].

4.2. Outlier-join operator using *kNN* and *RNN*

As described in this subsection, we apply the outlier measures proposed in Section 4.1 to detect multi-source outliers, which are classified into three types (see also *Type I*, *Type II* and *Type III* in Section 3). In the following sections, we elaborate three approaches to implementing *outlier-join* operator customized for the three types.

4.2.1. Formal expressions for outlier-join operator

First, we consider the *Type I* case, where given two datasets R and S , the intersection RS of the two datasets is non-empty. For all objects in set RS (i.e., $o \in RS$), if $den(o, R)$ is larger than ϵ'_R and $den(o, S)$ is larger than ϵ'_S (i.e., $den(o, R) > \epsilon'_R$ and $den(o, S) > \epsilon'_S$), object o is a multi-source outlier from the perspective of datasets R and S , where ϵ'_R and ϵ'_S are the deviation degrees in datasets R and S , respectively.

We use $\{R_{outlier}\}$ and $\{S_{outlier}\}$ to represent outliers sets in datasets R and S , respectively. $R_{outlier} \cap S_{outlier}$ is a joint set containing all the high-score outliers owing to outlier cross-certifications. Operator *outlier-join* (i.e., $R \propto S$) returns a multi-source outlier set by selecting outliers from datasets R and S .

The operator expressed in Eq. (3) is similar to that described in Eq. (4), except that we substitute R for the last S in Eq. (3) to form Eq. (4). The three types differ from one another in terms of the distinct correlation between datasets R and S (e.g., $R \cap S \neq \emptyset$ or $R \cap S = \emptyset$).

Type I. Given two datasets R and S , where the intersection RS is non-empty, we formally express the *outlier-join* operator $R \propto S$ for the first type of outliers in Eq. (3).

$$R \propto S^I = \{R_{outlier} \cap S_{outlier}\} = \{den(o, R) > \epsilon'_R | \forall o \in R\} \cap \{den(o, S) > \epsilon'_S | \forall o \in S\} \quad (3)$$

Type II. Given two datasets R, S , with R being a subset of S (i.e., $R \subset S$), we can express the *outlier-join* operator $R \propto S$ for the second outlier type in Eq. (4).

$$R \propto S^{II} = \{R_{outlier} \cap S_{outlier}\} = \{den(o, R) > \epsilon'_R | \forall o \in R\} \cap \{den(o, S) > \epsilon'_S | \forall o \in R\} \quad (4)$$

Type III. Considering two datasets R, S , the intersection of which is an empty set (i.e., $R \cap S = \emptyset$), we formally present the *outlier-join* operator $R \propto S$ for the third outlier type in Eq. (5).

$$R \propto S^{III} = \{den(o, R) > \epsilon'_R | \forall o \in R\} \cap \{den(o, S) > \epsilon'_S | \forall o \in R\} \cup \{den(o, S) > \epsilon'_S | \forall o \in S\} \cap \{den(o, R) > \epsilon'_R | \forall o \in S\} \quad (5)$$

4.2.2. The algorithm for outlier-join operator

At the heart of the aforementioned three formulas (see (3)–(5)), searching for the k nearest neighbors is vital step in computing the local density of query points. In particular, when all the query points in a dataset are linked to their k nearest neighbors from another dataset, the k nearest neighbors queries are converted to a k NN-join operation. Locality-Sensitive Hashing (a.k.a., *LSH*) can be used to search the approximate nearest neighbors in high-dimensional spaces. The *LSH* algorithm exhibits an extremely high efficiency in finding k nearest neighbors; hence, we employ the *LSH* algorithm to measure local density of query points, thereby detecting multi-source outliers.

Algorithm 3 illustrates the pseudocode of a solution that extends and optimizes the *LSH* algorithm. Algorithm 3 is comprised of three key phases (see Lines 1–11 for phase 1, Lines 12–24 for phase 2, and Lines 25–36 for phase 3).

Algorithm 3. LSH-based optimal algorithm: *MOD+*

Input: $R, S, k, \alpha, \epsilon'_R, \epsilon'_S$

Output: $RS_{outlier}$;

```

1:  $RS \leftarrow R \cap S$ ; // select same objects from  $R$  and  $S$ 
2: Create an empty hash table  $R\_HashTable$  for Dataset  $R$ ;
3: for (int  $i = 0$ ;  $i < |R|$ ;  $i++$ ) do //for each object  $o_i \in R$ 
4:    $HR_i \leftarrow \text{hash}(o_i)$ ; //compute object  $o_i$ 's hash value
5:    $R\_HashTable \leftarrow \text{insert}(o_i, HR_i)$ ; //  $o_i$  is inserted to  $R\_HashTable$  according to  $HR_i$ 
6: end for
7: Create an empty hash table  $S\_HashTable$  for Dataset  $S$ ;
8: for (int  $i = 0$ ;  $i < |S|$ ;  $i++$ ) do //for each object  $o_i \in S$ 
9:    $HS_i \leftarrow \text{hash}(o_i)$ ; //compute object  $o_i$ 's hash value
10:   $S\_HashTable \leftarrow \text{insert}(o_i, HS_i)$ ; //  $o_i$  is inserted to  $S\_HashTable$  according to  $HS_i$ 
11: end for
12: for (int  $i = 0$ ;  $i < |RS|$ ;  $i++$ ) do //for each object  $o_i \in RS$ 
13:    $HashValue \leftarrow \text{hash}(o_i)$ ; //compute object  $o_i$ 's hash value and save it to  $HashValue$ 
14:    $R\_neighbor[k] \leftarrow \text{find}(HashValue, R\_HashTable)$ ; //according to  $HashValue$ , find  $k$  nearest neighbors from  $R\_HashTable$ 
15:    $S\_neighbor[k] \leftarrow \text{find}(HashValue, S\_HashTable)$ ;
16:   for (int  $j = 0$ ;  $j < k$ ;  $j++$ ) do
17:      $R\_sum[i] \leftarrow R\_sum[i] + \text{distance}(o_i, R\_neighbor[j])$ ; //compute the sum of distances in  $R$  dataset
18:      $ID \leftarrow \text{get}(R\_neighbor[j])$ ; //get the ID of the  $j$ th neighbor
19:      $R\_count[ID]++$ ; //count the number of reverse nearest neighbors
20:      $S\_sum[i] \leftarrow S\_sum[i] + \text{distance}(o_i, S\_neighbor[j])$ ; //compute the sum of distances in  $S$  dataset
21:      $ID \leftarrow \text{get}(S\_neighbor[j])$ ; //get the ID of the  $j$ th neighbor in  $S\_neighbor[j]$ 
22:      $S\_count[ID]++$ ;
23:   end for
24: end for
25: for (int  $i = 0$ ;  $i < |RS|$ ;  $i++$ ) do //for each object  $o_i \in RS$ 
26:    $d_{min} \leftarrow \frac{R\_sum[i] \times \alpha + 1}{R\_count[i] \times (1 - \alpha) + 1}$ ; //compute the local density of object  $o_i$  from  $R$  dataset
27:   if ( $d_{min} \geq \epsilon'_R$ ) then //  $o_i$  is an outlier if  $d_{min}$  exceeds  $\epsilon'_R$ 
28:      $R_{outlier} \leftarrow \text{object } o_i$ ;
29:   end if
30:    $d_{min} \leftarrow \frac{S\_sum[i] \times \alpha + 1}{S\_count[i] \times (1 - \alpha) + 1}$ ; //compute the local density of object  $o_i$  from  $S$  dataset
31:   if ( $d_{min} \geq \epsilon'_S$ ) then //  $o_i$  is an outlier if  $d_{min}$  exceeds  $\epsilon'_S$ 
32:      $S_{outlier} \leftarrow \text{object } o_i$ ;
33:   end if
34: end for
35:  $RS_{outlier} \leftarrow R_{outlier} \cap S_{outlier}$ ;
36: output( $RS_{outlier}$ );

```

Phase 1. In the first phase, Lines 2 and 7 create empty hash tables $R_HashTable$ and $S_HashTable$, respectively; and Lines 4 and 9 compute the hash value of each object in datasets R and S , respectively. Next, Lines 5 and 10 insert these hash values into hash table $R_HashTable$ and $S_HashTable$, respectively. In Line 4, $\text{hash}(o_i)$, which is a hash function [11], calculates the hash value of object o_i . In hash table $R_HashTable$ and $S_HashTable$, objects with similar hash values are grouped and placed into the same bucket, which represent the objects and their nearest neighbors.

Phase 2. In the second phase, the algorithm accomplishes up two tasks. One is to search for the k nearest neighbors for each object in dataset RS , which is an intersection of datasets R and S ; the other task is to count the number of reverse nearest

neighbors of the objects. Each object o_i in dataset RS is assigned a *HashValue* by executing hash function $hash(o_i)$ (see Line 13). Next, o_i 's k NN can be obtained from the hash table by computing distances between o_i 's *HashValue* and those of the other points in the same bucket (see Lines 14 and 15). In a loop statement block, arrays $R_count[]$ and $S_count[]$ are reserved to store the RNN's number for each object in datasets R and S by scanning each point's k NN (see Lines 16–23).

Phase 3. The third phase identifies the outliers by omitting the intersection (i.e., $RS_{outlier}$) of outlier sets $R_{outlier}$ and $S_{outlier}$. Line 26 computes the local density d_{min} of each query point in RS , where $R_sum[i]$ and $R_count[i]$ are respectively obtained in Lines 17 and 19. If the value of d_{min} is larger than a given threshold ε'_R , object o_i will be envisioned as an outlier in R (see lines 27–29). Line 28 stores the newly identified outlier in $R_{outlier}$. Similarly, we can obtain outliers set $S_{outlier}$ by comparing d_{min} with threshold ε'_S (see Lines 30–33). After executing intersection of outliers sets $R_{outlier}$ and $S_{outlier}$ (see line 35), multi-source outliers set $RS_{outlier}$ is produced (see line 36).

5. Experimental results

This section is dedicated to assessing our proposed *MOD+*'s performance in terms of efficiency, accuracy, and precision. Before presenting the results of our empirical study, we prescribe the goals and setups of our experiments in Section 5.1). Section 5.2 is focused on the efficiency of *MOD+* and its two competitors. We discuss the accuracy and prevision measures of our solution in Sections 5.3 and 5.4, respectively. We verify the effectiveness of our methods in Section 5.5. We demonstrate in Section 5.6 a sample usage of multi-source outlier detection.

5.1. Goals and setups of the experiments

In our experimental evaluation, we consider the *Baseline*, *MOD*, and *MOD+* algorithms elaborated in the previous sections. The primary goal of conducting extensive experiments is to quantitatively examine the performance of the three methods. The secondary goal is to examine the accuracy and precision of the algorithms with various parameters, such as the *sample size*, the number n of outliers, and the number k of nearest neighbors.

In the recent decade, an increasing number of outlier detection methods have been proposed. However, the existing methods can detect outliers in only a single data source, and cannot directly detect outliers from multiple data sources. We can use data fusion methods to combine multi-source datasets into a single dataset; however, important and relevant information may be lost. Unlike traditional outlier detection, we introduce *outlier-join* to validate the behaviors of outliers detected from a dataset R in another dataset S . The two kinds of outlier detection schemes have different research issues and solutions. Therefore, the meaning of the outliers obtained using the proposed methods is likely different from that of the outliers obtained using traditional outlier detection methods.

We implement the three algorithms using Java JDK 10.0.2 and Eclipse IDE 4.7.0. on a server equipped with an Intel(R) Core (TM) i5-4570 CPU and 2 GB main memory. To test the effectiveness of the *Baseline*, *MOD*, and *MOD+* methods, we focus on *Type I* and *Type III* multi-source outliers in the experiments. We ignore *Type II* outliers, because *Type II* can be considered as a special case of *Type I*. We initiate the performance evaluation by testing the efficiency of the three algorithms while varying the sample size and parameters n and k (see Section 5.2). Next, we assess the impacts of the number of objects on the mining accuracy of multi-source outliers (see Section 5.3). We also quantify the precision performance of the three algorithms in Section 5.4.

We use the synthetic datasets in our experiments to easily manipulate crucial parameters (e.g., the number of objects) to evaluate the sensitivity of the algorithms performance against the parameters. In addition, we validate the experimental results obtained using the synthetic data through UCI machine learning repository datasets (see Section 5.5). Through an extensive analysis of the sample usage of multi-source outlier detection, we clarify the differences between multi-source outlier detection and traditional single-source outlier detection.

5.2. Impacts of parameters on mining efficiency

In this group of experiments, We evaluate the efficiency of the proposed algorithms in detecting *Type I* multi-source outliers, with the intersection of two datasets R and S being non-empty.

We randomly generate 40,000 (i.e., $n = 40,000$) points divided into two equal-sized clusters (i.e., R and S), where the data points satisfy a normal distribution with mean 0 and standard deviation 1. Each object in R and S has 200 attributes. Similarly, we randomly generate 10,000 data points augmented to datasets R and S . Next, 200 uniformly distributed data objects are inserted into datasets R and S as outlier data. Fig. 2 plots the impacts of the three critical parameters on the efficiency of the evaluated multi-outlier detection methods.

5.2.1. Impacts of sample size on efficiency

Fig. 2(a) shows that the running times of *MOD* and *MOD+* are less than those of the *Baseline* algorithm in cases with different sample-size cases. In particular, *MOD+* significantly improves the efficiency of the *Baseline* algorithm by an average of 40%. The *MOD* algorithm leverages the k NN-join scheme to reduce *Baseline*'s time complexity (see Section 3.3). The *MOD+*

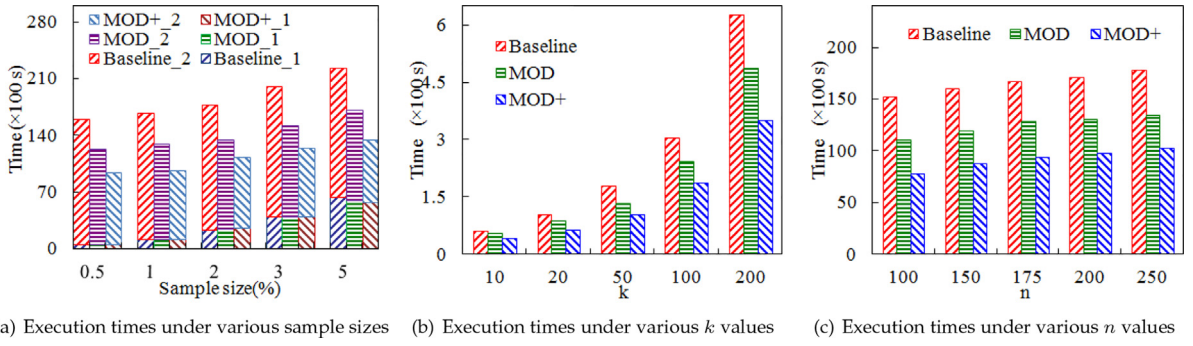


Fig. 2. Impacts of sample size, the number of nearest neighbors, and the number of outliers on the efficiency of the three multi-source outlier detection algorithms.

algorithm employs the *LSH* strategy to implement *kNN* queries, which further optimize the efficiency performance of the *MOD* algorithm by an average of 22%.

Fig. 2(a) shows the execution times for two phases in *Baseline*, *MOD* and *MOD+* algorithms. The first phase (i.e., *Baseline_1*, *MOD_1* or *MOD+_1*) computes the deviation degree ε in a small sample dataset, and the second phase (i.e., *Baseline_2*, *MOD_2* or *MOD+_2*) detects multi-source outliers. The second observation drawn from Fig. 2(a) is that sample size noticeably affects the mining efficiency of *Baseline*, *MOD*, and *MOD+*. The sample size is measured as a ratio between the size of sample dataset and its original dataset size; the measurement unit for the sample dataset is percentage units. The sample datasets are obtained using a random sampling method. The number k of nearest neighbors is set to 20; the number of outliers n is 200. Regardless of the algorithms (i.e., *Baseline*, *MOD*, and *MOD+*), the execution time continuously increases when the sample size varies from 0.5% to 5%. This performance trend is reasonable, because the deviation degree ε is computed in the sample datasets. The overhead of the calculated ε value marginally rises when the sample size is expanding. Moreover, Fig. 2(a) reveals that the running times of our *Baseline*, *MOD* and *MOD+* algorithms in the first phase change slightly at a given sample size because the three algorithms employ the same method to compute deviation degree ε .

5.2.2. Impacts of parameter k on efficiency

Fig. 2(b) reveals the execution times of the *Baseline*, *MOD*, and *MOD+* algorithms as functions of parameter k ranging from 10 to 200. Recall that k - a parameter in *kNN* scheme - is the number of nearest neighbors. The experimental results plotted in Fig. 2(b) show that regardless of the k value, *MOD+* consistently outperform *Baseline* and *MOD* owing to the deployment of the optimal *LSH* scheme. For example, when k is set to 200, *MOD+* cuts back the running times of *Baseline* and *MOD* by 44% and 28%, respectively. These results are consistent with those reported in Fig. 2(a).

The running times of the algorithms rapidly rise with an increasing value of k . In particular, the running-time growth rate becomes accelerated when k is greater than 100. The *kNN* calculation, a core part of the three algorithms, is implemented in the inner loop. Hence, the time complexity of the algorithms is proportional to the parameter k .

5.2.3. Impacts of parameter n on efficiency

Next, we analyze the effect of parameter n on the performance of the three algorithms. Fig. 2(c) shows that the mining times of the algorithms slightly climb as parameter n increases. The reason is two fold. First, the overhead of *Top-n* marginally goes up when the number n of outliers increases. Second, *Top-n* is a dominating factor in the calculation of threshold ε for a sample dataset compared to those for an entire dataset. Hence, from the perspective of the entire dataset, *Top-n* is not a leading player determining the overall mining performance. Not surprisingly, *MOD+* enhances the efficiency of *Baseline* and *MOD* by averages values of 44%, 26%, respectively.

5.3. Impacts of parameters on mining accuracy

In the second group of experiments, we focus on evaluating the mining accuracy of the three algorithms by testing the datasets listed at the beginning of Section 5.2. We define the mining accuracy as the ratio between the number of detected outliers and the total number of outliers. The accuracy is defined as

$$\text{Accuracy} = \frac{TP}{TP + FN}, \quad (6)$$

where TP and FN denote the number of real outliers detected and not detected by the proposed algorithms (i.e., true positives and false negatives), respectively.

We quantitatively assess the impacts of sample size, the number k of nearest neighbors, and the number n of outliers on mining accuracy, as described in Sections 5.3.1, 5.3.2, and 5.3.3, respectively.

5.3.1. Impacts of sample size on accuracy

Fig. 3(a) unravels that the sample size is positively correlated with the mining accuracy performance. As the sample size varies from 0.5% to 5%, the accuracy of *Baseline*, *MOD*, and *MOD+* are significantly improved. For instance, if we change the sample size from 1% to 3%, the accuracy improvements for the *Baseline*, *MOD*, and *MOD+* are 14.2%, 13.8%, and 13.3%, respectively. A large sample size allows these algorithms to accurately capture the data distribution of input datasets. Hence, the deviation degree threshold ε calculated in a large sample dataset is more accurate than that computed in a small sample dataset.

Moreover, Fig. 3(a) shows that the accuracy of *MOD+* is slightly lower than those of *Baseline* and *MOD*. *MOD+* exhibits a lower accuracy because it employs the *LSH* scheme that calculates *kNN* using approximate neighbors as authentic neighbors. The inaccurately determined *k* nearest neighbors deteriorate the *MOD+* algorithm's mining accuracy. Although *MOD+* algorithm has a slight decline in accuracy, the efficiency of outlier detection (see Section 5.2) is significantly increased. In short, *MOD+* algorithm improves the efficiency of outlier detection at low accuracy cost. Fortunately, *MOD* maintains good accuracy as high as that of *Baseline*. In a few cases, *MOD* marginally outperforms *Baseline* in terms of the accuracy (e.g., sample size is set to 0.5% and 1%). Combined with the experimental result of Section 5.2, we recommend *MOD+* algorithm to detect multi-source outliers when a high efficiency is recommended in real-world applications. In contrast, *MOD* algorithm is recommended in real-world applications with high accuracy requirements.

5.3.2. Impacts of parameter *k* on accuracy

Fig. 3(b) illustrates the accuracy performance of *Baseline*, *MOD*, and *MOD+* at various *k* values (i.e., *k* is set to 10, 20, 50, 100, and 200). The accuracy measures of the three algorithms are significantly boosted by increasing the *k* value. A large number *k* of nearest neighbors benefit the accuracy of calculating distance between any pair of two data points.

Interestingly, when *k* falls into a range between 50 and 200, the evaluated accuracy measures become insensitive to the number *k* of nearest neighbors. This trend suggests that increasing the *k* value further beyond 200 has negligible impacts on mining accuracy. As such, the empirical results plotted in Fig. 3(b) indicate that 50 is an optimal number *k* of nearest neighbors for the given dataset. It is evident that the optimal *k* values largely depend on the characteristics of datasets. One may apply sample techniques to capture the data distribution of a large dataset, thereby acquiring an optimal *k* value.

5.3.3. Impacts of parameter *n* on accuracy

Fig. 3(c) illustrates the impacts of parameter *n* – the number of outliers – on the accuracy performance of the three algorithms. The results show that enlarging the *n* value significantly enhances the accuracy of the algorithms. *Baseline*, *MOD* and *MOD+* detect a small number of outliers when parameter *n* is set to a low value. In this case, certain authentic outliers are output as normal data, because the small *n* value limits the number of output outliers. Given a fixed *n* value, the mining accuracy tends to be further reduced when the number of authentic outliers is considerably larger than *n*.

Similar to Fig. 3(b) and c shows that *MOD* and *Baseline* share similar accuracy measures. Although *MOD+* exhibits a slightly lower accuracy, the difference in *MOD+* and *MOD* is dramatically reduced when *n* varies from 100 to 250. The accuracy advantages of the *Baseline* and *MOD* over *MOD+* likely become negligible when *n* exceeds 300.

5.4. Precision using *F1-measure*

In order to distinguish true outliers from false outliers in detected results, we adopt the *F1-measure* [12] method to quantify outlier mining precision. Formally, the *F1-measure* is defined as follows:

$$F1 - measure = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (7)$$

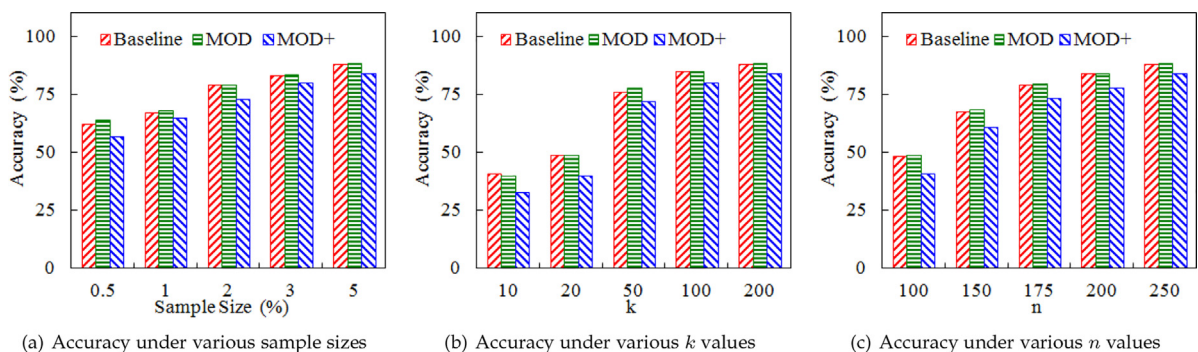


Fig. 3. Impacts of sample size, the number of nearest neighbors, and the number of outliers on the mining accuracy of the three multi-source outlier detection algorithms.

where TP is the number of real outliers detected by the proposed algorithms (i.e., true positives); FP is the number of normal data objects that are misclassified as outliers by our algorithms (i.e., false positives); and FN is the number of real outliers that are not detected by our algorithms (i.e., false negatives). The precision of the algorithms is evaluated by using Eq. (7). It is worth nothing that the precision expressed in Eq. (7) is different from the accuracy quantified by Eq. (6).

Fig. 4 depicts the relationships between the precision of the tested algorithms and three parameters (i.e., sample size, k , and n).

5.4.1. Impacts of sample size on precision

Fig. 4(a) shows that expanding sample size leads to enhanced measures of all three algorithms. Combining the results shown in Figs. 3(a) and 4(a), we can observe that the precision and accuracy of the three algorithms have a similar growing trend with respect to sample size. An intriguing observation is that these algorithms' precision measures are lower than their accuracy measures for a specific sample size. This observation is attributed to the fact that some undetected outliers and misclassified normal data deteriorate the precision performance.

5.4.2. Impacts of parameter k on precision

The precision trend plotted in Fig. 4(b) follows the pattern observed in Fig. 3(b). For example, *Baseline* and *MOD* exhibit a similar precision performance; *MOD+* exhibits a lower precision than *MOD*; and the precision gap between *MOD* and *MOD+* is curtailed when k exceeds 100.

The comparison between Fig. 4(b) and Fig. 3(b) indicates that the precision performance is worse than the corresponding accuracy performance for a given algorithm and k value.

5.4.3. Impacts of parameter n on precision

The comparison of Figs. 4(c) and 3(c) indicates that the precision trend is extremely different from that of the accuracy trend when parameter n varies from 100 to 250. Specifically, the precision measures continue moving up when we increase n from 100 to 175. Such an increasing trend halts at the n value of 175, from which the precision drops with the increasing n value. In contrast to the precision trend, accuracy keeps ascending as n is increased from 200 to 250. When the number TP of real outliers is less than n , an increasing number FP of normal data objects are misclassified as outliers with a growing n value. That is, a large FP value directly leads to a low precision (see Eq. (7)).

5.5. Three types of multi-source outlier detection

We assess the performance of our detection algorithms with respect to three types of multi-source outliers (*Type I–III*). We consider three real-life multi-source and high-dimensional datasets from the *UCI* machine learning repository in this group experiments. The datasets are configured in a way that offers considerable variability in terms of the number of objects and the number of attributes. These three datasets are referred to as *Internet Usage Data*, *Insurance Company Benchmark* (COIL2000), and *Musk* (Version 2), respectively. Table 2 summarizes the characteristics of the three datasets, to which a handful of outliers are injected. The numbers of outliers embedded in the datasets are 101, 90, and 66, respectively.

5.5.1. Outlier detection in Type I

In the case of *Type I* multi-source outliers, the intersection of two datasets R and S is non-empty (i.e., $R \cap S \neq \emptyset$). To resemble characteristics of *Type I* datasets, we divide each *UCI* dataset into two equal-sized clusters (i.e., R and S), where R and S share 10 percent of the same objects. Fig. 5 plots the efficiency and accuracy of the proposed algorithms in detecting multi-source outliers of *Type I*.

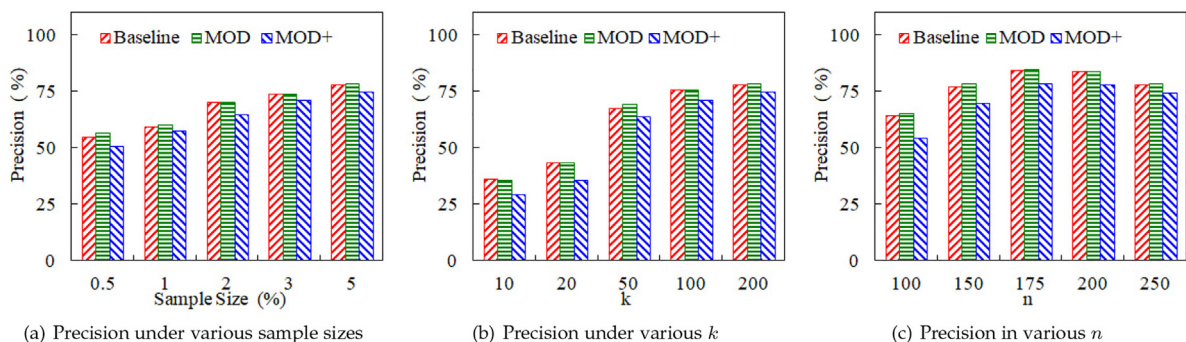


Fig. 4. Impacts of sample size, the number of nearest neighbors, and the number of outliers on the precision performance of the three multi-source outlier detection algorithms.

Table 2

The UCI datasets testing three types of multi-source outliers.

Parameters	InternetUsage Data	COIL 2000	Musk
The number of objects	10104	9000	6598
The number of attributes	72	86	168
The number of outliers	101	90	66

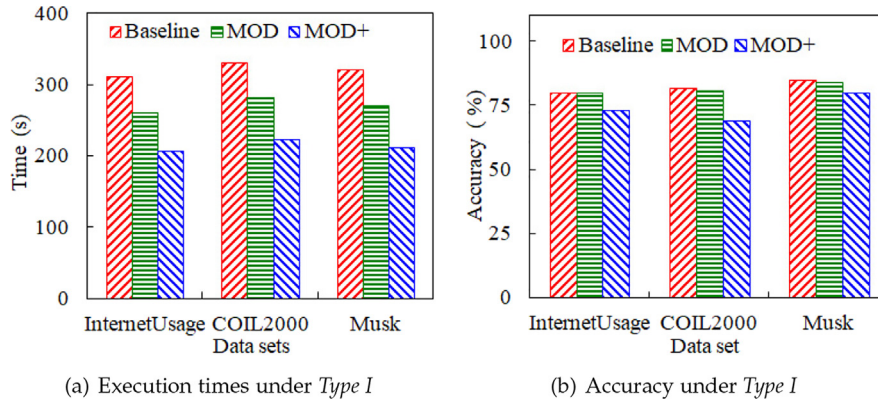
**Fig. 5.** The execution times and accuracy of the three algorithms detecting multi-source outlier Type I.

Fig. 5(a) unravels that compared with the *Baseline* algorithms, *MOD* and *MOD+* significantly accelerate the outlier detection process on the three UCI datasets. Specifically, *MOD* and *MOD+* curtail the execution time of the *Baseline* solution by up to 16.0% and 33.6%, with averages of 15.5% and 33.2%, respectively. The *MOD* algorithm leverages the *kNN*-join scheme to slash *Baseline*'s time complexity (see Section 3.3). With *MOD* in place, the *MOD+* algorithm employs the *LSH* strategy to implement *kNN* queries, which further optimize *MOD*'s efficiency performance.

Fig. 5(b) shows that all three algorithms can effectively detect multi-source outliers; the accuracy of these algorithms is maintained at a fairly high level. It is worth noting that the accuracy of *MOD+* is slightly lower than those of *Baseline* and *MOD*, because *MOD+* trades accuracy for speed by employing *LSH* to locate approximate neighbors of each object. The approximate feature of *LSH* is a key factor for the reduced accuracy of *MOD+*. Nevertheless, *MOD+* delivers a higher efficiency than *Baseline* and *MOD* (see Fig. 5(a)). We conclude that *Baseline*, *MOD* and *MOD+* are feasible and effective approaches to detect Type I multi-source outliers.

5.5.2. Outlier detection in Type II

In the case of Type II multi-source outliers, dataset R is a subset of S (i.e., $R \subset S$). To capture the dataset characteristics of Type II, we treat each UCI dataset (see Table 2) as dataset S while selecting 50% of objects in S to construct dataset R . Thus, we have $R \cap S = R$. The three UCI datasets (see Table 2) are processed using the above mentioned steps. Fig. 6 plots the efficiency and accuracy of the proposed algorithms handling multi-source outliers Type II.

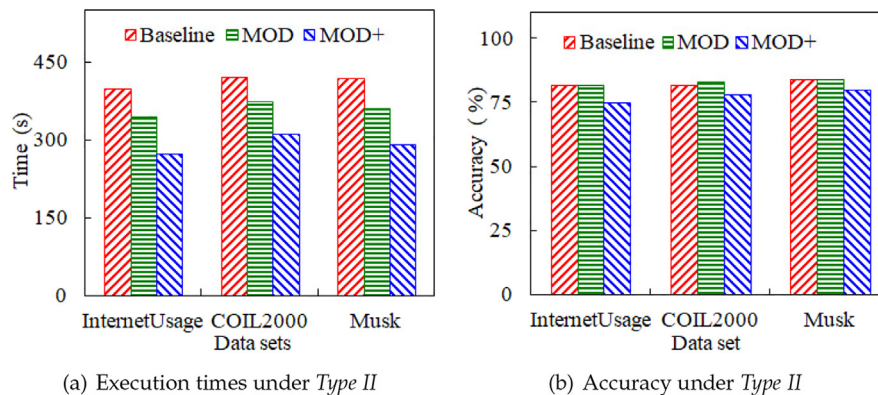
**Fig. 6.** The execution times and accuracy of the three algorithms detecting multi-source outlier Type II.

Fig. 6(a) exhibits the running times of the three algorithms discovering multi-source outliers *Type II*. The experimental results plotted in Fig. 6a indicate that *MOD* and *MOD+* deliver superb performance compared with *Baseline*. A trend similar to that illustrated in Fig. 5(a) is observed. The execution periods of the three algorithms in the case of *Type II* multi-source outliers are longer than those in the *Type I* case. The phenomenon occurs because the number of objects of $R \cap S$ in the *Type II* case is larger than that in the *Type I* case. The large number of objects leads to a notable processing time. Fig. 6(b) reveals that the accuracy of *MOD+* is slightly lower than those of the *Baseline* and *MOD* in multi-source outliers *Type II*. The accuracy trends plotted in Fig. 6(b) are consistent with those in Fig. 5(b). The experimental results confirm that similar to *Type I* multi-source outlier detection, the proposed algorithms are suitable for detecting *Type II* outliers.

5.5.3. Outlier detection in Type III

Recall that in multi-source outliers *Type III*, R and S are two independent datasets without sharing any objects (i.e., $R \cap S = \emptyset$). To construct datasets for the *Type III* scenarios, we divided each *UCI* dataset into two equal-sized clusters (i.e., R and S), where R and S share no objects. In this experiment, the *MOD* and *MOD+* algorithms are limited by the fact that the intersection of two datasets is not empty (i.e., $R \cap S \neq \emptyset$). Therefore, we only validate *Baseline* algorithm in the experiments. Fig. 7 plots the efficiency and accuracy of *Baseline* in the context of *Type III* multi-source outliers.

Fig. 7(a) reveals that *Baseline*'s efficiency in datasets *Internet Usage* and *COIL2000* is better than in dataset *Musk*. The reason is that the large number of attributes in dataset *Musk* leads to long time spent in computing distance between two objects due to its high complexity (i.e., $O(n \times n)$, n is the number of attribute), which enlarges the processing time of *Baseline*. Fig. 7(b) exhibits that the accuracy of *Baseline* in the three *UCI* datasets. Importantly, *Baseline* delivers much better accuracy in dataset *Musk* than in the *InternetUsage* and *COIL2000* cases, because the large number of attribute in *Musk* boosts accuracy of measuring distances among objects, which improves the accuracy of *kNN* calculation. In a nutshell, the *MOD* and *MOD+* algorithms are inapplicable for the *Type III* case, because computing the intersection between R and S is a cornerstone step of *MOD* and *MOD+*.

5.6. Multi- vs. single-source outliers: A sample usage

We demonstrate the use of our solutions to clarify the difference between multi-source outlier and single-source outlier (i.e., traditional outlier). The experiments are driven by two *IPUMS* Census Database (i.e., *ipums.la.97* and *ipums.la.98*), which contain unweighted *IPUMS* census data from the Los Angeles and Long Beach areas in 1997 and 1998. Datasets *ipums.la.97* and *ipums.la.98*, which share the same attributes, are regarded as datasets R and S , respectively; the data objects in *ipums.la.97* and *ipums.la.98* are merged to form $R \cup S$. Since *ipums.la.97* and *ipums.la.98* do not share any data object, the multi-source outlier detection belongs to the *Type III* case.

We apply the *Baseline* algorithm to detect multi-source outliers and obtain 175 outlier results, several of which are displayed in Table 3. The first multi-source outlier shows that a person with a total income of less than \$705 lives with his or her mother. This outlier phenomenon in 1997 became normal data in 1998. The experimental result indicates that \$705 was insufficient to buy a house in 1998, and thus, the individual was required to stay with her or his mother. The second multi-source outlier suggests that families with a total income of \$14,615 could own their own houses in 1997, but this phenomenon was rare in 1998. The result is justified by the significantly surge of the U.S. housing price in 1998.

The outlier results displayed in Table 3 can be represented by a Venn figure (see Fig. 8). Fig. 8 contains three circles named Outlier_ R , Outlier_ S , and Outlier_ $R \cup S$. Outlier_ R and Outlier_ S represent the outliers detected from R and S , respectively. Outlier_ $R \cup S$ is a set of outliers detected in a data set combining R and S . The three circles divide the Venn graph into seven regions, which are filled with seven colors. The outlier results of each region are different from those in the other regions, which can be explained as follows.

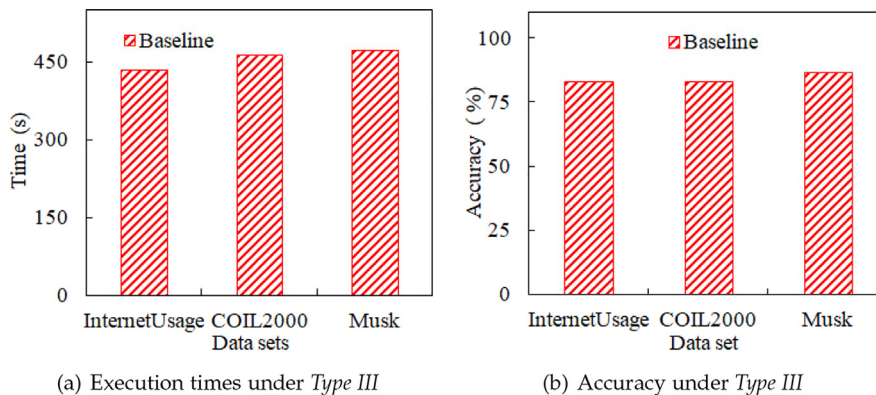
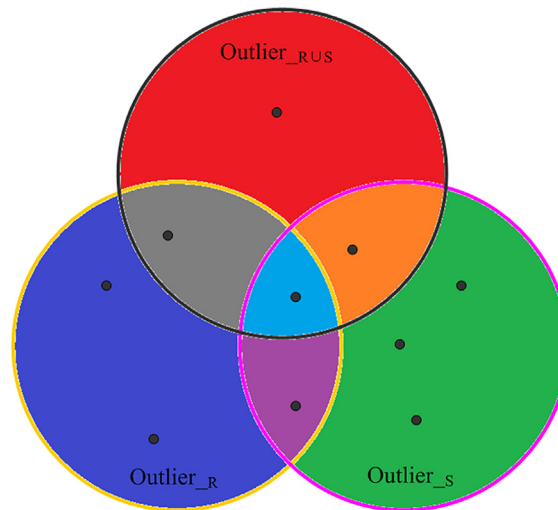


Fig. 7. The execution times and accuracy of the three algorithms detecting multi-source outlier *Type III*.

Table 3A sample usage: Multi-source outliers detected from *ipums.la.97* and *ipums.la.98*.

No.	Information object's feature	Is it outlier?		
		in R	in S	in $R \cup S$
1	inctot(705); momloc(1)	Yes	No	No
2	ownershg(1); ftotinc(14615)	No	Yes	No
3	ownershg(1); nchlt5(1)	No	Yes	No
4	steppop(1); educrec(4)	Yes	No	Yes
5	school(1); schltpe(1)	Yes	No	No
6	poploc(1); marst(0); inctot(12510)	No	Yes	No
7	inctot(54010); ftotinc(75000)	Yes	Yes	Yes
8	vetstat(2); inctot(17310); hrswork2(8)	Yes	Yes	No
9	steppop(1); inctot(54010)	No	No	Yes
10	ownershg(1); ftotinc(75000)	No	Yes	Yes

**Fig. 8.** A Venn diagram of multi-source outliers.

- The dots in blue (or green) region indicate that some objects are outliers in R (or S), but they are normal data point in S (or R) and $R \cup S$. These outliers can be used to study social phenomena in Los Angeles and Long Beach from 1997 to 1998.
- The dots in red (or violet) region indicate that several objects are global (or local) outliers, which can be used to analyze global (or local) social phenomena.
- The dots in other regions represent both global and local outliers, which can be used to analyze common social phenomena.

In other words, our algorithm can detect the multi-source outliers shown in the Venn figure. These multi-source outliers cannot be obtained using the existing data fusion technology because the outliers are hiding as normal data in a merged dataset (i.e., $R \cup S$). Please refer to Section 1.1 for detailed discussions on data fusion and multi-source outlier detection.

6. Future discussions

Recall that we addressed three types of multi-source outliers in Section 3.1. There exists a fourth type that is different from the three considered types of multi-source outliers. In this new type of outlier detection, datasets may appear independent of one another. Once information retrieved from multiple sources is integrated, new knowledge may be discovered. We describe an excellent example in which an organization utilizes mining data from various data sources to unveil valuable information that cannot be acquired from any individual data source. The detailed information regarding this type of multi-source data mining can be found in the literature [13].

An Example [13]. The Australian Taxation Office (a.k.a., ATO) mines outliers from multiple data sources, such as social media posts, private school records, and immigration data, to detect tax fraudulence. The mining outliers from multifaceted data sources has emerged as a sophisticated tool to detect tax cheats that yielded nearly 10 billion in 2016. For example, in an ordinary Australian family, a husband reported \$80,000 in taxable income each year, which placed the taxpayer at the second lowest tax bracket. This taxpayer's wife reported an annual income of \$60,000. The private school records displayed

that the family had three children at private schools, and the cost was up to \$75,000 per year. Moreover, the immigration records and social media posts indicated the occurrence of five business-class flights and a vacation in Canadian ski resorts. Consequently, a conclusion drawn from the multiple data sources that the income declared by this couple doesn't in line with the family's lifestyle. As a result, ATO contacted the family to confirm if the couple had any unpaid tax. Therefore, it is critical and indispensable to develop advanced data-mining techniques to discover insightful information from multiple data sources to facilitate decision-makings.

In this example depicted in Fig. 9, datasets R and S have a different attribute structure; however, R and S may share a few attribute dimensions (e.g., primary key). R and S can be connected through the shared attributes, and thus, a new set $R \sqcup S$ is constructed. $\langle O_1, O'_1 \rangle$ is an object pair, where O_1 and O'_1 have the same attribute value derived from datasets R and S respectively. The goal in this type of outlier detection is to pinpoint some object pairs (e.g., $\langle O_1, O'_1 \rangle$) that are outliers in $R \sqcup S$. This type of outlier detection plays a vital role in credit-card and tax fraudulence detection and similar detection services.

Algorithm 4 is the pseudocode of the fourth type of multi-source outlier detection, which is comprised of four steps.

- **Step 1.** Datasets R and S are connected using any shared attributes; a new dataset M is generated (see Line 1).
- **Step 2.** An empty hash table $hashTableM$ is initialized; all the objects in dataset M are mapped into table $hashTableM$ (see Lines 2–6).
- **Step 3.** k nearest neighbors of each object o_i in dataset M are identified; the sum of the distance d_{min} between o_i and its k neighbors is computed (see Lines 7–13).
- **Step 4.** Objects with d_{min} larger than a given threshold ε are output as multi-source outliers (see Line 14).

Algorithm 4. Type IV outlier detection algorithm: MOD^+

Input: R, S, k, ε
Output: $RS_{outlier}$; 1: $RS \leftarrow R \oplus S$; // Connect R and S by keyword, then generate a new dataset M .
 2: Create an empty hash table $HashTableM$;
 3: **for** (int $i = 0$; $i < RS$; $i++$) **do**
 4: $HM_i \leftarrow \text{hash}(o_i)$; // computing object o_i 's hash value
 5: $HashTableM \leftarrow \text{insert}(o_i, HM_i)$;
 6: **end for**
 7: **for** (int $i = 0$; $i < RS$; $i++$) **do**
 8: $neighbor \leftarrow \text{find}(\text{hash}(o_i), HashTableRS)$;
 9: $d_{min} \leftarrow \text{sum}(o_i, D_i)$; // select kNN from list D_i , and calculate sum between o_i and its kNN
 10: **if** ($d_{min} \geq \varepsilon$) **then**
 11: $RS_{outlier} \leftarrow \text{object } o_i$;
 12: **end if**
 13: **end for**
 14: **output**($RS_{outlier}$);

Although we propose multi-source outlier detection techniques to identify trustworthy outliers in multiple datasets, there are some shortcomings as follows. Firstly, we did not establish a unified model for multi-source data outlier, which can include four types of multi-source outliers. Secondly, our proposed algorithms are simple ways to solve multi-source outlier problem cannot be applied to high-dimensional data due to the limitation of k -nearest neighbor. We will study some optimization algorithms of multi-source outlier detection by using correlative subspace in future work. Finally, our algorithms cannot detect multi-source outliers from big data. We aim to implement a multi-source outlier parallel algorithm on a Spark cluster to exploit the in-memory computing environment to process large-scale multi-dimensional datasets.

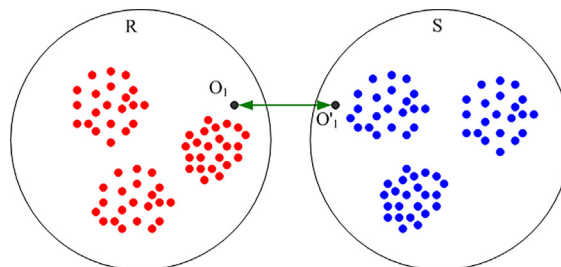


Fig. 9. A fourth type of multi-source outliers.

7. Related work

In this section, we review existing techniques in the areas of multi-source/multi-view outlier detection [14], and k NN- and RNN-based outlier detection [15].

7.1. Multi-source and multi-view outlier detection

Wang et al. reviewed recent progresses in the area of mining data from multiple data sources [13]. Recently, much attention has been paid towards preprocessing [16], association rules [17], clustering [18,19], and classification [20] in multiple sources. In contrast to the conventional outlier detection scheme involving the vertical relationship among data samples, Gao et al. leveraged the inconsistency of horizontal relationship among data samples from multiple data sources to mine outliers [21].

Multi-source outlier detection research is still in a nascent stage. Most existing multi-view outlier detection techniques classify outliers as (1) attribute-oriented outliers and (2) class-oriented outliers. The former one refers to outliers that have consistently abnormal behaviours in each view, whereas the latter one means outliers that are inconsistent among various views.

Wang et al. proposed a fuzzy-clustering-based distributed approach for multi-view anomaly detection, which simultaneously learns a membership degree matrix for each view and then detects anomalies for all parties [22]. Guo et al. solved the problem of partial multi-view outlier detection [23]. Gao et al. divided data set into multiple cubes, and detected outlier from these cubes [24]. Sheng et al. proposed a nearest-neighbor-based multi-view anomaly detection approach or *MUVAD* [25]. In the *MUVAD* study, an anomaly measurement criterion was introduced, followed by the formulating of the *MUVAD* objective to explicitly estimate the set of normal instances.

7.2. k NN and k NN-based outlier detection

Hinneburg et al. argued that the nearest-neighbor queries can be defined using commonly metrics such as Manhattan and Euclidean distances [26]. The determination of k nearest neighbors is a key step.

Several researchers have focused on nearest-neighbor queries in the context of high-dimensional datasets. A basic algorithm to realize nearest-neighbor queries is a branch-and-bound method using R^* -tree [27]. By the virtue of trading off accuracy for efficiency, approximate solutions to search k nearest neighbors have become a promising direction. Tao et al. developed a new access method known as the spatial-inverted index, which extends the conventional inverted index to multi-dimensional data [28]. In *voronoi*-based nearest neighbor searching, Zhang et al. extended the notion of *voronoi* cells to support uncertain objects [29]; the *PV-index* method was developed to evaluate probabilistic nearest-neighbor queries over uncertain data.

The fundamental idea of neighbor-based outlier detection techniques is to make use of distances or densities among data points to their nearest data points as a measurement metric to determine outliers. Such approaches can be viewed as an extension of distance-based methods or density-based methods. State-of-the-art k NN-based outlier detection algorithms can be used to filter out normal data and locate a small number of outliers.

Ramaswamy et al. proposed a novel formulation based on the distance of a point from its k th nearest neighbour. In this scheme, the top n points are defined as outliers. [30]. Wang et al. developed a new MST-inspired (minimum spanning tree) k NN-based outlier detection method that detects both global and local outliers [31]. Wang and Li proposed a weighted neighbourhood information network for outlier detection, where they fully consider the role of neighborhood relationship [32]. Yang and Huang put forward a k NN-based outlier detection algorithm, which consists of two phases [33]. This approach effectively avoids frequent passing an entire dataset to cut back unnecessary computations.

7.3. RNN and RNN-based outlier detection

The RNN query was first introduced by Korn [34]. A recent study conducted by Yang et al. compares an array of RNN query techniques [35]. Most of these techniques utilize a pruning and verification framework to significantly reduce the search space. For example, Tao et al. proposed TPL, which is arguably the most popular algorithm for RNN queries [36]. TPL is the first algorithm using the concept of *half-space* pruning to implement RNN queries. TPL inspired a handful of following-up studies.

The RNN search, which is one of the most fundamental operations, has been extensively studied in data-mining fields like outlier detection and clustering. The notion of reverse nearest neighbors is of importance in outlier detection. Radovanovic et al. designed an unsupervised outlier detection scheme based on reverse nearest neighbors [10]. Outliers are measured by detecting the frequencies in a nearest neighbor set. This method, being only applicable to unsupervised learning, cannot be extended to supervised and semi-supervised. Sahu et al. proposed to apply the reverse neighbour outlier factor to measure outlierness degrees of the objects in a dataset [37]. Zhang and Liang developed a data-stream outlier detection algorithm *SODRNN* taking into account reverse nearest neighbours [38]. An approach to detecting outliers based on reverse neighbours was thoroughly presented in [39], judging that a point is an outlier if it has a zero k -occurrence count. Tang et al. described a

simple outlier detection technique using the local kernel density estimation [40]. The local kernel density estimation seamlessly integrates the k -nearest neighbour and the reverse nearest neighbours techniques.

8. Conclusions and future work

In this study, we proposed the concept of multi-source outliers, the sample usages of which were demonstrated to highlight its significance. After designing a baseline approach to detect multi-source outliers, we developed an optimal algorithm *MOD* incorporating the k NN-join operator. Combining k NN and RNN, we proposed a new outlier measurement scheme *MOD* +, which was applied to identify multi-source outliers. Experimental results based on synthetic and real-world datasets validated the superiority of the proposed *MOD* and *MOD* + schemes against the *Baseline* algorithm in terms of running time throughout all the test scenarios. Compared with *MOD*, the *MOD* + algorithm employs the *LSH* strategy to implement k NN queries, which further optimize the outlier-detection efficiency.

The execution times of *MOD* and *MOD* + increase as the sample size increases because deviation degree ε is computed in sample datasets. The k NN calculation is a core part of the three algorithms and; consequently, the running times of the algorithms rapidly rise with an increasing value of k . The results indicate that the mining cost of the algorithms slightly goes up with an increasing value of parameter n due to the overhead of *Top-n*.

More importantly, when the sample size varies from 0.5% to 5%, the mining-accuracy performance of *MOD* and *MOD* + is significantly boosted. Increasing the k and n values dramatically enhance the accuracy and precision of *MOD* and *MOD* +.

We present a systematic plan to extend our work in four directions. First, we aim at developing optimal multi-source outlier algorithms by using related subspace [6]. Second, the fourth type of multi-source outlier detection must be further examined. Third, we intend to design a parallel algorithm for *MOD* and *MOD* +. We expect to implement the parallel algorithm on a Spark cluster, taking a full advantage of the in-memory computing environment to process large-scale multi-dimensional datasets. Finally, we will apply *MOD* and *MOD* + to solve real-world data mining problems. For example, it is intriguing to leverage *MOD* and *MOD* + as a toolkit to investigate astronomical spectral data and to identify unknown stars.

CRedit authorship contribution statement

Xujun Zhao: Data curation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of P. R. China (61876122), and the Natural Science Foundation for Young Scientists of Shanxi Province of China (No. 201901D211303), Scientific and Technological Innovation Team of Shanxi Province (No. 201805D131007). Xiao Qin's work is supported by the U.S. National Science Foundation under Grants IIS-1618669, CCF-0845257 (CAREER).

References

- [1] V.R. Ganji, S.N.P. Mannem, Credit card fraud detection using anti- k nearest neighbor algorithm, *International Journal on Computer Science and Engineering* 4 (6) (2012) 1035–1039.
- [2] Y. Ma, J. Zhang, J. Cai, H. Yang, X. Zhao, Parallel extraction and analysis of abnormal features of qso spectra based on sparse subspace, *Spectroscopy and Spectral Analysis* 41 (4) (2021) 1086–1091.
- [3] C. Qu, Y. Haifeng, C. Jianghui, X. Yaling, P-cygni profile analysis of the spectrum: Lamost j152238.11+333136.1, *Spectroscopy and Spectral Analysis* 40 (4) (2020) 1304–1308.
- [4] G. Kim, S. Lee, S. Kim, A novel hybrid intrusion detection method integrating anomaly detection with misuse detection, *Expert Systems with Applications* 41 (4) (2014) 1690–1700.
- [5] J. Li, J. Zhang, X. Qin, Y. Xun, Feature grouping-based parallel outlier mining of categorical data using spark, *Information Sciences* 504 (2019) 1–19.
- [6] J. Zhang, X. Yu, Y. Xun, S. Zhang, X. Qin, Scalable mining of contextual outliers using relevant subspace, *IEEE Transactions on Systems Man and Cybernetics Systems* 50 (3) (2020) 988–1002.
- [7] L. Zhang, S. Wang, G. Fu, Z. Wang, L. Cui, J. Hou, Multi-source manifold outlier detection, in: *International Conference on Computational Science*, Springer, 2019, pp. 44–58.
- [8] X. Zhao, J. Zhang, X. Qin, knn-dp: Handling data skewness in knn joins using mapreduce, *IEEE Transactions on Parallel and Distributed Systems* 29 (3) (2018) 600–613.
- [9] W. Jin, A.K. Tung, J. Han, Mining top- n local outliers in large databases, in: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2001, pp. 293–298.
- [10] M. Radovanović, A. Nanopoulos, M. Ivanović, Reverse nearest neighbors in unsupervised distance-based outlier detection, *IEEE Transactions on Knowledge and Data Engineering* 27 (5) (2014) 1369–1382.
- [11] A. Stupar, S. Michel, R. Schenkel, Rankreduce-processing k -nearest neighbor queries on top of mapreduce, in: *Large-Scale Distributed Systems for Information Retrieval*, ACM, 2010, pp. 13–18.
- [12] D.M.W. Powers, Evaluation: From precision, recall and f -measure to roc, informedness, markedness and correlation, *Journal of Machine Learning Technologies* 2 (1) (2011) 37–63.

- [13] R. Wang, W. Ji, M. Liu, X. Wang, J. Weng, S. Deng, S. Gao, C.-A. Yuan, Review on mining data from multiple data sources, *Pattern Recognition Letters* 109 (2018) 120–128.
- [14] Y. Peng, A. Tan, J. Wu, Y. Bi, Hierarchical edge computing: A novel multi-source multi-dimensional data anomaly detection scheme for industrial internet of things, *IEEE Access* 7 (2019) 111257–111270. .
- [15] J. Zhao, M. Shi, G. Hu, X. Song, C. Zhang, D. Tao, W. Wu, A data-driven framework for tunnel geological-type prediction based on tbn operating data, *IEEE Access* 7 (2019) 66703–66713. .
- [16] Y. Lin, X. Hu, X. Wu, Quality of information-based source assessment and selection, *Neurocomputing* 133 (2014) 95–102.
- [17] X. Wu, S. Zhang, Synthesizing high-frequency rules from different data sources, *IEEE Transactions on Knowledge and Data Engineering* 15 (2) (2003) 353–367.
- [18] Y. Yang, J. Cai, H. Yang, J. Zhang, X. Zhao, Tad: A trajectory clustering algorithm based on spatial-temporal density analysis, *Expert Systems with Applications* 139 (2020) 112846.
- [19] J. Cai, H. Wei, H. Yang, X. Zhao, A novel clustering algorithm based on dpc and pso, *IEEE Access* 8 (2020) 88200–88214. .
- [20] Y. Lin, X. Hu, X. Wu, Ensemble learning from multiple information sources via label propagation and consensus, *Applied Intelligence* 41 (1) (2014) 30–41.
- [21] J. Gao, W. Fan, D. Turaga, S. Parthasarathy, J. Han, A spectral framework for detecting inconsistency across multi-source object relationships, in: 2011 IEEE 11th International Conference on Data Mining, IEEE, 2011, pp. 1050–1055.
- [22] H. Wang, T. Chen, H. Wang, X. Shao, P. Su, Fuzzy clustering based anomaly detection for distributed multi-view data, in: 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, 2018, pp. 1–7.
- [23] J. Guo, W. Zhu, Partial multi-view outlier detection based on collective learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, AAAI Press, 2018, pp. 298–305. .
- [24] J. Gao, W. Ji, L. Zhang, A. Li, Y. Wang, Z. Zhang, Cube-based incremental outlier detection for streaming computing, *Information Sciences* 517 (2020) 361–376.
- [25] X.-R. Sheng, D.-C. Zhan, S. Lu, Y. Jiang, Multi-view anomaly detection: Neighborhood in locality matters, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 4894–4901. .
- [26] A. Hinneburg, C.C. Aggarwal, D.A. Keim, What is the nearest neighbor in high dimensional spaces?, in: 26th Internat Conference on Very Large Databases, 2000, pp. 506–515.
- [27] G.R. Hjaltason, H. Samet, Distance browsing in spatial databases, *ACM Transactions on Database Systems (TODS)* 24 (2) (1999) 265–318.
- [28] Y. Tao, C. Sheng, Fast nearest neighbor search with keywords, *IEEE Transactions on Knowledge and Data Engineering* 26 (4) (2013) 878–888.
- [29] P. Zhang, R. Cheng, N. Mamoulis, M. Renz, A. Züfle, Y. Tang, T. Emrich, Voronoi-based nearest neighbor search for multi-dimensional uncertain databases, in: 2013 IEEE 29th International Conference on Data Engineering (ICDE), IEEE, 2013, pp. 158–169. .
- [30] R. Sridhar, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data: Dallas, Texas, United States, vol. 15, no. 18, 2000, pp. 427–438. .
- [31] X. Wang, X.L. Wang, Y. Ma, D.M. Wilkes, A fast mst-inspired knn-based outlier detection method, *Information Systems* 48 (2015) 89–112.
- [32] Y. Wang, Y. Li, Outlier detection based on weighted neighbourhood information network for mixed-valued datasets, *Information Sciences* 564 (2021) 396–415.
- [33] P. Yang, B. Huang, Knn based outlier detection algorithm in large dataset, in: International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing, vol. 1, 2008, pp. 611–613.
- [34] F. Korn, S. Muthukrishnan, Influence sets based on reverse nearest neighbor queries, *ACM Sigmod Record* 29 (2) (2000) 201–212.
- [35] S. Yang, M.A. Cheema, X. Lin, W. Wang, Reverse k nearest neighbors query processing: experiments and analysis, *Proceedings of the VLDB Endowment* 8 (5) (2015) 605–616.
- [36] Y. Tao, D. Papadias, X. Lian, X. Xiao, Multidimensional reverse knn search, *The VLDB Journal* 16 (3) (2007) 293–316.
- [37] S.K. Sahu, S.K. Jena, M. Verma, K-nn based outlier detection technique on intrusion dataset, *International Journal of Knowledge Discovery in Bioinformatics (IJKDB)* 7 (1) (2017) 58–70.
- [38] Z.P. Zhang, Y.X. Liang, A data stream outlier detection algorithm based on reverse k nearest neighbors, in: Advanced Materials Research, Trans Tech Publications, 2011, pp. 1032–1035.
- [39] J. Lin, D. Etter, D. DeBarr, Exact and approximate reverse nearest neighbor search for multimedia data, in: Proceedings of the 2008 SIAM International Conference on Data Mining, SIAM, 2008, pp. 656–667.
- [40] B. Tang, H. He, A local density-based approach for outlier detection, *Neurocomputing* 241 (2017) 171–180.