Hindawi
Mathematical Problems in Engineering
Volume 2017, Article ID 2649535, 12 pages
https://doi.org/10.1155/2017/2649535

Research Article

A Distributed Algorithm for the Cluster-Based Outlier Detection Using Unsupervised Extreme Learning Machines

Xite Wang,[1] Mei Bai,[1] Derong Shen,[2] Tiezheng Nie,[2] Yue Kou,[2] and Ge Yu[2]

[1]*College of Information Science & Technology, Dalian Maritime University, Dalian, Liaoning 116000, China*
[2]*College of Information Science & Engineering, Northeastern University, Shenyang, Liaoning 110819, China*

Correspondence should be addressed to Xite Wang; xite-skywalker@163.com

Received 25 November 2016; Accepted 13 March 2017; Published 9 April 2017

Academic Editor: Alberto Borboni

Outlier detection is an important data mining task, whose target is to find the abnormal or atypical objects from a given dataset. The techniques for detecting outliers have a lot of applications, such as credit card fraud detection and environment monitoring. Our previous work proposed the Cluster-Based (CB) outlier and gave a centralized method using unsupervised extreme learning machines to compute CB outliers. In this paper, we propose a new distributed algorithm for the CB outlier detection (DACB). On the master node, we collect a small number of points from the slave nodes to obtain a threshold. On each slave node, we design a new filtering method that can use the threshold to efficiently speed up the computation. Furthermore, we also propose a ranking method to optimize the order of cluster scanning. At last, the effectiveness and efficiency of the proposed approaches are verified through a plenty of simulation experiments.

1. Introduction

Outlier detection is an important issue of data mining, and it has been widely studied by many scholars for years. According to the description in [1], "an outlier is an observation in a dataset which appears to be inconsistent with the remainder of that set of data." The techniques for mining outliers can be applied to many fields, such as credit card fraud detection, network intrusion detection, and environment monitoring.

There exist two primary missions in the outlier detection. First, we need to define what data are considered as outliers in a given set. Second, an efficient method to compute these outliers needs to be designed. The outlier problem is first studied by the statistics community [2, 3]. They assume that the given dataset follows a distribution, and an object is considered as an outlier if it shows distinct deviation from this distribution. However, it is almost an impossible task to find an appropriate distribution for high-dimensional data. To overcome the drawback above, some model-free approaches are proposed by the data management community. Examples include distance-based outliers [4–6] and the density-based outlier [7]. Unfortunately, although these definitions do not need any assumption on the dataset, some shortcomings still exist in them. Therefore, in this paper, we propose a new definition, the Cluster-Based (CB) outlier. The following example discusses the weaknesses of the existing model-free approaches and the motivation of our work.

As Figure 1 shows, there are a denser cluster C_1 and a sparse cluster C_2 in a two-dimensional dataset. Intuitively, points p_1 and p_2 are the outliers because they show obvious differences from the other points. However, in the definitions of the distance-based outliers, a point p is marked as an outlier depending on the distances from p to its k-nearest neighbors. Then, most of the points in the sparse cluster C_2 are more likely to be outliers, whereas the real outlier p_1 will be missed. In fact, we must consider the *localization* of outliers. In other words, to determine whether a point p in a cluster C is an outlier, we should only consider the points in C, since the points in the same cluster usually have similar characters. Therefore, in Figure 1, p_1 from C_1 and p_2 from C_2 can be selected correctly. The density-based outlier [7] also considers the localization of outliers. For each point p, they use the Local Outlier Factor (LOF) to measure the degree of being an outlier. To compute the LOF of p, we need to find the set of its k-nearest neighbors $nn_k(p)$ and all the k-nearest neighbors of each point in $nn_k(p)$. The

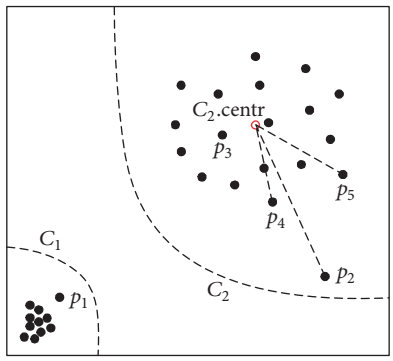


Figure 1: Example of outliers.

expensive computational cost limits the practicability of the density-based outlier. Therefore, we propose the CB outlier to conquer the above deficiencies. The formal definition will be described in Section 3.

To detect CB outliers in a given set, the data need to be clustered first. In this paper, we employ the unsupervised extreme learning machine (UELM) [8] for clustering. The extreme learning machine (ELM) is a novel technique proposed by Huang et al. [9–11] for pattern classification, which shows better predicting accuracy than the traditional Support Vector Machines (SVMs) [12–15]. Thus far, the ELM techniques have attracted the attention of many scholars and various extensions of ELM have been proposed [16]. UELM [8] is designed for dealing with the unlabeled data, and it can efficiently handle clustering tasks. The authors show that UELM provides favorable performance compared with the state-of-the-art clustering algorithms [17–20].

In [21], we have studied the problem of CB outlier detection using UELM in a centralized environment. Faced with the increasing data scale, the performance of the centralized method becomes too limited to meet the timeliness requirements. Therefore, in this paper, we develop a new efficient distributed algorithm for the CB outlier detection (DACB). The main contributions are summarized as follows:

(1) We propose a framework of distributed CB outlier detection, which adopts a master-slave architecture. The master node keeps monitoring the points with large weights on each slave node and obtains a threshold LB. The slave nodes can use LB to efficiently filter the unpromising points and accelerate the computational efficiency.

(2) We propose a new algorithm to compute the point weights on each slave node. Compared with our previous method in [21], the new algorithm adopts a filtering technique to further improve the efficiency. We can filter out a large number of unpromising points, instead of computing their exact weights.

(3) We propose a new method to optimize the order of cluster scanning on each slave node. With the help of this method, we can obtain a large LB early and improve the filtering performance.

The rest of the paper is organized as follows. Section 2 gives brief overviews of ELM and UELM. Section 3 formally defines the CB outlier. Section 4 gives the framework of DACB. Section 5 illustrates the details of DACB. Section 6 analyzes the experimental results. Section 7 gives the related work of outlier detection. Section 8 concludes the paper.

2. Preliminaries

2.1. Brief Introduction to ELM. The target of ELM is to train a single layer feedforward network from a training set with N samples, $\{\mathbf{X}, \mathbf{Y}\} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$. Here, $\mathbf{x}_i \in \mathbb{R}^d$, and $\mathbf{y}_i$ is an M-dimensional binary vector where only one entry is "1" to represent the class that $\mathbf{x}_i$ belongs to.

The training process of ELM includes two stages. In the first stage, we build the hidden layer with L nodes using a number of mapping neurons. In detail, for the jth hidden layer node, a d-dimensional vector $\mathbf{a}_j$ and a parameter b_j are randomly generated. Then, for each input vector $\mathbf{x}_i$, the relevant output value on the jth hidden layer node can be acquired using an activation function such as the Sigmoid function below:

$$g\left(\mathbf{x}_i, \mathbf{a}_j, b_j\right) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}_j^T \times \mathbf{x}_i + b_j\right)\right)}. \tag{1}$$

Then, the matrix outputted by the hidden layer is

$$\mathbf{H} = \begin{bmatrix} g\left(\mathbf{x}_1, \mathbf{a}_1, b_1\right) & \cdots & g\left(\mathbf{x}_1, \mathbf{a}_L, b_L\right) \\ \vdots & & \vdots \\ g\left(\mathbf{x}_N, \mathbf{a}_1, b_1\right) & \cdots & g\left(\mathbf{x}_N, \mathbf{a}_L, b_L\right) \end{bmatrix}_{N \times L}. \tag{2}$$

In the second stage, an M-dimensional vector $\boldsymbol{\beta}_j$ is the output weight that connects the jth hidden layer with the output node. The output matrix $\mathbf{Y}$ is acquired by

$$\mathbf{H} \cdot \boldsymbol{\beta} = \mathbf{Y}, \tag{3}$$

where

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_L \end{bmatrix}_{L \times M}, \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix}_{N \times M}. \tag{4}$$

We have known the matrices $\mathbf{H}$ and $\mathbf{Y}$. The target of ELM is to solve the output weights $\boldsymbol{\beta}$ by minimizing the square losses of the prediction errors, leading to the following equation:

$$\begin{aligned} \min_{\boldsymbol{\beta} \in \mathbb{R}^{L \times M}} \quad & L_{\text{ELM}} = \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2 + \frac{C}{2}\left\|\mathbf{Y} - \mathbf{H}\boldsymbol{\beta}\right\|^2 \\ \text{s.t.} \quad & \mathbf{H}\boldsymbol{\beta} = \mathbf{Y} - \mathbf{e}, \end{aligned} \tag{5}$$

Input. The training data: $\mathbf{X} \in \mathbb{R}^{N\times d}$.
Output. The label vector of cluster $\mathbf{y}_i$ corresponding to $\mathbf{x}_i$
(1) (a) Construct the graph Laplacian $\mathbf{L}$ of $\mathbf{X}$;
(2) (b) Generate a pair of random values $\{\mathbf{a}_i, b_i\}$ for each hidden neuron, and calculate the output matrix $\mathbf{H} \in \mathbb{R}^{N\times L}$;
(3) (c)
(4) **if** $L \leq N$ **then**
(5) Find the generalized eigenvectors $\mathbf{v}_2, \ldots, \mathbf{v}_{M+1}$ of Equation (11). Let $\boldsymbol{\beta} = [\tilde{\mathbf{v}}_2, \tilde{\mathbf{v}}_3, \ldots, \tilde{\mathbf{v}}_{M+1}]$.
(6) **else**
(7) Find the generalized eigenvectors $\mathbf{u}_2, \ldots, \mathbf{u}_{M+1}$ of Equation (13). Let $\boldsymbol{\beta} = H^T[\tilde{\mathbf{u}}_2, \tilde{\mathbf{u}}_3, \ldots, \tilde{\mathbf{u}}_{M+1}]$;
(8) (d) Calculate the embedding matrix: $\mathbf{E} = \mathbf{H}\boldsymbol{\beta}$;
(9) (e) Treat each row of $\mathbf{E}$ as a point, and cluster the N points into K clusters using the k-means algorithm. Let $\mathbf{Y}$ be the label vector of cluster index for all the points.
(10) **return** $\mathbf{Y}$;

Algorithm 1: UELM algorithm [8].

where $\|\cdot\|$ denotes the Euclidean norm, $\mathbf{e} = \mathbf{Y} - \mathbf{H}\boldsymbol{\beta} = [\mathbf{e}_1^T, \ldots, \mathbf{e}_N^T] \in \mathbb{R}^{N\times M}$ is the error vector with respect to the training samples, and C is a penalty coefficient on the training errors. The first term in the objective function is a regularization term against overfitting.

If $N \geq L$, which means $\mathbf{H}$ has more rows than columns and it is full of column rank, (6) is the solution for (5). Hence,

$$\boldsymbol{\beta}^* = \left(\mathbf{H}^T\mathbf{H} + \frac{\mathbf{I}_L}{C}\right)^{-1}\mathbf{H}^T\mathbf{Y}. \tag{6}$$

If $N < L$, a restriction that $\boldsymbol{\beta}$ is a linear combination of the rows of $\mathbf{H}$: $\boldsymbol{\beta} = \mathbf{H}^T\boldsymbol{\alpha}(\boldsymbol{\alpha} \in \mathbb{R}^{N\times M})$ is considered. Then, $\boldsymbol{\beta}$ can be calculated by

$$\boldsymbol{\beta}^* = \mathbf{H}^T\boldsymbol{\alpha}^* = \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T + \frac{\mathbf{I}_N}{C}\right)^{-1}\mathbf{Y}, \tag{7}$$

where $\mathbf{I}_L$ and $\mathbf{I}_N$ are the identity matrices of dimensions L and N, respectively.

2.2. Unsupervised ELM. Huang et al. [8] proposed UELM to process an unsupervised dataset, which shows good performance in clustering tasks. The unsupervised learning is based on the following assumption: if two points $\mathbf{x}_1$ and $\mathbf{x}_2$ are close to each other, their conditional probabilities $P(\mathbf{y} \mid \mathbf{x}_1)$ and $P(\mathbf{y} \mid \mathbf{x}_2)$ should be similar. To enforce this assumption on the data, we acquire the following equation:

$$L_m = \frac{1}{2}\sum_{i,j} w_{ij}\left\|P\left(\mathbf{y} \mid \mathbf{x}_i\right) - P\left(\mathbf{y} \mid \mathbf{x}_j\right)\right\|^2, \tag{8}$$

where w_{ij} is the pairwise similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, which can be calculated by Gaussian function $\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$.

Since it is difficult to calculate the conditional probabilities, the following can approximate (8):

$$\widehat{L}_m = \mathrm{Tr}\left(\widehat{\mathbf{Y}}^T\mathbf{L}\widehat{\mathbf{Y}}\right), \tag{9}$$

where $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix, $\widehat{\mathbf{Y}}$ is the predictions of the unlabeled dataset, $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is known as graph Laplacian, and $\mathbf{D}$ is a diagonal matrix with its diagonal elements $D_{ii} = \sum_{j=1}^{u} w_{ij}$.

In the unsupervised learning, the dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ is unlabeled. Substituting (9) into (5), the objective function of UELM is acquired:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^{L\times M}} \quad \|\boldsymbol{\beta}\|^2 + \lambda\,\mathrm{Tr}\left(\boldsymbol{\beta}^T\mathbf{H}^T\mathbf{L}\mathbf{H}\boldsymbol{\beta}\right), \tag{10}$$

where λ is a tradeoff parameter. In most cases, (10) reaches its minimum value at $\boldsymbol{\beta} = \mathbf{0}$. In [18], Belkin and Niyogi introduced an additional constraint $(\mathbf{H}\boldsymbol{\beta})^T\mathbf{H}\boldsymbol{\beta} = \mathbf{I}_M$. On the base of the conclusion in [8], if $L \leq N$, we can obtain the following equation:

$$\left(\mathbf{I}_L + \lambda\mathbf{H}^T\mathbf{L}\mathbf{H}\right)\mathbf{v} = \gamma\mathbf{H}^T\mathbf{H}\mathbf{v}. \tag{11}$$

Let γ_i be the ith smallest eigenvalues of (11) and $\mathbf{v}_i$ be the corresponding eigenvectors. Then, the solution of the output weights $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\beta}^* = \left[\tilde{\mathbf{v}}_2, \tilde{\mathbf{v}}_3, \ldots, \tilde{\mathbf{v}}_{M+1}\right], \tag{12}$$

where $\tilde{\mathbf{v}}_i = \mathbf{v}_i/\|\mathbf{H}\mathbf{v}_i\|$, $i = 2, \ldots, M+1$, are the normalized eigenvectors.

If $L > N$, (11) is underdetermined. We obtain the alternative formulation below:

$$\left(\mathbf{I}_N + \lambda\mathbf{L}\mathbf{H}\mathbf{H}^T\right)\mathbf{u} = \gamma\mathbf{H}\mathbf{H}^T\mathbf{u}. \tag{13}$$

Again, let $\mathbf{u}_i$ be the generalized eigenvectors corresponding to the ith smallest eigenvalues of (13). Then, the final solution is

$$\boldsymbol{\beta}^* = H^T\left[\tilde{\mathbf{u}}_2, \tilde{\mathbf{u}}_3, \ldots, \tilde{\mathbf{u}}_{M+1}\right], \tag{14}$$

where $\tilde{\mathbf{u}}_i = \tilde{\mathbf{u}}_i/\|\mathbf{H}\mathbf{H}^T\tilde{\mathbf{u}}_i\|$, $i = 2, \ldots, M+1$, are the normalized eigenvectors. Algorithm 1 shows the process of UELM.

3. Defining CB Outliers

For a given dataset P in a d-dimensional space, a point p is denoted by $p = \langle p[1], p[2], \ldots, p[d] \rangle$. The distance between two points p_1 and p_2 is $\text{dis}(p_1, p_2) = \sqrt{\sum_{i=1}^{d} (p_1[i] - p_2[i])^2}$. Suppose that there are m clusters $C_1, C_2, \ldots, C_m$ in P outputted by UELM. For each cluster C_i, the centroid point C_i.centr can be computed by the following equation:

$$C_i.\text{centr}[i] = \frac{\sum_{p \in C_i} p[i]}{|C_i|}. \quad (15)$$

Intuitively, in a cluster C, most of the *normal* points are closely around the centroid point of C. In contrast, an *abnormal* point p (i.e., outlier) is usually far from the centroid point and the number of points close to p is quite small. Based on this observation, the weight of a point is defined as follows.

Definition 1 (weight of a point). Given an integer k, for a point p in cluster C, one uses $nn_k(p)$ to denote the set of the k-nearest neighbors of p in C. Then, the weight of p is

$$w(p) = \frac{\text{dis}(p, C.\text{centr}) \times k}{\sum_{q \in nn_k(p)} \text{dis}(q, C.\text{centr})}. \quad (16)$$

Definition 2 (result set of CB outlier detection). For a dataset P, given two integers k and n, let R_{CB} be a subset of P with n points. If, $\forall p \in R_{\text{CB}}$, there is no point $q \in P \setminus R_{\text{CB}}$ that $w(q) > w(p)$, R_{CB} is the result set of CB outlier detection.

For example, in Figure 1, in cluster C_2, the centroid point is marked in red. For $k = 2$, the k-nearest neighbors of p_2 are p_4 and p_5. Because p_2 is an abnormal point and it is far from the centroid point, $\text{dis}(C_2.\text{centr}, p_2)$ is much larger than $\text{dis}(C_2.\text{centr}, p_4)$ and $\text{dis}(C_2.\text{centr}, p_5)$. Hence, the weight of p_2 is large. In contrast, for a normal point p_3 deep in the cluster, the distances from C_2.centr to its k-nearest neighbors are similar to $\text{dis}(C_2.\text{centr}, p_3)$. The weight of p_3 is close to 1. Therefore, p_2 is more likely to be considered as a CB outlier.

4. Framework of Distributed CB Outlier Detection

The target of this paper is to detect CB outliers in a distributed environment, which is constituted by a master node and a number of slave nodes. The master node is the coordinator, and it can communicate with all the slave nodes. Each slave node reserves a subset of the clusters in P outputted by UELM, and it is the main worker in the outlier detection. Figure 2 shows the framework of the distributed algorithm for CB outlier detection (DACB) proposed in this paper.

When a request of outlier detection arrives, each slave node starts to scan the local clusters. Basically, in a cluster C, we need to search the kNNs of each point p in C to compute the weight of p. Obviously, computing the kNNs of all the points is very time-consuming. Therefore, in our DACB, we propose a filtering method to accelerate the computation. Specifically, each slave node keeps tracking the local top-n points with the largest weights in the scanned points and sends them to the master node. From the received points, the master reserves the global top-n points with the largest weights. We choose the smallest one from these n weights as a threshold LB. The master broadcasts LB to the slave nodes to efficiently filter the unpromising points (the detailed filtering method is described in Section 5.1). On each slave node, if there emerges a point with weight larger than LB, the point will be sent to the master. The master termly updates the global top-n points and the threshold LB. At last, the n points stored in the master node are the CB outliers.

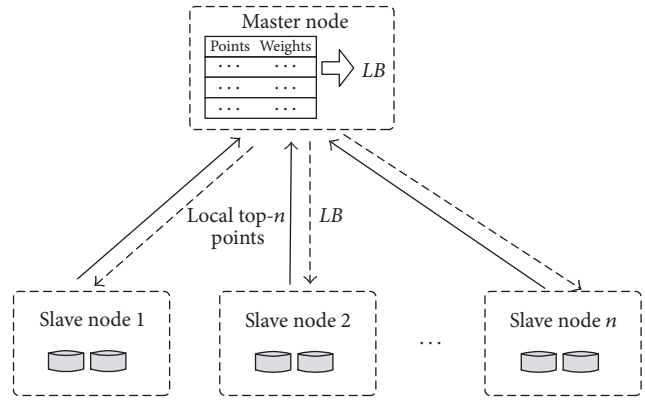


Figure 2: The framework of DACB.

5. DACB Description

5.1. The Computation of Point Weight. In this section, we introduce the techniques to compute the point weights on each slave node. According to Definitions 1 and 2, to determine whether a point p in a cluster C is an outlier, we need to search the k-nearest neighbors (kNNs) of p in C. In order to accelerate the kNN searching, we design an efficient method to prune the searching space.

For a cluster C, suppose that the points in C have been sorted according to the distances to the centroid point in an ascending order. For a point p in C, we scan the points to search to its kNNs. Let $nn_k^{\text{temp}}(p)$ be the set of k points that are the nearest to p from the scanned points, and let $k\,\text{dis}_{\text{temp}}(p)$ be the maximum value of the distances from the points in $nn_k^{\text{temp}}(p)$ to p. Then, the pruning method is described as follows.

Theorem 3. *For a point q in front of p, if* $\text{dis}(q, C.centr) < \text{dis}(p, C.centr) - k\,\text{dis}_{temp}(p)$, *the points in front of q and q itself cannot be the kNNs of p.*

Proof. For a point q' in front of q, because the points in C have been sorted, $\text{dis}(q', C.\text{centr}) < \text{dis}(q, C.\text{centr})$. Then, according to the triangle inequality, the distance from q' to p: $\text{dis}(q', p) > \text{dis}(p, C.\text{centr}) - \text{dis}(q', C.\text{centr}) > \text{dis}(p, C.\text{centr}) - \text{dis}(q, C.\text{centr}) > \text{dis}(p, C.\text{centr}) - (\text{dis}(p, C.\text{centr}) - k\,\text{dis}_{\text{temp}}(p)) = k\,\text{dis}_{\text{temp}}(p)$. Clearly, there exist k points closer to p than q', and thus q' cannot be the kNN of p. □

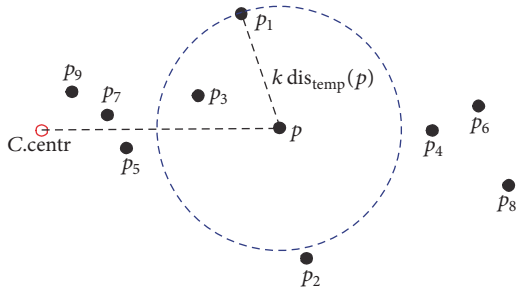


Figure 3: Example of kNN searching.

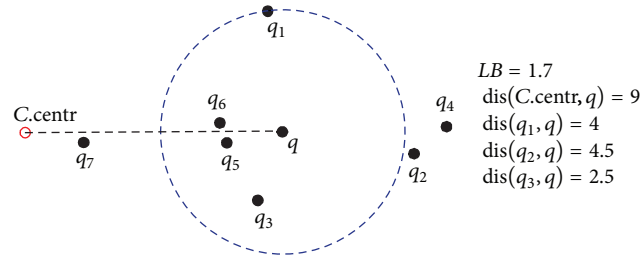


Figure 4: Example of filtering.

Similarly, for a point q at the back of p, if $\mathrm{dis}(q, C.\mathrm{centr}) > \mathrm{dis}(p, C.\mathrm{centr}) + k\,\mathrm{dis}_{\mathrm{temp}}(p)$, the points at the back of q and q itself cannot be the kNNs of p. For example, Figure 3 shows a portion of points in a cluster C. First, we sort the points according to the distances to the centroid point and obtain a point sequence p_9, p_7, p_5, p_3, p_1, p, p_2, p_4, p_6, p_8. For point p, we search its kNNs from p to both sides ($k = 2$). After p_1, p_2, and p_3 are visited, p_1 and p_3 are the current top-k nearest neighbors for p, and thus $k\,\mathrm{dis}_{\mathrm{temp}}(p) = \mathrm{dis}(p, p_1)$. When we visit p_4, $\mathrm{dis}(p_4, C.\mathrm{centr}) > \mathrm{dis}(p, C.\mathrm{centr}) + k\,\mathrm{dis}_{\mathrm{temp}}(p)$. Hence, the points behind p_4 in the sequence (i.e., p_4, p_6, p_8) cannot be the kNNs of p. Similarly, when p_5 is visited, we do not need to further scan the points before p_5 in the sequence because $\mathrm{dis}(p_5, C.\mathrm{centr}) < \mathrm{dis}(p, C.\mathrm{centr}) - k\,\mathrm{dis}_{\mathrm{temp}}(p)$. Therefore, the kNN searching stops, and the exact kNNs of p are p_1 and p_3.

Furthermore, after the weights of some points have been computed, we send the current top-n points with the largest weights to the master node to obtain the threshold LB (mentioned in Section 4). We can use LB to filter the points that cannot be the CB outliers, instead of searching the exact kNNs. The detailed method is stated as follows.

Theorem 4. *In the kNN searching of p in a cluster C, $nn_k^{temp}(p)$ is the set of the current k-nearest neighbors of p from the scanned points. $nn_k(p)$ is the set of the exact k-nearest neighbors. One sorts the points in $nn_k(p)$ and $nn_k^{temp}(p)$ according to the distances to p in an ascending order, respectively. Then, for the ith point in $nn_k(p)$: q and the ith point in $nn_k^{temp}(p)$: q', one asserts that* $\mathrm{dis}(q, C.centr) > \mathrm{dis}(p, C.centr) - \mathrm{dis}(p, q')$.

Proof. If q and q' are the same point, the theorem can be proven easily. Otherwise, $\mathrm{dis}(p, q) < \mathrm{dis}(p, q')$. Using the triangle inequality, $\mathrm{dis}(q, C.\mathrm{centr}) > \mathrm{dis}(p, C.\mathrm{centr}) - \mathrm{dis}(p, q) > \mathrm{dis}(p, C.\mathrm{centr}) - \mathrm{dis}(p, q')$. The theorem is proven. □

Corollary 5. *For a point p, if the current k-nearest neighbor set $nn_k^{temp}(p)$ meets the following condition, p cannot be the CB outlier:*

$$\frac{\mathrm{dis}\,(p, C.centr) \times k}{\sum_{q \in nn_k^{temp}(p)} (\mathrm{dis}\,(p, C.centr) - \mathrm{dis}\,(p, q))} \leq LB. \quad (17)$$

Proof. If (17) holds, the weight upper bound of p is smaller than or equal to LB according to Theorem 4. Therefore, p is not a CB outlier. □

By utilizing Corollary 5, a large number of points can be filtered out safely, and we do not need to compute the exact kNNs of them. For example, in Figure 4, for $k = 2$, we search the kNNs of q. After q_1, q_2, and q_3 are visited, the current k-nearest neighbors are q_1 and q_3. $\mathrm{dis}(C.\mathrm{centr}, q) \times k/(2 \times \mathrm{dis}(C.\mathrm{centr}, q) - \mathrm{dis}(q_1, q) - \mathrm{dis}(q_3, q)) \approx 1.57 < LB$. Then, we can assert that q is not a CB outlier even if the exact kNNs are not found. Therefore, q can be filtered out safely.

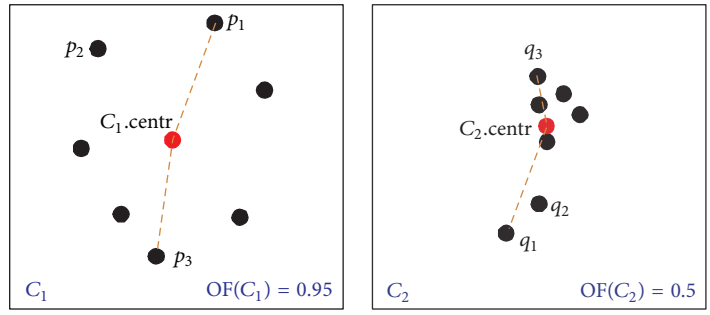
Algorithm 2 shows the process of CB outlier detection on each slave node. For each cluster C_i in SET_S, the points in C_i are sorted according to the distances to the centroid point in an ascending order (line (2)). Then, we scan the points in reversed order (line (3)), since the points far from the centroid point are more likely to have large weights and a good threshold LB can be obtained early. For each scanned point p, we visit the points from p to both sides to search the kNNs (line (8)). If the visited point q meets Theorem 3, a number of points can be erased from the visited list since they cannot be the kNNs of p (lines (10)–(13)). If the current kNNs of p meet Corollary 5, p is not an outlier, so we do not further search its kNNs (lines (16)–(18)). After all points are visited, we send p to the master node if p is still not filtered out (lines (19) and (20)). At last, the n points on the master node are the CB outliers.

5.2. The Order of Cluster Scanning. As Section 4 describes, each slave node needs to visit the local clusters one by one, and it computes the weights of points in each visited cluster. Meanwhile, the global top-n points with the largest weights thus far are kept in order to obtain the threshold LB. Clearly, the filtering effectiveness can be significantly improved if we obtain a large LB early, whereas a large LB is unlikely to be obtained if we visit the clusters in a random order, because we cannot guarantee that the early scanned points have large weights. As a consequence, we design a new ranking method for the clusters. Figure 5 illustrates the main idea of the ranking method.

A common idea is that points with large weights possibly emerge in a sparse cluster (the distance between every two points is large). However, this idea does not work well in many cases. For example, in Figure 5(a), the distances from the points to the centroid point in the sparse cluster are almost identical, and thus the weights of these points are not large. However, in Figure 5(b), in the dense cluster, although most of the points are very close, two abnormal points q_1 and q_2 are far from the centroid point, whose weights are large and

Input. The cluster set SET_S reserved on slave node S, integers k, n, the threshold LB
Output. The CB outliers in P
(1) **for** each cluster C_i in SET_S **do**
(2) Sort the points in C_i according to the distances to the centroid point in ascending order;
(3) Scan the points in reversed order;
(4) **for** each scanned point p **do**
(5) Initialize a heap nn; // to reserve the current kNNs of p
(6) $k\,\mathrm{dis}_{\mathrm{temp}} = \infty$; // the largest distance from the points in nn to p
(7) boolean *potential_outlier* = true;
(8) Visit the points from p to the both sides to search p's kNNs;
(9) **for** each visited point q **do**
(10) **if** q is before p and $\mathrm{dis}(q, C.\mathrm{centr}) < \mathrm{dis}(p, C.\mathrm{centr}) - k\,\mathrm{dis}_{\mathrm{temp}}$ **then**
(11) Erase the points before q from the visited list;
(12) **else if** q is behind p and $\mathrm{dis}(q, C.\mathrm{centr}) > \mathrm{dis}(p, C.\mathrm{centr}) + k\,\mathrm{dis}_{\mathrm{temp}}$ **then**
(13) Erase the points behind q from the visited list;
(14) **else**
(15) Update nn and $k\,\mathrm{dis}_{\mathrm{temp}}$;
(16) **if** nn meets the condition of Corollary 5 **then**
(17) *potential_outlier* = false;
(18) **break**
(19) **if** *potential_outlier* **then**
(20) Send p to the master node to update LB;

Algorithm 2: CB outlier detection on each slave node.



(a) A sparse cluster (b) A dense cluster

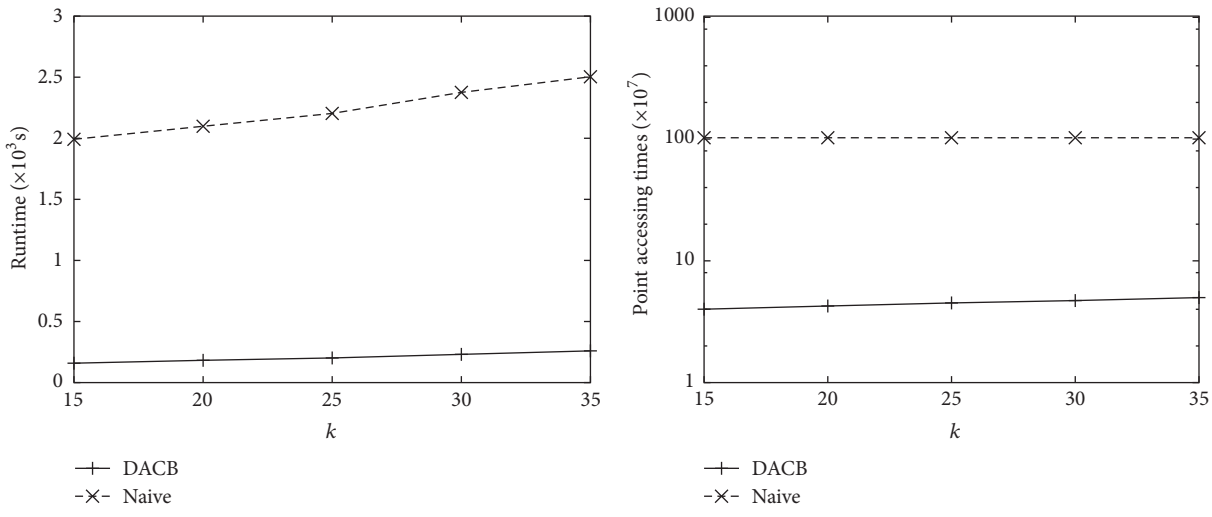
Figure 5: The ranking method of clusters.

suitable for the LB computation. From the observation in Figure 5(b), we can see that the distances of the points with large weights to the centroid point are large and quite different with those of the other points. Besides, note that we only need n large weights to compute LB. based on the description above, we propose the following ranking method.

Definition 6 (the outlier factor of a cluster). For a cluster C, suppose that the points have been sorted according to the distance to C.centr in a descending order. One uses dis_0^C to denote the distance between the first point and C.centr and dis_n^C to denote the distance between the $(n+1)$th point and C.centr. Then, the outlier factor of C is

$$\mathrm{OF}(C) = \frac{\mathrm{dis}_n^C}{\mathrm{dis}_0^C}. \tag{18}$$

For example, in Figure 5(a), $n = 2$, which means we consider at most 2 points in a cluster to obtain LB. Thus, we sort the points according to the distance to C_1.centr in a descending order, and we obtain the top-3 points, p_1, p_2, and p_3. Then, we compute the distance between the first point p_1 and C_1.centr: $\mathrm{dis}(p_1, C_1.\mathrm{centr})$, and the distance between the third point p_3 and C_1.centr: $\mathrm{dis}(p_3, C_1.\mathrm{centr})$. The outlier factor of C_1 is $\mathrm{OF}(C_1) = \mathrm{dis}(p_3, C_1.\mathrm{centr})/\mathrm{dis}(p_1, C_1.\mathrm{centr}) = 0.95$. Similarly, we get $\mathrm{OF}(C_2) = \mathrm{dis}(q_3, C_2.\mathrm{centr})/\mathrm{dis}(q_1, C_2.\mathrm{centr}) = 0.5$. Comparing the two outlier factors, we can see that $\mathrm{OF}(C_1)$ is large, which means many points are far from C_1.centr and they have similar distances to C_1.centr. Therefore, we can hardly know whether there are points with large weights in C_1. Conversely, the value of $\mathrm{OF}(C_2)$ is small. We can assert that, in C_2, most of the points closely surround the centroid



(a) Time cost versus k

(b) Point accessing times versus k

Figure 6: The effect of k for the centralized algorithm.

point and there is at least one abnormal point (e.g., p_1, p_2) far from C_2.centr. The weights of these abnormal points are large, and they contribute to selecting a large LB.

As a consequence, we preferentially visit the clusters with small outlier factors, instead of visiting in a random order. This method helps us to obtain a large LB early and improves the filtering effectiveness.

Table 1: Parameter settings for the centralized algorithm.

Parameter	Default	Range of variation
k	20	15–35
n	30	20–40
Data size $\|P\|$ ($\times 10^6$)	1	0.5–2.5
Dimensionality d	3	3–15

6. Experimental Evaluation

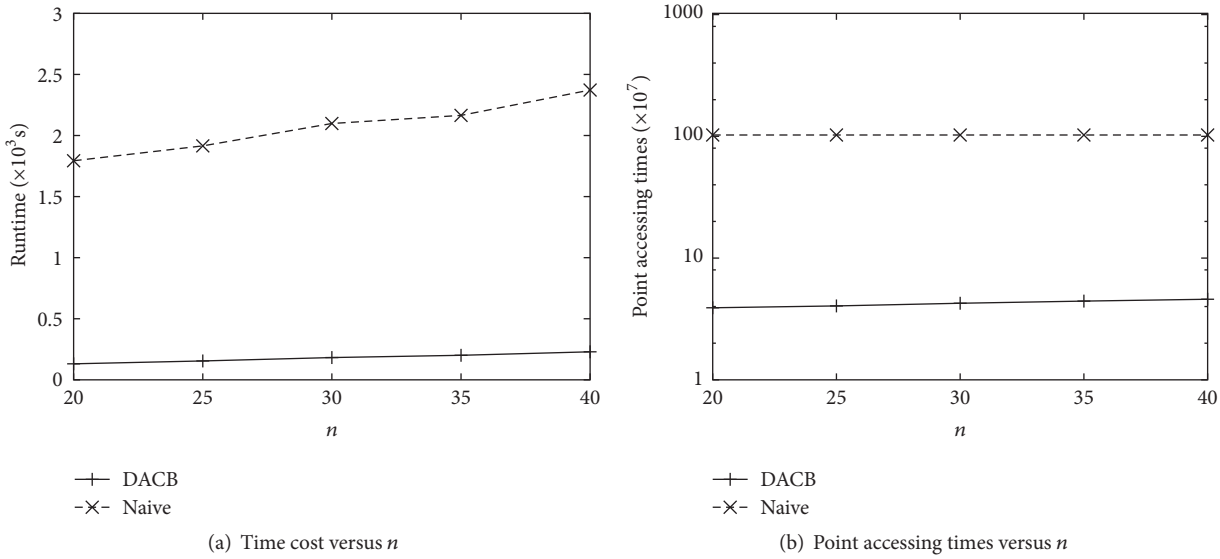
6.1. Method to Define Distance between Points. We use the method in [4] to define the distance between points in practical applications. First, we use a point with several measurable attributes to represent an object in the real world. All the objects can be mapped into European space. Then, for point p_1 representing object o_1 and point p_2 representing object o_2, we use the distance between p_1 and p_2 to measure the difference between o_1 and o_2. Thus, a point with large distances to others is more likely to be an outlier.

6.2. Result Analysis for the Centralized Algorithm. In this section, we first evaluate the performance of the proposed algorithm in a centralized environment using a PC with an Intel Core i7-2600 @3.4 GHz CPU, 8 GB main memory, and 1 TB hard disk. A synthetic dataset is used for the experiments. In detail, given the data size N, we generate $N/1000 - 1$ clusters and randomly assign each of them a center point and a radius. In average, each cluster has 1000 points following Gaussian distribution. At last, the remaining 1000 points are scattered into the space. We implement the proposed method to detect CB outliers (DACB) using JAVA programming language. A naive method (naive) is also implemented as a comparing algorithm, where we simply search each point's kNNs and compute its weight. In the experiments, we are mainly concerned with the runtime to represent the computational efficiency and the point accessing times (PATs) to indicate the disk IO cost. The parameters' default values and their variations are shown in Table 1.

As Figure 6(a) shows, DACB is much more efficient than the naive method because of the pruning and filtering strategies proposed in this paper. With the increase of k, we need to keep tracking more neighbors for a point, so the runtime of the naive method and the DACB becomes larger. Figure 6(b) shows the effect of k on the PATs. For the naive method, each point needs to visit all the other points in the cluster to find its kNNs. Hence, PATs are large. In contrast, for DACB, a point does not have to visit all the other points (Theorem 3), and a large number of points can be filtered out, instead of finding their exact kNNs (Corollary 5). Therefore, the PATs are much smaller.

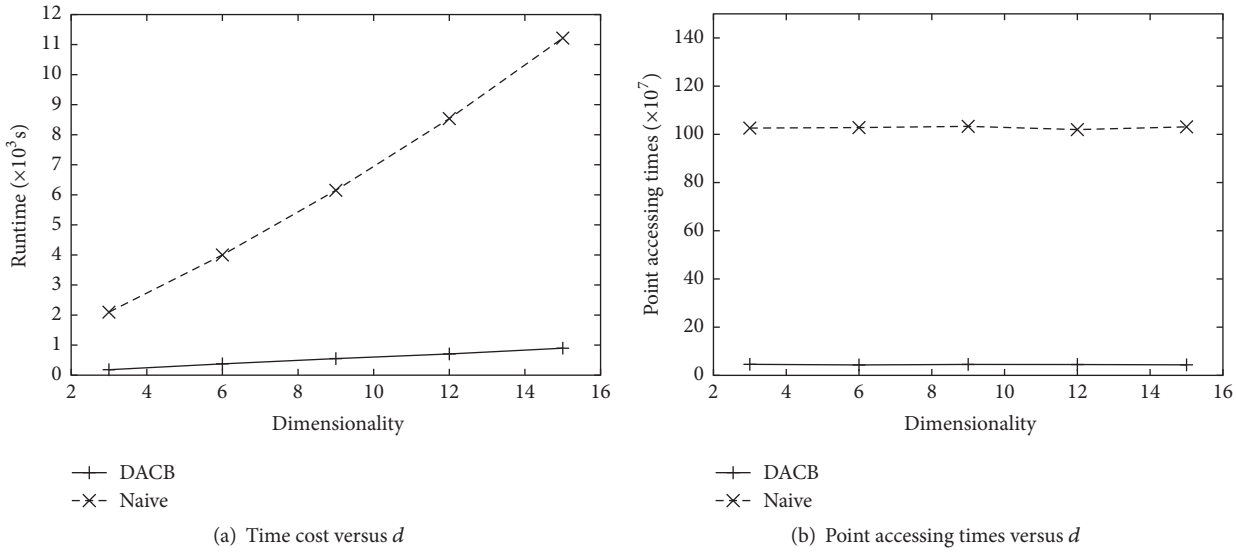
Figure 7 describes the effect of n. As n increases, more outliers are reported. Thus, the runtime of the naive method and the DACB becomes larger. The effect on the PATs is shown in Figure 7(b), whose trend is similar to that in Figure 6(b). Note that the PATs of DACB increase slightly with n, whereas the PATs of the naive method keep unchanged.

In Figure 8, with the increase of the dimensionality, a number of operations (e.g., computing the distance of two points) become more time-consuming. Hence, the time cost of the two methods becomes larger. But the variation of the dimensionality does not affect the PATs. The effect of the data size is described in Figure 9. Clearly, with the increase of the data size, we need to scan more points to find the outliers.



(a) Time cost versus n (b) Point accessing times versus n

Figure 7: The effect of n for the centralized algorithm.



(a) Time cost versus d (b) Point accessing times versus d

Figure 8: The effect of dimensionality for the centralized algorithm.

Therefore, both of the runtime and the PATs are linear to the data size.

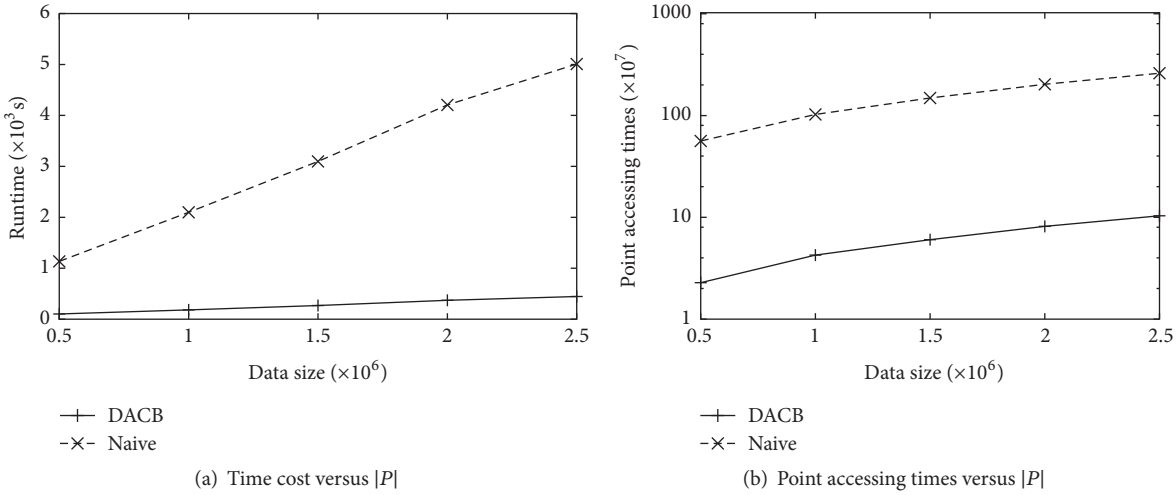
6.3. Result Analysis for the Distributed Algorithm. In this section, we further evaluate the performance of the proposed algorithm for distributed outlier detection. In the experiments, we mainly consider the time cost and the network transmission quantity (NTQ). The data size and the cluster scale are shown in Table 2. The other parameters settings are identical to those described in Section 6.2.

Figure 10 shows the effect of parameter k. The curve "with SOO" represents the DACB algorithm. The curve "without SOO" represents a basic distributed algorithm for outlier detection without the Scanning Order Optimization (SOO) method described in Section 5.2. As we can see, as the value of k increases, both algorithms cost more time (the reason has been discussed in Section 6.2). But LB is not sensitive to k, and thus NTQ changes very slightly. Comparing the two algorithms, we can see that, with the help of SOO, a large LB can be obtained early and a lot more points can be filtered out efficiently. As a result, we can reduce the network overhead and improve the computing efficiency.

In Figure 11, we evaluate the effect of parameter n. With the increase of n, we consider more points as outliers.

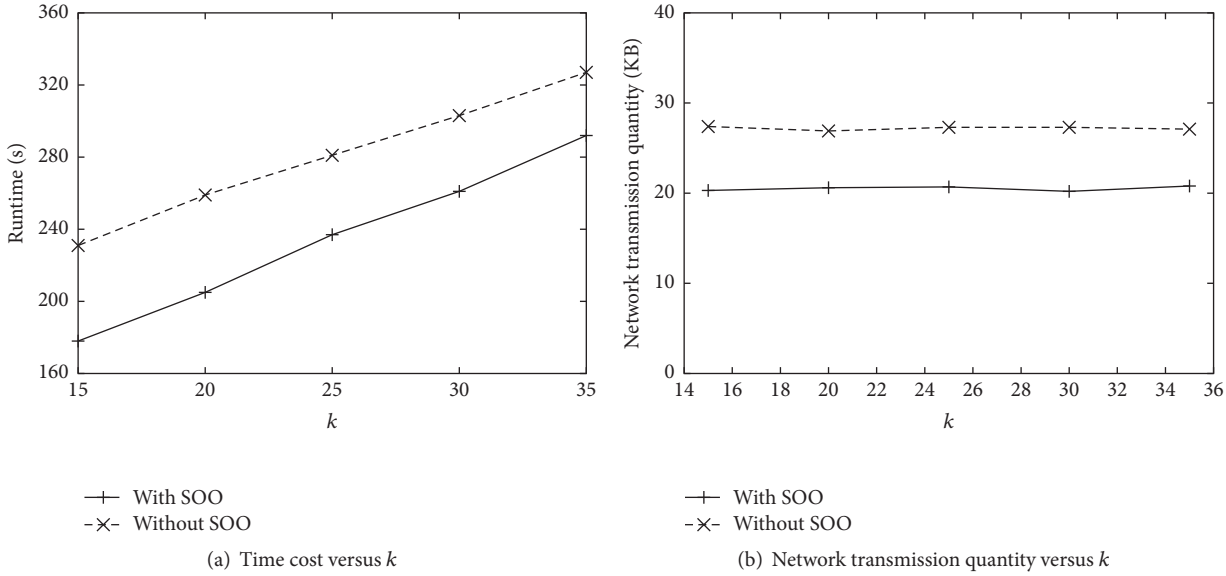
Table 2: Parameter settings for the distributed algorithm.

Parameter	Default	Range of variation
Data size ($\times 10^7$)	1	0.5–2.5
Number of slave nodes	10	6–14



(a) Time cost versus $|P|$ (b) Point accessing times versus $|P|$

Figure 9: The effect of data size for the centralized algorithm.



(a) Time cost versus k (b) Network transmission quantity versus k

Figure 10: The effect of k for the distributed algorithm.

Thus, the threshold LB becomes smaller and the filtering performance decreases. On the other hand, since a large n means that more points will be transmitted to the master node to compute LB, NTQ also increases.

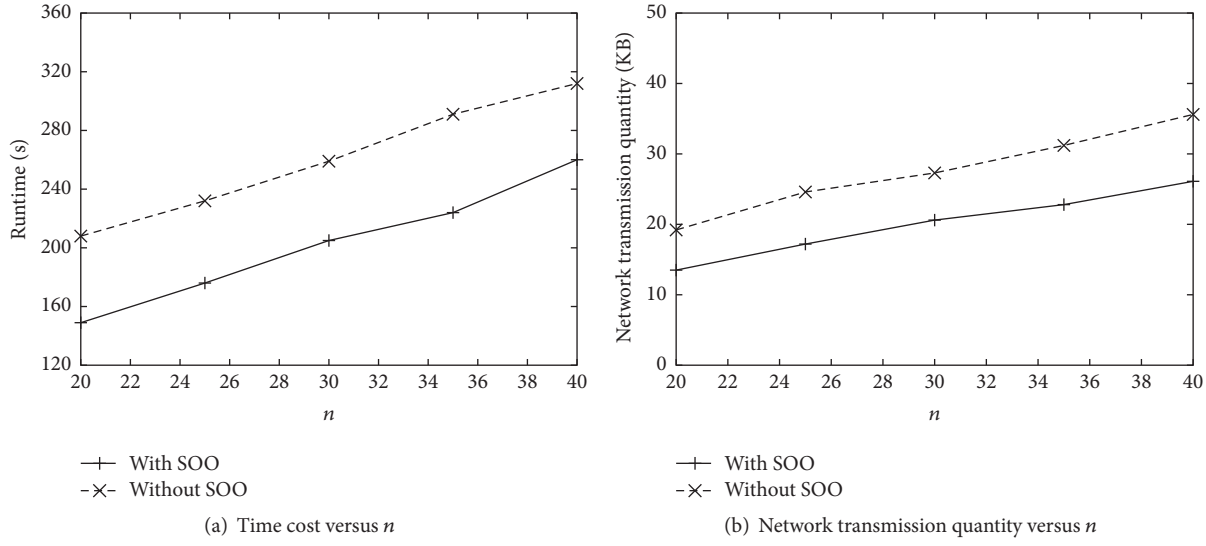
Figure 12(a) evaluates the effect of the dimensionality on the time cost, which shows the same trend as the result in Figure 8(a). In Figure 12(b), we test the effect of the dimensionality on NTQ. Clearly, each slave node needs to send the local top-n points with the largest weights to the master node. The transmission contents include points' ID, the values of all the dimensionality, and the weights. Therefore, more dimensionality leads to higher transmission quantity.

As Figure 13(a) shows, with the increase of the data size, more points need to be scanned to find the outliers; thus, the time cost becomes larger. We can also see that, in Figure 13(b), NTQ increases with the data size, but the change is very small. This is because the value of LB becomes stable after a certain amount of calculation, and it is not sensitive to the data size.

The effect of the cluster scale is tested in Figure 14. As more slave nodes are used, the workload on each node becomes smaller, and thus the computing speed is improved. However, to compute LB, the master node needs to collect n points from each slave node. Therefore, NTQ increases with the cluster scale. Note that, for the dataset with 10^7 points, NTQ is still maintained at a KB level. The network overhead of DACB is acceptable.

7. Related Work

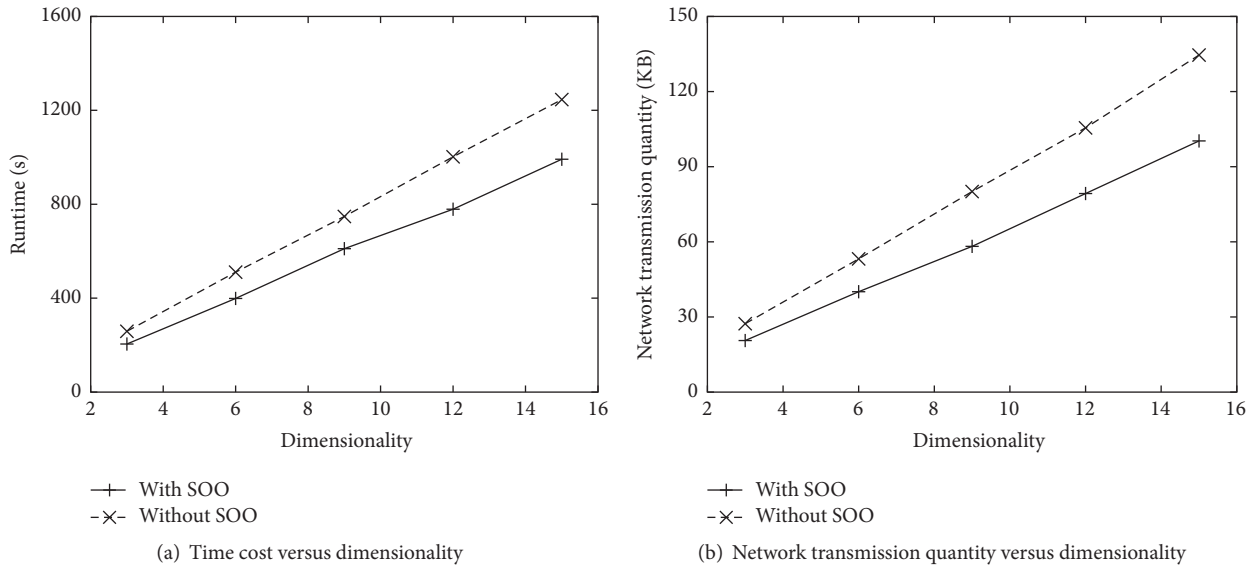
Outlier detection is an important task in the area of data management, whose target is to find the abnormal objects in a given dataset. The statistics community [2, 3] proposed the model-based outliers. The dataset is assumed to



(a) Time cost versus n

(b) Network transmission quantity versus n

Figure 11: The effect of n for the distributed algorithm.



(a) Time cost versus dimensionality

(b) Network transmission quantity versus dimensionality

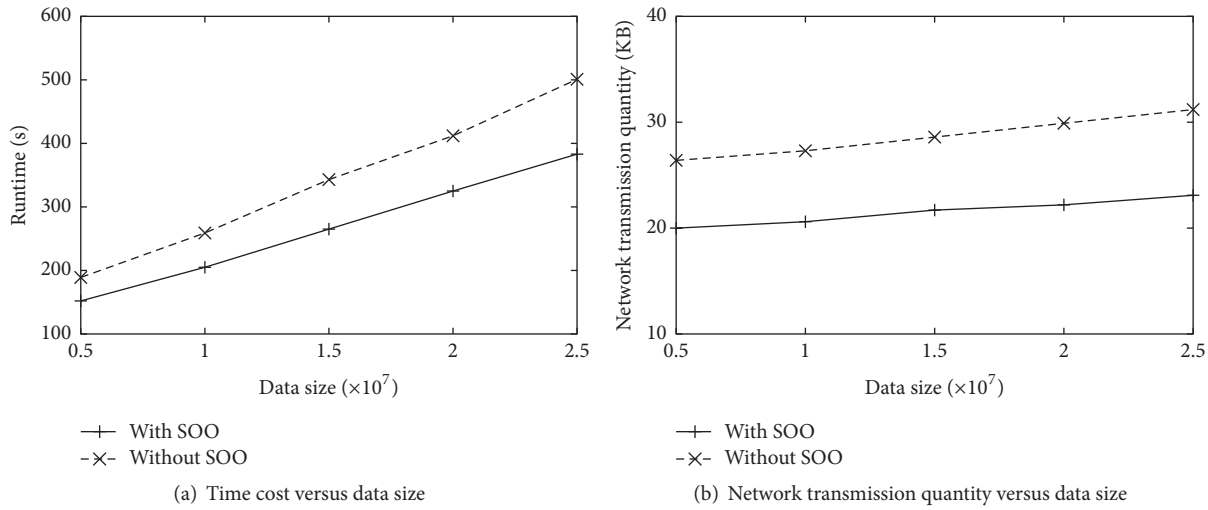
Figure 12: The effect of dimensionality for the distributed algorithm.

follow a distribution. An outlier is the object that shows obvious deviation from the assumed distribution. Later, the data management community pointed out that building a reasonable distribution is almost an impossible task for high-dimensional datasets. To overcome this weakness, they proposed several model-free approaches [22], including distance-based outliers [4–6] and density-based outliers [7].

A number of studies focus on developing efficient methods to detect outliers. Knorr and Ng [4] proposed the well-known nested-loop (NL) algorithm to compute distance-based outliers. Bay and Schwabacher [23] proposed an improved nested-loop approach, called ORCA. The approach efficiently prunes the searching space by randomizing the dataset before outlier detection. Angiulli and Fassetti [24] proposed DOLPHIN, which can reduce the disk IO cost through maintaining a small subset of the input data in main memory. Several researchers adopt the spatial indexes to further improve the computing efficiency. Examples include R-tree [25], M-tree [26], and grids. However, the performance of these methods is quite sensitive to the dimensionality. To improve the computing efficiency, some researchers attempt to use a distributed or parallel method to detect outliers. Examples include [27–29].

8. Conclusion

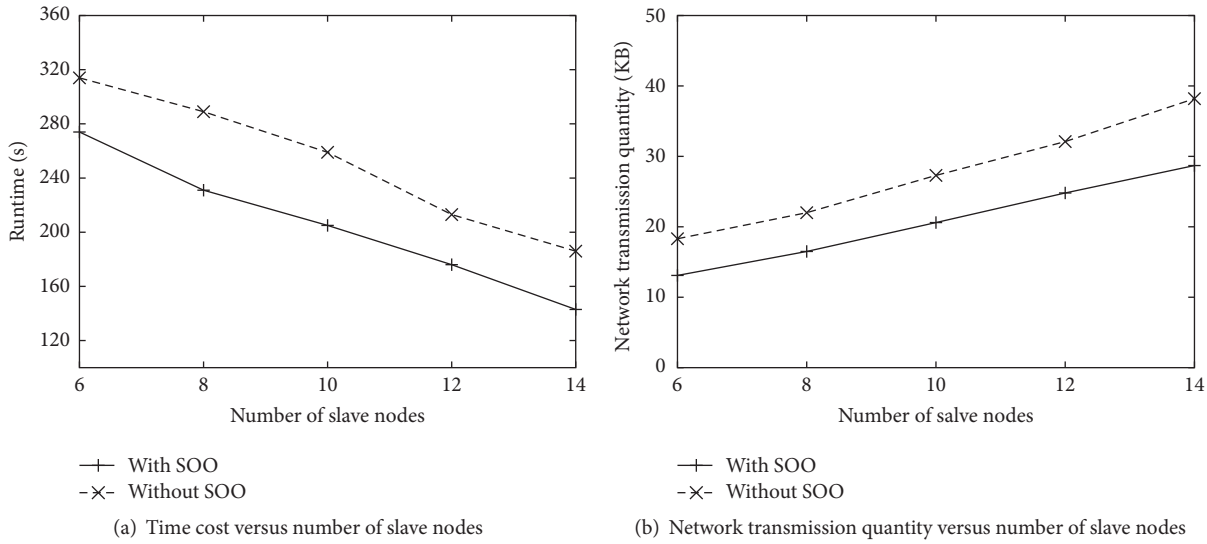
In this paper, we studied the problem of CB outlier detection in a distributed environment and proposed an efficient algorithm, called DACB. This algorithm adopts a master-slave architecture. The master node monitors the points with large weights on each slave node and computes a threshold.



(a) Time cost versus data size

(b) Network transmission quantity versus data size

Figure 13: The effect of data size for the distributed algorithm.



(a) Time cost versus number of slave nodes

(b) Network transmission quantity versus number of slave nodes

Figure 14: The effect of number of slave nodes for the distributed algorithm.

On each slave node, we designed a pruning method to speed up the kNN searching and a filtering method that can use the threshold to filter out a large number of unpromising points. We also designed an optimization method for cluster scanning, which can significantly improve the filtering performance of the threshold. Finally, we evaluated the performance of the proposed approaches through a series of simulation experiments. The experimental results show that our method can effectively reduce the runtime and the network transmission quantity for distributed CB outlier detection.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants nos. 61602076 and 61371090, the Natural Science Foundation of Liaoning Province under Grant no. 201602094, and the Fundamental Research Funds for the Central Universities under Grant no. 3132016030.

References

[1] D. M. Hawkins, *Identification of Outliers*, Springer, 1980.

[2] V. Barnett and T. Lewis, *Outliers in Statistical Data*, John Wiley & Sons, New York, NY, USA, 1994.

[3] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*, John Wiley & Sons, 2005.

[4] E. M. Knorr 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, New York, NY, USA, August 1998.

[5] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 427–438, 2000.

[6] 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.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 93–104, 2000.

[8] G. Huang, S. Song, J. N. D. Gupta, and C. Wu, "Semi-supervised and unsupervised extreme learning machines," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2405–2417, 2014.

[9] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 985–990, July 2004.

[10] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, 2006.

[11] G.-B. Huang, "What are extreme learning machines? Filling the gap between Frank Rosenblatt's Dream and John von Neumann's puzzle," *Cognitive Computation*, vol. 7, no. 3, pp. 263–278, 2015.

[12] V. Cherkassky, "The nature of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 8, no. 6, p. 1564, 1997.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

[15] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Online sequential fuzzy extreme learning machine for function approximation and classification problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 39, no. 4, pp. 1067–1072, 2009.

[16] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[17] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[18] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[19] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: analysis and an algorithm," in *Advances in Neural Information Processing Systems*, vol. 2, pp. 849–856, 2002.

[20] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–27, 2009.

[21] X. Wang, D. Shen, M. Bai, T. Nie, Y. Kou, and G. Yu, "Cluster-based outlier detection using unsupervised extreme learning machines," in *Proceedings of ELM-2015*, Springer, 2016.

[22] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1641–1650, 2003.

[23] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pp. 29–38, ACM, August 2003.

[24] F. Angiulli and F. Fassetti, "Very efficient mining of distance-based outliers," in *Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM '07)*, pp. 791–800, November 2007.

[25] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *ACM SIGMOD Record*, vol. 14, no. 2, pp. 47–57, 1984.

[26] M. Patella, P. Ciaccia, and P.-M. Zezula, "M-tree: an efficient access method for similarity search in metric spaces," in *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97)*, pp. 426–435, Athens, Greece, August 1997.

[27] F. Angiulli, S. Basta, S. Lodi, and C. Sartori, "Distributed strategies for mining outliers in large data sets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1520–1532, 2013.

[28] 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.

[29] E. Lozano and E. Acuña, "Parallel algorithms for distance-based and density-based outliers," in *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05)*, pp. 729–732, Houston, Tex, USA, November 2005.