A Distributed Density-based Outlier Detection Algorithm on Big Data

Lin Mei[1,2] and Fengli Zhang[1]
(Corresponding author: Fengli Zhang)

School of information and software engineering, University of Electronic Science and Technology of China[1]
Chengdu, China
School of Computer Science and Technology, Southwest Minzu University[2]
Chengdu, China
(Email: fzhang@uestc.edu.cn)

(Received Mar. 11, 2019; Revised and Accepted Nov. 6, 2019; First Online Jan. 29, 2020)

Abstract

As one of popular issues in data mining area, outlier detection aims to find the objects which show abnormal behaviors from original datasets' distribution, and it can be applied in various applications such as bank fraud, network intrusion detection, system health monitoring, medical care, public safety and security, and etc. Recently, the density-based outlier detection has been proposed which is the highly efficient and significant method for outlier detection processing. It adopts the relative density of an object to indicate the degree of an object is an outlier compared with its neighbors. Specifically, it aims at computing the Local Outlier Factor (LOF). In this paper, we propose a novel distributed density-based outlier detection method for large-scale data processing, namely IGBP. First, we split the data space into several grids and then allocates these grids into the data nodes with greedy algorithm in a distributed environment. Besides, we propose a distributed LOF computing method with KD-tree for detecting density-based outliers in parallel way. The validity of the proposed approaches is finally verified by experiments, Experimental results which demonstrate that our proposed method outperform the baselines.

Keywords: Density-based Outlier; Distributed Algorithm; Greedy Algorithm; Kd-Tree; LOF

1 Introduction

With the development of big data techniques in large scale data processing, outlier detection is one of importance but complex tasks in data mining area, it can be widely used in various applications such as bank fraud, network intrusion detection, system health monitoring, medical care and public security protection, and etc. Outlier detection can help us to discover valuable knowledge and abnormal patterns. Therefore, it has become one of hotspot directions in data mining, recently.

Hawkins who has addressed that an outlier is an observation that deviates obviously from other observations as to arouse suspicion that it was generated by a different mechanism [9]. In recent years, there are many studies about outlier detection. For example, distance-based outlier detection [13] and density-based outlier detection [5] are well representative works in traditional outlier detection methods. However, most of them only consider the centralized environments or single node processing. With the increasing scale of big data, the performance of these proposed methods cannot satisfy computing requirements of users. For instance, in the area of credit card fraud detection, we obtain the users' trade information as a dataset. If the credit card is stolen, its transaction pattern usually changes dramatically, especially that the locations of transactions and the purchased items are often unusual for the authentic card owner. Therefore, we define these abnormal transaction records as outliers. Then, the techniques of the outlier detection can help us to identify outliers to find out whether user accounts have been theft. And it can avoid the property damage. Besides, Outlier detection technology also can used to detect the cheating of game bots in Massively Multiplayer Online Role-playing Games [17].

Fortunately, there are some recent studies which attempt to utilize distributed computing environment to speed up the computation, and there are several related methods [10,11,14] for distributed outlier detection were proposed. For example, E. Lozano and E.Acufia [16] propose a master-slave architecture for distributed computing. More specifically, each slave node computes its neighborhood set and sends it to the master node. And the master node will collect all the partial neighborhood sets and compute LOFs of all the tuples. However, a large number of tuples in proposed method are aggregated to the master node, and then lots of calculations are needed to conduct the result. Thus, the master node will be the bottleneck when the data scale is increasing. And a high-performance master node is necessary. Recently, Mei Bai et al. adopt a coordinator and a number of datanodes for outlier detection problem [3]. The coordinator is responsible for the overall scheduling. Each datanode stores several data subsets in grids, and calculates LOFs in the data subsets. The coordinator in this frame only takes charge of the scheduling and all the actual computations are allocated to the datanodes. Thus, the coordinator would not be a bottleneck if the data scale is large. In order to reduce the network overhead, their grid allocation algorithm allocates the adjacent grids to the same datanodes. However, their proposed algorithm only need a small quantity of network communications between pairs of datanotes and the average numbers of tuples in each grid are likely to be very different. Hence, to address these limitations, we focus on improve the computational complexity which is the greatest bottleneck of this issue.

In this paper, we aim to model density-based outlier detection in a distributed manner. The general idea is that we attempt to compare the density around an object with the density around its local neighbors. The basic assumption of density-based outlier detection method is that the density around a non-outlier object is similar to the density around its neighbors, but the density around an outlier object is significantly different from the density around its neighbors [8]. Through our sufficient analysis, we discover that workload and network communications in computing architecture will increase while the scale of data is increasing. But the increased speed of workload is higher than network communication. Besides, if the dimensionality of data increases, the performance will be much better. Therefore, we propose an improved algorithm in this paper which aims at detecting density-based outliers in distributed environments efficiently compared to [3]. Moreover, our experiments are implemented to demonstrate the effectiveness and high efficiency. We will show the detailed description of algorithm in Section 4.

The rest of this paper is organized as follows. In Section 2, we will overview the related works. Section 3 states the problem of density-based outlier detection in a distributed environment. Section 4 detailedly presents our improved algorithm. Section 5 gives the experimental results. In the end, we conclude this paper in Section 6.

2 Related Work

We first make briefly overview of outlier detection in Section 2.1. Then the previous methods of distributed outlier computing are described in Section 2.2.

2.1 Outlier Detection

Hawkins firstly give a definition about outliers in 1980 [9]. Afterwards, Beckman and Cook [4] present more improved definition and survey. Markou and Singh [18, 19] present a review about statistical approaches for outlier detection. Especially in [19], they present a review about neural network based approaches for outlier detection. Fujimaki *et al.* present a semi-supervised outlier detection method by using a set of labeled "normal" objects [7]. Dasgupta and Majumdar also propose a semi-supervised method [6] for outlier detection task. Subsequently, distance-based outliers was developed by Knorr and Ng [13]. And the index-based, nested loop–based and grid-based approaches are well explored to speed up distance-based outlier detection [12, 13].

Other proximity-based approach is the density-based outlier detection [5], In order to detect a tuple whether is an outlier or not, a local outlier factor (LOF) which defines in [5] represents the degree of this tuple to be an outlier is assigned to each tuple [3]. LOF is based on a concept of a local density, where locality is given by k nearest neighbors, whose distance is used to estimate the density. By comparing the local density of a tuple to the local densities of its neighbors, one can identify the regions of similar density, and tuples that have a substantially lower density than their neighbors. These are considered to be outliers [8]. Besides, the HilOut algorithm was proposed by Angiulli and Pizzuti [2]. Aggarwal and Yu [1] develop the sparsity coefficient-based subspace outlier detection method. Kriegel *et al.* proposed angle-based outlier detection [15].

2.2 Outlier Detection in Distributed Environments

Lozano and Acufia propose a distributed algorithm to compute density-based outliers [16]. However, owing to all the tuples are transferred to the master node, the workload on the master node is quite heavy. Thus, this method is unable to achieve good performance when the data scale is huge.

Mei *et al.* adopt a coordinator and a number of datanodes [3]. And the coordinator is responsible for the overall scheduling. Each datanode stores several data subsets in grids, and calculates LOFs in the data subsets. After comparison, the coordinator in this frame is only in charge of the scheduling and all the actual computations are allocated to the datanodes. Hence, the coordinator would not be a bottleneck if the data scale is huge. In order to reduce the network overhead, their gird allocation algorithm allocates the adjacent grids to the same datanodes. However, in fact, their algorithm only need a small amount of network communications between pairs of datanotes. In addition, the time and space complexity of LOF are very high. In this paper, we present an improved algorithm based on theirs to efficiently detect density-based outliers in distributed environments. Moreover, our experiments are implemented to demonstrate the validity.

3 Preliminaries

3.1 Problem Formalism

we will show some definitions to better solve our problem and they will be applied in our proposed method.

Local density: It is estimated by the typical distance at which a tuple can be reached from its neighbors. And distance is usually adopted in LOF, which is an additional measure to produce more stable results within clusters.

Reachability distance: Let $d(A, B)$ be the distance between A and B, k-distance(A) be the distance of the object A to the k-th nearest neighbor. To simplify the description, we define the distance as Euclidean distance in the rest of this paper similar to [21]. Hence,

1) There are at least k tuples A' such that $d(A, A') \leq d(A, B)$.
2) There are at most $k-1$ tuples A'' such that $d(A, A'') < d(A, B)$.

Note that the set of the k nearest neighbors includes all tuples at this distance, which can in the case of a "tie" be more than k tuples. We denote the set of k nearest neighbors as $N_k(A) = A'|d(A, A') \leq d(A)$. This distance is used to define what is called reachability distance:

$$Rd_k(A, B) = \max\{k\text{-distance}(A), d(A, B)\}.$$

As shown in following Figure 1, it illustrates the original intention of reachability distance. Tuples B and D have the same reachability distance (k=3), while G is not a k nearest neighbor.

Thus, the reachability distance of an object A from B is the true distance of the two objects, but at least the k-distance(A). Objects that belong to the k nearest neighbors of A are considered to be equally distant. The reason for this distance is to get more stable results. Note that this is not a distance in the mathematical definition, since it is not symmetric.

The local reachability density of an object A is defined by

$$LRD(A) = \frac{1}{\frac{\sum_{B\in N_k(A)} Rd_k B, A}{|N_k(A)|}}$$

where the inverse of the average reachability distance of the object A from its neighbors. Note that it is not the average reachability of the neighbors from A (which by definition would be the k-distance(A)), but the distance at which A can be "reached" from its neighbors. With duplicate points, this value can become infinite.

The local reachability densities are then compared with those of the neighbors using

$$LOF_k(A) = \frac{\sum_{B\in N_k(A)} \frac{LRD(B)}{LRD(A)}}{|N_k(A)|}$$

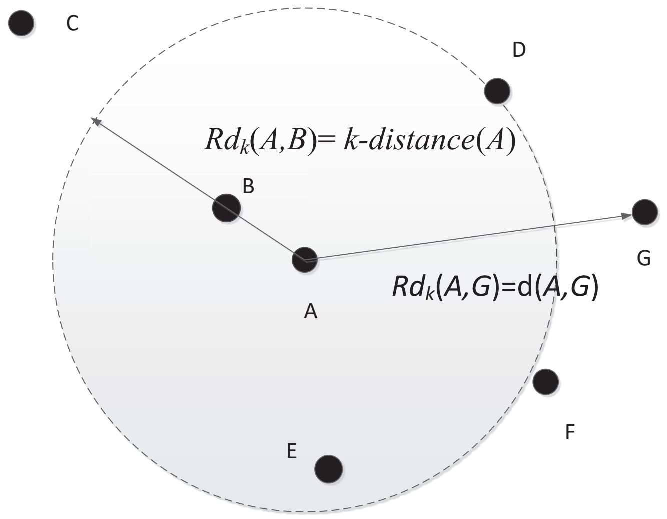

Figure 1: k-distance neighborhood and reachability distance when k=3

where the average local reachability density of the neighbors divided by the object's own local reachability density. A value of approximately a given threshold indicates that the object is comparable to its neighbors (and thus it is not an outlier). A value less than the threshold indicates a denser region (which would be an inlier), while values significantly larger than the threshold indicate outliers.

The traditional methods usually form all pair Euclidean distance matrix, and then run KNN query to proceed further. It is $\Theta(n^2)$ in terms of both space and time complexity. However, it can be improved with KD-tree [20].

3.2 Distributed Environment

As Figure 2 shows, we utilize a distributed framework that consists of a coordinator and a number of datanodes. The coordinator is for the overall scheduling similar to [3]. Each datanode is to store a portion of a complete data set. Most of previous algorithms utilize a master-slave architecture for outlier detection, while lots of computations are performed on the master node. Following [3], we also do that the coordinator in our frame only takes charge of the scheduling and all the actual computations are allocated to the datanodes. Besides, Density-based outlier detection is to compute the LOF of each tuple for a given integer k. First, we use improved GBP (IGBP) algorithm to split the data set into several subsets and assign them to the datanodes. After that, our algorithm work via two main steps. First, each datanode processes the local tuples. And LOFs of some tuples can be computed directly in the local nodes. Second, we will output LOFs of the rest of the tuples by a few necessary network communications compared with [3].

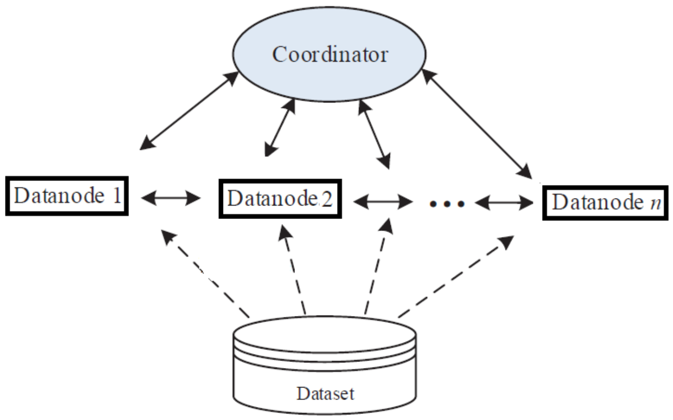

Figure 2: Computation frame

4 The Improved GBP Algorithm (IGBP)

4.1 Grid-based Partition

In our IGBP algorithm, we first attempt to split the whole d-dimensional space into several isometric grids. While grid-based partition method conducts limitations in the high-dimensional data, Hence, we cut each dimension into several equal segments (the number of segments is denoted by s). After that, the space is partitioned into s^d grids. Let $g_{x_1,x_2,\cdots,x_d}$ be the gird that is at the x_i-th position for dimension i. Next, we give the definition of adjacent grid:

$$N(g_{x_1,x_2,\cdots,x_d}) = \{g_{y_1,y_2,\cdots,y_d} | \max(1, x_i - 1) \leq y_i \leq \min(s, x_i + 1), g_{y_1,y_2,\cdots,y_d} \neq g_{x_1,x_2,\cdots,x_d}\}.$$

Next, we allocate these grids to the datanodes. In order to speed up the computations of density-based outliers, we propose an allocation method through considering the following two factors.

1) To obtain high parallelism, we set the number of tuples on each datanode almost the same (balance the workload).

2) According to Section 3.1, we compute the k-distance neighborhood for each tuple and reduce the network overhead if we allocate the adjacent grids to the same datanodes as possible. The details of the proposed method are shown in Algorithm 1.

Algorithm 1 Grid allocation

```
Input: The grid set $G$; The datanode set $N$;
Output: output: Allocation plan;
1: sort the grids in $G$ according to the number of tuples in a grid in the descending order;
2: for each grid $g$ in $G$ do
3:     if there exist datanodes with no grid then
4:         n ← randomly choose a datanode with no grid;
5:     else
6:         Initialize a datanode set $N'$;
7:         for each datanode $n_i$ in $N$ do
8:             if $n_i$ has the least tuples then
9:                 insert $n_i$ into $N'$;
10:            end if
11:            if there exists at least one datanode in $N'$ with grids which belong to adjacent grids $N_g$ then
12:                $n$ ← choose the datanode in $N'$ with the largest number of grids which belong to adjacent grids$N_g$;
13:            else
14:                n←randomly choose a datanode in $N'$;
15:            end if
16:        end for
17:    end if
18:    allocate $g$ to $n$;
19: end for
```

4.2 Distributed LOF Computing

In above part, we have split the data set into several grids and allocated them to the corresponding datanodes. Next, we turn to compute the LOF for each tuple in parallel way. By analyzing the definitions in Section 3, it is clear that LRD is the premise of LOF. In order to calculate LRDs effectively, we have to compute the k-distances and k-distance neighborhoods for all the tuples first, which is the core part of this section.

However, in distributed environments, the situation is more complex. It's difficult to compute the actual k-distances of all the tuples. For example, in Figure 3, considering the tuples in grid g_1, the local k-distance neighborhood of tuple A is identical to its actual k-distance neighborhood. However, tuple B shows a complex situation. Its local k-distance neighborhood is C, D, E, which cannot be computed unless D, E is transmitted from g_2 to gird g_1.

For previous work in [3], they classify the tuples in a grid into 2 categories. A tuple whose neighborhoods can be computed in local grid is a grid-local tuple. Otherwise, it is a cross-grid tuple. After that, they proposed an algorithm to solve this problem. This part is not the most significant point in our paper. For distributed LOF computing, we will make a example to explain our proposed method which is different from previous work.

First, as shown in Figure 4, there is a set P with 120 tuples in 2-dimensional space, and the number of datanodes is 10. Thus, we set $s = 4$ and split the space into 16

Table 1: Improved algorithm process

Sequence number	Grid ID	Number of tuples	Allocated datanode	Average number of tuples
1	$g_{1,2}$	11	n_1	/
2	$g_{1,1}$	10	n_2	/
3	$g_{3,2}$	10	n_3	/
4	$g_{1,4}$	10	n_4	/
5	$g_{3,4}$	9	n_5	/
6	$g_{3,3}$	9	n_6	/
7	$g_{3,1}$	8	n_7	/
8	$g_{4,1}$	8	n_8	/
9	$g_{4,4}$	7	n_9	/
10	$g_{1,3}$	7	n_{10}	8.9
11	$g_{2,2}$	6	n_{10}	9.5
12	$g_{2,4}$	6	n_9	10.1
13	$g_{2,1}$	6	n_7	10.7
14	$g_{4,2}$	5	n_8	11.2
15	$g_{4,3}$	4	n_6	11.6
16	$g_{2,3}$	4	n_5	12

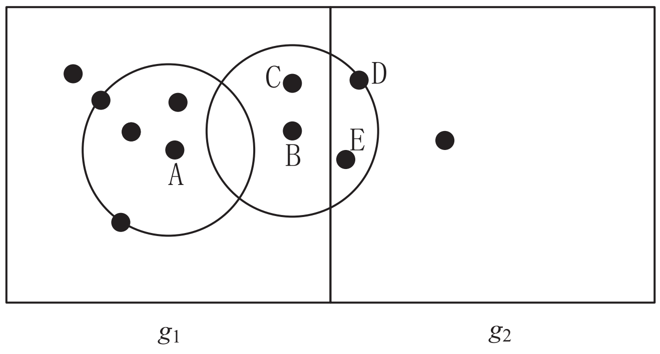

Figure 3: Example of DLC (k=3)

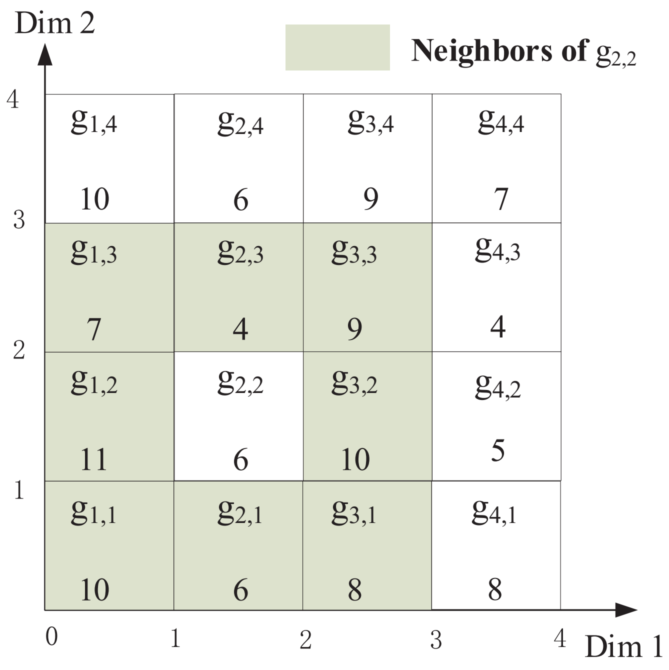

Figure 4: Example of grids

grids. The related number of tuples is shown at the bottom of each grid. According to the algorithm 1, we sort the grids by the number of tuples in a grid, and the result is shown in Table 1. Then, for each of the first 10 grids, we randomly allocate it to a datanode with no grid. After that, totally 89 tuples have been allocated, and the average number of tuples per datanode is 8.9. When allocating the 11th grid $g_{2,2}$, there are 4 datanodes whose numbers of tuples are not larger than 8.9, including n_7; n_8; n_9; n_{10}. We choose n_{10} because the number of tuples in n_{10} is smallest. Using the same method, we allocate all the grids to the corresponding datanodes.

After allocation, the number of tuples in data notes $\{n_1, n_2, \cdots, n_{10}\}$ is $\{11, 10, 10, 10, 13, 13, 14, 13, 13, 13\}$. We use σ (standard deviation) to indicate how spread out a data distribution is. A low standard deviation means that this algorithm balances the workload well. We use computational formula of standard deviation to obtain:

$$\sigma = \sqrt{\frac{11^2+10^2+10^2+10^2+13^2+13^2+14^2+13^2+13^2+13^2}{10}} \approx 1.483$$

Second, according to the previous algorithm in [3], the number of tuples in datanotes $\{n_1, n_2, \cdots, n_{10}\}$ is $\{11, 10, 10, 10, 15, 13, 14, 13, 11, 13\}$. We also obtain $\sigma \approx$ 1.732.

5 Experimental Evaluation

We implement our proposed approaches using python programming language, and evaluate the performance in a cluster (with 4 data nodes and 1 coordinator) where each node (coordinator or datanode) has a Intel Core i5 @ 2.53 GHz CPU, 32G main memory. We first use a synthetic

Table 2: The influence of data scale

Method	**100000 tuples**		**500000 tuples**		**1000000 tuples**	
	k=10	k=20	k=10	k=20	k=10	k=20
In [16]	782ms	1243ms	5341ms	7988ms	13568	27755
In [3]	512ms	910ms	4122ms	6278	9742	22495
In IGBP	547ms	932ms	3907ms	5611	8472	16625

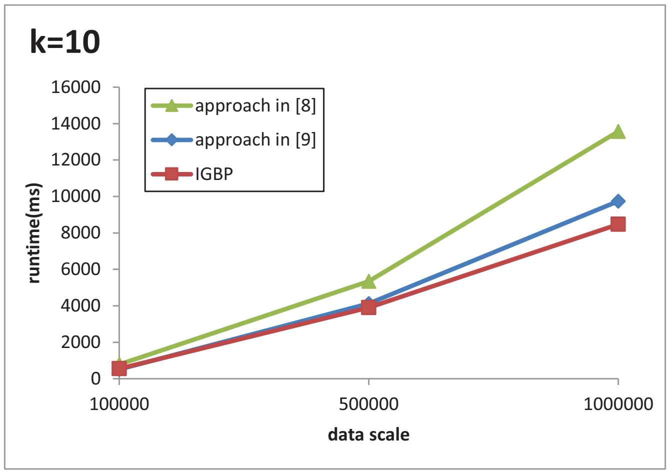

Figure 5: Runtime comparison

Table 3: Open datasets

	Number of Instances	Number of Attributes
Shuttle	58000	9
Census	48842	14
Drug	215063	6

Table 4: Results in three datasets

Methods	**Runtime(ms)**		
	Shuttle	Census	Drug
In [16]	12339	34987	47788
In [3]	3374	7120	9759
IGBP	**3115**	**5780**	**6880**

dataset to verify the efficiency of our proposed method IGBP. Then we choose three open datasets Shuttle, Census and Drug which have been widely used in outlier detection to further show the highly efficiency and effectiveness of IGBP.

5.1 IGBP with Synthetic Data

We generate various synthetic data sets to analyze the performance of our methods. Specifically, we compare our proposed method with [3, 16]. We generate several clustering center points. And the tuples in each cluster follow a Gaussian distribution. Finally, we add some generated noise into dataset, and the dimension of the data set is 3. The detailed parameter settings and the runtime are summarized in Table 2.

As shown in the table, with the increase of data scale, the runtime for our algorithm show better performance than previous work [3, 16]. Besides, experimental results demonstrate that workload balance is more significant than network overhead in this issue. Figure 5 illustrates that the impact of different data scale, with the increase of data scale, our proposed method outperform better baselines.

5.2 IGBP with Open Datasets

In this part, we use three popular real-world datasets to evaluate our proposed method. Shuttle, Census and Drug which have been widely used in outlier detection. The details has shown in following Table 3.

As Table 4 shows that our proposed method have low time consumption than [3, 16] based on three real-world datasets. More specifically, with the increase of data scale, we find that our proposed method have higher efficiency compared with [3, 16].

6 Conclusions

In this paper, we focus on the problem of density-based outlier detection in distributed environments for high-dimensional and large-scale data sets. We first introduce the basic concept of LOF. We summarized the approach in [3] to solve the problem of distributed LOF computing in detail. The advantages and disadvantages of the approach are discussed. Then, we propose an improved algorithm based on greedy algorithm, namely IGBP. With the experimental results, we show the efficiency and effectiveness of the proposed approaches compared with previous work. The results demonstrate that our algorithm outperform baseline. In future, we will considerate more efficient policies to balance the time consumption and space consumption.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (61472064,61602096) and the Sichuan Provincial Science and Technology Department Project (2018GZ0087,2016FZ0002).

References

[1] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," in *ACM Sigmod Record*, vol. 30, pp. 37–46, 2001.

[2] F. Angiulli and C. Pizzuti, "Outlier mining in large high-dimensional data sets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 2, pp. 203–215, 2005.

[3] M. Bai, X. Wang, J. Xin, and G. Wang, "An efficient algorithm for distributed density-based outlier detection on big data," *Neurocomputing*, vol. 181, pp. 19–28, 2016.

[4] R. J. Beckman and R. D. Cook, "Outlier . . . . s," *Technometrics*, vol. 25, no. 2, pp. 119–149, 1983.

[5] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *ACM Sigmod Record*, vol 29, pp. 93–104, 2000.

[6] D. Dasgupta and N. S. Majumdar, "Anomaly detection in multidimensional data using negative selection algorithm," in *Proceedings of the Congress on Evolutionary Computation (CEC'02)*, vol. 2, pp. 1039–1044, 2002.

[7] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pp. 401–410, 2005.

[8] J. Han, J. Pei, and M. Kamber, "Data mining: Concepts and techniques," *A volume in the Morgan Kaufmann Series in Data Management Systems*, 2012. (https://www.sciencedirect.com/book/9780123814791/data-mining-concepts-and-techniques)

[9] D. Hawkins, "Identification of outliers," *Monographs on Statistics and Applied Probability*, vol. 11, 1980.

[10] Q. He, Y. Ma, Q. Wang, F. Zhuang, and Z. Shi, "Parallel outlier detection using kd-tree based on mapreduce," in *IEEE Third International Conference on Cloud Computing Technology and Science*, pp. 75–80, 2011.

[11] E. Hung and D. W. Cheung, "Parallel mining of outliers in large database," *Distributed and Parallel Databases*, vol. 12, no. 1, pp. 5–26, 2002.

[12] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 8, no. 3-4, pp. 237–253, 2000.

[13] E. M. Knox and R. T. Ng, "Algorithms for mining distancebased outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases*, pp. 392–403, 1998.

[14] A. Koufakou, J. Secretan, J. Reeder, K. Cardona, and M. Georgiopoulos. "Fast parallel outlier detection for categorical datasets using mapreduce," in *IEEE International Joint Conference on Neural Networks*, pp. 3298–3304, 2008.

[15] H. P. Kriegel, A. Zimek, *et al.*, "Angle-based outlier detection in high-dimensional data," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 444–452, 2008.

[16] E. Lozano and E. Acufia, "Parallel algorithms for distance-based and density-based outliers," in *Fifth IEEE International Conference on Data Mining*, pp. 4, 2005.

[17] Y. Lu, Y. Zhu, M. Itomlenskis, S. Vyaghri, and H. Fu, "Mmoprg bot detection based on traffic analysis," *International Journal of Electronics and Information Engineering*, vol. 2, no. 1, pp. 18–26, 2015.

[18] M. Markou and S. Singh, "Novelty detection: A review—part 1: statistical approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[19] M. Markou and S. Singh, "Novelty detection: A review—part 2:: Neural network based approaches," *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, 2003.

[20] F. P. Miller, A. F. Vandome, and J. McBrewster, *Kd-Tree*, 2009. (https://www.amazon.fr/kd-tree-Frederic-P-Miller/dp/6130094590)

[21] E. Schubert, A. Zimek, and H. P. Kriegel, "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 190–237, 2014.

Biography

LinMei received B.Sc and M.Sc in computer science and technology from University of Electronic Science and Technology of China in 2003 and 2006, He is currently a lecturer in Southwest Minzu University. And pursing Ph.D degree in University of Electronic Science and Technology of China. His research interests include network security and cloud computing.

Fengli Zhang received B.Sc, M.Sc and Ph.D in computer science and technology from University of Electronic Science and Technology of China in 1983, 1986 and 2007. She is currently a professor in University of Electronic Science and Technology of China, her research interest include database management, network security and big data processing.