Fast Memory Efficient Local Outlier Detection in Data Streams

Article *in* IEEE Transactions on Knowledge and Data Engineering · December 2016

DOI: 10.1109/TKDE.2016.2597833

Fast Memory Efficient Local Outlier Detection in Data Streams

Mahsa Salehi, Christopher Leckie, James C. Bezdek, *Life Fellow, IEEE*,
Tharshan Vaithianathan and Xuyun Zhang

Abstract—Outlier detection is an important task in data mining, with applications ranging from intrusion detection to human gait analysis. With the growing need to analyze high speed data streams, the task of outlier detection becomes even more challenging as traditional outlier detection techniques can no longer assume that all the data can be stored for processing. While the well-known *Local Outlier Factor* (LOF) algorithm has an incremental version, it assumes unbounded memory to keep all previous data points. In this paper, we propose a *memory efficient incremental local outlier* (MiLOF) detection algorithm for data streams, and a more flexible version (MiLOF_F), which has an accuracy close to Incremental LOF but within a fixed memory bound. Our experimental results show that both proposed approaches have better memory and time complexity than Incremental LOF while having comparable accuracy. In addition, we show that MiLOF_F is robust to changes in the number of data points, the number of underlying clusters and the number of dimensions in the data stream. These results show that MiLOF/MiLOF_F are well suited to application environments with limited memory (e.g., wireless sensor networks), and can be applied to high volume data streams.

Index Terms—Outlier detection, Stream data mining, Local outlier, Memory efficiency

1 Introduction

Outlier detection (also known as rare event or anomaly detection) has received considerable attention in the field of data mining because of the need to detect unusual events in a variety of applications, such as fraud detection, human gait analysis and intrusion detection. A variety of outlier detection algorithms have been proposed for use on static data sets which have a finite number of samples. However, outlier detection on streaming data is particularly challenging, since the volume of data to be analyzed is effectively unbounded and cannot be stored indefinitely in memory for processing [1]. This is a significant problem that arises in many real applications. For example, an outlier detection system in wireless sensor networks must work with the limited memory in each sensor node in order to detect rare events in near real time. The cost of communication is an important factor in such systems, which limits the scope for downloading data to a central server for archiving and analysis. Hence a memory efficient outlier detection algorithm is needed for streaming data, as it satisfies the memory constraint in the sensor nodes and avoids the communication cost of having to transfer data to external storage. Data streams are generated at a high data rate, hence making the task of outlier detection even more challenging.

- *M. Salehi, C. Leckie and J. C. Bezdek are with NICTA Victoria and the Department of Computing and Information Systems, University of Melbourne, Parkville Victoria, 3010 Australia.*
  E-mail: {salehim,caleckie,jbezdek}@unimelb.edu.au
- *T. Vaithianathan is with the Department of Electrical and Electronic Engineering, University of Melbourne, Parkville Victoria, 3010 Australia.*
  E-mail: tva@unimelb.edu.au
- *X. Zhang is with NICTA Victoria, West Melbourne Victoria, 3003 Australia.*
  E-mail: xyzhanggz@gmail.com

While researchers have mostly studied distance based outlier detection in streaming environments [2], [3], [4], [5], the literature pays less attention to finding local outliers in data streams [5], [6]. The *local outlier factor* (LOF) detection technique [7] and many of its variates have been proposed for static datasets [8], [9], [10]. In terms of detecting local outliers in data streams, researchers in [11] proposed an *incremental LOF* (iLOF). However, this technique needs all previous data points to detect outliers precisely. In the case of data streams, where the number of data points is unbounded and can arrive at a high rate, keeping all data points is impossible. Hence, such an approach has limited use in practice.

Simply deleting data points which cannot fit into memory, as in sliding window techniques, is not useful as it can influence the detection accuracy for new incoming data points. Deleting past data points due to memory limitations causes two problems: i) we cannot differentiate new events from events that have already been seen, and ii) the accuracy of the estimated local outlier factor of data points will drop as we delete the history of earlier data points. The synthetic data set in Fig. 1 illustrates this point. Figs. 1a-c show a two dimensional data set that evolves over time intervals T_i $(i = 1 : 3)$ from left to right. Suppose that during T_1 a set of incoming data points are from a single distribution and form a cluster C_1 (Fig. 1a). Then during T_2 a second set of data points form another cluster C_2 (Fig. 1b). During time $T_1 + T_2$, the iLOF technique is successful in estimating LOF values and finding the new events. However, suppose that due to the limited amount of memory available (which could occur when processing relatively high volume data streams), the data points in C_1 are deleted by the iLOF algorithm at time $T_1 + T_2$. Thereafter, if during T_3 a set of new data points ($C_1^{'}$) arrive with the same distribution as

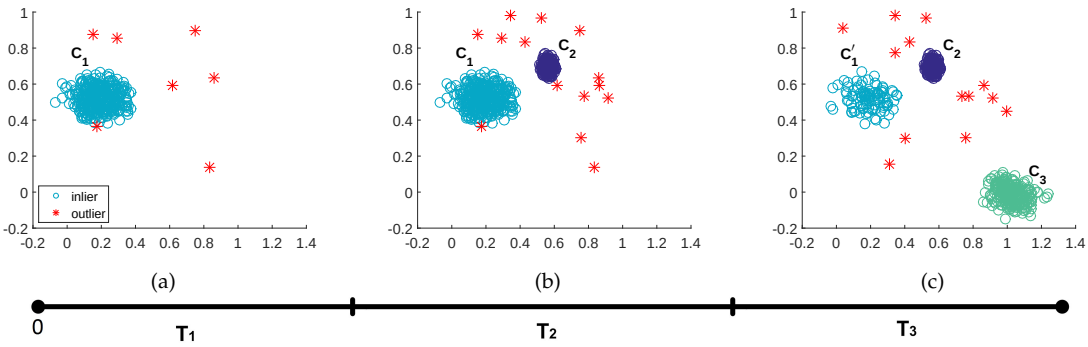


Fig. 1. Scatter plots of a two-dimensional synthetic data set evolving over time, containing normal data points (circles) and outliers (stars)

C_1, along with a new unseen set of data points C_3 (Fig. 1c), iLOF fails to differentiate which one is new and which one is old as no history of C_1 was kept. The estimated *LOF* values for the new incoming set of data points $C_1^{'}$ may not be accurate because there are not enough data points for estimation and this could result in the incorrect classification of normal points as outliers. This example demonstrates the need for an outlier detection framework that can perform accurately in the presence of recurring events and also accommodate new events in environments with finite memory.

To the best of our knowledge none of the state-of-the-art approaches are designed for detecting local outliers in limited memory systems. In this paper, we propose an approximation algorithm and its extension to tackle this problem. In particular, we have developed a novel algorithm called *memory efficient incremental local outlier detection* (MiLOF) for data streams in environments with limited memory. The goal is to compute LOF values that are similar to those computed by iLOF, within the given memory constraints. The algorithm summarizes the previous data points that exceed the memory constraint, taking into consideration their LOF scores. We have also proposed a more efficient approach which is an extension for MiLOF called MiLOF_F. This approach is capable of dynamically finding the number of summaries to keep in memory. In addition, it is more stable in terms of changes to the available memory size and more accurate in detecting outliers. Our experiments on both synthetic data sets and various real-life data sets show that our MiLOF framework and its extension MiLOF_F can successfully detect outliers while decreasing both the memory requirements and time complexity of iLOF. We have tested the scalability of our framework in terms of the number of data points, the number of dimensions and the number of clusters in the data sets. This paper substantially extends an earlier conference paper by the authors [12]. The contributions of this paper are as follows:

- A local outlier detection framework is proposed which works in limited memory environments. This *unsupervised* approach is particularly well-suited to detecting outliers in wireless sensor networks, in which nodes have limited memory and communication capacity.
- A novel adaptive clustering based approach for building summaries of data points for the local outlier detection problem is proposed.
- Our proposed algorithms improve the time complexity of iLOF to near linear complexity.
- Our proposed general framework has the potential to be applied to other variants of local outlier detection based techniques, such as Loci [8] and LoOP [10].

In the remaining sections, we first present related work on outlier detection algorithms in data streams in Section 2. In Section 3 we review the LOF and iLOF algorithms to compute LOF values and define the problem we address. Next we describe our proposed framework and its extension in Section 4. In Section 5, we show our experimental results and finally conclude the paper in Section 6, with some suggestions for future research.

2 Related Work

Comprehensive surveys of outlier detection are presented in [13], [14]. However, most of the algorithms described there focus on static environments, where all data points are available at once. In streaming environments where large volumes of data are generated at high speed, we need to detect outliers with a single scan (one pass) and in a limited time. Three categories of approaches have been proposed to detect outliers in data streams [11], [15], [16]: *distribution based*, *clustering based* and *distance based approaches*.

In *distribution based* approaches, the aim is to learn a probabilistic model based on the streaming data [17], [18]. However, *a priori* knowledge regarding the underlying distribution of the data set is required, which is almost always unavailable.

Clustering based approaches build clusters to model the underlying data distribution. Then, depending upon the algorithm, the data points that form small clusters or those that are far from their cluster centroid are considered outliers. However, the main purpose of designing such algorithms is to build a clustering model rather than detecting outliers, e.g., [19], [20], [21] and in high dimensional data streams [22], they are not necessarily optimized for outlier detection. Cluster histograms are used for modeling and mining data streams along with some guidelines for outlier detection applications [23]. In [24], another clustering based approach is specifically used for outlier detection, while their focus is on data streams with varying arrival rates. Recently, an ensemble-based algorithm has been proposed using a pool of clustering models over data streams to detect outliers [25].

Distance based approaches are designed to detect outliers by computing the distance of a point with respect to the other points in the data set. The detected outliers can be either 'global' or 'local'. A data point p is a distance-based 'global' outlier if less than k data points out of all n data points (where n is the number of points in the static data set) are within the distance d from p [26]. In [2] and [3], a sliding window is used to detect global distance-based outliers in data streams with respect to the current window. The authors in [27] use a data editing approach [28] to detect outliers in fast data streams and resource limited sensors. However, their approach is supervised and needs normal data for the data editing phase. The authors in [4] improved the time complexity and memory consumption in comparison to [2] and [3]. Finally in [5], a faster and more general framework called LEAP has been proposed for high volume high dimensional data streams by optimizing the search space. In addition, different types of distance-based outliers [29], [30] have been detected by this framework. Since the outliers are computed with respect to the last n data points, the outliers that are detected by these approaches are called 'global' outliers. Hence these approaches fail to detect outliers in *non-homogeneous* densities. Finally, [31], [32] and [33] are examples of outlier detection models for streaming data that combine *clustering* and *distance based* approaches.

In contrast to distance-based 'global' outliers, distance-based 'local' outliers are data points that are outliers with respect to their k nearest neighbors. Local outliers were discussed in [7] and a measure of being an outlier - the Local Outlier Factor (LOF) - was assigned to each data point in a static environment. Since the LOF technique achieves good detection accuracy in non-homogeneous densities (often the case in practice) without assumptions regarding the underlying distribution of the data set, it has become a popular approach and many variants of this technique have been proposed. Some techniques improve the detection accuracy of LOF by changing the way k-NNs are computed [34] so that it can cover a wider range of outlier types, and finding a symmetric neighborhood relationship [35], while others improve the time complexity of the initial LOF technique via approximation [8]. Since the time complexity of LOF is quadratic with respect to the number of data points (n), the authors of [36] try to find the top $m << n$ outliers, instead of computing the LOF factor over the whole data set. Other variants of LOF focus on covering more complex data types, e.g., categorical [37] or spatial [38] data sets. In [10], another variation of LOF is introduced by normalizing the outlier scores and finding the local outlier probabilities. In addition to detecting outliers, the problem of interpreting outliers to find important feature subspaces [39], as well as unifying and evaluating outlier scores [40], [41] in order to find better ensemble techniques have been studied. Finally, in [42] the problem of local outlier detection has been formalized by studying various relevant techniques, and a schema for this problem along with several of its applications have been proposed.

Pokrajak et al. [11] were the first to propose an incremental version of the LOF technique (iLOF) that can be used for data streams. All previous versions required the entire data set to compute LOF values for the data points, but in the iLOF technique the outlier factor is computed for each incoming data point. For each incoming data point p, the iLOF finds the *k-nearest neighbors* (k-NNs) of p, computes the local outlier factor of p based on the outlier factors of its k-NNs, and updates the k-NNs of past data points along with their local outlier factors if needed. Pokrajak et al. showed that only a few data points are required updating and therefore the algorithm was efficient.

Although iLOF addresses a significant issue with local outlier detection algorithms, it still has a major limitation. In order to detect outliers accurately, all past data points need to be retained to compute the outlier factor for each new incoming data point. Therefore, iLOF suffers from large memory requirements, as well as high time complexity.

3 Background on LOF and iLOF

Before describing our memory efficient framework, we briefly describe the original LOF algorithm to compute local outliers in static data sets. We also describe the incremental version of LOF (iLOF), which computes LOF values in data streams. Interested readers are referred to the original papers for more detailed descriptions [7], [11].

3.1 LOF algorithm

In the original LOF technique, the LOF values are computed for all data points according to the following definitions, assuming that all data points are available and the number of nearest neighbors is k.

- *k-distance*(p): the distance between a data point p and its k^{th} nearest neighbor (k^{th}-NN).
- *Reachability distance* (*reach-dist*) of a data point p with respect to another data point o:

$$\textit{reach-dist}_k(p,o) = max\{\textit{k-distance}(o), d(p,o)\} \quad (1)$$

  where $d(p,o)$ is the Euclidean distance between p and o.
- *Local reachability density* (lrd) of a data point p:

$$lrd_k(p) = \left(\frac{1}{k} \sum_{o \in N_{(p,k)}} \textit{reach-dist}_k(p,o) \right)^{-1} \quad (2)$$

  where $N_{(p,k)}$ is the set of k nearest neighbors of p.
- *Local outlier factor* of a data point p

$$LOF_k(p) = \frac{1}{k} \sum_{o \in N_{(p,k)}} \frac{lrd_k(o)}{lrd_k(p)} \quad (3)$$

In this paper we drop the subscript k in the above definitions for convenience, so we compute *k-distance*, *reach-dist*, lrd and LOF values assuming that the number of nearest neighbors is k.

3.2 iLOF algorithm

Let $T = [t_1, t_n]$ be a continuous interval of time, and let $p_t \in \Re^D$ denote a data vector collected at time $t \in T$ where D is the number of data dimensions. In the iLOF technique, an insertion algorithm is used to compute the LOF value for each incoming data point p_t:

1) Compute $N_{(p_t,k)}$ and *k-distance*(p_t)
2) For all $o \in N_{(p_t,k)}$ compute *reach-dist*(p_t, o)
3) Update the *k-distance*, *reach-dist*, lrd and LOF values for all Reverse k-NNs of p_t (the Reverse k-NNs of a data point p_t are the set of data points that have p_t in their k-NNs) and all other affected data points in the set of existing data points up to p_t, i.e., $\{p_1, ..., p_{t-1}\}$. Note that a point can be affected if one or more of its k-NNs has been updated.
4) Compute $lrd(p_t)$ and $LOF(p_t)$

The outputs of the iLOF algorithm are the LOF values for all data points. The LOF values can then be used to select outliers in the dataset.

3.3 Problem Definition

Given data vector $p_t \in \Re^D$ collected at time $t \in T$, the goal of MiLOF and MiLOF_F is to assign an LOF value to p_t, under the constraint that the available memory stores only a fraction $m << n$ of the n points that have been observed up to time T. In other words, it is impractical to store all n data vectors and their corresponding LOF values in memory. Hence we need to choose a strategy to summarize the previous data points so that the LOF values of new points can be calculated.

One approach to accommodate limited memory would be to just keep the latest m data points at time t ($\{p_{t-m+1}, ..., p_t\}$) and delete the rest (i.e., a sliding window). Since the $LOF(p_t)$ value for each incoming data point is computed based on its k-nearest neighbors, if we delete the old data points we might remove some nearest neighbors of the incoming data points. This can affect the LOF values and subsequent detection accuracy. In the next section, we describe two proposed approaches to address this problem.

4 MiLOF: Memory Efficient iLOF Algorithm and the MiLOF_F variant

In this section we describe the main phases of MiLOF and MiLOF_F. We assume that the memory limit allows m data points with their corresponding *k-distance*, *reach-dist*, lrd and LOF values to be stored in memory. In the remainder of this section we describe the details of each phase of the algorithm. Fig. 2 shows the different phases of MiLOF on a 2-dimensional data stream: 1) Summarization, 2) Merging and 3) Revised Insertion.

The pseudo code of the MiLOF algorithm is shown in Alg 1. The algorithm is similar to the damped window model (a.k.a. time-fading window model) [43], in which the latest data points are more significant than older data points. In this algorithm we allocate one part of the memory to store b data points, and the remaining part to store c summaries of older data points. For each incoming data point the LOF value is computed using the *revised insertion* algorithm of iLOF. Once the memory limit is reached, the algorithm *summarizes* the first $\frac{b}{2}$ data points and deletes them from memory. The summary of these deleted points is in the form of a set of c prototype vectors in $\Re^D$. If there exists a set of summary prototypes from a previous window in time, the two sets of prototypes are merged to build a single summary of past data points. The algorithm continues until it reaches the end of the data stream. This insures that the the number of data points stored in memory to detect outliers is not more than $m = b + c$.

The choice of b depends on memory limitation of the implementation platform. Time complexity will be further investigated theoretically and experimentally in Sections 4.5 and 5.3.1 respectively. We will also show that MiLOF/MiLOF_F's detection accuracies are roughly stable with respect to choosing different values for b.

Algorithm 1 MiLOF

1. **Input:** set of data points $P = \{p_1, .., p_n\}$
2. memory size limit of m (choose $b : c = m - b$)
3. **Output:** set of $LOF = \{LOF(p_1), .., LOF(p_n)\}$ values
4. $i \leftarrow 0$; {step}
5. **for all** $p_t \in P$ **do**
6.
7. $LOF(p_t) \leftarrow$ Revised Insertion(p_t)
8. **if** Number of data points in memory $= b$ **then**
9. $C^i \leftarrow \{p_{i\frac{b}{2}+m} | m \in \{1..\frac{b}{2}\}\}$
10. $(V^i, N^i) \leftarrow c$-means(C^i) {or Flexible c-means(C^i) in MiLOF_F, Algorithm 2}
11. **for all** $v^i_j \in V^i$ **do**
12. Compute *k-distance*(v^i_j), $lrd(v^i_j)$, $LOF(v^i_j)$ using Defs. 1, 3 and 4
13. **end for**
14. Delete C^i
15. **if** $i>0$ **then**
16. $(Z, W) \leftarrow$ Weighted c-means$(V^i \cup V^{i-1}, N^i \cup N^{i-1})$
17. **for all** $z_J \in Z$ **do**
18. Compute *k-distance*(z_J) using Equation 12 and $lrd(z_J)$, $LOF(z_J)$ similarly
19. **end for**
20. $V^i \leftarrow Z$
21. $N^i \leftarrow \{\sum_{w_I \in W_1} w_I, ..., \sum_{w_I \in W_{c'}} w_I\}$
22. Delete V^{i-1}, Z
23. Update $k-$NNs of remaining points if deleted
24. **end if**
25. $i \leftarrow i + 1$;
26. **end if**
27. **end for**
28. **return** LOF

4.1 Summarization

Every time the number of points in memory reaches the limit b, the algorithm invokes the summarization phase. This phase includes building a summary over the past data points along with their corresponding values (k-distance, lrd and LOF), and deleting them from memory. Half (the first $\frac{b}{2}$) of the b points in time are summarized for low resolution history retention, while the most recent $\frac{b}{2}$ points in time are kept intact to provide higher resolution of the most current data in the stream. Since the underlying distribution of data points can evolve gradually over time, the most recent data points are most important and therefore we keep them in memory for computing the next outlier factors. If a larger history at lower resolution is desired, the

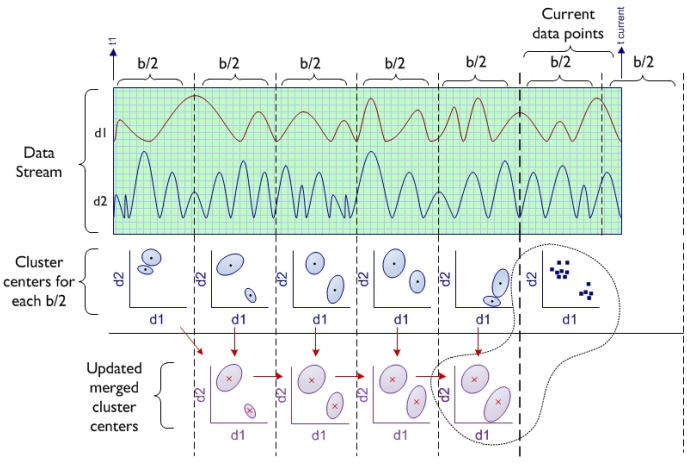


Fig. 2. Three phases of MiLOF on a 2 dimensional data stream with dimensions d_1 and d_2. 1) Summarization: every $\frac{b}{2}$ data points are summarized and cluster centers are generated. 2) Merge: current cluster centers (last diagram of middle row) with the previously merged cluster centers (last diagram of bottom row) forming updated cluster centers. 3) Revised Insertion: Assigning the LOF scores are based on current data points and updated cluster centers

summarization can be done less frequently. This can save time at the cost of decreasing the detection performance. On the other hand, if a smaller percentage of data is chosen for historical retention, this step is needed more often. As a result, the computation time increases. As the width of the summarization window decreases, MiLOF begins to resemble iLOF, and in the limit (when there is no historical retention by summarization), MiLOF reduces to iLOF. In this sense MiLOF is a direct generalization of iLOF. In this paper, we cluster the first half of the data points ($\frac{b}{2}$) and keep the second half ($\frac{b}{2}$).

The details of the summarization step are as follows: When the memory limit is reached, MiLOF clusters the first $\frac{b}{2}$ data points in memory. There are a virtually unlimited number of choices for the clustering algorithm [44]. MiLOF uses the c-means (a.k.a. "k-means" in much of the literature, but we have reserved k for the number of NNs, so we use c to avoid confusion about these two integers) algorithm. If the number of clusters is unknown (and possibly evolving with time), we try to overestimate c. Here we are not necessarily looking to find the apparent cluster substructure in the evolving data. Instead, we want to find similar data points and average them to keep only one point (the cluster center) as a representative of each cluster. MiLOF_F introduces a new flexible clustering algorithm based on c-means which summarizes the historical data points in a better way. The proposed algorithm leads to a higher detection accuracy and is independent of the number of clusters. It is called *Flexible c-means* and its description is in Section 4.1.1.

Step i of the algorithm begins at the time $t = i\frac{b}{2}$, so the first time for a point to arrive in step i is at $t = i\frac{b}{2} + 1$. To avoid using this awkward notation, the $\frac{b}{2}$ points arriving in step i are called $C^i = \{p_{i\frac{b}{2}+m} | m \in \{1..\frac{b}{2}\}\}$. Then the algorithm summarizes C^i. At the termination of c-means (or Flexible c-means) at step i of the algorithm, C^i is partitioned into c clusters, $C^i = \{C^i_1 \cup C^i_2 \cup ... \cup C^i_c\}$, having the cluster centers $V^i = \{v^i_1, v^i_2, ..., v^i_c\} \in \Re^D$. Then *k-distance*, lrd and LOF are computed for the new cluster centers V^i using the following definitions.

Definition 1. The *k-distance of a cluster center* $v^i_j \in V^i$ is defined as:

$$\textit{k-distance}(v^i_j) = \frac{\sum_{p \in C^i_j} \textit{k-distance}(p)}{|C^i_j|} \quad (4)$$

where $|C^i_j|$ is the number of data points in C^i_j.

Note that at this point, lrd and LOF values for all the data points in C^i_j have already been computed. Hence all properties of the cluster centers in V^i can be computed independently, and it is not necessary to compute *reach-dist* of cluster centers in V^i with respect to the remaining data points. However, for a newly incoming data point p_t where $t > b + ib$, it may be necessary to compute the *reach-dist* of data point p_t with respect to one of the cluster centers in V^i, according to the following definition.

Definition 2. The reachability distance of a data point p with respect to a cluster center $v^i_j \in V_i$ is defined as:

$$\textit{reach-dist}_k(p, v^i_j) = max\{\textit{k-distance}(v^i_j), d(p, v^i_j)\} \quad (5)$$

where $d(p, v^i_j)$ is the Euclidean distance between p and cluster center v^i_j.

We define the lrd and LOF of new clusters similarly.

Definition 3. The lrd of a cluster center $v^i_j \in V^i$ is defined as:

$$lrd_k(v^i_j) = \frac{\sum_{p \in C^i_j} lrd_k(p)}{|C^i_j|} \quad (6)$$

Definition 4. The LOF of a cluster center $v^i_j \in V^i$ is defined as:

$$LOF_k(v^i_j) = \frac{\sum_{p \in C^i_j} LOF_k(p)}{|C^i_j|} \quad (7)$$

In addition to computing these properties for all cluster centers in C^i, the algorithm retains the number of data points in each cluster, i.e., $N^i = \{|C^i_1|, ..., |C^i_c|\}$. Finally, it deletes the first half of the data points $\{p_{i\frac{b}{2}+1}, ..., p_{i\frac{b}{2}+\frac{b}{2}}\}$ from memory.

4.1.1 *Flexible c-means in MiLOF_F*

To increase the detection accuracy in the above approach, we decided to store previous inputs more selectively, by retaining those points that are most useful for subsequent outlier detection. To achieve this end, we consider the density distribution of the previous data stream. There may be some regions in the previous data stream with a high probability of containing outliers. Flexible c-means prunes these regions during summarization, while keeping the less probable ones. The intuition behind this approach is density reduction of outlier regions. Since the local outlier factor shows whether a data point is similar to its neighbors in terms of local density, by pruning the regions with high probability of being outliers, while keeping the summary of inliers, we can further decrease the similarity of a future outlier data point and its neighbours, and hence increase the

LOF. This means, the probability of assigning higher LOF values to future outliers increases, which make them more distinguishable from inliers.

Fig. 3 shows the output of Flexible c-means on our previous synthetic example in Fig. 1, assuming that the number of data points that have arrived at time intervals T_1 and T_2 are equal. The crosses (x's) in Fig. 3a corresponds to the cluster centers of the data points over time interval T_1 (Fig. 1a). Rather than storing all of the cluster centers, if we prune the summaries of the regions with a high probability of outlierness, i.e., the ellipses in Fig. 3a, these regions become less dense. Hence, future outliers in these regions (e.g. the stars in Fig. 1c) have a higher probability to be assigned high LOF scores. As a result, we only keep the selected cluster centers with high probability of inlierness (crosses in Fig. 3b) in addition to the data points at time interval T_2 (circles in cluster C_2 in Fig. 3b).

A similar approach appears in [45], where the accuracy of static LOF is improved by using random subsampling. MiLOF_F uses this pruning method to avoid random subsampling past summaries. In addition, our proposed algorithm does not require the exact number of clusters as input. In summary, our Flexible c-means algorithm has two main properties: 1) The algorithm builds a summarization by clustering while decreasing the density of outlier regions and keeping summaries of inliers. 2) The number of clusters (c) does not need to be known; instead the algorithm starts with an overestimation of c and dynamically decreases it through pruning.

Algorithm 2 Flexible c-means

1. **Input:** set of data points C^i
2. **Output:** set of cluster centers V^i
3. set of cluster member count N^i
   {Initial clustering of points into c clusters}
4. $(V^i, \{C_1^i \cup C_2^i \cup ... \cup C_c^i\}) \leftarrow c$-means$(C^i)$
   {Prune clusters with high probability of being outliers}
5. $\lambda \leftarrow \{$*k-distance*$(p) | p \in C^i\}$
6. $POT_{C^i} \leftarrow \mu(Y) + 3\sigma(Y)$ where $Y \in \lambda$
7. **for all** C_j^i in C^i **do**
8. **if** Majority of j^{th} cluster members' *k-distance* $> POT_{C^i}$ **then**
9. Remove relevant cluster center from V^i
10. **end if**
11. **end for**
12. $N^i \leftarrow \{|C_1^i|, ..., |C_{c^i}^i|\}$
13. **return** V^i and N^i

The pseudo code of this algorithm is shown in Alg 2. As mentioned, C^i is the set of data points in the time window to be summarized at step i. Using the c-means algorithm, initially C^i is partitioned into c clusters, $C^i = \{C_1^i \cup C_2^i \cup ... \cup C_c^i\}$, having the cluster centers $V^i = \{v_1^i, v_2^i, ..., v_c^i\} \in \Re^D$. Next the clusters need to be pruned. Here we need two definitions:

Definition 5. Prune Outlier Threshold (POT) for a set of data points C is defined as:

$$POT_C = \mu(Y) + 3\sigma(Y) \quad (8)$$

where Y is a random variable that takes sample values from the set $\lambda = \{$*k-distance*$(p) | p \in C\}$, and $\mu(Y)$ and $\sigma(Y)$ are its mean and standard deviation, respectively. Since the *k-distance* of data points follows the gamma distribution [46], we have defined POT using three standard deviations.

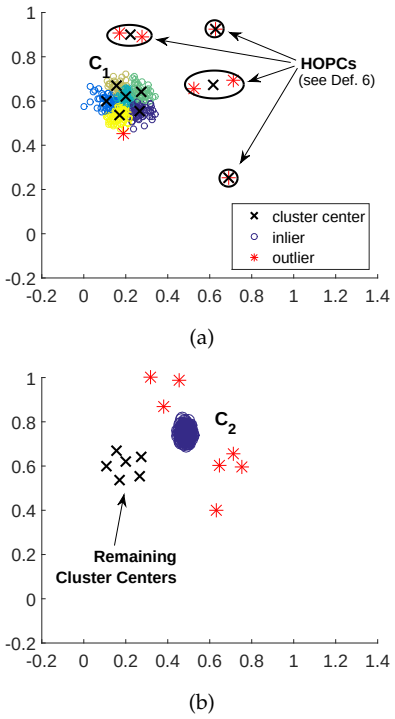


Fig. 3. Flexible c-means on the data points in Fig. 1a: (a) Cluster centers (crosses) and HPOCs (ellipses) in time interval T_1, (b) Remaining data points and cluster centers at time $T_1 + T_2$.

Definition 6. Assume C is a set of data points partitioned into c clusters: $C = \{C_1 \cup ... \cup C_c\}$. C_j is a High-Probable Outlier Cluster (HPOC) if:

$$|\{p | p \in C_j \quad \text{and} \quad \textit{k-distance}(p) > POT_C\}| > \frac{|C_j|}{2} \quad (9)$$

Using the above definitions, the algorithm deletes all HPOCs from C^i and their relevant cluster centers from V^i (see HPOCs in Fig. 3a). In other words, a cluster is omitted if the majority of its members' *k-distance*s are higher than the Prune Outlier Threshold (POT_{C^i}). Note that we look at the distribution of the points' *k-distance*s to compute this threshold and find HPOCs. Hence, pruning is based on 'global' outliers, which are more safely removed, than 'local' ones.

At the termination of Flexible c-means at step i of the MiLOF algorithm, C^i is partitioned into c^i clusters, $C^i = \{C_1^i \cup C_2^i \cup ... \cup C_{c^i}^i\}$, having the cluster centers $V^i = \{v_1^i, v_2^i, ..., v_{c^i}^i\} \in \Re^D$, where $c^i \leq c$. Note that in MiLOF_F the number of clusters in each step may vary from one step to another, in contrast to MiLOF in which the number of clusters in all steps is fixed.

4.2 Merging

Since summarization is performed every time $\frac{b}{2}$ new data points are received, a new set of cluster centers will be

generated after each step. The cluster centers V^i in step i are merged with the existing cluster centers V^{i-1} from the previous step $i-1$ so that there is only a single set of cluster centers maintained by the anomaly detection framework after each step.

In order to implement the merge, we cluster the cluster centers $X = \{V^i \cup V^{i-1}\}$ generated at steps i and $i-1$ to get c' new cluster centers $Z = \{z_1, z_2, ..., z_c\}$ using a *weighted clustering* algorithm, where $c' = c$ in MiLOF whereas $c' = max\{c^{i-1}, c^i\}$ in MiLOF_F. This technique is inspired from [21]. In the weighted clustering algorithm, each data point has a weight that shows its importance, and data points are clustered by taking into consideration those weights. Here a cluster center's weight is equal to the number of data points in that cluster. Set $\{N^i \cup N^{i-1}\}$ denotes the cluster centers' corresponding weights at steps i and $i-1$. We use a weighted c-means algorithm with a fixed number of iterations.

Let X be the cluster centers to be clustered and let W be their corresponding weights. At each iteration of weighted c-means, X is partitioned into c' clusters, $X = \{X_1 \cup X_2 \cup ... \cup X_{c'}\}$ with their corresponding weights $W = \{W_1 \cup W_2 \cup ... \cup W_{c'}\}$, having the new cluster centers $Z = \{z_1, z_2, ..., z_{c'}\} \in \Re^D$. Initially Z is equal to the latest set of cluster centers V^i. Thereafter, the algorithm iteratively clusters the set of cluster centers X such that the following equation is minimized:

$$TotalEnergy = \sum_{1 \le J \le c'} \sum_{x_I \in X_J, w_I \in W_J} w_I ||x_I - z_J||^2 \quad (10)$$

At each iteration the cluster centers $z_J \in Z$ are updated as follows:

$$z_J = \frac{\sum_{x_I \in X_J, w_I \in W_J} w_I x_I}{\sum_{w_I \in W_J} w_I} \quad (11)$$

4.2.1 Updating $kdist$, lrd and LOF

At the termination of weighted c-means, the *k-distance*, lrd and LOF for the new cluster centers $z_J \in Z$ is updated. The *k-distance* of the new cluster center $z_J \in Z$ is computed by averaging the weighted *k-distances* of its corresponding data points:

$$\textit{k-distance}(z_J) = \frac{\sum_{x_I \in X_J, w_I \in W_J} w_I \textit{k-distance}(x_I)}{\sum_{w_I \in W_J} w_I} \quad (12)$$

The lrd and LOF of the merged clusters are computed the same way.

4.2.2 Updating V^i and N^i

The latest cluster centers V^i are replaced with the new merged cluster centers Z. The number of data points in each cluster is also updated to the sum of the weights of each cluster's corresponding data points: $N^i = \{\sum_{w_I \in W_1} w_I, ..., \sum_{w_I \in W_{c'}} w_I\}$.

4.2.3 Updating the second $\frac{b}{2}$ data points

At this point the first half of the set of b data points are deleted. Since the LOF values have already been computed, these do not need an update. However, the k-nearest neighbors of the remaining data points should be updated with the new cluster centers in Z if they include the deleted data points or cluster centers from the previous step, V^{i-1}. At the end of the merge step V^{i-1} and Z are deleted from memory.

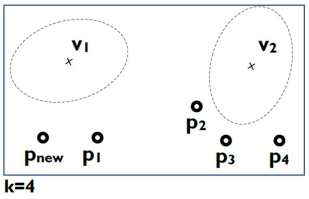


Fig. 4. The crosses v_1 and v_2 are cluster centers. The 4-nearest neighbors of a new data point p_{new} are p_1 and the remainder are in the cluster having v_1 as its center.

4.3 Revised Insertion

The revised insertion algorithm is similar to iLOF's insertion algorithm with slight modifications. In iLOF the LOF values are computed according to the previous "data points", whereas in MiLOF the LOF values are computed based on both the recent "data points" and "cluster centers".

Similar to iLOF there are two main steps:

1) Compute LOF value of the new incoming data point p: Find the k-NNs of each incoming data point. In the process of finding the k-NNs, if the i-th nearest neighbor (i<k) of p is a cluster center, we do not find the rest of the nearest neighbors. We assume that k is smaller than the number of data points in each cluster. Hence, if a cluster center is the ith nearest neighbor of a data point p while i<k, the rest of the neighbors would be in the same cluster and we stop searching for the rest of the nearest neighbors. Fig. 4 shows this situation for $k = 4$ and p_{new} is the new incoming data point. Since v_1 is its second nearest neighbor, the algorithm stops searching for the rest of the neighbors (all the points summarized by v_1 are also closer to p_{new} than the other points in Fig. 4) and considers $\{p_1, v_1\}$ as the 4-NNs of p_{new}. Note, this saves time as well. The process of finding *k-distance*, lrd and LOF for an incoming point are exactly the same as iLOF. However, in order to compute the reachability distance of a data point p with respect to a cluster center v, the algorithm uses Definition 2.
2) Update the *k-distance*, *reach-dist*, lrd and LOF values for the existing data points if required: For data points, the process of updating is the same as iLOF. However, if in the process of updating neighbors a cluster center becomes a candidate, we postpone it to the next merge. Since the *k-distance*, lrd and LOF values associated with each cluster center are the average of many data points in that area, the cluster is expected to be reasonably stable. Thus, a single data point will not affect the values as the updates are weighted with respect to the number of data points in the cluster. As we show in our experimental results, this strategy is successful in terms of detecting outliers.

TABLE 1
Comparison of time complexity in different implementations

		Efficient Implementation		Simple Implementation	
Algorithm	**Existing data points**	**Insertion**	**Total**	**Insertion**	**Total**
MiLOF/MiLOF_F	$b+c$	$log(b+c)$	$nlog(b+c)$	$b+c$	$n(b+c)$
iLOF	n	$log(n)$	$nlog(n)$	n	n^2

4.4 Time Complexity

The MiLOF algorithm consists of three phases. We compute the time complexity of each phase separately. The total number of D dimensional observations that have been seen so far in the data stream is n, k is the number of nearest neighbors, b is the number of observations that can be stored in memory and c is the number of summaries of older observations that can be stored in memory.

Summarization: The algorithm summarizes every $\frac{b}{2}$ set of incoming data in memory. Hence the summarization phase is repeated $\lfloor \frac{2n}{b} \rfloor - 1$ times. In each step, $\frac{b}{2}$ data points are clustered using the c-means algorithm and the relevant LOF values are computed. Therefore, the time complexity of each step is $O((I_1 c + 1)D\frac{b}{2})$ where c is the number of clusters and I_1 is the maximum number of c-means iterations. The summarization phase with all steps can be done in:

$$T_{Summarization} = O((I_1 c + 1)Dn) \tag{13}$$

which is linear with respect to the number of data points and the dimension D.

Merge: This phase is done after each summarization, except for the first step. Therefore, merge repeats $\lfloor \frac{2n}{b} \rfloor - 2$ times. In each step, a weighted c-means algorithm is applied on $2c$ weighted data points and the relevant LOF values are computed. Here the time complexity is $O((I_2 c + 1)D(2c))$ where c is the number of clusters and I_2 is the number of iterations of weighted c-means. In addition, the k-NNs of existing data points are updated, which takes $O(\frac{b}{2}k)$. Altogether, the merge phase is done in:

$$T_{Merge} = O((4c(I_2 c + 1)D/b + k)n) \tag{14}$$

which is linear with respect to the number of data points and the dimension D.

Revised Insertion: The time complexity of revised insertion in MiLOF is similar to the insertion algorithm that is used in iLOF but there are a few modifications that must be accounted for. The revised insertion algorithm is repeated $\lceil \frac{2n}{b} \rceil$ times. In each step, $\frac{b}{2}$ data points are inserted and LOF values are computed based on the existing set of data points with maximum size b and cluster centers c ($b + c$). For the first step the existing set of data points is a maximum $\frac{b}{2}$ as no previous set of data points of size $\frac{b}{2}$ nor cluster centers are available. We compute the time complexity of each step.

For each incoming data point, computing k-NNs, reverse k-NNs and inserting the data point all depend on the implementation. Table 1 shows two different implementations along with their time complexity. We discuss the time complexity of revised insertion in MiLOF for the two implementations separately:

1) In MiLOF we need to go through the existing data points as well as c cluster centers and if indexing structures are used, computing k-NNs, reverse k-NNs and inserting the data point can be done in $O(log(b + c))$. However inserting a data point may require updates of a few data points. As shown in [11], the maximum number of updates F does not depend on the number of data points. However, F is exponentially proportional to the number of data dimensions D. According to [11], the time complexity for each incoming data point considering all updates is $O(kFlog(b + c) + F^2k)$, and for all $b/2$ data points in each step is $O(\frac{b}{2}(kFlog(b+c)+F^2k))$ except for the first step which is $O(\frac{b}{2}(kFlog(\frac{b}{2}) + F^2k))$. Altogether, the time complexity of revised insertion of MiLOF after n data points using indexing structures is:

$$T_{Revised\ Insertion\ Efficient} = O(kFlog(b+c)n+F^2kn) \tag{15}$$

which is linear with respect to the number of data points, and the impact of dimension D is exponential.

2) On the other hand, if we use a simple implementation (no indexing structures), computing k-NNs, reverse k-NNs and inserting the data point can be done in $O(b + c)$ (in the first step it is $\frac{b}{2}$) and hence the revised insertion algorithm can be done in:

$$T_{Revised\ Insertion} = O(kF(b + c)n + F^2kn) \tag{16}$$

This approach is also linear with respect to the number of data points.

Considering all steps of the algorithm, as shown in Table 1, by using an efficient implementation, the MiLOF algorithm is $O(nlog(b+c))$, which is linear with respect to n, while iLOF is done in $O(nlog(n))$. However, if an indexing structure is not used, the time complexity of MiLOF is $O(n(b + c))$, which is again linear with respect to n and better than iLOF with $O(n^2)$. Note that both algorithms are exponentially proportional to D, the number of dimensions of the input vectors. In conclusion, MiLOF does reduce the time complexity in terms of the number of data points in comparison to iLOF.

The time complexity of MiLOF_F is computed in a similar way to MiLOF, while there are some changes in the summarization phase in which Flexible c-means is used. However, that does not affect the overall time complexity, which is still $O(n)$.

4.5 Upper Bound and Lower Bound for Parameter b in High Volume Data Streams

As discussed earlier, since $m < n$ where $m = b + c$, MiLOF/MiLOF_F is more efficient than iLOF in terms of

memory consumption. In other words, with a constant number of clusters c, the fixed parameter b can be viewed in terms of a fraction of the n points seen so far in the data stream $x = b/n$, where $0 < x < 1$. On the other hand, MiLOF/MiLOF_F might not be more time efficient than iLOF for all values of x in the above range. We show in Theorem 4.1 that MiLOF/MiLOF_F remains more time efficient than iLOF in the above range for *high volume* data streams.

Theorem 4.1. Given a high volume data stream with n number of data points, where n is large, MiLOF/MiLOF_F is more efficient than iLOF in terms of time complexity for any non-zero fraction b of the number of data points.

Proof: To find the conditions under which MiLOF is more efficient, we need to find for which values of x, the time complexity of MiLOF is less than the time complexity of iLOF, i.e., the values of x for which the following equation is satisfied (Here we just prove the theorem for MiLOF, but we can use the same logic for MiLOF_F):

$$T_{Summarization} + T_{Merge} + T_{Revised\ Insertion} < T_{iLOF} \quad (17)$$

where $T_{Summarization}$, T_{Merge} and $T_{Revised\ Insertion}$ are the time complexities of the three phases of MiLOF as described in Section 4.4 using the simple implementation (Equations 13, 14 and 16). Given the definition of $x = b/n$, we can replace b with xn. We also omit the effect of the constant values c, I_1 and I_2 for large n. Hence we have:

$$(D + \frac{D}{xn} + k + kFxn)n < kFn^2 \quad (18)$$

Since F is exponentially proportional to D, we can replace D with F in Equation 18:

$$(F + \frac{F}{xn} + k + kFxn)n < kFn^2 \quad (19)$$

Solving Equation 19 for x, we find x is in the range $f_1(n) - f_2(n) < x < f_1(n) + f_2(n)$, where $f_1(n) = \frac{1}{2} - \frac{1}{2Fn} - \frac{1}{2kn}$ and $f_2(n) = \frac{\sqrt{((k+F-nkF)^2 - 4kF^2)n^2}}{2kFn^2}$.

Let $g(n) = f_1(n) \pm f_2(n)$. For high volume data streams, $\lim_{n\to\infty} f_1(n) = \frac{1}{2}$ and $\lim_{n\to\infty} f_2(n) = \frac{1}{2}$ so letting $n \to \infty$ in $g(n)$ shows that $\lim_{n\to\infty} g(n) = \frac{1}{2} \pm \frac{1}{2}$ =0 or 1, i.e., $0 < x < 1$. ■

5 Experimental Results

In this section the performance of MiLOF and MiLOF_F are compared to iLOF. First, we describe the data sets that we used.

5.1 Data sets

Table 2 summarizes the data sets used in our experiments. Five are publicly available real-life data sets and there are 3 groups of synthetic data.

UCI Data sets: We choose three labeled data sets with different properties from the UCI machine learning repository[1]: Vowel, Pendigit and Letter. The first three rows in Table 2 show their properties. We add 5% noise to all three data sets, and for Pendigit and Letter we perform the experiments over the test sets.

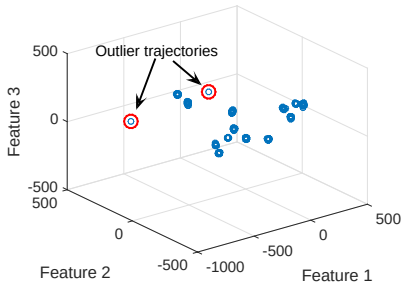


Fig. 5. Scatter plot of Motion Trajectory dataset showing two unusual trajectories (hollow circles inside the red donuts)

IBRL Data set: The IBRL data set consists of measurements from 54 sensors which were deployed in the Intel Berkeley Research Lab[2]. In our experiments we choose the measurements from 22 March until 24 March 2004 of sensor 45, which stopped working at the end of this period. We used two features/dimensions: humidity and temperature. We used the *Mahalanobis distance* of each data point to the aggregated sample set X to label the data set, i.e., (i) We initialize X with all the data points in IBRL and compute the grand mean μ of X; (ii) Compute the sample covariance matrix $\sigma(X)$; and (iii) For each $x \in X$, compute *Mahalanobis distance* $d(x, \mu) = (x - \mu)^T(\sigma(X))^{-1}(x - \mu)$. We then label $x^* \in X$ as an outlier in the IBRL data set when $d(x^*, m) > t$. In this experiment we take t = 10.

Motion Trajectory: This data set was used by Pokrajac et.al. [11] to evaluate the performance of their iLOF algorithm. The data set consists of 238 motion trajectories, which are represented by three features extracted from surveillance videos[3]. Fig. 5 shows a scatter plot of this dataset. There are two outlier trajectories, shown as hollow circles in Fig. 5 that were successfully detected by iLOF.

Synthetic: Three groups of synthetic data sets of Gaussian clusters were generated. Each data set consists of samples from a mixture of c underlying component distributions with random means and covariances. Each set was designed to test the scalability and robustness of MiLOF to one of three different parameters: the number of data points (n), number of data dimensions (D) and number of underlying clusters in the data set (c). Each data set is named Synthetic followed by the parameter that we varied in that data set.

In the first group (Synthetic-n), we generated 11 different data sets with increasing n, with parameters D and c fixed (Table 2, 6^{th} row). The smallest data set had n=2000 data points and was subsequently increased in 2000 sample increments to n=18,000, indicated in Table 2 by the notation (2000:2000:18,000). We also generated two high volume data sets with 10^5 and 10^6 data points.

The second group is called Synthetic-D in which D is varied while n and c are fixed (Table 2, 7^{th} row). We generated 10 Synthetic-D data sets ranging from 5 to 50 increments of 5.

1. http://archive.ics.uci.edu/ml

2. http://db.lcs.mit.edu/labdata/labdata.html

3. http://www-users.cs.umn.edu/~aleks/inclof

TABLE 2
Data sets properties

	Dataset	n = # data points	D = # dimensions	c = # of labeled subsets	Notes
1	UCI Vowel	1040	10	11	Add 5% noise
2	UCI Pendigit	3600	16	10	Add 5% noise
3	UCI Letter	4100	16	26	Add 5% noise
4	IBRL	3000	2	Unlabeled	*Mahalanobis distance*
5	Motion Trajectory	238	3	Unlabeled	Unusual trajectories
6	Synthetic-n	Variable (2000:2000:18000) 100000 and 1000000	2	3	Add 5% noise
7	Synthetic-D	2000	Variable(5:5:50)	5	Add 5% noise
8	Synthetic-c	500 in cluster	2	Variable(5:5:45)	Add 5% noise

The last group of synthetic data sets is called Synthetic-c, in which we change the number of clusters in each data set. Here, instead of keeping n the same for all data sets, we keep the number of data points in each cluster the same, because otherwise the number of points in each cluster would become less upon increasing c. There are 9 data sets in this group and the number of underlying clusters increases from 5 to 45 increments of 5 (Table 2, last row).

5.2 Performance Measures

Three performance measures are used to compare MiLOF/MiLOF_F to iLOF. We depict a ROC curve (false positive rate vs. true positive rate) for the LOF values in all experiments such that each point on the curve is related to a threshold of the resulting LOF values for both algorithms. The Area Under the ROC Curve (AUC) is the first performance measure and we refer to it as *detection accuracy*. The second measure is the *run time* of each algorithm and the last measure is the *number of data points held in memory*.

We compared the scalability of MiLOF_F to iLOF using the same measures.

5.3 Results and Discussion

In all of the experiments, the number of nearest neighbors k in MiLOF, MiLOF_F and iLOF is set to 10, except for the IBRL data set. In this data set, the detection accuracy with iLOF is low, so k=30 is used to enable higher detection accuracy for iLOF. Both algorithms are implemented similarly (simple implementation) in Matlab R2014a and all experiments were done on a Core i7-2600 CPU 3.40GHz running Windows 7. The 5% noise added to the UCI and synthetic data sets is uniformly randomly distributed between the minimum and maximum values observed in each dimension. Finally, we start with an overestimation of c equal to 50 for all 8 data sets in MiLOF_F.

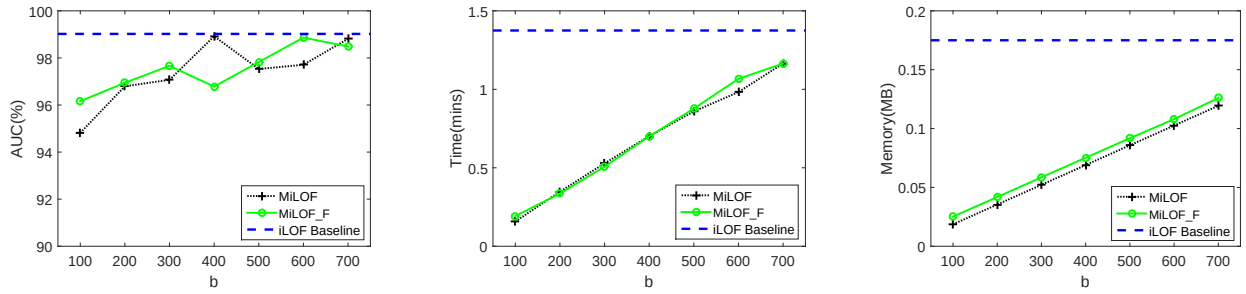
5.3.1 Effect of Parameter b

In the first experiment we change the parameter b and investigate its effect on the AUC, time and memory of MiLOF and MiLOF_F over all real life data sets. In the experiments we increase b until the time consumption of MiLOF exceeds iLOF. We have also applied iLOF on each real life data sets once, using the assumption of unlimited memory.

Figs. 6a-c show the results on the Vowel data. The horizontal axis in all three views is MiLOF/MiLOF_F's step size b, with $n = 1040$ for Vowel. The two graphs in the three views show AUC (Fig. 6a), computation time (Fig. 6b) and memory consumption (Fig. 6c) over increasing parameter values of b for MiLOF and MiLOF_F respectively. There are also dashed horizontal lines in each figure which are the results of iLOF. In Fig. 6a, the AUC of iLOF is 99% while the AUC for different values b for MiLOF and MiLOF_F are slightly lower than the baseline. However, the difference between iLOF's baseline and MiLOF's mean value is 1.64% whereas the difference between iLOF's baseline and MiLOF_F's mean value is 1.48%. These results suggest that the detection accuracy of our methods are comparable with iLOF. The mean AUC of MiLOF_F is slightly better than MiLOF, and is less sensitive to changes in b (the standard deviation of MiLOF is 1.39%, while the standard deviation of MiLOF_F is 0.96%). On the other hand, Figs. 6b and 6c illustrate that the computation time and memory consumption in MiLOF and MiLOF_F are always lower than the iLOF baseline in the chosen b range, which is a considerable improvement.

The trend of the graphs in Fig. 6-b suggests that if b is increased beyond $\frac{2n}{3} \sim 693$, the computation time of both MiLOF and MiLOF_F would be more than iLOF. After extensive tests with many values, we suggest limiting the *upper bound* for b to roughly $\frac{2n}{3}$ in smaller data sets. The *lower bound* can be any positive value with which we can build a meaningful summarization with half of that value. On the other hand, in high volume data streams with a large number of data points, by Theorem 4.1 MiLOF/MiLOF_F remains more time efficient than iLOF for any non-zero fraction b of the number of data points. Moreover, if $b+c < n$ where c is the memory allocated to cluster centers and n is the data set size, then for large n the memory consumption of MiLOF/MiLOF_F is less than iLOF. The same experiment was done on the other four real life data sets. The computation time and memory consumption diagrams for the four data sets are similar to the Vowel data set's diagrams (Fig. 6b and 6c), hence we do not plot them here to avoid redundancy. Since the number of clusters in the IBRL and Motion Trajectory data sets are not known, in the MiLOF algorithm we set $c = 4$, which might be an inaccurate estimation. For the real life data sets, we conclude that the MiLOF/MiLOF_F approaches are more efficient than iLOF in terms of time and memory in the suggested b range.

We now consider the detection accuracy of MiLOF/MiLOF_F over these data sets in more detail. Fig. 7 summarizes the AUC of the five real life data sets.



(a) Area Under ROC Curve (b) Computation Time (c) Memory Consumption

Fig. 6. Comparison of MiLOF (in dotted line) and MiLOF_F (in solid line) with iLOF baseline (in dashed line) for Vowel data set while changing b

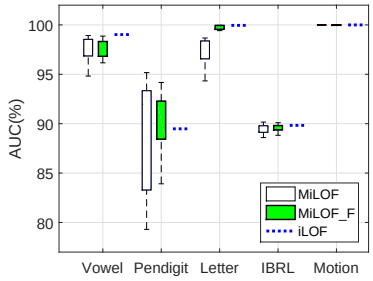


Fig. 7. Boxplot of MiLOF/MiLOF_F's detection accuracy over five real life data sets while changing the parameter b, in comparison to iLOF's baseline detection accuracy

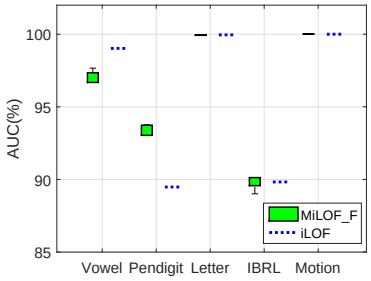


Fig. 8. Boxplot of MiLOF_F's detection accuracy over five real life data sets while changing the parameter c, in comparison to iLOF's baseline detection accuracy

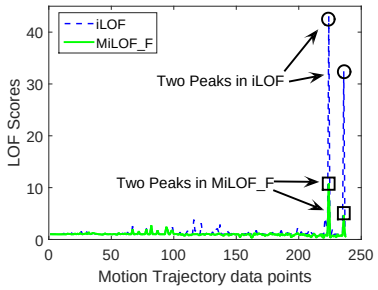


Fig. 9. LOF values on Motion Trajectory dataset, by MiLOF_F (in solid line) and iLOF (in dashed line) with $b = 100$

In this figure, each boxplot illustrates the changes in MiLOF/MiLOF_F's AUC over a real life data set while varying b. For each data set on the horizontal axis in the figure there are two boxplots and a dotted horizontal line. The left boxplot in each data set is MiLOF's AUC, the second boxplot from left is MiLOF_F's AUC and the dotted horizontal line on the right hand side of the boxplots is iLOF's baseline AUC.

For the Vowel, Letter and IBRL data sets, MiLOF/MiLOF_F's median AUC is slightly lower than iLOF's baseline. In the Vowel data set, the difference between MiLOF's and MiLOF_F's median from iLOF are 1.49% and 1.35% respectively. In the Letter data set these differences are 2.31% and 0.13% respectively, and in the IBRL data set they are 0.12% and 0.06% respectively. In all three data sets the standard deviations are very low, and these values in MiLOF_F are lower than MiLOF. Hence MiLOF_F is more accurate than MiLOF and less sensitive to changes in b.

In Pendigit, considering the standard deviation, MiLOF fluctuates a lot from the iLOF baseline by changing b. However, MiLOF_F is much less sensitive. In addition, the median AUC of both MiLOF/MiLOF_F is higher than iLOF with differences of 2.48% and 1.46% respectively.

Finally, in the last data set, i.e., Motion Trajectory, the median in MiLOF/MiLOF_F and the baseline in iLOF are all equal to 100%. According to the data set's boxplot, all three algorithms successfully detect the two unusual trajectories highlighted in Fig. 5.

All of MiLOF's boxplots in Fig. 7 (except one) are narrow, which means that the standard deviation is small. However, all MiLOF_F's boxplots are even narrower, which suggests a higher stability to changes in b. Although there are some deviations, the detection accuracy in MiLOF and MiLOF_F are comparable with iLOF in all real life data sets, while having better time and memory consumption than iLOF. Finally, MiLOF_F is more efficient than MiLOF in terms of stability of changes in stepsize b.

5.3.2 *Effect of Parameter c on MiLOF_F*

We have investigated how sensitive MiLOF_F's AUC is to the initial value of c. We change the parameter c between 30 and 50 for all real life data sets while keeping the parameter b fixed (For each data set we picked the median AUC and set b to its corresponding value). Fig 8 shows the AUC results for MiLOf_F using boxplots and for iLOF using dotted horizontal lines. The source of variance in the boxplots is changing parameter c. For each data set on the horizontal axis in the figure there are a boxplot and a dotted horizontal line. The left boxplot in each data set is MiLOF_F's AUC and the dotted horizontal line on the right hand side of each boxplot is iLOF's relevant AUC. As expected, while getting similar results to Fig 7 in terms of AUC, the standard deviation in all data sets are very low (two of them are zero). Specifically, we depicted the LOF values assigned to data points by iLOF (dashed diagram) and MiLOF_F (solid diagram) in Fig. 9 for the Motion Trajectory data set (in which c is unknown). The solid diagram shows the MiLOF_F's mean LOF value for all values of c. It can be seen that the LOF values of MiLOF_F are generally similar to those of iLOF. Both algorithms assign a relatively high LOF score (two peaks in both diagram) to the two unusual

trajectories in comparison to normal trajectories, which are shown in Fig. 5.

Both figures suggest that our flexible c-means algorithm is effective in finding the number of summaries of older data points dynamically and hence finding outliers, independent of c's initial value. In other words, the initialization of c has little effect on MiLOF_F's detection accuracy.

5.4 Scalability

To investigate the scalability of our MiLOF_F approach, data sets 6-8 vary number of data points, the number of underlying clusters and the number of data dimensions. The stepsize b is fixed, b=500, for these tests, so memory is always conserved by MiLOF_F.

Figs. 10a-c show the results of applying MiLOF_F and iLOF to the Synthetic-n data sets. It can be seen in Fig. 10a that as the number of data points increases, the detection accuracy in MiLOF_F remains roughly stable whereas there is a steady decrease in the detection accuracy of iLOF, especially for the high volume data sets, i.e., 10^5 and 10^6 data points. The mean difference for the two methods is 4.83% which shows that MiLOF_F outperforms iLOF in terms of detection accuracy in this group of data sets. As more data points arrive, the density of outlier and inlier regions increases. Due to the pruning procedure in the outlier regions in MiLOF_F the LOF scores that were assigned to outliers is higher than in iLOF, which helps to improve accuracy. The second graph (Fig. 10b) shows that the computation time in iLOF increases non-linearly because of the time complexity of the k-NN search over the entire data set for each incoming data point, while in MiLOF_F there is a slight rise as expected based on the derived time complexities. Finally in Fig. 10c, the occupied memory in MiLOF_F is fixed and is proportional to $b+c$, while in iLOF it depends on the number of data points in the data set which is increasing. These results show that MiLOF_F achieves better detection accuracy than iLOF while requiring substantially less time and memory in the Synthetic-n data sets.

Figs 11a-c show the results on the second group of synthetic data sets, Synthetic-D. As can be inferred from Fig. 11a, the detection accuracy in MiLOF_F is higher than iLOF while MiLOF_F remains more stable than iLOF. As the data dimensionality increases the data sets become more sparse. Again pruning the outlier regions helps MiLOF_F to distinguish outliers better. The AUC mean difference between the two algorithms is 11.09%. Since in this group of data sets the number of data points are kept roughly similar, all data sets require roughly similar time (Fig. 11b). However, as the number of existing data points in memory in MiLOF_F is less than iLOF, the time complexity is lower in MiLOF_F. In addition, we find as the number of dimensions increases the summarization phase needs fewer iterations for clustering and hence MiLOF_F's time consumption remains stable. The last graph (Fig 11c) shows that the required memory in MiLOF_F is less than iLOF, while they both rise when increasing the number of dimensions.

The results on the Synthetic-c group of data sets is shown in the last set of graphs (Figs 12a-c). The first graph (Fig. 12a) shows that the detection accuracy in both methods changes approximately with the same pattern and is stable as the number of clusters increases. However, the detection accuracy in MiLOF_F is higher than iLOF for higher numbers of clusters. Since the number of data points also increases by increasing the number of clusters, similar to Fig. 10a the AUC of MiLOF_F outperforms iLOF beyond some point. The standard deviation for MiLOF_F and iLOF are 3.06% and 3.27% respectively. This suggests that the detection accuracy in both methods are robust to changes in the number of clusters. In addition, since the number of data points in each cluster is kept constant, by increasing the number of clusters, the number of data points in increases, which leads to a higher computation time in iLOF in comparison to MiLOF_F (Fig 12b). Similarly, the memory consumption in iLOF is increasing, while it remains constant in MiLOF_F (Fig 12c).

To summarize, the results in Fig. 10, Fig. 11 and Fig. 12 show that MiLOF_F achieves comparable accuracy to iLOF while requiring substantially less time and memory. Moreover, MiLOF_F is more robust to changes in n, D and c, and is well suited to high volume data streams.

6 Conclusions

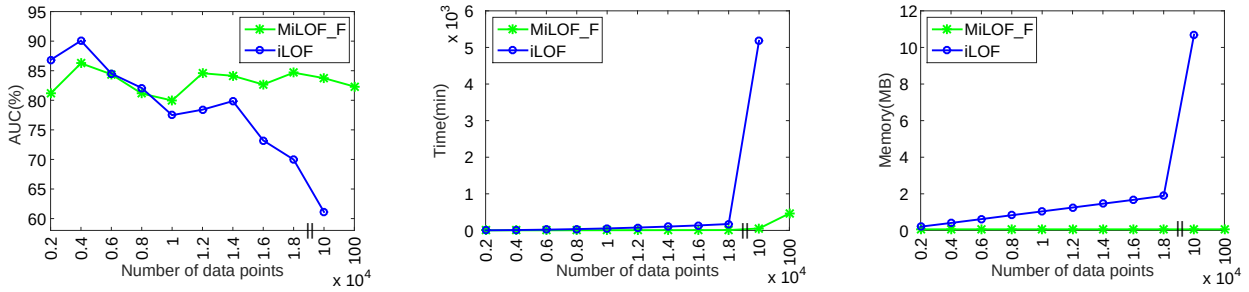
Memory efficiency is an important requirement in the field of outlier detection for data streams. In this paper, we proposed two approaches to tackle the problem of identifying local outliers. Our first algorithm (MiLOF) has substantial practical significance in improving widely cited methods (LOF and incremental LOF). iLOF requires saving all previous data points to compute local outlier factors. MiLOF method avoids this difficulty by summarizing subsets of the previous data and accumulating an evolutionary history of the streaming information. In this way MiLOF saves computation time as well as memory. For the data sets used in our experiments, MiLOF required substantially less computation time and memory than iLOF, while achieving comparable accuracy.

In our second approach (MiLOF_F), a more efficient and flexible algorithm reduces the density of outliers in the previous data points to increase the detection accuracy for future data points, while remaining fast and memory efficient. Experimental results demonstrate that MiLOF_F is more stable than MiLOF as it is less sensitive to the size of available memory and independent of number of data summaries. In addition we showed that MiLOF_F is more scalable with respect to the dataset size, the number of data dimensions and the number of clusters.

Consequently, both MiLOF/MiLOF_F seem well suited to application environments with limited memory. In the future, we intend to apply MiLOF_F to wireless sensor networks. Since memory constraints are generally a shortcoming of local outlier detection methods, we will test our approach on other variants of the LOF family of outlier detection algorithms.

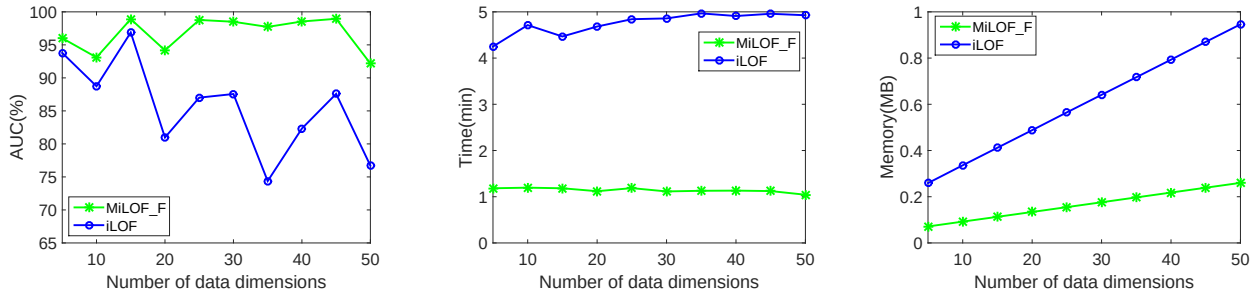
Acknowledgments

The authors would like to thank National ICT Australia (NICTA) for providing funds and support.



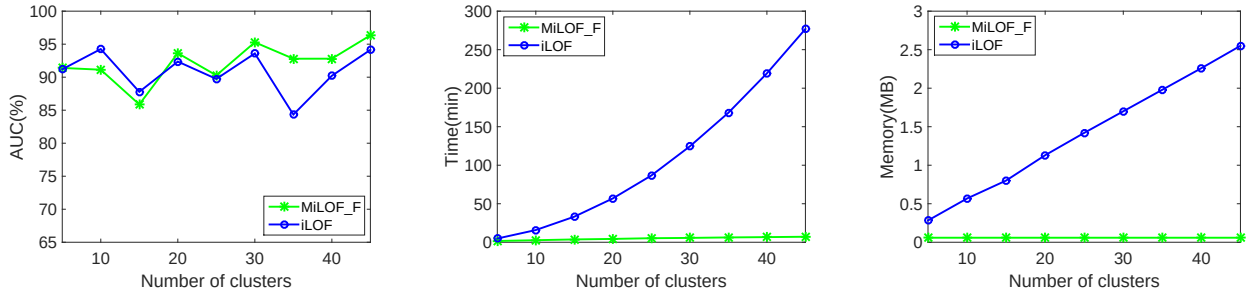
(a) Detection accuracy (b) Computation Time (c) Memory Consumption

Fig. 10. Comparing MiLOF_F with iLOF on Synthetic-n data sets. Note the discontinuity in the horizontal axis for the last two high volume data sets. iLOF exceeds a reasonable time limit for the last data set (with one million data records) so there is no value shown for it in all three plots



(a) Detection accuracy (b) Computation Time (c) Memory Consumption

Fig. 11. Comparing MiLOF_F with iLOF on Synthetic-D data sets



(a) Detection accuracy (b) Computation Time (c) Memory Consumption

Fig. 12. Comparing MiLOF_F with iLOF on Synthetic-c data sets

REFERENCES

[1] S. Sadik and L. Gruenwald, "Research issues in outlier detection for data streams," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 1, pp. 33–40, 2014.

[2] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *ACM Conference on Information and Knowledge Management*, 2007, pp. 811–820.

[3] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Advances in Database Technology*, 2009, pp. 529–540.

[4] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *International Conference on Data Engineering*, 2011, pp. 135–146.

[5] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *International Conference on Data Engineering*, 2014, pp. 76–87.

[6] L. Cao, Q. Wang, and E. A. Rundensteiner, "Interactive Outlier Exploration in Big Data Streams," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, 2014.

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

[8] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "Loci: Fast outlier detection using the local correlation integral," in *International Conference on Data Engineering*, 2003, pp. 315–326.

[9] K. Zhang, M. Hutter, and H. Jin, "A new local distance-based outlier detection approach for scattered real-world data," in *Advances in Knowledge Discovery and Data Mining*, 2009, pp. 813–822.

[10] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "LoOP: local outlier probabilities," in *ACM Conference on Information and Knowledge Management*, 2009, pp. 1649–1652.

[11] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *Computational Intelligence and Data Mining*, 2007, pp. 504–515.

[12] M. Salehi, C. Leckie, J. C. Bezdek, and T. Vaithianathan, "Local outlier detection for data streams in sensor networks: Revisiting the utility problem," in *IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, April 2015.

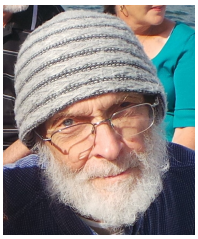
[13] S. Rajasegarar, C. Leckie, and M. Palaniswami, "Anomaly detection in wireless sensor networks," *IEEE Wireless Communications*, vol. 15, no. 4, pp. 34–40, 2008.

[14] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A

survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.
[15] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier Detection for Temporal Data: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 1–20, 2013.
[16] C. C. Aggarwal, "Outlier Analysis," 2013.
[17] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, "Online unsupervised outlier detection using finite mixtures with discounting learning algorithms," in *SIGKDD*, 2000, pp. 320–324.
[18] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *SIGKDD*, 2002, pp. 676–681.
[19] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *VLDB*, 2003, pp. 81–92.
[20] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *SIAM Conference on Data Mining*, 2006, pp. 328–339.
[21] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.
[22] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *International Conference on Very Large Data Bases-Volume 30*, 2004, pp. 852–863.
[23] C. C. Aggarwal, "A segment-based framework for modeling and mining data streams," *Knowledge and information systems*, vol. 30, no. 1, pp. 1–29, 2012.
[24] I. Assent, P. Kranen, C. Baldauf, and T. Seidl, "Anyout: Anytime outlier detection on streaming data," in *Database Systems for Advanced Applications*, 2012, pp. 228–242.
[25] M. Salehi, C. A. Leckie, M. Moshtaghi, and T. Vaithianathan, "A Relevance Weighted Ensemble Model for Anomaly Detection in Switching Data Streams," in *Advances in Knowledge Discovery and Data Mining*, 2014, pp. 461–473.
[26] E. M. Knox and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *International Conference on Very Large Data Bases*, 1998, pp. 392–403.
[27] V. Niennattrakul, E. Keogh, and C. A. Ratanamahatana, "Data editing techniques to allow the application of distance-based outlier detection to streams," in *IEEE International Conference on Data Mining*. IEEE, 2010, pp. 947–952.
[28] E. Pkalska, R. P. Duin, and P. Paclik, "Prototype selection for dissimilarity-based classifiers," *Pattern Recognition*, vol. 39, no. 2, pp. 189–208, 2006.
[29] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *ACM SIGMOD Record*, vol. 29, no. 2, 2000, pp. 427–438.
[30] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *Principles of Data Mining and Knowledge Discovery*, 2002, pp. 15–27.
[31] M. Elahi, K. Li, W. Nisar, X. Lv, and H. Wang, "Efficient clustering-based outlier detection algorithm for dynamic data stream," in *International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 5, 2008, pp. 298–304.
[32] M. Moshtaghi, J. C. Bezdek, T. C. Havens, C. Leckie, S. Karunasekera, S. Rajasegarar, and M. Palaniswami, "Streaming analysis in wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 14, no. 9, pp. 905–921, 2014.
[33] M. Moshtaghi, J. C. Bezdek, C. Leckie, S. Karunasekera, and M. Palaniswami, "Evolving Fuzzy Rules for Anomaly Detection in Data Streams," *IEEE Transactions on Fuzzy Systems*, 2014.
[34] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung, "Enhancing effectiveness of outlier detections for low density patterns," in *Advances in Knowledge Discovery and Data Mining*, 2002, pp. 535–548.
[35] W. Jin, A. K. Tung, J. Han, and W. Wang, "Ranking outliers using symmetric neighborhood relationship," in *Advances in Knowledge Discovery and Data Mining*, 2006, pp. 577–593.
[36] W. Jin, A. K. Tung, and J. Han, "Mining top-n local outliers in large databases," in *SIGKDD*, 2001, pp. 293–298.
[37] J. X. Yu, W. Qian, H. Lu, and A. Zhou, "Finding centric local outliers in categorical/numerical spaces," *Knowledge and Information Systems*, vol. 9, no. 3, pp. 309–338, 2006.
[38] P. Sun and S. Chawla, "On local spatial outliers," in *International Conference on Data Mining*, 2004, pp. 209–216.
[39] X. H. Dang, B. Micenková, I. Assent, and R. T. Ng, "Local Outlier Detection with Interpretation," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 304–320.
[40] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Interpreting and Unifying Outlier Scores," in *SIAM International Conference on Data Mining*, 2011, pp. 13–24.
[41] E. Schubert, R. Wojdanowski, A. Zimek, and H.-P. Kriegel, "On Evaluation of Outlier Rankings and Outlier Scores," in *SIAM International Conference on Data Mining*, 2012, pp. 1047–1058.
[42] E. Schubert, A. Zimek, and H.-P. Kriegel, "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 190–237, 2014.
[43] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *International Conference on Very Large Data Bases*, 2002, pp. 358–369.
[44] C. C. Aggarwal and C. K. Reddy, *Data Clustering: Algorithms and Applications*. CRC Press, 2013.
[45] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 428–436.
[46] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," in *Conference on Information and Knowledge Management*, 2008, pp. 669–678.

Mahsa Salehi received two B.Sc degrees in computer engineering and IT in 2006 and 2008, the M.Sc degree in software engineering in 2009, all from Amirkabir University of Technology, Iran. Since March 2012, she has been a PhD researcher with the University of Melbourne and Machine Learning group at NICTA, Australia. Her research interests include stream data mining, outlier detection and ensembles models.

Christopher Leckie received his PhD in computer science in 1992 from Monash University, Australia. He joined Telstra Research Lab in 1988, where he conducted research into artificial intelligence techniques. In 2000, he joined the University of Melbourne, where he is currently Professor with the Department of Computing and Information Systems. His research interests include using artificial intelligence for sensor networks and intrusion detection, and data mining.



James Bezdek PhD Cornell, 1973, Applied Mathematics. Past president of NAFIPS, IFSA and IEEE CIS. Founding editor of Int'l. Jo. Approximate Reasoning, and IEEE Transactions on Fuzzy Systems. Life fellow in the IEEE and IFSA. IEEE 3rd Millennium, IEEE CIS Fuzzy Systems Pioneer, IEEE Frank Rosenblatt TFA medals; IPMU Kempe de Feret award.

Tharshan Vaithianathan completed his Ph.D. (2001) from University of New South Wales. He spent 5 years at the University College London as a Researcher and before returning to Australia in 2004. He has been leader of biomedical devices and signal processing at NICTA Victoria, and is currently is a Honorary Fellow at the University of Melbourne. His other interests include developing low cost biomedical instrumentation for medical application.

Xuyun Zhang is currently a data analytics researcher at the Machine Learning Group, NICTA, Australia. He received his PhD degree from University of Technology Sydney, and the M.Sc and B.Sc degrees in Computer Science from Nanjing University, China. His research interests include scalable machine learning and data mining, big data, cloud computing, privacy and security. He received the Chinese Government Award for Outstanding Self-Financed Student Abroad in 2012.