

Continuous Adaptive Outlier Detection on Distributed Data Streams

Liang Su, Weihong Han, Shuqiang Yang, Peng Zou, and Yan Jia

School of Computer Science National University of Defense Technology Changsha
410073, China

liangsumail@gmail.com, 13808419839@hnmcc.com, sqyang9999@126.com,
zpeng@nudt.edu.cn, jiayanjy@vip.sina.com

Abstract. In many applications, stream data are too voluminous to be collected in a central fashion and often transmitted on a distributed network. In this paper, we focus on the outlier detection over distributed data streams in real time, firstly, we formalize the problem of outlier detection using the kernel density estimation technique. Then, we adopt the *fading strategy* to keep pace with the transient and evolving natures of stream data, and *mico-cluster technique* to conquer the data partition and “one-pass” scan. Furthermore, our extensive experiments with synthetic and real data show that the proposed algorithm is efficient and effective compared with existing outlier detection algorithms, and more suitable for data streams.

1 Introduction

Following with the fast improvement of hardware and communication technologies, many modern data acquisition systems are essentially automatic, distributed and continuous. For example, networking applications (multiple web/blog crawlers, intrusion detection, network monitoring), financial services (distributed fraud detection, financial monitoring, click stream analysis) and military application(soldier location streams), etc. In these environments, all stream data generally have these characters: continuous and unbounded, distributed and evolvable over time. So, distributed data stream model was provided, which is suitable for these applications very well.

In this paper we study a quintessential monitoring problem on continuously changing distributed data sources, namely, *outlier detection*, which is an important part of data mining. And the outliers may point out some surprising and suspicious activities or observations which appear to be inconsistent with the remainder data. Formally, we have $m + 1$ distributed nodes (one leader node and m child nodes), illustrated in Figure 1. Each child node i has a changing source of data S_t^i at time t , and individual data items in the sources may be high dimensional including numerical values, text, audio or video. We are more concerned about monitoring some desirable properties of the union $\bigcup_{i=1}^m S_i^t$ of data on all nodes in real time. In order to understand the challenge in monitoring outliers over distributed nodes, there are two straightforward “solutions”:

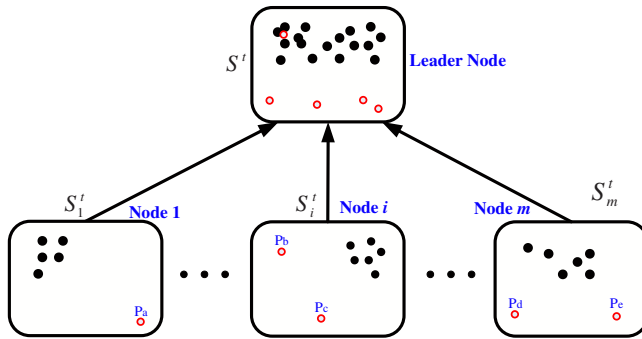


Fig. 1. Continuous Distributed Outlier Detection (red points are local outliers)

1. Each node sends the newly gathered data to the central node. The central node updates the outliers of $\bigcup_{i=1}^m S_i^t$ whenever a new point is received.

2. Periodically, each node sends the newly gathered data to the central node, every τ seconds. The central node updates the outliers every τ seconds.

The first method has communication and central processing bottleneck because all data are aggregated to central node; The second solution, which is similar to a batch processing of the first method, will have latency because of the time τ and also does not overcome the communication problem. Furthermore, the outliers would change radically within the time period. So these two solutions are not suitable for monitoring applications. Therefore, we develop efficient online outlier detection techniques. And our contributions are as follows:

1. We propose two novel outlier measures, using *kernel density estimation technique* [1] to approximate the stream data distribution, which are compatible with distance-based outlier and density-based outlier (see Definition 2 and Definition 3).

2. We adopt the *fading strategy* to keep pace with the transient and evolving natures of stream data, and *mico-cluster technique* (commonly used for data compression or summarization) to deal with the data partition.

3. Finally, extensive experiments including synthetic and real data sets, demonstrate that our proposed methods are efficient and effective.

The rest of this paper is organized as follows. Section 2 introduces the related researches of outlier detection. Section 3 presents some preliminaries and the problem definitions. Section 4 provide the effective algorithm (**MOD** algorithm). Section 5 evaluates the effectiveness and efficiency of the algorithm compared with existing algorithms. Section 6 concludes the paper.

2 Related Work

Methods for outlier detection are drawing increasing attention. The salient approaches can be classified as either distribution-based, clustering, distance-based, or density-based.

The distribution-based approach [2,3] assumes the data following a distribution model (e.g. Normal) and flags as outliers those objects which deviate from the model. Thus, such approaches do not work well in moderately high-dimensional (multivariate) spaces, and have difficult to find a right model to fit the evolving data stream. To overcome these limitations, researchers have turned to various non-parametric approaches (clustering, distance-based, and density-based).

Most clustering algorithms (e.g. CLARANS [4], DBSCAN [5], BIRCH [6], WaveCluster [7], CLIQUE [8]), are to some extent capable of handling exceptions. However, since the main objective is to find clusters, they are developed to optimize clustering, and regard outliers as “by-products”.

Distance based approaches [9,10,11,12,13,14,15], first proposed by E.M. Knorr and R.T. Ng [9], attempt to overcome limitations of distribution-based approach and they detect outliers by computing distances among points. A point p in a data set T is a distance-based outlier ($DB(\rho, r)$ -outlier) if at most a fraction ρ of the points in T lie within distance r from p . It has an intuitive explanation that an outlier is an observation that is sufficiently far from most other observations in the data set. However, it will be no effect when the data points exhibit different densities in different regions of the data space or across time, because this outlier definition is based on a single, global criterion determined by the parameters ρ and r , so more robust density-based techniques were provided.

Density-based approaches proposed originally by M. Breunig, et al. [16] which defines a local outlier factor (LOF) for each point depending on the local density of its neighborhood. In general, points with a high LOF are flagged as outliers. It has attracted considerable attention [17,18,19], and a large number of algorithms have been developed which concern how to define the density or accelerate the efficiency, such as, LOCI [17] method.

In the distributed data stream environments, Babcock and Olston presented an original algorithm for distributed top- k monitoring [20]. Amit Manjhi, et al. [21] thought of finding recently frequent items. S. Subramaniam, et al. [22] cared about the outlier over sensor stream data. Graham Cormode, et al. [23] considered continuous clustering.

3 Preliminaries and Problem Statement

In this section, first, we introduce the two preliminaries: distributed data stream model and kernel density estimation. Then, we propose two novel outlier measures which are compatible with distance-based and density-based outlier.

3.1 Distributed Data Stream Model

A data stream consists of an unbounded sequence D_1, D_2, \dots of numeric values which $D_i = (D_i^1, D_i^2, \dots, D_i^d)$ and $D_i^j \in \mathbb{R}$ for $i, j \in \mathbb{N}, j \in [1, d]$. Then, distributed data stream model composes of $m + 1$ nodes: a central *leader node* N_c , and m remote stream monitor nodes N_1, N_2, \dots, N_m . Each node just monitors one stream data, illustrated in Figure 1. Except otherwise stated, we assume

the stream data as *independent and identical distributed*(**iid**) observations of an unknown continuous random variable which is reasonable for most distributed data stream applications. On the other hand, in many data stream applications, only the most recent elements are important in data mining, while the old ones are very little or not. For example, in a stream of stock market data, people are more concerned about the moving average of the stock price over all observations made in the last hour. So, we only care about the data in the *sliding window* (W) which contains the most recent N elements of stream.

3.2 Kernel Density Estimation

Kernel density estimation is to reveal the unknown density of a distribution by selecting a suitable density estimator and has been successfully applied in diverse application scenarios [1]. Theoretically, kernel density function $f_h(x)$ is sure to converge to the real density $f(x)$ function for arbitrary distribution. More formally, assume that we have a data set \mathcal{D} , containing n points whose values are $\overline{X}_1, \overline{X}_2, \dots, \overline{X}_n$. We can approximate the underlying distribution $f(x)$ using the following kernel function ($\mathcal{K}(x)$, *kernel function*, is a function of random variable X . h , *kernel width*, determines the smoothing level of kernel function):

$$\tilde{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n \mathcal{K} \left(\frac{x - \overline{X}_i}{h} \right), \quad x, \overline{X}_i \in \mathbb{R}, \quad \int_{x \in \mathbb{R}} \mathcal{K}(x) dx = 1 . \quad (1)$$

$$\mathcal{K} \left(\frac{x - \overline{X}_i}{h} \right) = \begin{cases} \frac{3}{4} \cdot \frac{1}{h} \left(1 - \left(\frac{x - \overline{X}_i}{h} \right)^2 \right), & \left| \frac{x - \overline{X}_i}{h} \right| < 1 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Common used kernel functions are Epanechnikov, Gaussian, Quartic and Triweight kernel, etc. Since the Gaussian kernel is unbounded ($x \in (-\infty, +\infty)$), it exacerbates the cost of computing integral. Quartic and Triweight kernel are fourth and sixth function each other. So, we choose the Epanechnikov kernel (see Equation 2) which is a square function, more easy to integrate and has bound.

3.3 Novel Outlier Measures

Intuitively, outliers can be defined as given by Hawkins [24] in Definition 1. There are several formal definitions of an outlier in Section 2. In our work, combining the meaning of Definition 1 with kernel density estimation, naturally we can come to a conclusion — the probability of one point is more close to zero, the point is more likely to be an outlier. So, we use the following Equation 3 to measure the outlier degree in this setting. Then, two variations based on the commonly-used outlier definitions are presented.

Definition 1 (Hawkins-Outlier). *An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.*

$$\Phi(p, r) = P[p - r, p + r] = \int_{p-r}^{p+r} \hat{f}_h(\mathbf{x}) d\mathbf{x} . \quad (3)$$

Distance-based Kernel Estimation Outlier: It is a variation of *Distance-based Outlier* [9]. Naturally, we present the definition of *Distance-based Kernel Estimation Outlier* (DisKE-Outlier) (see Definition 2). However, it also has the same limitations as distance-based methods. So we provide a more robust outlier measure: *DenKE-Outlier* in Definition 3.

Definition 2 (DisKE-Outlier). A point is a “DisKE-Outlier” if $\Phi(p, r) \leq \rho$.

Density-based Kernel Estimation Outlier: It is a variation of *Density-based Outlier* [16]. Spiros Papadimitriou, et al. [17] provided an outlier metric — Multi-Granularity Deviation Factor (MDEF). For any given point, MDEF is a measure of how the neighborhood count of p (in its counting neighborhood) compares with that of the points in its sampling neighborhood. A point is flagged as an outlier, if its MDEF is significantly different from that of the local averages. r is the sampling neighborhood distance and αr ($\alpha \in (0, 1)$) is the range over which the neighborhood counts are considered. Correspondingly, we can define a *Density-based Kernel Estimation Outlier* (DenKE-Outlier) as follows:

Definition 3 (DenKE-Outlier). A point is defined as a “DenKE-Outlier” that $\frac{\Phi(p, \alpha r)}{\sum_{q \in \Omega_r} \Phi(q, \alpha r) / |\Omega_r|} < \xi$, ξ is a real value parameter to define how significant the point p is to the average of its neighbors. Ω_r is a point set that contains all points which distances are below r to point p . $|\Omega_r|$ is the count of Ω_r .

4 Micro-cluster Based Outlier Detection Algorithm

In this section, firstly, we discuss the selection of kernel width which significantly affects the accuracy of algorithm. Secondly, the fading strategy is proposed to conquer the evolving nature of stream data. Thirdly, we provide the approximate JS-divergence technique to save the traffic between the leader node and other child nodes. Finally, we use the micro-cluster CF-tree [6] data structure to condense the stream data and meet the “one-pass” scan.

4.1 Selecting Kernel Width

The kernel density estimation in the Figure 2 indicate that the *kernel width* significantly affects the shape of a kernel function. If the width is chosen too high, the estimation is over-smoothed and hides important details. Otherwise, the estimation is under-smoothed and brings illusive details, resulting in heavy computation. A widely used rule for approximating the kernel width is the Scott’s rule [1] (h in Equation 4). However, these strategies depend on the complete sample, which is impracticable in data stream scenario. To overcome this problem, we adopt an approximate solution, only considering the data in sliding window. The number of sliding window (N) is more large, the approximate kernel width

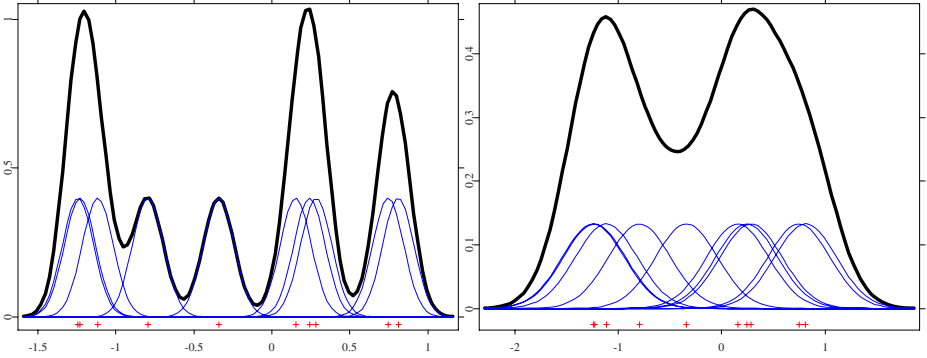


Fig. 2. Kernel density estimation, underlying standard normal kernel and kernel width with $h = 0.1$ (left) and $h = 0.3$ (right)

\tilde{h} is more close to h . σ is the standard deviation of whole stream data and $\tilde{\sigma}$ is the sample standard deviation of sliding window.

$$h \approx 2.345\sigma n^{-1/5} \approx \tilde{h} \triangleq 2.345\tilde{\sigma}(N)^{-1/5}, \quad \hat{h} = \min \left\{ 2.345\tilde{\sigma}n^{-1/5}, D_x^k \right\}. \quad (4)$$

Definition 4 (*k*-nearest neighbor set). D_p^k is the distance between point p and its k -nearest neighbor. Δ_p , named “ k -nearest neighbor set of point p ”, is defined as a set that contains all points which distance is not larger than D_p^k . Δ_S , named “ k -nearest neighbor set of set S ”, is defined as $\bigcup_{p \in S} \Delta_p$.

The \tilde{h} width will increase very fast following by the $\tilde{\sigma}$ and n in Equation 4. In order to balance the quality and efficiency, we consider the distance between the point and its k -nearest neighbors. So an improved kernel width (\hat{h}) is listed in Equation 4. Extensive experiments demonstrate its efficiency.

4.2 Capturing Evolution over Data Stream

Because of the dynamic nature of stream data, it is the inherent obstacle that must be conquered by an effective and robust stream analysis technique. To capture the evolution, we are conscious of the problem in Equation 1: each kernel entry is equally weighted with constant $\frac{1}{n}$. So we present a *fading strategy* that couple the kernel density estimation with exponential smoothing [25]. The basic idea of fading strategy is to give older data less weight and to gradually discount the history data, which is widely used in the area of time series analysis and forecasting. Then, Equation 1 is adjusted to Equation 5. Notice that the sum of weights in Equation 5 is $\sum_{i=1}^N \frac{\omega^{N-i}}{\sum_{j=1}^N \omega^{j-1}} = 1$, which is equal to $\sum_{i=1}^n \frac{1}{n} = 1$ in Equation 1. When $\omega = 1$, Equation 5 comes back to Equation 1. The *fading factor* (ω) determines the rate of fading. The higher the value of ω the lower importance of the historical data compared to more recent data. The sliding window becomes an *evolving sliding window*. And also, we can tune the impact degree of past and current data through ω to match different applications.

$$\hat{f}_h(x) = \frac{1}{h} \sum_{i=1}^N \left(\frac{\omega^{N-i}}{\sum_{j=1}^N \omega^{j-1}} \cdot \mathcal{K} \left(\frac{x - \bar{X}_i}{h} \right) \right), \quad x \in \mathbb{R}, \quad \omega \in (0, 1]. \quad (5)$$

4.3 Comparison for Different Distributions

The difference of two distributions can seriously influence the traffic between the leader node and other child nodes. Intuitively, the less change of distribution in one node, the less traffic between this node and the leader node. So it is very important to compare the difference between two distributions. Several methods have been proposed to quantify the difference between probability density distributions [1]. One widely used measure is Ali-Silvey divergences, which is more natural than ℓ_p norms. $p(x)$ and $q(x)$ are probability distribution functions over random variable x , Ω is the value set of x . Two most common divergences are the asymmetric KL-divergence and the symmetric JS-divergence in Equation 6. However, because kernel density estimation method may assign probability of zero to some regions, KL-divergence is undefined and meaningless for the \ln function. So, we choose the JS-divergence which is meaningful to any $x \in \Omega$.

$$D_{JS}(p, q) = \sum_{i=1}^n \left(p_i \ln \frac{2p_i}{p_i + q_i} + q_i \ln \frac{2q_i}{p_i + q_i} \right). \quad (6)$$

$$\hat{D}_{JS}(p, q) = \sum_{i \in \Delta_{S_{in} \cup S_{out}}} \left(p_i \ln \frac{2p_i}{p_i + q_i} + q_i \ln \frac{2q_i}{p_i + q_i} \right). \quad (7)$$

Theoretically, a new data incoming the sliding window or an old data being dropped out will change the kernel density for all data points in the whole sliding window. Obviously, it is too inefficient and unscalable to suit the evolving data stream over time. The incoming data and dropped data are a very small proportion to the capacity of sliding window. The points are more close to the changeable data, the influence is more serious. To balance the accuracy and complexity, we provide an approximate JS-divergence ($\hat{D}_{JS}(p, q)$ in Equation 7) which is only concerned about the changeable data (S_{in} and S_{out}) and their k -nearest neighbors set. As the number $|\Delta_{S_{in} \cup S_{out}}|$ is much less than n , time complexity is reduced to $O(d(|\Delta_{S_{in} \cup S_{out}}| + \log n))$ which $\log n$ is the time spending to search k -nearest neighbor set. $\hat{D}_{JS}(p, q)$ is non-negative real number and unbounded. In practical, we can select a parameter λ (*divergence threshold*) as the upper-bound of $\hat{D}_{JS}(p, q)$. If $\hat{D}_{JS}(p, q) > \lambda$, there is a significant change and corresponding node transfer its kernel estimation function to the leader node. Otherwise, no transformation.

4.4 Micro-cluster Definition and MOD Algorithm

In this section, we describe the *Micro CF-Tree Based Distributed Outlier Detection Algorithm* (**MOD** Algorithm) that can deal with the two outlier categories (DisKE-Outlier and DesKE-Outlier), in a distributed manner. We

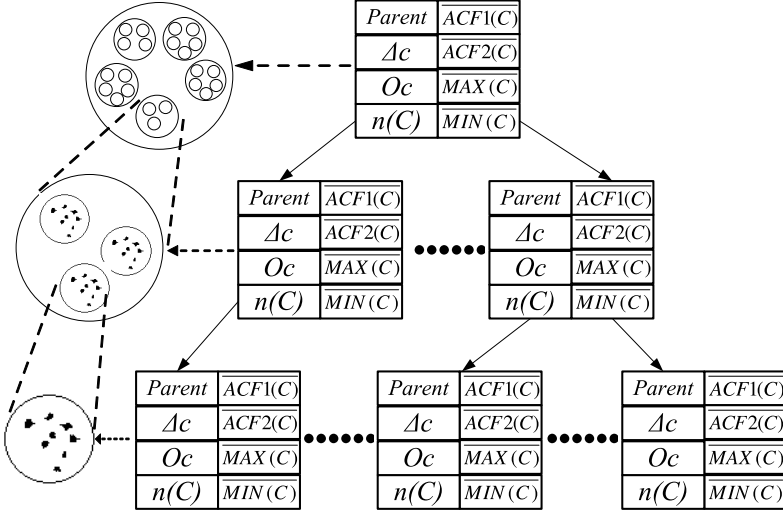


Fig. 3. Extended Micro-Cluster Feather Tree

want to identify outliers in the leader node. By observation of the definition of DisKE-Outlier, it is easy to find that the leader node need only to examine the values that have been marked as outliers by its child nodes. All the other data values can be safely ignored, since they cannot possibly be outliers. However, DesKE-Outliers are non-decomposable because the neighbors of one point may be distributed in different child nodes. A feasible solution is that all child nodes have a copy of the kernel density function of leader node. The child node transfers its kernel estimation function to the leader node when its $\hat{D}_{JS}(p, q) > \lambda$, and the leader node broadcasts its kernel estimation function to its child nodes when $\hat{D}_{JS}(p, q) > \lambda$ in N_c . We define the concept of micro-cluster-based outlier more precisely in Definition 5 which is a *CF-Tree* [6]. It is very easy to create and maintain the clusters in a single pass. The whole algorithm is described in algorithm (1).

Definition 5. An extended micro *CF-tree* for a set of d -dimensional points $D_{i_1}, D_{i_2}, \dots, D_{i_n}$ with time-stamp $T_{i_1}, T_{i_2}, \dots, T_{i_n}$ and each point $D_j = (d_j^1, d_j^2, \dots, d_j^d)$, is defined as the $(4d + 1 + |\Delta_c| + |O_c|)$ -tuple $(\overline{ACF1(C)}, \overline{ACF2(C)}, \overline{MIN(C)}, \overline{MAX(C)}, \Delta_c, O_c, n(C))$. $(\overline{ACF1(C)}, \overline{ACF2(C)}, \overline{MIN(C)}, \overline{MAX(C)})$ are four vectors of d entries. The definition of each of these entries is as follows, Figure 3 is the whole data structure:

- The p -th entry of $\overline{ACF1(C)}$ is equal to $\sum_{j=1}^{n(C)} d_{i_j}^p$, The p -th entry of $\overline{ACF2(C)}$ is equal to $\sum_{j=1}^{n(C)} (d_{i_j}^p)^2$;
- The p -th entry of $\overline{MIN(C)}$ is equal to $\min_{j=1}^{n(C)} d_{i_j}^p$, The p -th entry of $\overline{MAX(C)}$ is equal to $\max_{j=1}^{n(C)} d_{i_j}^p$;

Algorithm 1. CF-Tree Based Distributed Outlier Detection Algorithm

Input: m data streams S_1, \dots, S_m in N_1, \dots, N_m , the divergence threshold λ , ρ is the outlier proportion in DisKE-Outlier, the distance r , α is the distance proportion in DesKE-Outlier.

Output: the outlier set O_c

Function OutlierDetetion()

- 1: $BDisKE \leftarrow$; // variable $BDisKE = true$ indicates using DisKE-Outlier detection method, otherwise using DesKE-Outlier detection method.
- 2: **LeaderProcess()**; // initiate the process for the leader node;
- 3: **for**($i = 1$ **to** m) **do**
- 4: **ChildProcess()**; // initiate the process for each child node;

Function LeaderProcess()

- 1: find the closest micro-cluster from the CF-Tree to the point p which a new point from a child node, assumed the cluster as GC which is a leaf node in the CF-Tree;
- 2: **if**($p \notin O_{GC}$) **then isOutlier**(p, GC);
- 3: update $\tilde{f}_h(x)$; // using Equation 1 and kernel width in Equation 4;
- 4: **if**($\widehat{D}_{JS}(\tilde{f}_h^{new}, \tilde{f}_h^{old}) > \lambda$) **then** broadcast \widehat{f}_h^{new} to the all child node;
- 5: **if**($p \in O_{GC}$, or is marked as an outlier) **then** report point p as an outlier;
- 6: **else** insert p to micro-cluster GC ;

Function ChildProcess()

- 1: find the closest micro-cluster from the CF-Tree to the new point p , assumed as LC which is a leaf node in the CF-Tree;
- 2: **if**($p \notin O_{LC}$) **then isOutlier**(p, LC);
- 3: update $\tilde{f}_h(x)$; // using Equation 1 and kernel width in Equation 4;
- 4: **if**($\widehat{D}_{JS}(\tilde{f}_h^{new}, \tilde{f}_h^{old}) > \lambda$) **then** send the point p and \widehat{f}_h^{new} to the leader node;
- 5: **if**($p \notin O_{GC}$, and is not marked as an outlier) **then** insert p to LC ;

Function isOutlier(point p , micro cluster MC)

- 1: **if**($BDisKE =$)
 - 2: **if**($\Phi(p, r) < \rho$) **then** mark p as an outlier and add to O_{MC} ;
 - 3: **else** sum up to $\Omega_r = \bigcup_{q \in MC} \Delta_q$;
 - 4: **if**($\frac{\Phi(p, \alpha r)}{\sum_{q \in \Omega_r} \Phi(q, \alpha r) / |\Omega_r|} < \xi$) **then** mark p as an outlier and add to O_{MC} ;
 - 5: update $\tilde{f}_h(x)$; // using Equation 1 and kernel width in Equation 4;
-

• The Δ_c is defined in Definition 4, O_c holds the outliers in cluster C , $n(C)$ is the count number in Micro-Cluster C .

5 Experiments

In this section, we will present the experimental results for our algorithms comparing with two typical algorithms. One is the distanced-based algorithm — NL algorithm [9] which is proposed by E.M. Knorr and R.T. Ng, and the other is an density-based algorithm — LOCI algorithm [17] which is presented by Spiros Papadimitriou, et al. There are three primary purposes: (1) Comparing the precision and recall ratio in four real data sets. (2) Verifying the algorithm

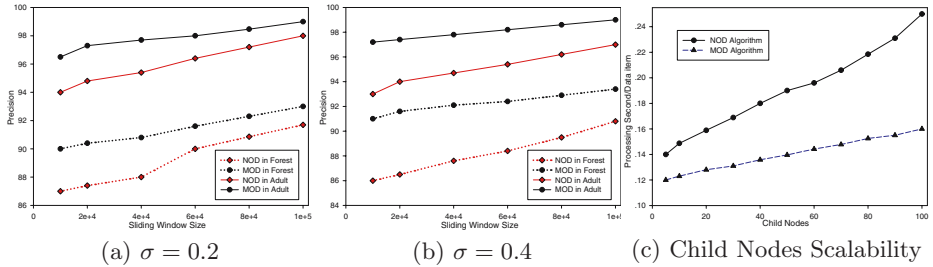


Fig. 4. Precision when varying the sliding window size, and nodes scalability

scalability in high dimensions and large numbers of child nodes. (3) Checking the efficiency of two kernel widths (\tilde{h} and \hat{h} in Equation 4).

All experiments were conducted on a 3.2 GHz PentiumIV PC with 1GB memory, running Microsoft Windows XP. To evaluate the efficiency and scalability of our two algorithms, both real and synthetic data sets are used. Real data sets include adult, KDD cup 99, forest and stock price series. The last data set consists of the price for a single stock taken at frequent intervals over a six year period. Total count is 330K values. The other three data sets come from the UCI machine learning repository. All algorithms are implemented by Microsoft Visual C++ 6.0. The **Naïve Outlier Detection Algorithm** (NOD Algorithm) use the \tilde{h} kernel width and C++ STL (NOT the CF-tree), which is to proof the MOD algorithm scalability.

Selecting the Kernel Width: Kernel width is a virtual parameter in kernel density estimation. An proper width can save computing time and improve the accuracy very large. In this experiment, we select two very large data sets (adult and forest cover). We consider the precision change with the sliding window size. The sliding window varied from 10000 to 100000. Unless particularly mentioned, the nearest neighbor $k = 8$, divergence threshold $\lambda = 0.1$ and fading factor $\omega = 0.2$. In Figure 4 (a) the $\sigma = 0.2$ in the real data selected and Figure 4 (b) is 0.4. The results show the MOD algorithm has more better precision than the NOD algorithm, especially in Figure 4 (b). The precision in MOD algorithm is less sensitive than NOD algorithm following with the change of sliding window. This owes to the kernel width. In MOD algorithm, we considered the influence of k -nearest neighbors which can prune effectively off the kernel width.

Efficiency of MOD Algorithm: Our first set of experiments focused on the efficiency of three algorithms. We use two measures, namely *precision* and *recall*, defined as follows. *Precision* represents the fraction of the values reported by our algorithms as outliers that are true outliers. *Recall* represents the fraction of the true outliers that our algorithms identified correctly. We set the sliding window size $N = 10000$. We selected five real value columns in the real data sets (The stock price series data set only has one column and we constructed five columns by simply duplicating the column). The experiment results in Fig. 5 (a) and (b) indicate that our NOD algorithm is better than the corresponding distance-based

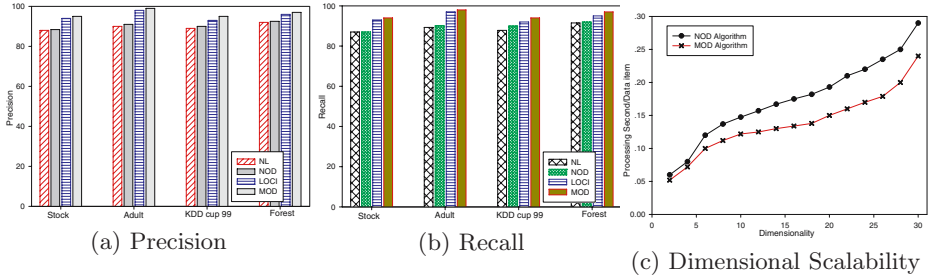


Fig. 5. Precision and Recall in four real datasets, and dimensional scalability

NL algorithm, the MOD algorithm also exceeds the LOCI, although only 0.2 ~ 3.1% improvement. It is not surprising that our NOD and MOD algorithms are essentially improved from the distance-based and density-based algorithms.

Algorithm Scalability: The scalability is an important criteria to judge whether an algorithm is suitable for distributed environment or not. Our synthetic data were random sample columns from adult data sets. we used ten PCs to simulate the 100 child nodes (each node simulate ten child nodes at most) and one PC as the leader node in Figure 1. each In Figure 4(c) and Figure 5(c), we change the dimension(d) and child nodes(N) to check the precision of NOD and MOD algorithms. The results show that: NOD algorithm is exponential proportional to the dimension, but the MOD is linear to it. This phenomenon is more obvious in Figure 5(c). The processing time per data item of MOD algorithm in Figure 5(c) only varied range of 0.12 ~ 0.16 second, but 0.14 ~ 0.25 second in NOD algorithm. The ratio of fluctuation difference ($\frac{0.25-0.14}{0.16-0.12} \approx 3$) approximate to three. So, the MOD algorithm has more scalability than NOD algorithm, mainly because of the micro-cluster compressed data structure.

6 Conclusions and Future Work

In this paper, we study the problem of continuous adaptive outlier detection on distributed data streams. We propose two novel outlier measures and an algorithm which can deal with the distributed and evolving natures of stream data. The mathematical analysis and extensive experiments show that the proposed methods are both efficient and effective comparing with existing outlier detection algorithms. In the future, we will study cluster over distributed data streams.

Acknowledgements

This work is supported by the National High Technology Development 863 Program of China under Grant No.2004AA112020, and the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321804.

References

1. Scott, D.W.: *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley-Interscience, Chichester (2001)
2. Barnett, V., Lewis, T.: *Outliers in statistical data*, 3rd edn. Wiley, Chichester (2001)
3. Eskin, E.: Anomaly detection over noisy data using learned probability distributions. In: *ICML (2000)*
4. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: *VLDB (1994)*
5. Ester, M., Kriegel, H.P., Xu, X.: A database interface for clustering in large spatial databases. In: *KDD (1995)*
6. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: *SIGMOD (1996)*
7. Sheikholeslami, G., Chatterjee, S., Zhang, A.: Wavecluster: A multi-resolution clustering approach for very large spatial databases. In: *VLDB (1998)*
8. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: *SIGMOD (1998)*
9. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: *VLDB (1998)*
10. Knorr, E.M., Ng, R.T.: Finding intensional knowledge of distance-based outliers. In: *VLDB (1999)*
11. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: *SIGMOD (2000)*
12. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: *KDD (2003)*
13. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high-dimensional datasets. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) *PAKDD 2006*. LNCS (LNAI), vol. 3918, Springer, Heidelberg (2006)
14. Wu, M., Jermaine, C.: Outlier detection by sampling with accuracy guarantees. In: *KDD (2006)*
15. Tao, Y., Xiao, X., Zhou, S.: Mining distance-based outliers from large databases in any metric space. In: *KDD (2006)*
16. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: *SIGMOD (2000)*
17. Papadimitriou, S., Kitagawa, H., Gibbons, P.B., Faloutsos, C.: Loci: Fast outlier detection using the local correlation integral. In: *ICDE (2003)*
18. Lazarevic, A., Kumar, V.: Feature bagging for outlier detection. In: *KDD (2005)*
19. Jin, W., Tung, A.K.H., Han, J., Wang, W.: Ranking outliers using symmetric neighborhood relationship. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) *PAKDD 2006*. LNCS (LNAI), vol. 3918, Springer, Heidelberg (2006)
20. Babcock, B., Olston, C.: Distributed top-k monitoring. In: *SIGMOD (2003)*
21. Manjhi, A., Shkapyenyuk, V., Dhamdhere, K., Olston, C.: Finding (recently) frequent items in distributed data streams. In: *ICDE (2005)*
22. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Online outlier detection in sensor data using non-parametric models. In: *VLDB (2006)*
23. Cormode, G., Muthukrishnan, S., Zhuang, W.: Conquering the divide: Continuous clustering of distributed data streams. In: *ICDE (2007)*
24. Hawkins, D.: *Identification of Outliers*, 1st edn. Springer, Heidelberg (1980)
25. Gijbels, I., Pope, A., Wand, M.: *Automatic forecasting via exponential smoothing: Asymptotic properties (1997)*