DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams

Gyoung S. Na, Donghyun Kim, Hwanjo Yu*
POSTECH (Pohang University of Science and Technology), South Korea
{ngs00, kdh5377, hwanjoyu}@postech.ac.kr

ABSTRACT

With precipitously growing demand to detect outliers in data streams, many studies have been conducted aiming to develop extensions of well-known outlier detection algorithm called Local Outlier Factor (LOF), for data streams. However, existing LOF-based algorithms for data streams still suffer from two inherent limitations: 1) Large amount of memory space is required. 2) A long sequence of outliers is not detected. In this paper, we propose a new outlier detection algorithm for data streams, called DILOF that effectively overcomes the limitations. To this end, we first develop a novel density-based sampling algorithm to summarize past data and then propose a new strategy for detecting a sequence of outliers. It is worth noting that our proposing algorithms do not require any prior knowledge or assumptions on data distribution. Moreover, we accelerate the execution time of DILOF about 15 times by developing a powerful distance approximation technique. Our comprehensive experiments on real-world datasets demonstrate that DILOF significantly outperforms the state-of-the-art competitors in terms of accuracy and execution time. The source code for the proposed algorithm is available at our website: http://di.postech.ac.kr/DILOF.

KEYWORDS

Outlier detection; Data streams; Density-based sampling

ACM Reference Format:
Gyoung S. Na, Donghyun Kim, Hwanjo Yu*. 2018. DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220022

1 INTRODUCTION

Local outlier factor (LOF) [3] is a fundamental and popular density-based outlier detection algorithm for static data whose size is fixed. LOF has been widely used as it effectively detects outliers on data with heterogeneous densities [11, 15, 16]. However, LOF requires a large amount of memory and is not applicable to data stream whose size is incessantly increasing. Specifically, LOF has $O(n^2)$ space complexity to detect outliers because it computes and stores the distances of every pair of data points. Such $O(n^2)$ space complexity is a critical issue when handling data streams. To reduce the space complexity, MiLOF was proposed, which stores only a small number of data points in memory by clustering past data [11]. However, this approach inevitably leads to a significant degradation in outlier detection accuracy, since it clusters past data using the k-mean algorithm which does not preserve the density of data [5].

Another problem of existing LOF-based algorithms is that *they do not detect sequences of outliers in data streams.* Detecting a sequence of outliers is crucial in many fields because a sequence of outliers arises when catastrophic problems occur, such as sensor malfunctions, system failures, and network attacks. As a representative example, the KDD Cup 99 http dataset contains many sequences of outliers. However, detecting a sequence of outliers is challenging because sequences of outliers show a seemingly similar pattern to the new class emergence [7]. Furthermore, detecting a sequence of outliers is especially hard in LOF due to the fundamental conflict between the definition of outlier in LOF and the characteristics of sequences of outliers. To be specific, LOF determines data points in low-density regions to be outliers, but the density of the region where outliers exist inevitably increases when outliers arise in sequence.

To overcome the two limitations – the memory problem and the problem of detecting a sequence of outliers – of LOF and its extensions, we propose a new outlier detection algorithm for data streams called DILOF (Density summarizing Incremental LOF). DILOF consists of two phases – (1) detection phase and (2) summarization phase. The detection phase computes and updates the neighbor information of past data in memory and determines if a current data point p_t is an outlier. The summarization phase reduces the memory consumption by sampling from past data while preserving the density of past data. For convenience of readers, we explain the summarization phase first in the order that we described the two limitations of LOF and its extensions.

For the summarization phase of DILOF, we develop a novel density-based sampling algorithm. Our density-based sampling algorithm samples past data while preserving their density without any prior knowledge or assumptions on data distribution. Due to this sampling, DILOF processes only a small number of data points and the time and space complexities are substantially reduced. In addition, we develop a novel distance approximation technique for the density-based sampling algorithm of DILOF, which accelerates the execution time of DILOF by 12∼15x without compromising the outlier detection accuracy.

The detection phase of DILOF combines an incremental methodology with a new strategy called skipping scheme. For data stream that contains n data points, LOF requires $O(n^2)$ time complexity to update neighbor information whenever a new data point is inserted. To avoid such excessive time complexity, we adopt the incremental

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220022

insertion of incremental LOF (iLOF) [9] that selectively updates neighbor information of past data points. We then devise the skipping scheme that preserves the low-density of the regions where outliers exist by considering the context of outliers even if a sequence of outliers arises.

We demonstrate effectiveness and efficiency of DILOF by comprehensive experimental evaluations on real-world datasets. The contributions of this paper are as follows:

- To the best of our knowledge this paper is the first work that tackles the two fundamental limitations of LOF in data streams.
- The proposed algorithm can effectively and efficiently detect outliers using only a small amount of memory without any prior knowledge or assumptions on data distribution.
- In Section 3.1.3, we develop distance approximation technique that accelerates the execution time of the proposed algorithm by 15x.
- In Section 3.2.2, we introduce a new scheme that detects sequences of outliers and demonstrate its effectiveness with comprehensive experiments.
- We experimentally demonstrate on real-world datasets that the proposed algorithm significantly outperforms the state-of-the-art competitors in terms of both outlier detection accuracy and execution time.

2 RELATED WORK AND PRELIMINARIES

2.1 LOF and Its Extensions

LOF is well-known outlier detection algorithm for static data. The goal of LOF is to compute the *LOF* score of each data point, which indicates how much the data point is close to outlier. The *LOF* score of a data point p is defined as follows.

Definition 2.1. $d_k(p)$, which is also written as $k\text{-}dist(p)$, is the Euclidean distance between p and its k^{th} nearest neighbor.

Definition 2.2. For given two data points p and x, $reach\text{-}dist_k(p,x)$ is defined by

$$reach\text{-}dist_k(p,x) = \max\{d_k(x), d(p,x)\} \tag{1}$$

where $d(p,x)$ is the Euclidean distance between two data points.

Definition 2.3. Local reachability density, lrd_k is given by

$$lrd_k(p) = \left(\frac{1}{k}\sum_{x\in N_k(p)} reach\text{-}dist_k(p,x)\right)^{-1} \tag{2}$$

where $N_k(p)$ is a set of the k-nearest neighbors of p.

Definition 2.4. Local outlier factor, LOF_k is given by

$$LOF_k(p) = \frac{1}{k}\sum_{x\in N_k(p)} \frac{lrd_k(x)}{lrd_k(p)}. \tag{3}$$

If $LOF_k(p)$ is higher than a given threshold θ, the data point p is classified as outlier.

Unfortunately, LOF is inapplicable for processing data streams since it must update the *LOF* scores of all past data points whenever a new data point is inserted, which takes $O(n^2)$ time complexity. To handle such $O(n^2)$ time complexity, iLOF [9] was proposed. It reduces the time complexity of LOF to $O(n)$ by incrementally updating the scores. However, iLOF still requires $O(n^2)$ space complexity like LOF since it stores the complete neighbor information of all past data points to detect outliers.

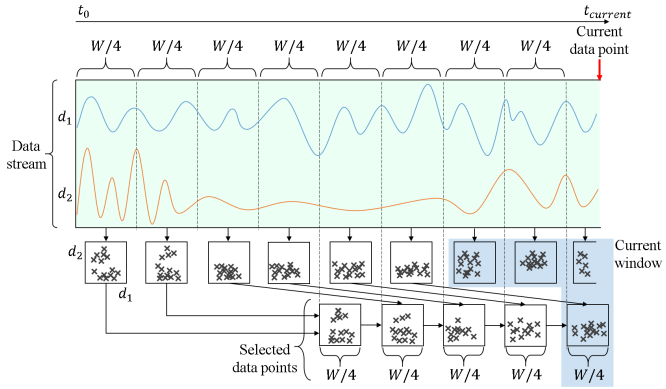


Figure 1: The behaviour of NDS from the time t_0 to $t_{current}$ for a 2-dimensional data stream.

To reduce the memory consumption of iLOF, MiLOF was proposed. However, MiLOF significantly degrades outlier detection accuracy since it clusters past data using flexible c-means [11] based on k-means clustering, which does not preserve the density of data. Since the *LOF* score is computed based on the local density of data, failure to preserve the density of data is a serious problem in LOF-based algorithms. Readers are referred to [11] for more detailed description about MiLOF.

2.2 Problem Definition

For an infinite data stream $P = \{p_1, p_2, ..., p_t, ...\}$, the goal of DILOF is to compute the *LOF* scores for all data points and detect outliers even under the following constraints.

- The number of data points that can be hold in memory is extremely smaller than the number of all data points.
- The outlier detection for a data point p_t must be completed at the current time t. That is, outlier detection algorithms cannot look at future data points in order to detect current outliers. This contatint makes the buffer-based approaches inapplicable.
- There are no prior knowledge or assumptions on data distribution.

Under the these constraints, outlier detection becomes more challenging. To the best of our knowledge MiLOF is the only algorithm that can detect outliers in data streams under these constraints.

3 PROPOSED ALGORITHM

The proposed algorithm, DILOF, consists of the two phases – (1) detection phase and (2) summarization phase. We refer to the detection phase as *last outlier-aware detection* (LOD) and the summarization phase as *nonparametric density summarization* (NDS). Algorithm 1 formally describes DILOF with a given maximum window size W.

(1) In LOD, for current data point p_t, it computes $LOF_k(p_t)$ on the current window of data and determines if p_t is an outlier

(line 5). Then p_t is added to the window (line 7) no matter whether it is an outlier or not.

(2) In NDS, if the number of past data points in window reaches W, NDS selects $W/4$ data points from the oldest $W/2$ data points such that their density difference is minimized (Figure 1). Then the oldest $W/2$ data points are removed in memory, and the oldest $W/4$ of window becomes to contain the selected data points by NDS.

(3) The above two phases are repeated. Note that LOD executes on every data insertion while NDS is executed the number of data points in memory reaches W.

Note that the codes in line 3 and line 6 of Algorithm 1 are used for the skipping scheme, which is described in Section 3.2.2.

Algorithm 1: DILOF

Input : infinite data stream $P = \{p_1, p_2, ..., p_t, ...\}$,
maximum window size W,
threshold of LOF scores θ,
step size η,
regularization constant λ,
maximum number of iteration I

1 $X \leftarrow \{\}$ // data points in memory
2 $O \leftarrow \{\}$ // set of detected outliers
3 $ENABLE\_SKIP \leftarrow$ false
4 **foreach** $p_t \in P$ **do**
5 $\quad LOF_k(p_t) \leftarrow \text{LOD}(p_t, o, O, \theta)$
6 $\quad$ **if** $LOF_k(p_t) > 0$ **then**
7 $\quad\quad X \leftarrow X \cup \{p_t\}$
8 $\quad\quad$ **if** $|X| = W$ **then**
9 $\quad\quad\quad Z \leftarrow \text{NDS}(X, \eta, \lambda, I)$
10 $\quad\quad\quad$ Remove oldest $W/2$ data points in X
11 $\quad\quad\quad X \leftarrow X \cup Z$
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 **end**

Since NDS summarizes the oldest $W/2$ data points to $W/4$ data points whenever the number of data points reaches W, the number of data points to be processed by DILOF is always bounded by W. The size W is typically determined by the memory limitation. The analysis about time complexity according to W is described in Section 3.3. As in the order in which the problems of LOF are mentioned in the previous section, we explain NDS first in Section 3.1 and then explain LOD in Section 3.2.

3.1 NDS: Nonparametric Density Summarization

This section describes NDS in the order of 1) problem formulation, 2) optimization, and 3) further execution time acceleration. The NDS algorithm is formally described in Algorithm 2.

3.1.1 ***Problem formulation.*** NDS selects a small number of representative data points such that the density difference between the selected and the original data is minimized, by solving a combinatorial optimization problem. Specifically, 1) NDS is defined as a

Algorithm 2: NDS

Input : set of data points $X = \{x_1, x_2, ..., x_W\}$,
step size η,
regularization constant λ,
the maximum number of iteration I

Output : set of the selected data points Z

1 $Y = \{y_1, y_2, ..., y_{(W/2)}\}$ // Decision variables
2 **foreach** $y \in Y$ **do**
3 $\quad y = 0.5$
4 **end**
5 **for** $t = 1 : I$ **do**
6 $\quad \eta = \eta \times 0.95$
7 $\quad$ **for** $n = 1 : W/2$ **do**
8 $$\quad\quad y_n^{(t+1)} = y_n^{(t)} - \eta \left\{ \sum_{x_i \in C_{k,n}} y_i^{(t)} + \frac{\rho_k(x_n)}{v_k(x_n)} - e^{LOF_k(x_n)} + \psi'_{0,1}(y_n^{(t)}) + \lambda \left(\sum_{i=1}^{W/2} y_i^{(t)} - \frac{W}{4} \right) \right\}$$
9 $\quad$ **end**
10 **end**
11 Project Y into binary domain
12 **for** $n = 1 : W/2$ **do**
13 $\quad$ **if** $y_n = 1$ **then**
14 $\quad\quad Z \leftarrow Z \cup \{x_n\}$
15 $\quad$ **end**
16 **end**
17 **return** Z

combinatorial optimization problem by extending a nonparametric divergence estimator [8], 2) decision variables are introduced in order to transform the combinatorial optimization problem to a solvable form, and 3) a new component called shape term is designed to tighten NDS.

In order to formulate NDS as a combinatorial optimization problem, we need a metric that computes the density difference between two data. However, since the actual density function of data is unknown, we cannot exploit divergence metrics such as KL-divergence that require prior knowledge about data. To compute the density difference without any prior knowledge or assumptions on data distribution, we extend the previous study about the nonparametric Rényi divergence estimator [8]. For given two datasets $Z = \{z_1, z_2, ..., z_N\}$ and $X = \{x_1, x_2, ..., x_M\}$ where Z is a proper subset of X, the density difference $\widehat{D}_\alpha(Z|X)$ can be computed by the divergence estimator as the following Equation (4): $\rho_k(z_n)$ is the Euclidean distances between data point z_n and its k^{th} nearest neighbor in Z, and $v_k(z_n)$ is the distance between z_n and its k^{th} nearest neighbor in X. α is a hyper-parameter.

$$\widehat{D}_\alpha(Z|X) \doteq \frac{1}{N} \sum_{n=1}^{N} \left(\frac{(N-1)\rho_k^d(z_n)}{M v_k^d(z_n)} \right)^{1-\alpha} B_{k,\alpha} \tag{4}$$

where $B_{k,\alpha} \doteq \frac{\Gamma(k)^2}{\Gamma(k-\alpha+1)\Gamma(k+\alpha-1)}$ with $k > |\alpha - 1|$ and Γ is the gamma distribution. For a given maximum window size $W > 4$, the purpose of NDS is to select the proper subset Z of X minimizing $\widehat{D}_\alpha(Z|X)$ where X is the set of the oldest $W/2$ data points and Z is the set of the selected $W/4$ data points. Therefore, NDS can be

defined as a combinatorial optimization problem of Equation (5) when the hyper-parameter α is fixed to 0.

$$\begin{aligned} Z &= \arg\min_{Z \subset X} \frac{1}{(W/4)} \sum_{n=1}^{W/4} \frac{((W/4)-1)\rho_k^d(z_n)}{(W/2)\nu_k^d(z_n)} B_{k,0} \\ &= \arg\min_{Z \subset X} \sum_{n=1}^{W/4} \frac{\rho_k(z_n)}{\nu_k(z_n)} \end{aligned} \tag{5}$$

Now we introduce decision variables y_n with a value of 1 when the data point x_n is selected, in order to transform the combinatorial optimization problem in Equation (5) to a solvable form. Finally, NDS is defined as the following binary constrained optimization problem.

$$\begin{aligned} &\min_{\mathbf{y}} \sum_{n=1}^{W/2} y_n \frac{\rho_k(x_n)}{\nu_k(x_n)} \\ &\text{s. t.} \sum_{n=1}^{W/2} y_n = \frac{W}{4} \\ &y_n \in \{0,1\}. \end{aligned} \tag{6}$$

Many literature about the density-based sampling or clustering show that preserving the shape of data distribution helps to improve sampling or clustering performance [17]. However, the objective function in Equation (6) cannot sufficiently look at the shape of data distribution because it can be minimized when we select only the data points in the high-density region. Thus, it is necessary to modify this objective function to take into account the shape of data distribution as well as the density. To this end, we design a new component of the objective function called shape term based on the *LOF* scores as follows.

$$\begin{aligned} &\min_{\mathbf{y}} \sum_{n=1}^{W/2} y_n \frac{\rho_k(x_n)}{\nu_k(x_n)} - \underbrace{\sum_{n=1}^{W/2} y_n e^{LOF_k(x_n)}}_{\text{shape term}} \\ &\text{s. t.} \sum_{n=1}^{W/2} y_n = \frac{W}{4} \\ &y_n \in \{0,1\}. \end{aligned} \tag{7}$$

Since the data points located on the boundary of data distribution generally have high *LOF* score [3], the shape term acts to preserve the shape of data distribution by selecting the data points in the boundary. Surprisingly, the shape term can be computed using only the *LOF* scores already computed in LOD, i.e. computing the shape term does not require any additional cost.

3.1.2 ***Optimization***. Since the decision variables y_n should have binary value, we cannot solve the problem in Equation (7) using an analytical optimization techniques. Furthermore, we cannot exploit the dynamic programming because $\rho_k(x_n)$ can be computed only after the values of all decision variables are determined. For these reasons, we transform the binary constrained optimization problem in Equation (7) to an unconstrained optimization problem as shown in Equation (8) by applying relaxation to the decision variables. Then, we find the values of the decision variables y_n using the gradient descent method.

$$\min_{\mathbf{y}} \sum_{n=1}^{W/2} y_n \frac{\rho_k(x_n)}{\nu_k(x_n)} - \sum_{n=1}^{W/2} y_n e^{LOF_k(x_n)} + \sum_{n=1}^{W/2} \psi_{0,1}(y_n) + \frac{\lambda}{2}\left(\sum_{n=1}^{W/2} y_n - \frac{W}{4}\right)^2 \tag{8}$$

where

$$\psi_{0,1} = \begin{cases} (y_n - 1)^2, & \text{if } y_n > 1 \\ {y_n}^2, & \text{if } y_n < 0 \\ 0, & \text{otherwise.} \end{cases}$$

To derive the derivative of the objective function in Equation (8) for computing gradients with respect to the decision variables, we can express $\rho_k(x_n)$ as a computable form as the following equation.

$$\rho_k(x_n) = \{y_i || x_n - x_i ||_2 | i = 1, 2, ..., \frac{W}{2}\}_{[(W/4)-k+1]} \tag{9}$$

where $A_{[m]}$ is the m^{th} largest element in a set A. As a result, the update rule for y_n is given by

$$\begin{aligned} y_n^{(t+1)} = y_n^{(t)} - \eta \Bigg\{ &\sum_{x \in C_{k,n}} \frac{\rho_k(x)}{\nu_k(x)} + \frac{\rho_k(x_n)}{\nu_k(x_n)} - e^{LOF_k(x_n)} + \psi'_{0,1}(y_n^{(t)}) \\ &+ \lambda \left(\sum_{i=1}^{W/2} y_i^{(t)} - \frac{W}{4} \right) \Bigg\} \end{aligned} \tag{10}$$

where η is step size and $C_{k,n}$ a set of data points that have x_n as their k^{th} nearest neighbor in Z. After the update of the decision variables is completed, they are projected into the binary domain to obtain a feasible solution. In this projection step, the largest $W/4$ decision variables are set to 1, and the rest are set to 0. Then, the data points corresponding to the decision variables that have the value of 0 are removed from memory.

Unfortunately, since $C_{k,n}$ in Equation (10) are dependent on the decision variables, they should be recomputed whenever a decision variable is updated. Therefore, NDS requires $O((W/2)^3)$ time complexity because it should compute k-nearest neighbors of all $W/2$ data points on every update of decision variables. To reduce the time complexity, we develop a heuristic approximation technique formally described in Algorithm 3: First, we approximate the k^{th} nearest neighbor of x_i with two ways according to s_i that is the sum of the *LOF* scores for the k-nearest neighbors of x_i. Outliers will be removed in NDS because they distort the density as well as the shape of data distribution. Thus, if a data point x has outliers in its k-nearest neighbors, x has a high probability that the k^{th} nearest neighbor of x is changed in Z. For this reason, only the k^{th} nearest neighbor of x_i that have $s_i > \bar{s}$ is changed (line 8). Second, for a data point x_i with $s_i > \bar{s}$, we estimate a proper range of the Euclidean distance between x_i and its k^{th} nearest neighbor in Z. Our decision is that if a data point x_n is in the estimated range of x_i, the k^{th} nearest neighbor of x_i in Z is x_n. To this end, we compute the lower and upper bounds of the estimated range using the exact lower bound, $\nu_k(x_i)$, and a heuristically computed upper bound with the sigmoid function σ (line 10).

Algorithm 3: Approximation of $C_{k,n}$

Input : data points in memory X
Output: $C_{k,1}, C_{k,2}, ..., C_{k,|X|}$

```
1  for n = 1 : |X| do
2      foreach q ∈ N_k(x_n) do
3          s_n = s_n + e^{σ(LOF_k(q))}
4      end
5  end
6  s̄ = (1/|X|) Σ_{i=1}^{|X|} s_n
7  for i = 1 : |X| do
8      if s_i > s̄ then
9          for n = 1 : |X| do
10             if v_k(x_i) < d(x_i, x_n) < 2e^{σ(LOF_k(x_i))} v_k(x_i) then
11                 C_{k,n} ← C_{k,n} ∪ {x_i}
12                 break
13             end
14         end
15     end
16 end
```

3.1.3 ***Approximation of*** $\rho_k(x_n)$. In Equation (10), $\rho_k(x_n)$ are also dependent on the decision variables like $C_{k,n}$. Thus, $\rho_k(x_n)$ should also be recomputed on every update of decision variables. If we compute value of a $\rho_k(x_n)$ exactly, we should sort the decision variables for projecting into binary domain, then compute the k^{th} nearest neighbor of x_n in Z. Therefore, exactly computing the value of all $\rho_k(x_n)$ requires $O((W/2)^2 \log(W/2))$ time complexity.

However, we can use the residually approximated $\rho_k(x_n)$ in Algorithm 3 since the k^{th} nearest neighbor in Z of all data points is already computed in the procedure for approximating $C_{k,n}$. But, such approximated $\rho_k(x_n)$ could hurt sampling performance since they are heuristically computed. To more accurately estimate $\rho_k(x_n)$, we develop a distance approximation technique. Our distance approximation technique estimates a mathematically proven range of $\rho_k(x_n)$ and computes the value of $\rho_k(x_n)$ using the estimated range. The distance approximation technique for $\rho_k(x_n)$ is described through Equations (11)~(13).

Lemma 3.1. *The lower and upper bounds of* $\rho_k(x_n)$ *is given by*

$$v_k(x_n) \le \rho_k(x_n) \le \max_{x \in X \setminus \{x_n\}} ||x_n - x||_2. \tag{11}$$

We can estimate $\rho_k(x_n)$ as a parametric form using Lemma 3.1, and it is given by

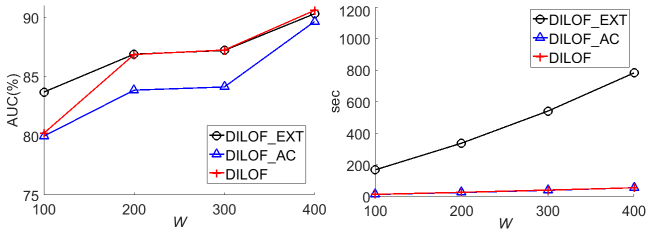
$$\rho_k(x_n) = v_k(x_n) + \beta \left(\max_{x \in X \setminus \{x_n\}} ||x_n - x||_2 - v_k(x_n) \right) \tag{12}$$

where β is a hyper-parameter with $0 \le \beta \le 1$.

As we mentioned in Section 3.1.2, outliers will be removed in NDS. Thus, $\rho_k(x_n)$ will increase if there are outliers in the k-nearest neighbors of the data point x_n. Based on this property, NDS automatically sets β using the LOF scores as follows.

$$\beta = \frac{\sum_{x \in N_k(x_n)} e^{\sigma(LOF_k(x))}}{\sum_{i=1}^{W/2} e^{\sigma(LOF_k(x_i))}}. \tag{13}$$

To demonstrate the effectiveness and efficiency of our distance approximation technique for $\rho_k(x)$, we compare outlier detection performance of three versions of DILOF: DILOF_EXT, DILOF_AC, and the original DILOF. The first version is DILOF_EXT that accurately computes $\rho_k(x_n)$ with an excessive $O((W/2)^2 \log(W/2))$ time complexity. The second version is DILOF_AC using $\rho_k(x_n)$ that are generated as by-product of Algorithm 3. Lastly, the original DILOF, is that approximates $\rho_k(x_n)$ using the proposed distance approximation technique in this section. For the comparison, the KDD Cup 99 smtp dataset is used, the properties of this dataset are given by Table 1 in Section 4.1.



(a) Outlier detection accuracy **(b) Execution time**

Figure 2: Outlier detection performance of DILOF_EXT, DILOF_AC, and DILOF for the KDD Cup 99 smtp dataset.

Figure 2 shows Area Under the ROC Curve (AUC) and execution time of the three version of DILOF. Understandably, DILOF_EXT shows the best AUC because it accurately computes $\rho_k(x_n)$, but it spends a lot of time. The AUC of DILOF_AC reaches the AUC of DILOF_EXT as W increased, and it indirectly shows that the heuristic approximation for $C_{k,n}$ in Algorithm 3 is somewhat reasonable. Surprisingly, DILOF shows the equivalent AUC of DILOF_EXT when $W \ge 200$. In the case of the execution time, both DILOF_AC and DILOF are 15x faster than DILOF_EXT. Therefore, we can confirm that our distance approximation technique accelerate the proposed algorithm by 15x faster while fully maintaining the outlier detection accuracy.

3.2 LOD: Last Outlier-aware Detection

The goal of LOD is to compute the LOF score for a new data point and determine if it is an outlier. To do this efficiently, LOD combines the incremental insertion based on iLOF with our new scheme called skipping scheme. Algorithm 4 formally describes LOD for a new data point p_t. The code in line 6~26 are for the incremental insertion, and the code in line 1~5 and 29~32 are for the skipping scheme.

3.2.1 ***Incremental update***. LOF should updates the LOF scores of all data points whenever a new data point is inserted. In contrast, iLOF selectively updates the scores of the data only affected by the insertion of a new data. Just as iLOF, LOD searches for the data points whose neighbor information will be changed as the new data point p_t is inserted (line 10~19). After that, LOD selectively updates lrd_k and LOF_k of them (line 20~25). For more details about the incremental insertion of iLOF, readers are referred to [9].

3.2.2 ***Skipping scheme***. LOF-based algorithms cannot detect a long sequence of outliers because they determine only the data

Algorithm 4: LOD

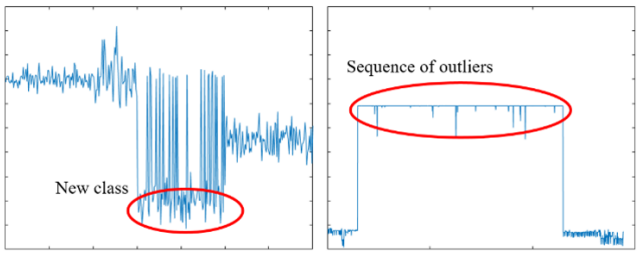
Input : data point p_t at time t,
least recently detected outlier o,
set of detected outliers O,
given threshold θ for LOF score
Output: LOF score of data point p_t

```
1  if Skipping_enabled then
2      if Skipping_scheme(p_t, o, O) then
3          return −1
4      end
5  end
6  Compute N_k(p_t) and d_k(p_t)
7  foreach x ∈ N_k(p_t) do
8      Compute reach-dist_k(p_t, x) using Equation (1)
9  end
10 X_update ← RN_k(p_t) // set of reverse k-NNs of p_t
11 foreach x ∈ X_update do
12     Update d_k(x)
13     foreach q ∈ N_k(x) do
14         Update reach-dist_k(q, x)
15         if x ∈ N_k(q) then
16             X_update ← X_update ∩ {q}
17         end
18     end
19 end
20 foreach x ∈ X_update do
21     Update lrd_k(x)
22     foreach q ∈ RN_k(x) do
23         Update LOF_k(q)
24     end
25 end
26 Compute lrd_k(p_t) and LOF_k(p_t)
27 if LOF_k(p_t) > θ then
28     O ← O ∪ {p_t}
29     if Skipping_enabled then
30         o ← p_t
31         ENABLE_SKIP ← true
32     end
33 end
34 return LOF_k(p_t)
```

points in the low-density region to be outliers. To solve this problem, we can simply remove the incoming outliers from the window to keep the low-density of the region where outliers are located. But unconditionally removing the data points that seem to be outliers could hurt the detection accuracy especially when a new class emerges [7] because the data points in the newly emerging class may look like outliers but they are actually inlier. Thus, we need an appropriate scheme that effectively distinguishes outliers from the data points in newly emerging classes.

The probability that data points consecutively occurs in the low-density region is very low in typical environments. For instance, in communication systems, clients occasionally send large data, such as image or video, to the server, and this situation is very natural. However, if most clients constantly send only large data, we have to doubt network attacks or system failures. The KDD Cup 99 http dataset clearly shows this phenomenon. Note that Figure 3-(b) represents data distribution of KDD Cup 99 http dataset when only one feature called src_byte is considered, and the portion corresponding to a sequence of outliers is actually labeled as outlier. To generalize this observation, we make Assumption 1.

Assumption 1. If data points that seem to be outliers appear alternately with inliers, they are the data points of a new class (Figure 3-(a)). However, if only data points that seem to be outlier appear consecutively, they are a sequence of outliers since this is a non-natural phenomenon called abnormal situation (Figure 3-(b)).



(a) New class emergence **(b) Sequence of outliers**

Figure 3: An example data that outliers appear alternately with inliers (a) and Data that only outliers appear consecutively (b).

Based on Assumption 1, we develop the skipping scheme as formally described in Algorithm 5. For all data points $x \in X$, the skipping scheme computes $d_1(x)$, i.e. the Euclidean distance between a data point x and its first nearest neighbor, and calculates $\overline{d_1(X)}$ which is the average of $d_1(x)$. If $d(p_t, o)$, i.e. the Euclidean distance between the current data point p_t and the last detected outlier o, is less than $\overline{d_1(X)}$, we set o to p_t and return true value. Otherwise, we set $ENABLE\_SKIP$ to false and return false value. As shown in line 2~4 of Algorithm 4, if the skipping scheme returns true value, LOD is immediately terminated.

3.3 Time Complexity

In this section, we analyze the time complexity of DILOF by dividing into two phases – LOD and NDS. Both the time and space complexities of LOD are O(W) due to NDS where W is the maximum window size. The detection phase of MiLOF have the same time and space complexity as LOD. In the case of NDS, the time complexity is O($(W/2)^2$) because NDS scans W data points twice (line 7~9 of Algorithm 2). On the other hand, the summarization phase of MiLOF requires the time complexity of k-means clustering, i.e. O($IDCW/2$), where I is the maximum number of iterations, C is the number of clusters, and D is the dimensionality of data. Note that the time complexity of MiLOF is dependent on the dimension of data whereas that of DILOF is not.

The time complexity of NDS looks higher than the summarization phase of MiLOF, but DILOF runs faster than MiLOF in practice,

Algorithm 5: Skipping scheme

Input : data point p_t at time t,
the least recently detected outlier o,
set of detected outliers O
Output: true or false value

```
1  SKIP_INSERTION ← false
2  if ENABLE_SKIP then
3      Compute d_1(x)
4      if d(p_t, o) < d_1(x) then
5          O ← O ∪ {p_t}
6          o ← p_t
7          SKIP_INSERTION ← true
8      end
9      else
10         ENABLE_SKIP ← false
11     end
12 end
13 return SKIP_INSERTION
```

as W is typically set to a small value. For instance, if we handle the KDD Cup 99 smtp dataset where D is 3 and C is 10 [11], with the maximum window size $W = 400$, then $(W/2)^2$ is 40,000. But $ICDW/2$ is 600,000 with $I = 100$, which is the setting of the original paper of MiLOF. However, since NDS is more frequently executed than the summarization phase of MiLOF, DILOF is not 15x faster than MiLOF. To compare the execution time of DILOF and MiLOF more precisely, we conduct several experiments in Section 4.2.2.

4 EXPERIMENT

In this section, we experimentally compare the performance of DILOF with the state-of-the-art competitors. In our comparison, we also include DILOF_NS, i.e. DILOF with the skipping scheme *disabled*. Note that there is a fundamental conflict between the definition of outlier and the characteristics of a sequence of outliers, and thus the skipping scheme may hurt the detection accuracy depending on the domains. Thus, we let user to enable/disable the skipping scheme in DILOF as user configuration (line 1 and 29 of Algorithm 4). The comprehensive experiments are conducted to answer the following research questions:

- **RQ 1** To what extent do DILOF and DILOF_NS improve the outlier detection accuracy compared to the state-of-the-art competitors?
- **RQ 2** Is our skipping scheme effective on detecting a sequence of outliers?
- **RQ 3** To what extent do DILOF and DILOF_NS improve the execution time compared to the state-of-the-art competitors?
- **RQ 4** How well does NDS preserve the characteristics of datasets compared to the other density-based sampling algorithms?
- **RQ 5** How does NDS improve outlier detection accuracy?

4.1 Experiment Settings

We compare DILOF and DILOF_NS with the state-of-the-art competitors, iLOF, MiLOF, and MiLOF_F, in terms of outlier detection accuracy and execution time. The accuracy is measured using AUC. Execution time is defined to be the time for checking all data points in the data stream. All experiments are conducted on a machine with Intel Core i7-6700, 64GB RAM, and Windows 10 64bit.

Datasets. The real-world datasets used in the experiments are described in Table 1. The UCI Vowel dataset is modified to data stream format like [1], and the class 1 is down-sampled to 50 outliers. In the UCI Pendigit dataset, 5 percent [0, 0.5] uniform noise is added, and the noized data points is set to be outliers. For the KDD Cup 99 datasets, we followed the settings of [14] and set the network attacks to be outliers. Recall that the KDD Cup 99 http dataset contains many sequences of outliers.

Table 1: Properties of the four real-world datasets

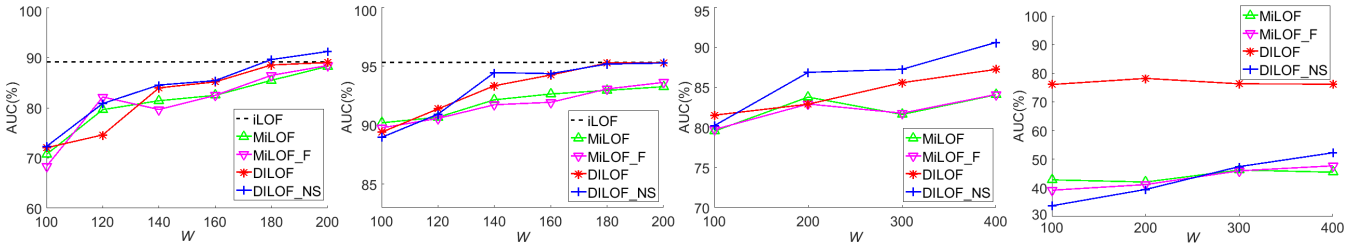
Dataset	N = # of data points	D = dimension	C = # of classes
UCI Vowel	1,456	12	11
UCI Pendigit	3,498	16	10
KDD Cup 99 smtp	95,156	3	unknown
KDD Cup 99 http	567,479	3	unknown

Parameter settings. The hyper-parameters of MiLOF and MiLOF_F are set to be similar to [11], and the value of C is given in Table 1. For datasets where the number of classes is unknown, C is set to 10 like [11]. The hyper-parameters of DILOF, η and λ in Equation (10) are fixed to 0.3 and 0.001 for all datasets, respectively. In all algorithms, the value of k is set to 19 for the UCI Vowel dataset and to 8 for UCI Pendigit dataset. To obtain a higher accuracy, we differently set k in MiLOF (and MiLOF_F) to 10 and 8 for KDD Cup 99 smtp and http datasets, respectively. In DILOF (and DILOF_NS), k is fixed to 8 for both KDD Cup 99 smtp and http datasets. Note that k is experimentally set in all experiments. In addition, since DILOF, DILOF_NS, MiLOF, and MiLOF_F execute the summarization phase whenever the number of data points reaches W, the outlier detection performances are measured while W is changing. For the UCI Vowel and Pendigit datasets that contain a relatively small number of data points, $W = \{100, 120, 140, 160, 180, 200\}$. For the KDD Cup 99 datasets, $W = \{100, 200, 300, 400\}$.

4.2 Experimental Results

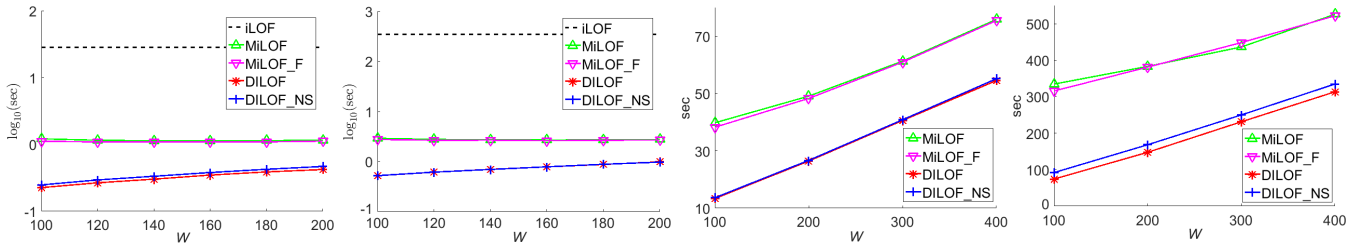
4.2.1 ***Outlier detection accuracy (RQ 1, RQ 2).*** We evaluate AUC of iLOF, MiLOF, MiLOF_F, DILOF, and DILOF_NS. However, AUC of iLOF for the KDD Cup 99 smtp and http datasets cannot be measured due to the unacceptable execution time. Figure 4 shows AUC of the five outlier detection algorithms. For all datasets, DILOF and DILOF_NS generally show higher AUC than MiLOF and MiLOF_F. Moreover, for the UCI Vowel and Pendigt datasets, the AUC of DILOF and DILOF_NS reaches the AUC of iLOF, which stores all data points, as W increased. In the case of the KDD Cup 99 smtp dataset, DILOF_NS surpasses MiLOF and MiLOF_F, and DILOF also shows higher AUC than MiLOF and MiLOF_F when $W \geq 300$.

In particular, for the KDD Cup 99 http dataset, MiLOF, MiLOF_F, and DILOF_NS show terribly low AUC that cannot be used as an outlier detector. In contrast, DILOF shows 76~78% AUC with the comparable AUC of DILOF_NS for other datasets. Therefore, we



(a) UCI Vowel (b) UCI Pendigit (c) KDD Cup 99 smtp (d) KDD Cup 99 http

Figure 4: Outlier detection accuracy comparisons for the four real world datasets while varying the maximum window size W.



(a) UCI Vowel (b) UCI Pendigit (c) KDD Cup 99 smtp (d) KDD Cup 99 http

Figure 5: Execution time comparisons for the four real world datasets while varying the maximum window size W.

can confirm that the proposed skipping scheme effectively detects a sequence of outliers without a significant performance degradation in detecting other types of outliers.

4.2.2 ***Execution time (RQ 3).*** Figure 5 shows the execution times of iLOF, MiLOF, MiLOF_F, DILOF, and DILOF_NS. Due to the unacceptable execution time of iLOF as mentioned in Section 4.2.1, iLOF cannot be evaluated for the KDD Cup smtp and http datasets. Note that the execution times are scaled by log in the experiments on UCI Vowel and Pendigit datasets for the convenience of readers. In the experiments, iLOF spends 28.23 and 346.38 seconds for the UCI Vowel and Pendigit datasets, respectively. In contrast, MiLOF and MiLOF_F spend 1.12 and 2.72 seconds. Moreover, DILOF and DILOF_NS spend just 0.42 and 0.98 seconds. For the UCI Vowel and Pendigit datasets, MiLOF and MiLOF_F show the 25.21x and the 127.35x speed up, while DILOF and DILOF_NS show higher improvement with 67.21x and 353.45x speed up. For the KDD Cup 99 smtp and http datasets, DILOF and DILOF_NS are also 1.5~3x faster than MiLOF and MiLOF_F.

4.2.3 ***Quantitative analysis of NDS (RQ 4).*** Since the purpose of NDS is to select data points while preserving the density of data, NDS can be regarded as a kind of density-based sampling algorithm. In this experiment, we compare the sampling performance of NDS and another density-based sampling called DENDIS [10]. with extensive experiments, [10] shows that DENDIS is better than other sampling algorithms based on various approaches [6, 13] in terms of sampling results and execution time. Note that since DENDIS cannot fix the number of samples that affects memory consumption, it is unable to be directly applied to DILOF instead of NDS. For the experiments, the well-known benchmark datasets in UCI Repository [1] and two synthetic datasets, as shown in Table 2, are used. Data distribution of the two synthetic datasets are shown in Figure 6-(a) and 7-(a). Due to the limitation of page in submission, the properties of datasets and parameter settings for these sampling experiments are described in our website [2].

To quantitatively compare the sampling performance, Earth mover's distance (EMD) is used. EMD is a widely used metric in many fields [4, 12] to compute the density difference between two data. Although computing EMD requries huge computational cost, the previous study [2] shows that EMD is more accurate than other divergence metrics such as total variation distance, KL-divergence, and JS-divergence. In these experiments, we compare EMD of random sampling, DENDIS, and NDS with 10 times execution for each dataset. Table 2 shows EMD and its standard deviation of the three sampling algorithms. As a reult, NDS shows the lowest EMD for all datasets. Moreover, the standard deviation of NDS is always zero because there are no random processes in NDS.

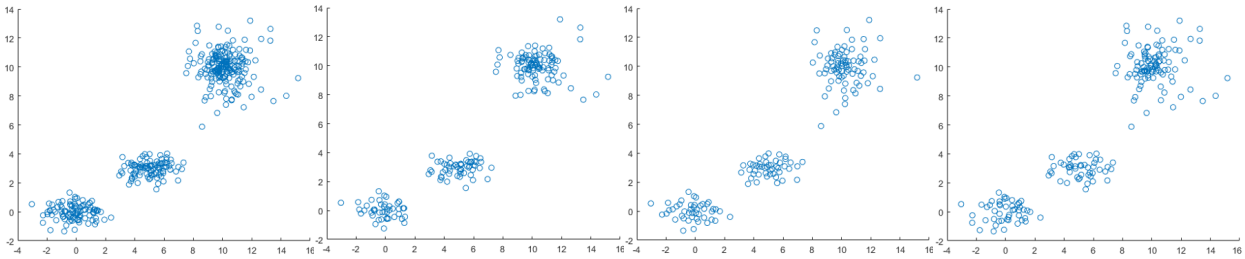
Table 2: EMD for the five datasets.

Dataset	Random sampling	DENDIS	NDS
Iris	367 (109)	270 (91)	**188 (0)**
Vowel	5,704 (891)	5,460 (587)	**4,434 (0)**
Abalone	5,326 (145)	5,028 (367)	**4,575 (0)**
Synthetic-normal	3,077 (276)	2,426 (250)	**1,672 (0)**
Synthetic-noisy	5,457 (579)	5,080 (305)	**4,450 (0)**

In addition, Table 3 shows the execution time of DENDIS and NDS in milliseconds. In the experiments about execution time, NDS 10~35x faster than DENDIS. These two experiments clearly show that NDS is more effective and efficient than a state-of-art algorithm in the field of density-based sampling.

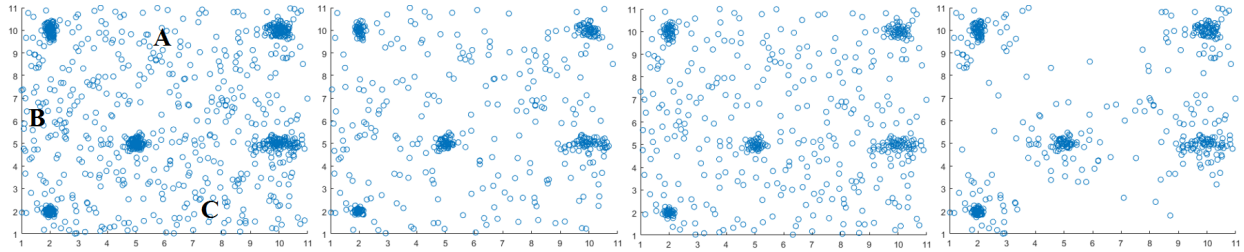
[1] https://archive.ics.uci.edu/ml/index.php

[2] http://di.postech.ac.kr/DILOF



(a) Original dataset (b) Random sampling (c) DENDIS (d) NDS

Figure 6: Sampling results for the Synthetic-normal dataset.



(a) Original dataset (b) Random sampling (c) DENDIS (d) NDS

Figure 7: Sampling results for the Synthetic-noisy dataset.

Table 3: Execution time (millisecond) of the sampling algorithms for the five datasets

Dataset	DENDIS	NDS
Iris	56	**5**
Vowel	7,032	**144**
Abalone	268,477	**3,250**
Synthetic-normal	467	**23**
Synthetic-noisy	6,250	**192**

4.2.4 ***Qualitative analysis of NDS (RQ 5).*** In this section, we visualize the sampling results for the 2-dimensional synthetic datasets (Synthetic-normal and Synthetic-noisy) to understand how does NDS improve the outlier detection accuracy. Figures 6 and 7 show the sampling results of the random sampling, DENDIS, and NDS. For the Synthetic-normal dataset, both DENDIS and NDS take samples successfully, but the random sampling distorts the shape of data distribution since it select samples just uniformly regardless of the characteristics of data distribution. In the case of the Synthetic-noisy which contains many noisy data points, DENDIS selects data points just uniformly in all regions, and shows the sampling result similar to the random sampling. In contrast, NDS intensively selects data points in the high-density regions, and removes data points in **A**, **B**, and **C** of Figure 7-(a). This result shows that NDS makes the boundaries between the high-density regions and the low-density regions clearer. Due to such effect of NDS, outlier detection algorithms are able to easily distinguish outliers even if data is very noisy.

5 CONCLUSIONS

In data streams, LOF has two fundamental limitations about memory consumption and detecting a long sequence of outliers. In this study, we propose a novel outlier detection algorithm called DILOF for data streams. DILOF effectively and efficiently overcomes two fundamental limitations of LOF using the density-based sampling approach and a new strategy called skipping scheme. In addition, the powerful distance approximation technique that accelerates the execution time of DILOF by 15 times is developed. Our comprehensive experimental evaluations demonstrate that DILOF significantly outperforms the state-of-the-art competitors in terms of both detection accuracy and execution time. In particular, DILOF shows incomparably higher detection accuracy than the state-of-the-art competitors for the dataset containing many sequences of outliers.

ACKNOWLEDGEMENTS

This research was supported by the National Research Foundation of Korea (NRF) (No. 2016R1E1A1A01942642), Next-Generation Information Computing Development Program (No. 2012M3C4A7033344), Institute for Information & communications Technology Promotion (IITP) (No. 2018-0-00584), and the Naver Corporation.

REFERENCES

[1] C. C. Aggarwal and S. Sathe. 2015. Theoretical Foundations and Algorithms for Outlier Ensembles. *ACM SIGKDD Exploarations Newsletter* 17, 1 (2015).
[2] M. Arjovsky, S. Chintala, and L. Bottou. 2017. Wasserstein GAN. *arXiv:1701.07875* (2017).
[3] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*. ACM, 93–104.
[4] K. Grauman and T. Darrell. 2004. Fast Counter Matching Using Approximate Earth Mover's Distance. In *CVPR*.
[5] A. K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31, 8 (2010).
[6] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. 2003. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering* 15, 5 (2003).
[7] X. Mu, F. Zhu, J. Du, E. Lim, and Z. Zhou. 2017. Streaming Classification with Emerging New Class by Class Matrix Sketching. In *Proceedings of the Thiry-First AAAI Conference on Artificial Intelligence*. 2373–2379.

[8] B. Poczos, L. Xiong, and J. Schneider. 2011. Nonparametric Divergence Estimation with Applications to Machine Learning on Distributions. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. 599–608.
[9] D. Pokrajac, A. Lazarevic, and L. J. Latecki. 2007. Incremental Local Outlier Detection for Data Streams. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining*. 504–515.
[10] F. Ros and S. Guillaume. 2016. DENDIS: A new density-based sampling for clustering algorithm. *Expert Systems With Applications* 56, 1 (2016).
[11] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang. 2016. Fast Memory Efficient Local Outlier Detection in Data Streams. *IEEE Transactions on Knowledge and Data Engineering* 28, 12 (2016).
[12] Y. Tang, L. H. U, Y. Cai, N. Mamoulis, and R. Cheng. 2013. Earth Mover's Distance based Similarity Search at Scale. *Proceedings of the VLDB Endowment* 7, 4 (2013).
[13] Y. Xiao, B. Liu, Z. Hao, and L. Cao. 2014. A K-Farthest-Neighbor-based approach for support vector data description. *Applied Intelligence* 41, 1 (2014).
[14] K. Yamanishi, J. Takeuchi, and G. Williams. 2000. On-line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms. In *KDD*. 320–324.
[15] Y. Yan, L. Cao, C. Kuhlman, and E. A. Rundensteiner. 2017. Distributed Local Outlier Detection in Big Data. In *KDD*. 1225–1234.
[16] Y. Yan, L. Cao, and E. A. Rundensteiner. 2017. Scalable Top-n Local Outlier Detection. In *KDD*. 1235–1244.
[17] D. Yang, E. A. Rundensteiner, and M. O. Ward. 2011. Summarization and matching of density-based clusters in streaming environments. *Proceedings of the VLDB Endowment* 5, 2 (2011).