Knowledge-Based Systems 204 (2020) 106186

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Scalable KDE-based top-*n* local outlier detection over large-scale data streams

Fang Liu [b], Yanwei Yu [a,*], Peng Song [b], Yangyang Fan [c], Xiangrong Tong [b]

[a] Department of Computer Science and Technology, Ocean University of China, Qingdao, Shandong 266100, China
[b] School of Computer and Control Engineering, Yantai University, Yantai, Shandong 264005, China
[c] Shanghai Key Lab of Advanced High-Temperature Materials and Precision Forming, Shanghai Jiao Tong University, Shanghai 200240, China

ARTICLE INFO

Article history:
Received 9 December 2019
Received in revised form 6 May 2020
Accepted 22 June 2020
Available online 30 June 2020

Keywords:
Local outlier detection
Data streams
Kernel Density Estimation
Upper-bound based pruning

ABSTRACT

The detection of local outliers over high-volume data streams is critical for diverse real-time applications in the real world, where the distributions in different subsets of the data tend to be skewed. However, existing methods are not scalable to large-scale high-volume data streams owing to the high complexity of the re-detection of data updates. In this work, we propose a top-*n* local outlier detection method based on Kernel Density Estimation (KDE) over large-scale high-volume data streams. First, we define a KDE-based Outlier Factor (KOF) to measure the local outlierness score for the data points. Then, we propose the upper bounds of the KOF and an upper-bound-based pruning strategy to quickly eliminate the majority of the inlier points by leveraging the upper bounds without computing the expensive KOF scores. Moreover, we design an Upper-bound pruning-based top-*n* KOF detection method (**UKOF**) over data streams to efficiently address the data updates in a sliding window environment. Furthermore, we propose a Lazy update method of UKOF (**LUKOF**) for bulk updates in high-speed large-scale data streams to further minimize the computation cost. Our comprehensive experimental study demonstrates that the proposed method outperforms the state-of-the-art methods by up to 3,600 times in speed, while achieving the best performance in detecting local outliers over data streams.

1. Introduction

Outlier detection aims to identify the data points that significantly differ from the majority of the points in a data space, which is one of the most important tasks in both the data mining and machine learning areas. Nowadays, diverse high-speed and huge-volume data streams are being generated from different applications (e.g., location-based online services, mobile payments, online shopping systems, and social networks). Outlier analysis over large-scale data streams is critical for real-time applications ranging from fraud detection and network intrusion monitoring to fault detection. For example, a credit card transaction for a large amount passed in a physical location distant from where it typically is used could be a fraud. Credit card companies must continuously monitor the enormous number of card transactions made by different users to discover financial frauds within an actionable time. Therefore, it is critical to be able to efficiently detect outliers from such huge-volume streaming data in a near real-time fashion for multiple domains [1–3].

In recent years, numerous studies have been conducted to detect outliers in datasets [4,5], such as distance-based outlier detection [3,6–8], neighbor-based outlier detection [9,10], distribution-based outlier detection [11,12], and cluster-based outlier detection [13,14]. However, none of these methods can address the outlier detection problem in skewed data; the distributions in real-world datasets tend to be skewed [4]. Hence, the concept of local outlier factor (i.e., LOF) [15] is proposed to detect local outliers by comparing the density difference between a data point and its local neighborhood instead of using a global density [7,16]. Several studies [17–19] have employed LOF to detect local outliers from data streams. In particular, [17] proposes an incremental LOF algorithm that incrementally updates the densities of points affected by newly inserted points to detect local outliers precisely. Because the LOF score of each point is computed based on the local densities of the point, and the density of each point is also based on the reachable distances from the point to its *k* nearest neighbors (*k*NNs), the complexity of *k*NNs recomputation for a large number of points is extremely expensive in terms of time and memory space. To reduce the memory consumption, [18] only stores the information for the clusters of the past points to search the *k*NNs for the future points. However, this method sacrifices outlier detection accuracy to save

* Corresponding author.
E-mail addresses: liufang0812@163.com (F. Liu), yuyanwei@ouc.edu.cn (Y. Yu), pengsongseu@gmail.com (P. Song), yfan.mse@gmail.com (Y. Fan), txr@ytu.edu.cn (X. Tong).

https://doi.org/10.1016/j.knosys.2020.106186
0950-7051/

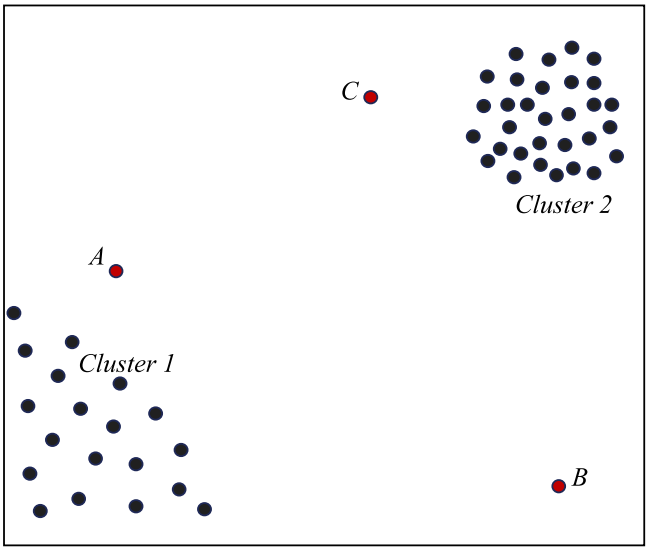


Fig. 1. Example with two clusters of different densities.
Source: From [21].

memory consumption because it clusters past data using the k-means algorithm, which does not preserve the density of the data. Recently, [19] proposes a density summarizing incremental LOF detection method for data streams. It reduces the memory consumption by sampling from the past data while preserving its density. However, the sampling process consumes a significant time resource. Therefore, existing streaming local outlier solutions [17–19] are not scalable to high-volume data streams.

Although LOF applies more weight to the objects with small k-distances in dense regions and less weight to points in sparse regions, [20] determine that LOF and its variants using a heuristic approach are not well supported by theory. [21] also verifies that Kernel Density Estimation (KDE) based on established probability density approximation yields a more robust local density estimation than the LOF method. As in the example from [21] display in Fig. 1, the distance from point *A* to *Cluster* 1 is similar to the k-distances of the points in *Cluster* 1. The LOF method cannot easily distinguish point *A* from *Cluster* 1 using LOF score because its local density is similar to its neighbors' local densities. However the KDE-based method can clearly identify point *A* from *Cluster* 1 by adjusting the bandwidth parameter in the KDE model to obtain a greater outlierness score than those of the points in *Cluster* 1. Several studies [20,22] have been proposed in recent years to utilize the KDE model to detect local outliers in static datasets. Although these KDE methods can be applied to streaming data, they are inefficient because they are not designed to detect local outliers in the context of streaming data.

In this paper, we propose a new local outlier detection method that uses KDE to detect the top-n local outliers for data streams in a sliding window environment. First, we define a KDE-based Outlier Factor (KOF) to measure the degree of outlierness for each data point in the context of streaming data. Secondly, we propose an upper-bound-based pruning strategy to minimize the computation cost for the top-n local outlier detection. Thirdly, we present an efficient top-n KOF detection method leveraging upper-bound-based pruning over the data streams called **UKOF** (Upper-bound pruning-based KOF). Moreover, to further reduce the detection cost, we design an optimized R-tree index for each slide in the sliding window environment, which significantly accelerates both kNNs and reverse k nearest neighbors (RkNNs) searches across the slides for the data points. Furthermore, we propose a Lazy update version of UKOF, named **LUKOF**, for bulk updates in a large sliding window for high-speed large-scale data streams. Finally, extensive experiments are conducted on ten real-world and synthetic datasets. The experiment results confirm that the proposed method is up to 3600 times faster than the state-of-the-art methods, while achieving the best performance in detecting local outliers for data streams.

To summarize, we offer the following contributions:

- We are the first to formally define the problem of top-n local outlier detection based on defined KOF in the context of streaming data.
- We propose the upper bounds of the KOF and an upper-bound-based pruning strategy to efficiently eliminate the inlier (normal) data points without computing expensive KOF scores.
- We present an efficient top-n KOF detection method using upper-bound-based pruning over data streams with an optimized R-tree index.
- We further propose a lazy update version of the top-n KOF detection method for bulk updates in high-speed large-scale data streams to minimize the computation cost.
- Experimental evaluations on ten real-world and synthetic datasets demonstrate that the proposed method outperforms the state-of-the-art methods by a factor of several thousand times in speed, and achieves the best performance in detecting local outliers over data streams.

2. Related work

In this section, we briefly introduce the related work in three areas: LOF and its variants in static datasets, LOF detection in streaming data, and KDE-based local outlier detection in static datasets.

2.1. LOF and its extensions in static datasets

Breunig et al. [15] first proposed the concept of LOF. Compared with distance-based outlier detection [16,23] and KNN outlier detection [24], LOF implements relative density to measure the degree of outlierness based on the density of a data point relative to its local neighbors. The greater the LOF score, the greater the possibility that it is an outlier. Data points with greater LOF scores tend to be considered as outliers. Because outliers are the absolute minority in a dataset, Jin et al. [25] proposed the concept of top-n local outliers, that is, the n points with the greatest LOF scores in a dataset. Varieties of LOF variations [26–29] have been sequentially proposed to detect local outliers in static datasets. Tang et al. [30] proposed a Connectivity-based Outlier Factor (COF) method, which uses the relative connectivity between data points and their kNNs to measure the outlierness degree of the data points. Another variant, Local Distance-based Outlier Factor (LDOF) is proposed in [31]. LDOF uses the mean distance between the data point and its kNNs to define the local density of the data points, and then calculates the relative density based on its kNNs to obtain the outlier factor. [29] proposed a simpler variant, simplified-LOF, which directly uses the k-distance to estimate the densities of the data points, instead of the reachable distance in LOF, to reduce the detection cost; however it has the consequence of yielding less stable results on inliers.

2.2. LOF in data streams

To address continuously evolving streaming data, Pokrajac et al. [17] first proposed an incremental LOF method (iLOF) over data streams. It incrementally updates the LOF scores of the data points. However, all the previous points must be stored to

compute the LOF scores for the new points, which requires a large memory space and high time complexity. Salehi et al. [18] proposed a memory efficient incremental local outlier detection algorithm (MiLOF). To reduce the memory consumption of iLOF, it uses flexible c-means based on k-means [32] to cluster the past data points. However it computes the approximate LOF scores for the data points, which reduces the outlier detection accuracy. Most recently, [19] proposed a Density summarizing Incremental LOF algorithm called DILOF. DILOF reduces the memory consumption by sampling from the past data while preserving the density of this past data. More Specifically, when the number of data points in the current window reaches w, it selects $w/4$ expired data points from the oldest $w/2$ data points to minimize their density difference; thus, it is sensitive to window size and is not suitable for large windows. Moreover, DILOF is inefficient for the detection of local outliers in data streams owing to the high time complexity for the density summarization for the past data.

2.3. KDE for outlier detection in static datasets

Several methods have been proposed to detect local outliers using the KDE model [5,33,34]. Latecki et al. [21] first used KDE to detect local outliers by extending LOF with KDE, named Local Density Factor (LDF). However, LDF does not use the original kernel; rather, it replaces the actual distance between a data point and its neighbors with its local reachable distance in the density estimation, and therefore the desired properties of the KDE are lost. Gao et al. [33,35] proposed a Robust Kernel-based Outlier Factor (RKOF) method, which uses a weighted neighborhood density estimation to improve the robustness of the parameter k. The outlierness score of each data point is the ratio of its local density estimate and the weighted density estimate of its neighbors. Schubert et al. [20] proposed the KDEOS method, which computes the KDE values of each data point using multiple values of parameter k, and uses the average z-score of the KDE values of each point in its local densities to measure its outlierness degree. Tang et al. [22] proposed a Relative Density-based Outlier Score method (RDOS). The method not only uses kNNs, it further considers the reverse nearest neighbors and shared nearest neighbors of the data points in the density estimation. Pavlidou and Zioutas [36] proposed a KDE-based outlier detection method. It first computes a weighted kernel density probability estimation for each data point. Next, the data points having the smallest density values are reported to be outliers. In their follow-up work [37], they proposed two algorithms for outlier detection using KDE in skewed data, which use the Mahalanobis distance to estimate the robust kernel density probability. The first algorithm uses the Mahalanobis distance in the estimation of the kernel density probability; the second uses the minimum covariance determinant to obtain a robust covariance matrix and then incorporates it to obtain a more robust Mahalanobis distance. Recently, [34] proposed an adaptive kernel density based local outlier detection method. Specifically, the method selects relatively large kernel widths in high-density regions to reduce the difference between normal data points, and selects relatively small kernel widths in low-density regions to intensify the abnormality of anomalous data points. However the adaptive setting actually smooths the variance of the KDE-based outlierness scores of normal points and potential local outliers. In summary, none of these methods is designed to address the local outlier detection problem for data streams.

3. Problem definitions

In this section, we first define the key definitions used in the paper and then provide a formal problem definition for KDE-based top-n local outlier detection for data streams. In this work, we use $dist(p, q)$ to denote the Euclidean distance between data points p and q.

Definition 1 (*k-distance*)**.** Given a data point p and any positive integer k, the k-distance of p is the distance from p to its kth nearest data point, denoted as $dist_k(p)$ [15].

Definition 2 (*k Nearest Neighbors, kNNs*)**.** Given a data point p and the k-distance of p, the kNNs of point p contain every point whose distance from p is not greater than the k-distance, denoted as $\mathcal{N}_k(p)$ [15].

Definition 3 (*Reverse k Nearest Neighbors, RkNNs*)**.** Given a data point p, the RkNNs of p contains all points q for which p is among their k nearest neighbor, namely, if $p \in \mathcal{N}_k(q)$, q is a reverse nearest neighbor of p, denoted as $\mathcal{N}_r(p)$ [17].

The kNNs of the kNNs of a point (or a point's kNNs' kNNs) is called the second-order kNNs for the point. More specifically, for each point q in $N_k(p)$, all points in $N_k(q)$ are included in p's second-order kNNs. For simplicity, we denote the second-order kNNs of p as $\mathcal{N}_k(\mathcal{N}_k(p))$.

Similarly, the second-order RkNNs of a point is the RkNNs of the RkNNs of the point. Namely, p's second-order RkNNs contains all points in the RkNNs of each point in $N_r(p)$. We denote the second-order RkNNs of p as $\mathcal{N}_r(\mathcal{N}_r(p))$.

In this paper, we use KDE to estimate the local density. KDE is a non-parametric method to estimate the probability density of a random variable. Following the kernel density estimator, we define the density value for a data point p with respect to its kNNs as follows:

$$KDE(p) = \frac{1}{k} \sum_{q \in \mathcal{N}_k(p)} \frac{1}{h^d} K\left(\frac{dist(p, q)}{h}\right), \tag{1}$$

where h represents the bandwidth, d is the dimension of the data points, and K is the kernel function. Here, we adopt the Gaussian kernel in the KDE model. The density of p is estimated by the average density estimation with respect to all its local neighbors; thus, if p is in a high-density area, it has a large KDE value. Otherwise, it has a relative small KDE value.

In skewed data, the distribution of points in different subsets is different; hence, it is unreasonable to use a fixed bandwidth for all points. Inspired by [21], we use a different bandwidth h_p for each point p: $h_p = h * dist_k(p)$, where h is a fixed bandwidth. That is, we determine an adaptive bandwidth for each data point according to its k-distance. If a data point is in a dense area, its k-distance is relatively small, thus its bandwidth is small, which makes its kNNs give a relatively large density estimation to it. Conversely, if a data point is in a sparse area, its k-distance is relatively large, thus its bandwidth is relatively large, meaning that its kNNs also provide a density contribution but relative less. As indicated in Fig. 2, p_1 is located in a high-density area; thus, we use a small bandwidth for p_1 such that its nearest neighbors can provide a larger density contribution. As p_2 is in a low-density area, it receives a large bandwidth, which results in it obtaining a relative low density. In this way, the adaptive bandwidth based on k-distance effectively differentiates the local density estimation for data points in dense and sparse areas.

Hence, the KDE model used in our paper becomes:

$$KDE(p) = \frac{1}{k} \sum_{q \in \mathcal{N}_k(p)} \frac{1}{(2\pi)^{\frac{d}{2}} * (h_p)^d} e^{-\frac{dist(p,q)^2}{2(h_p)^2}} \tag{2}$$

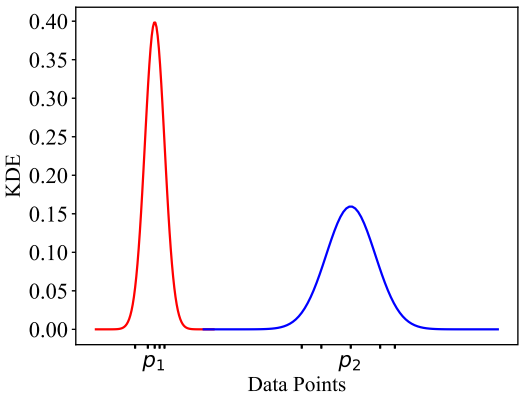


Fig. 2. Example for KDE of two data points with different bandwidths.

Definition 4 (*KDE-based Outlier Factor, KOF*)**.** The KOF of data point p is defined as:

$$KOF(p) = \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE(q)}{KDE(p)} \tag{3}$$

Similar to LOF [15], KOF is the mean ratio between the KDE values of kNNs of each data point and its KDE value. When $KOF(p)$ is closer to 1, the density of p is closer to the density of its kNNs. As discussed in [15], for the points inside a dense cluster, the LOFs of the points is bounded, that is, the LOFs of the data points in a dense cluster are close to 1. Similarly, the KOFs of the data points in the dense areas also tends to 1 in our model. When $KOF(p)$ is greater than 1, the density of p is smaller than its kNNs, that is, point p is located at the edge of a high-density region or in a sparse region. Therefore, the higher the score of $KOF(p)$, the more likely p is a local outlier.

To capture the KDE-based local outliers over data streams, we use the periodic sliding window semantics as proposed by [38] to define the sub-stream of the infinite data stream. There are two types of sliding windows: a count-based window and time-based window. In both window types, each sliding window has a window size w and a slide size s. For time-based sliding windows, each window W has a starting time $W.T_{start}$ and an ending time $W.T_{end} = W.T_{start} + w$. Periodic sliding of the current window W_c causes an increase of $W_c.T_{start}$ and $W_c.T_{end}$ with s. For count-based sliding windows, the window always maintains w data points. Periodically, the current window W_c slides s new data points. We model streaming data with different scales by setting different window sizes w and slide sizes s. The greater the window size w, the greater the scale of the streaming data. The greater the slide size s, the faster the streaming data is updated. In this paper, we present the proposed method using a count-based sliding window; however, time-based sliding windows apply to the proposed method as well.

Now, we define our problem as follows:

Problem (*KDE-based Top-n Local Outlier Detection Over Data Streams*)**.** Given a data stream DS, sliding window W, number of nearest neighbors k, and number of outliers n, KDE-based top-n local outlier detection outputs the top n data points with the highest KOF scores in each window over data stream DS.

4. KDE-based top-*n* local outlier detection

In this section, we first introduce how to use KOF to detect the top-n local outliers. Then, we present the upper bounds of KOF used in this paper and how to leverage the upper bounds to prune normal points. Finally, we introduce the proposed method for top-n local outlier detection over data streams.

4.1. Using KOF to detect top-n outliers

As defined in our problem, we aim to detect the top-n data points with the highest KOF scores, namely the top-n local outliers. A traditional method is to calculate the KOF scores of all points and sort these KOF scores in descending order. Then, the first n points are the top-n local outliers. However, we only focus on quickly detecting the n points with the highest KOF scores, thus it is not necessary to rank all the KOF scores of all the data points. Therefore, the cutoff threshold-based method can be used to select top-n local outliers.

Given a dataset D, n data points are first randomly selected from D and their KOF scores are calculated. Then the smallest KOF score is selected as the cutoff threshold, denoted as ct. Next, for any point p in the remaining dataset, if $KOF(p) < ct$, p cannot be a top-n local outlier; hence, p can be directly pruned. If $KOF(p) > ct$, p could be a top-n local outlier, and therefore, p is placed into the top-n outlier candidate list, the point with the smallest KOF score in the list is deleted, and ct is updated. The top-n local outliers can be obtained by traversing the entire dataset.

However, by Definition 4, the computation complexity of the KOF score for all data points is extremely high. In our problem, we only wish to detect the top-n local outliers; hence, we are not required to calculate the KOF scores for all the data points. To minimize the computation cost, we next present the proposed pruning strategy, which uses the upper bounds of the KOF with less time complexity to quickly eliminate the normal data points.

4.2. Pruning strategy based on upper bound

4.2.1. Upper and lower bound of KDE

According to Eqs. (1) and (2), we first set the upper and lower bounds of KDE with a lower computation complexity as follows:

Lemma 1 (*Upper Bound of KDE*)**.** *Given a data point p, it holds that $KDE(p) \le KDE_{max}(p) = \frac{1}{(h_p)^d}K(\frac{dist(p,o)}{h_p})$, where o is the nearest neighbor of p. $KDE_{max}(p)$ is called the upper bound of the KDE value for p.*

Proof. $\forall q \in \mathcal{N}_k(p)$, $dist(p,q) \ge dist(p,o)$, therefore, by Eq. (2), we have $KDE(p) = \frac{1}{k}\sum_{q\in\mathcal{N}_k(p)}\frac{1}{(h_p)^d}K(\frac{dist(p,q)}{h_p}) \le \frac{1}{k}\frac{k}{(h_p)^d}K(\frac{dist(p,o)}{h_p})$ $= \frac{1}{(h_p)^d}K(\frac{dist(p,o)}{h_p}) = KDE_{max}(p)$.

Therefore, $KDE(p) \le KDE_{max}(p)$ holds.

Lemma 2 (*Lower Bound of KDE*)**.** *Given a data point p, it holds that $KDE(p) \ge KDE_{min}(p) = \frac{1}{(h_p)^d}K(\frac{dist_k(p)}{h_p})$. $KDE_{min}(p)$ is called the lower bound of the KDE value for p.*

Proof. This proof is similar to Lemma 1. $\forall q \in \mathcal{N}_k(p)$, $dist(p,q) \le dist_k(p)$. Hence, $KDE(p) = \frac{1}{k}\sum_{q\in\mathcal{N}_k(p)}\frac{1}{(h_p)^d}K(\frac{dist(p,q)}{h_p}) \ge \frac{1}{k} * k * \frac{1}{(h_p)^d}K(\frac{dist_k(p)}{h_p}) = \frac{1}{(h_p)^d}K(\frac{dist_k(p)}{h_p}) = KDE_{min}(p)$.

Therefore, $KDE(p) \ge KDE_{min}(p)$ holds.

Based on the upper and lower bounds of KDE, we next propose the upper bounds of the KOF for the data points.

4.2.2. Upper bounds of KOF

Lemma 3 (*General Upper Bound of KOF*)**.** *Given a data point p, it holds that $KOF(p) \le UB(p) = \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)}{KDE_{min}(p)}$. $UB(p)$ is called the general upper bound of KOF for p.*

Proof. Because $KOF(p) = \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE(q)}{KDE(p)}$, and we also know $KDE(q) \le KDE_{max}(q)$ by Lemma 1 and $KDE(p) \ge KDE_{min}(p)$ by Lemma 2, $KOF(p) \le \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)}{KDE(p)} \le \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)}{KDE_{min}(p)}$. Therefore, $KOF(p) \le UB(p)$ holds.

Clearly, the time complexity of the general upper bound is less than that of the KOF score because we only involve the nearest neighbor and kth nearest neighbor in $UB(p)$. The KDE_{max} and KDE_{min} of p can be immediately obtained as long as the kNNs of p are found; thus the complexity is the cost of the kNNs search in the current window (i.e., $O(wlogk)$ without any index usage). Therefore, the complexity of $UB(p)$ is $O(kwlogk)$, whereas the $KOF(p)$'s complexity includes the kNNs search and KDE calculation, namely $O(kwlogk+k^2)$. To reduce the computation cost, we can use the upper bound of KOF to detect the top-n local outliers. That is, to determine whether a point p is a top-n local outlier, we first calculate the upper bound of p. If $UB(p) < ct$, it means that p cannot be a top-n local outlier, then p can be directly pruned. If $UB(p) > ct$, we must further validate the status of p by computing its KOF score.

To prune more inlier points using the KOF upper bound, we next propose two tighter upper bounds for the KOF based on the previously obtained KDE values.

Lemma 4 (*Tighter Upper Bounds of KOF*)**.** *Given a data point p, and data points $Q \subseteq \mathcal{N}_k(p)$ and their KDE values, it holds that $KOF(p) \le UB_{T_1}(p) = \frac{\sum_{q\in Q} KDE(q)+\sum_{q\in(\mathcal{N}_k(p)-Q)} KDE_{max}(q)}{k*KDE_{min}(p)} \le UB(p)$. If the KDE value of p has been previously obtained, it holds that $KOF(p) \le UB_{T_2}(p) = \frac{\sum_{q\in Q} KDE(q)+\sum_{q\in(\mathcal{N}_k(p)-Q)} KDE_{max}(q)}{k*KDE(p)} \le UB(p)$.*

Proof. We know that $KDE(q) \le KDE_{max}(q)$ by Lemma 1; therefore $\sum_{q\in\mathcal{N}_k(p)} KDE(q) \le \sum_{q\in Q} KDE(q) + \sum_{q\in(\mathcal{N}_k(p)-Q)} KDE_{max}(q) \le \sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)$. Because $UB(p) = \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)}{KDE_{min}(p)}$, and we also know $KDE(p) \ge KDE_{min}(p)$ by Lemma 2, then $KOF(p) \le \frac{\sum_{q\in Q} KDE(q)+\sum_{q\in(\mathcal{N}_k(p)-Q)} KDE_{max}(q)}{k*KDE_{min}(p)} \le UB(p)$ holds. Namely, $KOF(p) \le UB_{T_1}(p) \le UB(p)$. Similarly, $KOF(p) \le UB_{T_2}(p) \le UB(p)$ holds.

For simplicity, we use $UB(p)$ to refer to all upper bounds of the KOF score for data point p below.

4.3. Local outlier detection over data streams

The data points in data streams are constantly updated; that is, new points continuously arrive and old points constantly expire. By Definition 3, we know that the kNNs of the RkNNs of a point o must contain o, thus the kNNs of $\mathcal{N}_r(o)$ are changed if o is inserted or deleted. Therefore, the newly arrived and expired points can influence the KDE values of their RkNNs, which further affects the KOF scores of their RkNNs. Furthermore, by Definition 4, to calculate the KOF score of a point, we must calculate the KDE values of its kNNs. Therefore, if its second-order kNNs contain new and/or expired points, its KOF score is also changed. That is, the KOF scores of the second-order RkNNs of the newly arrived and expired points are changed.

The effect of the new and expired points on the upper bound and KOF score is discussed below.

4.3.1. Effect of new point

The insertion of a new data point o influences the k-distances of the data points that have o in their kNNs, i.e., the RkNNs of o, the k-distances of these points must be updated. We denote the new k-distance as $dist_k^{new}(p)$ for data point p:

$$dist_k^{new}(p) = \begin{cases} dist(p,o), & o \text{ is the } k\text{th}NN \text{ of } p, \\ dist_{k-1}(p), & otherwise \end{cases}$$

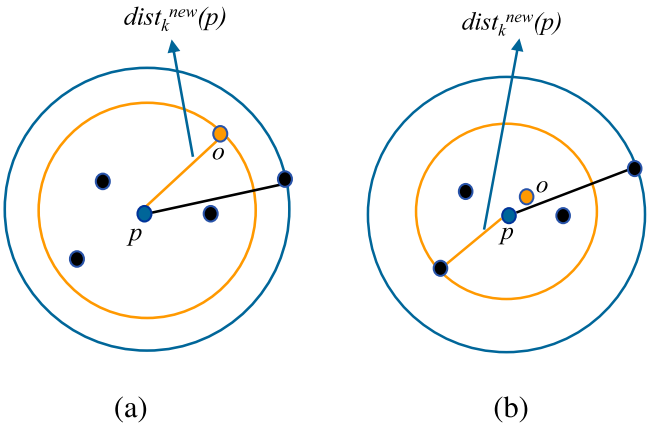


Fig. 3. Updated k-distance of point p w.r.t new point o.

As indicated in Fig. 3(a), if the newly added point o is the new kth nearest neighbor of p, namely, $dist(p,o) < dist_k(p)$ and $dist(p,o) > dist_{k-1}(p)$, the new k-distance of p is $dist(p,o)$. Otherwise, if the distance between p and o is less than its old $(k-1)$th distance, the updated k-distance of p is its old $(k-1)$th distance (Fig. 3(b)).

As discussed above, when a new point o arrives, we must update the k-distance of the RkNNs of o, namely, the kNNs of the RkNNs of o change. The KDE values of the RkNNs of o can increase; thus the upper bounds and KOF scores of o's RkNNs and second-order RkNNs could be updated.

More specifically, if $p \in \mathcal{N}_r(o)$, only o is the nearest neighbor of p, the distance between p and its nearest neighbor decreases compared to the previous value, namely $KDE_{max}(p)$ increases. Otherwise, $KDE_{max}(p)$ remains unchanged. For $p \in \mathcal{N}_r(o)$, because the k-distance of p decreases compared to the previous value, both $KDE_{min}(p)$ and $KDE(p)$ increase. For $p \in (\mathcal{N}_r(\mathcal{N}_r(o)) - \mathcal{N}_r(o))$, because the kNNs of p does not change, $KDE_{min}(p)$ and $KDE(p)$ remain unchanged. Based on this analysis, we can offer the following Lemmas:

Lemma 5. *Given a new point o, for point $p \in \mathcal{N}_r(o)$, if $\forall q \in \mathcal{N}_k(p)$, o is not the nearest neighbor of q, then the KOF upper bound of p need not be updated.*

Proof. because $p \in \mathcal{N}_r(o)$, $KDE_{min}(p)$ and $KDE(p)$ increase. If $\forall q \in \mathcal{N}_k(p)$, o is not the nearest neighbor of q, then $KDE_{max}(q)$ remains unchanged. Thus $UB(p) = \frac{\frac{1}{k}\sum_{q\in\mathcal{N}_k(p)} KDE_{max}(q)}{KDE_{min}(p)}$ only decreases. Therefore, the KOF upper bound of p need not be updated.

Lemma 6. *Given a new point o, for point $p \in \mathcal{N}_r(o) \cup \mathcal{N}_r(\mathcal{N}_r(o))$, if $\exists q \in \mathcal{N}_k(p)$, o is the nearest neighbor of q, then the KOF upper bound of p must be updated.*

Proof. For $p \in \mathcal{N}_r(o)$, $KDE_{min}(p)$ increases. If $\exists q \in \mathcal{N}_k(p)$ such that o is the nearest neighbor of q, then $KDE_{max}(q)$ increases. That is, the denominator and numerator of $UB(p)$ both increase at the same time; hence, we cannot determine whether $UB(p)$ increases or decreases. Therefore, it is necessary to update $UB(p)$.

For $p \in (\mathcal{N}_r(\mathcal{N}_r(o)) - \mathcal{N}_r(o))$, $KDE_{min}(p)$ remains unchanged. Then, if $\exists q \in \mathcal{N}_k(p)$, $KDE_{max}(q)$ increases, then $UB(p)$ increases. p may change from a normal point to an outlier; therefore, $UB(p)$ must also be updated.

Next, we can easily derive Lemma 7 as follows:

Lemma 7. *Given a new point o, for point $p \notin \mathcal{N}_r(o) \cup \mathcal{N}_r(\mathcal{N}_r(o))$, the upper bound and the KOF score of p need not be updated with respect to o.*

Proof. Because $p \notin \mathcal{N}_r(o) \cup \mathcal{N}_r(\mathcal{N}_r(o))$, the kNNs of p remains unchanged, namely $KDE(p)$ remains the same. For $q \in \mathcal{N}_k(p)$, because $p \notin \mathcal{N}_r(o) \cup \mathcal{N}_r(\mathcal{N}_r(o))$, $q \notin \mathcal{N}_r(o)$. This is because that if $\exists q \in \mathcal{N}_k(p)$ such that $q \in \mathcal{N}_r(o)$, then $p \in \mathcal{N}_r(q)$, i.e., $p \in \mathcal{N}_r(\mathcal{N}_r(o))$. This contradicts with the known conditions. Because $q \notin \mathcal{N}_r(o)$, the kNNs of q also remains unchanged and $KDE(q)$ remains the same. Therefore, the KOF score and its upper bound of p remain the same.

4.3.2. Effect of expired point

When a point o expires, it also causes its RkNNs to change their kNNs, which further influences the upper bounds and KOF scores of the o's RkNNs and second-order RkNNs.

As in Section 4.3.1, only when o is the nearest neighbor of p, does $KDE_{max}(p)$ decrease after removing o; otherwise, $KDE_{max}(p)$ remains unchanged. Moreover, for $p \in \mathcal{N}_r(o)$, because the k-distance of p increases, the $KDE_{min}(p)$ and $KDE(p)$ both decrease. For $p \in (\mathcal{N}_r(\mathcal{N}_r(o)) - \mathcal{N}_r(o))$, $KDE_{min}(p)$ and $KDE(p)$ remain the same.

Lemma 8. *Given an expired point o, for point $p \notin \mathcal{N}_r(o)$, the upper bound and KOF score of p need not be updated with respect to o.*

Proof. Because $p \notin \mathcal{N}_r(o)$, the kNNs of p remain unchanged; hence, $KDE(p)$ remains the same.

For $q \in \mathcal{N}_k(p)$, if $q \in \mathcal{N}_r(o)$, after removing o, then $KDE_{max}(q)$ remains unchanged or decreases, and $KDE_{min}(q)$ and $KDE(q)$ decrease. Otherwise, the kNNs of q and $KDE(q)$ remain the same.

By Eq. (3), Lemmas 3 and 4, the upper bound and KOF score of p either decrease or remain the same. Therefore, we need not update the upper bound and KOF score for p.

We next offer Corollary 1 by Lemma 8.

Corollary 1. *Given an expired point o, for point $p \in \mathcal{N}_r(o)$, the upper bound and KOF score of p must be updated.*

Because $p \in \mathcal{N}_r(o)$, $KDE_{min}(p)$ and $KDE(p)$ decrease. For $q \in \mathcal{N}_k(p)$, $KDE_{max}(q)$ and $KDE(q)$ either decrease or remain unchanged after removing o. Therefore, we cannot determine whether $UB(p)$ and $KOF(p)$ increase or decrease, thus it is necessary to update the upper bound and KOF score for p.

Finally, according to Lemmas 5–8 and Corollary 1, we offer Theorem 1, which indicates what points should have their upper bounds and KOF scores updated with respect to the new and expired points.

Theorem 1. *Given a new point o_{new} and an expired point o_{exp}, for point $p \in \mathcal{N}_r(o_{new}) \cup \mathcal{N}_r(\mathcal{N}_r(o_{new}))$, if $\exists q \in \mathcal{N}_k(p)$, o_{new} is the nearest neighbor of q, the KOF upper bound of p must be updated. For point $p \in \mathcal{N}_r(o_{exp})$, the upper bound and KOF score of p must be updated.*

4.3.3. Upper-bound pruning-based local outlier detection

In this section, we present UKOF method for local outlier detection over data streams in a sliding window environment. The pseudo-code of UKOF is displayed as Algorithm 1.

First, we build an R-tree index for the new slide S_{new} (line 1). Then, for each new point o in the new slide, we first quickly obtain o's kNNs, RkNNs, and second-order RkNNs using the proposed optimized R-tree index (line 3). By Lemma 6 and Theorem 1, for $p \in \mathcal{N}_r(o) \cup \mathcal{N}_r(\mathcal{N}_r(o))$, only if $\exists q \in \mathcal{N}_k(p)$, and o is the nearest neighbor of q, does the KOF upper bound of p require be updating. Therefore, as indicated in lines 5–18, we first update

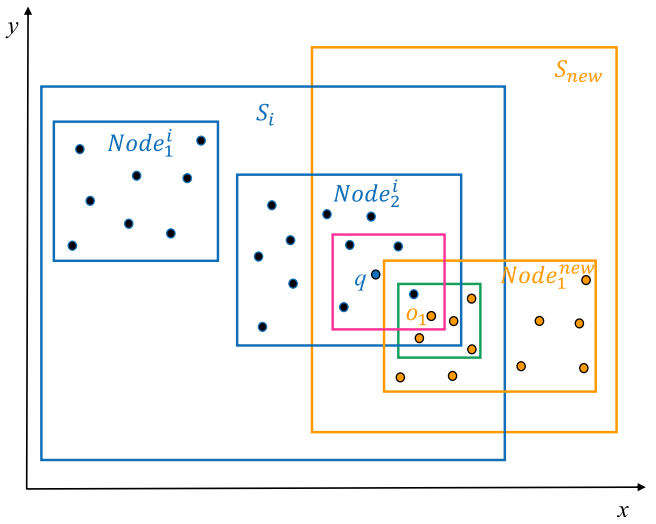


Fig. 4. Range query of kNNs and RkNNs: each rectangle indicates a data range of a node, each small circle denotes a data point, magenta rectangle is the data range of $\mathcal{N}_k(q)$, and green rectangle is the data range of $\mathcal{N}_k(o_1)$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$UB(p)$. If $UB(p) < ct$, we directly prune p. Otherwise, we further calculate the KOF score for p. If $KOF(p)$ remains not less than ct, then p could be a top-n outlier. We then update the top-n local outlier list Top_n using p as indicated in line 13. $Top_n.replace(p)$ denotes updating the Top_n list with respect to p, that is, if p is already in Top_n, we only update p's KOF score and ct. Otherwise, we use p to replace the point with the lowest KOF score in Top_n and update ct.

Secondly, for each expired point o in the expired slide, we first update the RkNNs of o (lines 21–22). Next, by Corollary 1, only when $p \in \mathcal{N}_r(o)$, p is updated. Similarly, we first use the updated KOF upper bound to prune p. If p cannot be pruned according to ct, we further compute its KOF score. If $KOF(p) < ct$, we eliminate point p; otherwise, p is a top-n local outlier and thus we update the Top_n list and ct using the method described above (lines 23–33). Finally, UKOF outputs the top-n local outliers with the highest KOF scores (line 35).

To accelerate the range query for all the data points in the proposed method, we implement an optimized R-tree index for kNNs and RkNNs search across the slides for data points. We first build an R-tree index for each new slide in the sliding window. Accordingly, we obtain an R-tree data range for each slide.

For the kNNs search of a data point, we first search its kNNs in S_{new} and calculate the data range of its kNNs. We then compare the data range of its kNNs with the other slides in the current window. We only traverse the slides with overlapping range to determine the final kNNs of the point across the slides in the current window. More specifically, in other R-trees, we only search the nodes that have an overlap with the data range of its kNNs in S_{new}. For example, for the kNNs search of point o_1 in Fig. 4, we first find the data range of $\mathcal{N}_k(o_1)$ in S_{new}, i.e., the green area. There exists a slide S_i overlap with the green area in the data space; thus we continue to search the kNNs in the R-tree of slide S_i for point o_1.

If $o \in \mathcal{N}_k(q)$, then $q \in \mathcal{N}_r(o)$. Namely, if a data point o falls in the data range of data point q's kNNs, then q could be an RkNN of o. Therefore, for the RkNNs search of a new data point, we traverse the nodes in each R-tree that has an overlap range with the new slide. If the point does not fall within the maximum data range of a node in the R-tree, there is no RkNNs in the node for the

Algorithm 1 UKOF algorithm

Input: Data Stream DS, sliding window W, new slide S_{new}, expired slide S_{exp}, k, n;
Output: top-n KOF outliers

```
1: createRtree(S_new);
2: for each o ∈ S_new do
3:     get N_k(o); get N_r(o); get N_r(N_r(o));
4:     for each p ∈ (N_r(o) ∪ N_r(N_r(o)) ∪ {o}) do
5:         for each q ∈ N_k(p) do
6:             if o is the nearest neighbor of q then
7:                 update UB(p);
8:                 if UB(p) < ct then
9:                     prune(p);
10:                else if KOF(p) < ct then
11:                    prune(p);
12:                else
13:                    Top_n.replace(p);
14:                    update ct;
15:                end if
16:                break;
17:            end if
18:        end for
19:    end for
20: end for
21: for each o ∈ S_exp do
22:    update N_r(o);
23:    for each p ∈ N_r(o) do
24:        update UB(p);
25:        if UB(p) < ct then
26:            prune(p);
27:        else if KOF(p) < ct then
28:            prune(p);
29:        else
30:            Top_n.replace(p);
31:            update ct;
32:        end if
33:    end for
34: end for
35: return Top_n;
```

point; therefore, it is not necessary to traverse all the child nodes of the node. In this manner, we can quickly find the minimal nodes that the point falls into, where the potential RkNNs can be searched for the new point. For example, in Fig. 4, for the new point o_1, we first search its RkNNs in S_{new}. If o_1 falls into the data range of kNNs of another point q, q could be an RkNN of o_1. Next, we continue to search o_1's RkNNs in S_i overlapping with S_{new} by traversing the R-tree of S_i.

In fact, we search the kNNs and RkNNs for new points simultaneously by sharing the same traverse in the R-trees, which significantly reduces the search time.

4.3.4. Lazy update of UKOF for bulk updates in large data streams

The new/expired points in the same slide can have common RkNNs and second-order RkNNs. If a data point arrives/expires, we immediately update the upper bounds or KOF scores of its RkNNs and second-order RkNNs; these points can be updated multiple times, which are redundant computations. Therefore, we propose a Lazy update version of the UKOF method, called LUKOF, for bulk updates in large-scale data streams. That is, we no longer update the new/expired data one by one; rather, we update the data in one piece. The pseudo-code of LUKOF is presented in Algorithm 2. Again, we first build an R-tree index for the new slide.

Then, for expired points in the expired slide, we first update their RkNNs, and place their RkNNs together into a set Set_{up_1}. Because the RkNNs of the expired points can contain expired points, we remove these expired points from Set_{up_1} (lines 3–7). For new points in the new slide, we first quickly find their kNNs, RkNNs, and second-order RkNN using the optimized R-tree index, and place their RkNNs and second-order RkNNs into a set set_{up_2}. The expired and new points can have common RkNNs. To update the upper bounds or KOF scores of these common points only once, we remove these common points from Set_{up_2} (lines 8–12).

Next, similar to Algorithm 1, we update the upper bounds and KOF scores for the points in Set_{up_1} according to the processing method of expired points (lines 13–23), and then process the points in Set_{up_2} according to that of the new points (lines 24–39).

Algorithm 2 LUKOF algorithm

Input: Data Stream S, sliding window W, new slide S_{new}, expired slide S_{exp}, k, n;
Output: top-n KOF outliers

```
1: createRtree(S_new);
2: Set_up1 ← ∅; Set_up2 ← ∅;
3: for each o ∈ S_exp do
4:     update N_r(o);
5:     Set_up1 ← Set_up1 ∪ N_r(o);
6: end for
7: Set_up1 ← Set_up1 − S_exp;
8: for each o ∈ S_new do
9:     get N_k(o); get N_r(o); get N_r(N_r(o));
10:    Set_up2 ← Set_up2 ∪ (N_r(o) ∪ N_r(N_r(o)) ∪ {o}) − S_exp;
11: end for
12: Set_up2 ← Set_up2 − Set_up1;
13: for each p ∈ Set_up1 do
14:    update UB(p);
15:    if UB(p) < ct then
16:        prune(p);
17:    else if KOF(p) < ct then
18:        prune(p);
19:    else
20:        Top_n.replace(p);
21:        update ct;
22:    end if
23: end for
24: for each p ∈ Set_up2 do
25:    for each q ∈ N_k(p) do
26:        if ∃o ∈ S_new is the nearest neighbor of q then
27:            update UB(p);
28:            if UB(p) < ct then
29:                prune(p);
30:            else if KOF(p) < ct then
31:                prune(p);
32:            else
33:                Top_n.replace(p);
34:                update ct;
35:            end if
36:            break;
37:        end if
38:    end for
39: end for
40: return Top_n;
```

At last, we use an example to further illustrate the difference between the proposed UKOF and LUKOF methods. As shown in Fig. 5, the current sliding window contains three slides (i.e., S_1, S_2 and S_3), where S_1 is the expired slide and S_3 is the new slide.

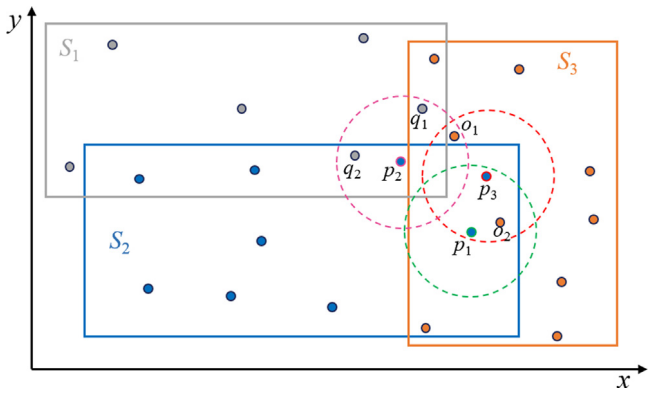


Fig. 5. Example for the proposed UKOF and LUKOF methods.

The points in each circle belongs to $\mathcal{N}_k(p_i)$, where p_i is the center point. For simplicity, we only discuss partial selected expired points q_1, q_2 and new points o_1, o_2. From the example, we know that $\mathcal{N}_r(q_1) = \{p_2\}, \mathcal{N}_r(q_2) = \{p_2\}, \mathcal{N}_r(o_1) = \{p_2, p_3\}$, and $\mathcal{N}_r(o_2) = \{p_1, p_3\}$. According to Theorem 1, we only focus on the data points in S_2 and S_3 that fall into $\mathcal{N}_r(o_{new}) \cup \mathcal{N}_r(\mathcal{N}_r(o_{new}))$ and $\mathcal{N}_r(o_{exp})$.

For UKOF, when point q_1 expires, its RkNNs p_2 needs to be updated. And when q_2 expires, the upper bound/KOF score of p_2 also need to be updated. Similarly, when o_1 and o_2 arrive, p_3 needs to be updated twice, p_1 and p_2 are updated once. In addition, the second-order RkNNs of o_1 and o_2, i.e., $\mathcal{N}_r(\mathcal{N}_r(o_1)) \cup \mathcal{N}_r(\mathcal{N}_r(o_2))$, may be updated multiple times. Therefore, the expired points q_1, q_2 and the new points o_1, o_2 make the upper bound/KOF score of p_2 being updated three times and p_3 twice in total, which causes redundant calculations.

To reduce redundant calculations, LUKOF uses a delayed update method for points that need to be updated. That is, we first find the RkNNs that need to be updated for q_1 and q_1 and put them into set S_1, namely $Set_1 = \{p_2\}$, then we put the RkNNs and second-order RkNNs of o_1 and o_2 into set Set_2, and delete the points shared by Set_1 and Set_2 from Set_2, namely $Set_2 = \{p_1, p_3, \dots\}$. Finally, we uniformly update the upper bounds and KOF scores of the points in Set_1 and Set_2, which makes all points that need to be updated only updated once.

4.3.5. Time complexity analysis

In this section, we analyze the time complexity of the proposed method. Given a window size w, the time complexity of the kNNs search for each point is $O(wlogk)$ without any index usage. For KDE computation, the complexity is $O(wlogk + k)$.

To compute the KOF score of point p, we must first find p's kNNs, and then calculate the KDE values of its kNNs. Therefore, the complexity of computing $KOF(p)$ is $O(kwlogk + k^2)$. For $UB(p)$, we obtain the kNNs of p to calculate $KDE_{max}(p)$ and $KDE_{min}(p)$ with time complexity $O(wlogk)$; thus the complexity of computing $UB(p)$ is $O(kwlogk)$. By Theorem 1, we are only required to update the upper bounds and KOF scores for the partial RkNNs and second-order RkNNs of the newly added points and/or expired points. We assume the average number of RkNNs for each point is ϕ, and the average number of second-order RkNNs per point is ψ. Because the nearest neighbors for a point share the most kNNs in the local areas, $\phi < \psi \ll \phi^2$. Therefore, the total time complexity of UKOF is $O(s(\psi + \phi)(kwlogk + k^2))$ in the worst case with respect to a slide size s.

However, we use the optimized R-tree to search the kNNs and RkNNs for each point; hence, the time complexity of the kNNs search for the proposed method is considerably less than $O(wlogk)$. The worst-case time complexity of the kNNs search becomes $O(\frac{w}{s}(s + log_M s))$ with the optimized R-tree index in the proposed method, where M is the maximum number of entries in each node. Furthermore, the proposed method first uses the upper bound of KOF with lower time complexity to detect the outlierness degree of each point. Moreover, according to Lemmas 5–8 and Corollary 1, certain RkNNs and second-order RkNNs of the new and expired points need not be updated, thus the number of points that must be updated is less than $s(\psi + \phi)$. Therefore, UKOF has a lower time complexity than $O(s(\psi + \phi)(k\frac{w}{s}(s + log_M s) + k^2))$ in reality.

LUKOF updates the affected points in one action for all newly arrived and expired points. We assume the number of all points require update is $\alpha * w$ when a window slides, where $0 \leq \alpha \leq 1$. Because the newly arrived and expired points at the same time point typically share numerous common RkNNs and second-order RkNNs, $\alpha * w < s(\psi + \phi)$. Hence, LUKOF has a lower time complexity compared to $O(\alpha w(k\frac{w}{s}(s + log_M s) + k^2))$ in the worst case.

5. Experiment

In this section, we evaluate the effectiveness and efficiency of the proposed method compared to the state-of-the-art local outlier detection approaches. All experiments are conducted on a computer with a 4.20 GHz i7-7700k processor, 8 GB memory, and the Windows 7 operating system. The source code of the proposed method is available at GitHub.[1]

5.1. Data description

Datasets. We use the ten real-world and synthetic datasets to evaluate the proposed method and baselines. The detailed information of ten datasets is displaced in Table 1.

- Vowels [39–41]: The original Japanese Vowels dataset from the UCI machine learning repository is multivariate time series data, where nine male speakers uttered two Japanese vowels /ae/ successively. Each utterance by a speaker forms a time series whose length is in the range 7–29, and each point of a time series includes 12 features. This is a classification dataset to classify the speakers. For outlier detection, each frame in the training data is considered as an individual data point, whereas the UCI repository considers a block of frames (utterance) as an individual point. In this case, class (speaker) 1 is down-sampled to 50 outliers. The inliers contained classes 6, 7 and 8. The other classes are discarded.
- KDDCup [42]: The KDDCup 1999 dataset from UCI Machine Learning Repository is a network intrusion dataset wildly used in outlier detection studies. This dataset contains 41 attributes (34 continuous, and 7 categorical). The original dataset has 3,925,651 attacks (80.1%) out of 4,898,431 records. A smaller set is forged with only 3377 attacks (0.35%) of 976,157 records, where the attribute 'logged_in' is positive. Using the 'service' attribute, the data is divided into {http, smtp, ftp, ftp_data, and others} subsets. From this forged dataset, the 623,091 'http' service data is used to construct the KDDCup dataset, which contains 249,236 attacks (0.4%).
- Subhttp and Smtp [19,43–45]: These two datasets are extracted from the KDDCup 1999 dataset. Following [19], KDDCup 1999 dataset are reduced to 4 attributes (service, duration, src_bytes, and dst_bytes) as these attributes are regarded as the most basic attributes, where only 'service' is categorical. From the forged dataset, 95,156 'smtp' service

[1] https://github.com/LiuFang0812/TopNKOF.

Table 1
Experimental datasets.

Dataset	#Points	#Dim	Size	Outlier	Ratio (%)
Vowels	1456	12	645 k	Class1	3.4%
KDDCup	623,091	41	708M	Class0	0.4%
Subhttp	4079	3	773 k	Class0	2%
Smtp	95,156	3	17M	Class0	0.03%
ForestCover	286,048	10	28.3M	Class4	0.9%
Mobike	1,082,732	2	43.1M	×	×
GeoLife	11,065,399	2	851M	×	×
Interchanging RBF	200,000	2	5.6M	Class15	0.5%
Moving Squares	200,000	2	4.2M	Class4	0.5%
Mixture RBF	200,000	2	5.1M	Class10	0.5%

data is used to construct the Smtp dataset, and sampled 4079 'http' service data is used to construct the Subhttp dataset.

- ForestCover [46,47]: This dataset includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, such that the existing forest cover types are more a result of ecological processes rather than forest management practices. This dataset has 54 attributes (10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables). Here, the outlier detection dataset is created using only 10 quantitative attributes. Instances from class 2 are considered as normal points and instances from class 4 are anomalies. The anomalies ratio is 0.9%. Instances from the other classes are omitted.
- Mobike: The real-time GPS location data of all mobikes within the six-ringed road in Beijing is crawled on the Mobike platform.[2] This dataset includes 1,082,732 locations of up to 50,000 mobikes on one day.
- GeoLife [48]: This dataset is extracted from the GeoLife project of Microsoft Research Asia. It collects the GPS trajectories of 182 users in a period of over three years (from April 2007 to August 2012). Each GPS point is composed of the longitude, latitude, and a timestamp.

In addition to seven real-world datasets, we also generate three datasets to empirically validate our proposed UKOF and LUKOF.

- Interchanging RBF [49,50]: Fifteen Gaussians with random covariance matrices are replacing each other every 3000 samples. Thereby, the number of Gaussians switching their position increases each time by one until all are simultaneously changing their location. This allows to evaluate an algorithm in the context of abrupt drift with increasing strength.
- Moving Squares [49,50]: Four equidistantly separated, squared uniform distributions are moving in horizontal direction with constant speed. The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. This allows to evaluate an algorithm in the context of incremental concept drift.
- Mixture RBF: We use the random RBF generator with 10 different parameter settings to generate a synthetic dataset of mixture Gaussian distributions with random initial positions, weights and standard deviations.

Notice that we add 0.5% uniformly distributed noise as outliers in these three synthetic datasets.

[2] https://api.mobike.com/.

5.2. Evaluation methods and metrics

Baselines. We compare the proposed UKOF and LUKOF with the following baselines:

- iLOF [17]. iLOF is an incremental LOF algorithm for landmark windows that computes the LOF score for each new added point and updates the LOF scores of the affected data points.
- DILOF [19]. DILOF is a state-of-the-art LOF-based local outlier detection method over data streams.
- sRKOF [33,35]. RKOF is a kernel-based local outlier detection method in a static dataset.
- sRDOS [22]. RDOS measures the local outlierness of data points using a local KDE model based on extended nearest neighbors.
- sKDEOS [20]. KDEOS uses KDE in local density estimations to compute a local outlierness score for each data point.
- sKDD [37]. KDD uses the Mahalanobis distance to estimate the kernel density for each point to measure its outlierness.
- sR-KDD [37]. R-KDD utilizes the robust Mahalanobis distance in the estimation of the kernel density probability.
- MCOD [3]. MCOD is a distance-based outlier detection method over data streams. It incorporates micro-cluster to detect distance-based outliers.

RKOF, RDOS, KDEOS, KDD and R-KDD typically use the KDE model to detect local outliers in a static dataset. To compare them with the proposed method, they are extended to support streaming data. More specifically, they are applied repeatedly in each new window to update the outlier status for all data points. We denote the extended RKOF, RDOS, KDEOS, KDD and R-KDD for data streams as sRKOF, sRDOS, sKDEOS, sKDD and sR-KDD, respectively.

DILOF is designed to detect local outliers over data streams; however, it samples $w/4$ expired points in the oldest $w/2$ points to minimize the density difference for each data update. To support any slide size s and large window size w, we extend DILOF to sample s expired points in the oldest $w/2$ points to preserve the density of the past data for each window slide.

Evaluation metrics. In terms of the effectiveness evaluation, we measure the quality of the reported outliers by Precision and Recall defined as follows:

$$Precision = \frac{|R \cap D_o|}{|R|},$$

$$Recall = \frac{|R \cap D_o|}{|D_o|},$$

where D_o denotes the set of the real outliers in a dataset, and R is the outliers detected by an anomaly detection method.

For the efficiency evaluation, we measure the average CPU running time per window for all methods on each dataset. The average execution times are recorded after ten repeats for each experiment.

Table 2
The time complexity comparison of all methods.

Method	Complexity
iLOF	$O(N^2)$
sKDD	$O(w^2d^2 + w^2)$
sR-KDD	$O(wbhd^2 + w^2)$
DILOF	$O((w/2)^2 + \beta w * (k^2w + k^2))$
sKDEOS	$O(w * (wlogk + k * \Delta k))$
sRDOS	$O(w * ((m+1)wlogw + m^2))$
sRKOF	$O(w * ((k+1)wlogk + k^2))$
UKOF	$O(s(\psi + \phi)(k\frac{w}{s}(s + log_M s) + k^2))$
LUKOF	$O(\alpha w(k\frac{w}{s}(s + log_M s) + k^2))$

5.3. Time complexity discussion

In this section, we discuss the time complexity of all baselines. The time complexity of all methods is displayed in Table 2.

- iLOF [17]. Similar to LOF, its time complexity is $O(N^2)$, where N is the total number of data points in dataset D.
- DILOF [19]. DILOF reduces the memory consumption by sampling from past data while preserving the density of the past data. The complexity is $O((w/2)^2)$. Because the distance between two points in a window is obtained after sampling, the complexity of the kNNs search for each point is $O(w)$. Next, it detects the outlierness degree of the points that must be updated, the time complexity is $O(\beta w * (k^2w + k^2))$, where β is the ratio of points that must be updated. Therefore, the time complexity of DILOF is $O((w/2)^2 + \beta w * (k^2w + k^2))$.
- sRKOF [33,35]. sRKOF must calculate the KDE values of each point and its kNNs to obtain the local outlierness factor; thus, the computational complexity is $O((k+1)wlogk + k^2)$. Therefore, its total time complexity is $O(w * ((k+1)wlogk + k^2))$.
- sRDOS [22]. sRDOS uses all extended nearest neighbors to compute the local KDE. We assume that the number of each point's extended nearest neighbors is m, similar to the time complexity of calculating the KOF score; the time complexity of the outlierness score for each point is $O((m+1)wlogw + m^2)$. Therefore, the time complexity of sRDOS is $O(w * ((m+1)wlogw + m^2))$.
- sKDEOS [20]. sKDEOS computes the KDE value of each point using multiple values of parameter k and uses the average z-score of the KDE values to measure its outlierness degree; thus the computation requires $O(wlogk + k * \Delta k)$, where Δk is the number of different k values. The complexity of sKDEOS is $O(w * (wlogk + k * \Delta k))$.
- sKDD [37]. sKDD uses the Mahalanobis distance to estimate the density of each point. The computational complexity of calculating the Mahalanobis distance is $O(w^2d^2)$, where d is the dimension of the data. The kernel density of each point is computed by scanning all points in the sliding window with complexity $O(w)$. Therefore, the time complexity of KDD is $O(w^2d^2 + w^2)$.
- sR-KDD [37]. Similar to KDD, R-KDD uses the minimum covariance determinant to calculate the Mahalanobis distance of points. Because it must loop b times and selects h points from the current window to calculate the robust Mahalanobis distance, the computational complexity is $O(wbhd^2)$. Therefore, its complexity is $O(wbhd^2 + w^2)$.

5.4. Effectiveness evaluation

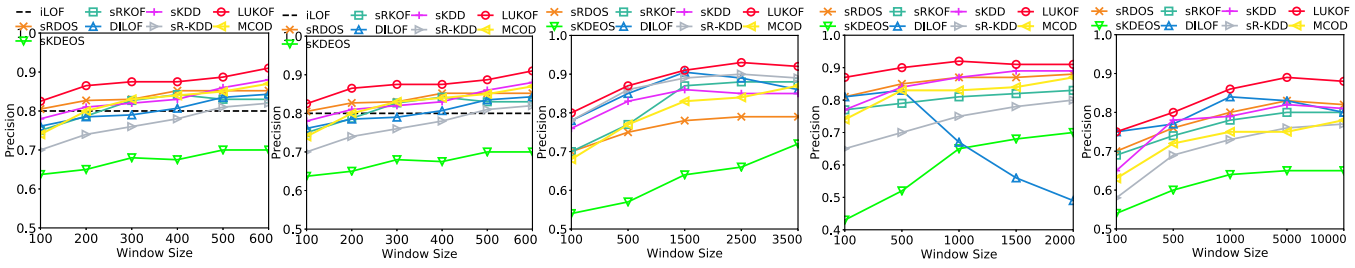
We first evaluate the effectiveness of the proposed method in term of the *precision* on the five datasets with outlier labels (i.e., Vowels, Subhttp, Smtp, ForestCover and KDDCup) and compare the results with all baselines. The experimental results are displayed in Fig. 6. Because the number and distribution of data points in the five datasets are different, we set k to 15, 15, 15, 10 and 15, h to 0.1, 1, 1, 1.5 and 1 on Vowels, Subhttp, Smtp, ForestCover and KDDCup, respectively. For the Vowels and Subhttp data, the window size w is varied from 100 to 600. For the Smtp data, w is varied from 100 to 3500. For the ForestCover data, w is varied from 100 to 2000. For the KDDCup data, w is varied from 100 to 10,000. All methods are set to the same parameters on the same dataset. We only display LUKOF in Fig. 6 because UKOF and LUKOF indicate the same performance in precision. iLOF could not be evaluated on three large datasets (i.e., Smtp, ForestCover and KDDCup) owing to its huge memory space and time consumption requirement.

From Fig. 6, we can see that LUKOF significantly outperforms all baselines in all datasets. MCOD is a distance-based outlier detection method that uses global outlier standards to process data points in the data streams. But, the distribution of data in the real world is not necessarily a globally balanced, but tends to be skewed, thus it cannot effectively detect the local outliers in the skewed data. iLOF and DILOF use the LOF score to detect local outliers; however, [21] verifies that LOF-based methods cannot accurately detect outliers with a lower outlierness degree. On ForestCover, the detection precision of DILOF decreases with window size $w \geq 500$. This is because DILOF samples s expired data points from the oldest $w/2$ data points in each new window to minimize the density difference between two consecutive windows. However, a larger the window size results in a greater density difference, and thus the searched kNNs for new points in DILOF could be inaccurate, causing a decrease in precision. Although sRKOF and sRDOS also use KDE variants to detect the outliers, the proposed method is superior to both of these, indicating that the original KDE that is well supported by theory is more effective practically. sRDOS not only uses the kNNs of the data points, but also considers RkNNs and shared nearest neighbors; however, there could be identical points in these three types of neighbors, which influences the accurate local density estimation. sRKOF uses the weighted neighborhood density estimation to make it more robust to the variations of parameter k, which is not necessarily effective for the detection of local outliers for a k value in the acceptable range. Moreover, because the distributions of data points vary in the different areas, and the local density of the same data point is evolved over data streams, it is unreasonable to use a fixed bandwidth to estimate the local density for data points. sKDD and sR-KDD use the median absolute deviation and density of data points to determine outlierness, which cannot accurately detect outliers with a lower outlierness degree. Moreover, similar to RKOF, because the distributions of data points vary in the different datasets, it is unreasonable to use a fixed bandwidth to estimate points' density for all datasets. The proposed KOF model uses an adaptive bandwidth based points' k-distance, which effectively differentiates the local density of highly dense points and low dense points. For sKDEOS, instead of estimating the relative density of each point p by its neighbors, it determines whether p is an outlier by comparing a z-score for p in its local neighborhood densities. Therefore, it can only achieve acceptable precision when the number of neighbors k is large.

In summary, the proposed method achieves the best performance in effectively detecting local outliers for large streaming data.

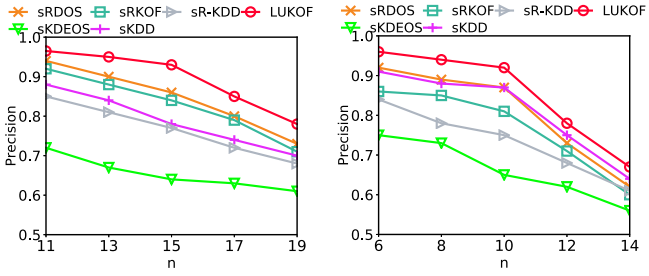
5.5. Effectiveness w.r.t. top-n

We next evaluate the effectiveness of our method in both of precision and recall by varying the number of detected outliers



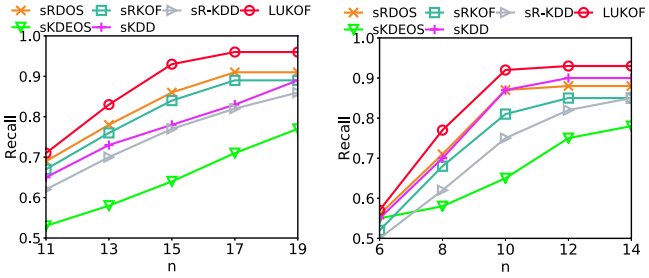
(a) Vowels (b) Subhttp (c) Smtp (d) ForestCover (e) KDDCup

Fig. 6. Precision comparison of all methods.



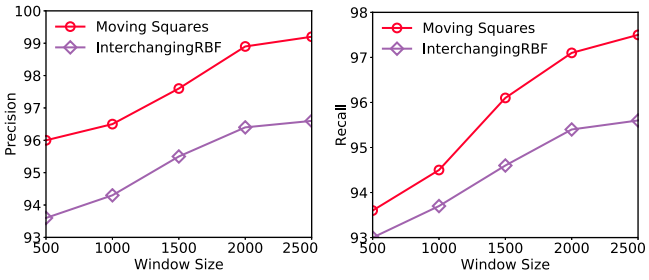
(a) Vowels (b) ForestCover

Fig. 7. Precision comparison w.r.t the number of outliers n.



(a) Vowels (b) ForestCover

Fig. 8. Recall comparison w.r.t the number of outliers n.



(a) Precision (b) Recall

Fig. 9. Precision and recall w.r.t. concept drift.

n (i.e., top-n) on two datasets (i.e., Vowels and ForestCover). All parameter settings are the same as in the last experiment. The experimental results are depicted in Figs. 7 and 8. For baselines iLOF and DILOF, they set a fixed cutoff threshold ct, if the outlierness score of a data point is greater than ct, then the point is an outlier. For MCOD, if the number of all neighbors of a point p w.r.t. the distance threshold r is less than k, then p is an outlier. Therefore, iLOF, DILOF and MCOD cannot set the parameter n when detecting outliers, so we omit them in Figs. 7 and 8.

As can be observed, the precision and recall are significantly better than all baselines in all datasets. The reason is the same as explained in above experiments. To enhance the robustness of parameter k, sRKOF uses the weighted neighborhood density estimation to detect outliers. And sKDEOS uses multiple k values for each point to estimate its KED values separately, and uses these KDE values to calculate its outlierness score. However, sRKOF and sKDEOS can only achieve acceptable precision when the number of neighbors k is large. sRDOS uses three types of neighbors (i.e., kNNs, RkNNs and shared nearest neighbors) to measure points' outlierness degree. These three neighbors may have common points, which affects the accurate local density estimation. sKDD and sR-KDD use the Mahalanobis distance to estimate the kernel density for each point, which cannot accurately detect outliers with a lower outlierness degree.

In addition, we can see that as the number of detected outliers n increases, the precision of all methods gradually decreases, and the recall increases. This is because the larger the parameter n, the more data points are detected as outliers and the smaller the Cutoff threshold ct. But, compared with real outliers, as n increases, more inlier points with lower outlierness degree would be misclassified as outliers. For the recall of methods, as the parameter n increase, the number of real outliers detected by each method increases or remains the same, thus the recall rate of each method gradually increases and then remains unchanged when n is large enough.

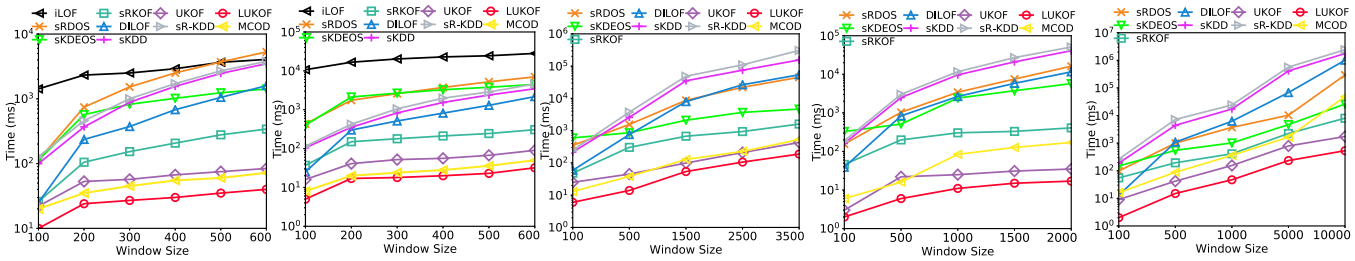
5.6. Effectiveness w.r.t. concept drift

We evaluate the effectiveness of our proposed LUKOF with respect to the concept drift in the data streams on two synthetic datasets (i.e., Interchanging RBF and Moving Squares). Experimental results are illustrated in Fig. 9.

As we see, our proposed LUKOF achieves significant outlier detection performance in both *precision* and *recall* on the two synthetic datasets in context of concept drift. Namely, our LUKOF can handle both types of incremental and abrupt drifts in data streams. More specifically, LUKOF reaches 99% precision and 97.3% recall when window size $w = 2500$ on Moving Squares, meaning that only 0.01% concept drifts are mistaken as outliers. Additionally, LUKOF performs better on Moving Squares than Interchanging RBF. This is because abrupt concept drifts are more likely to be mistaken as anomalies than the incremental drifts.

5.7. Efficiency evaluation

In this section, we first evaluate the efficiency of all the methods with respect to two important parameters (i.e., window size w and slide size s) in streaming data. Then we study the efficiency of all the methods in large-scale streaming data.



(a) Vowels (b) Subhttp (c) Smtp (d) ForestCover (e) KDDCup

Fig. 10. Efficiency comparison w.r.t. window size.

5.7.1. Efficiency w.r.t. window size w

We measure the running time per window of all methods implemented in five datasets (Vowels, Subhttp, Smtp, ForestCover and KDDCup) by varying the window size from 100 to 10,000. All parameter settings are the same as in the last experiment. The results are displayed in Fig. 10. iLOF is an incremental method for landmark windows, thus its execution time does not depend on window size. To compare with the other streaming methods, we report the average running time by dividing the total time by the number of windows used in the other methods for iLOF.

From Fig. 10, we can observe that the proposed method are significantly faster than all the baselines in every tested case for all datasets. Specifically, LUKOF consistently outperforms iLOF by 373 times on two small datasets, i.e., Vowels and Subhttp (Fig. 10(a) and (b)). Moreover, LUKOF is 275.6 ×, 332.9× and 896.7× faster than the state-of-the-art streaming local outlier detection method DILOF for the three larger datasets, i.e., Stamp, ForestCover and KDDCup, respectively. LUKOF also achieves an average of 1009.5×, 950.2×, 327.3×, 161.8× and 37.8× speedup compared to the KDE-based methods sR-KDD, sKDD, sRDOS, sKDEOS and sRKOF for the three larger datasets, respectively. The reason is that the proposed LUKOF uses the proposed upper bounds of the KOF with lower time complexity to prune the number of inlier points, and reuses the already obtained kNNs and KDE values to compute the KOF scores for the data points. For streaming local outlier detection in a sliding window, LUKOF only updates the upper bounds of the points that are affected by the newly added and expired points. Moreover, the proposed optimized R-tree index accelerates the kNNs and RkNNs search across the slides in a sliding window environment for data points, which significantly reduces the time consumption for large numbers of range queries. Although DILOF is designed to detect local outliers over data streams, it must sample s expired points in the oldest $w/2$ to minimize the density difference; thus it is sensitive to the window size w. From Fig. 10, DILOF is relatively efficient under small window sizes. However, it is difficult to apply to detecting local outliers over data streams in large-scale sliding windows owing to its high complexity $O((w/2)^2 + \beta w * (k^2 w + k^2))$. sKDD, sR-KDD, sRDOS, sKDEOS and sRKOF compute the KDE values and/or KOF scores for all data points from scratch in each new window. Therefore, they are all inefficient for streaming data outlier detection. Moreover, sKDD and sR-KDD must calculate the covariance matrix to obtain the kernel density of each point. To obtain a robust covariance matrix, sR-KDD also must calculate the covariance matrix multiple times; thus sR-KDD requires more time resource than sKDD. Moreover, in addition to kNNs, sRDOS also must search more categories of neighbors (i.e., RkNNs and shared nearest neighbors) to calculate the relative outlierness factor; thus its time complexity is $O(w * ((m+1)w\log w + m^2))$ in Table 2, which is greater than sKDEOS and sRKOF; hence, it is slower than sKDEOS and sRKOF. sRKOF is 6.3× faster than sKDEOS, as it computes the KOF score only once for each data point in each window, although it uses weighted density estimation. To obtain more robust results, sKDEOS computes the average outlierness score for each data point in its local densities using multiple values of k, meaning that it must compute multiple KDE values for each point.

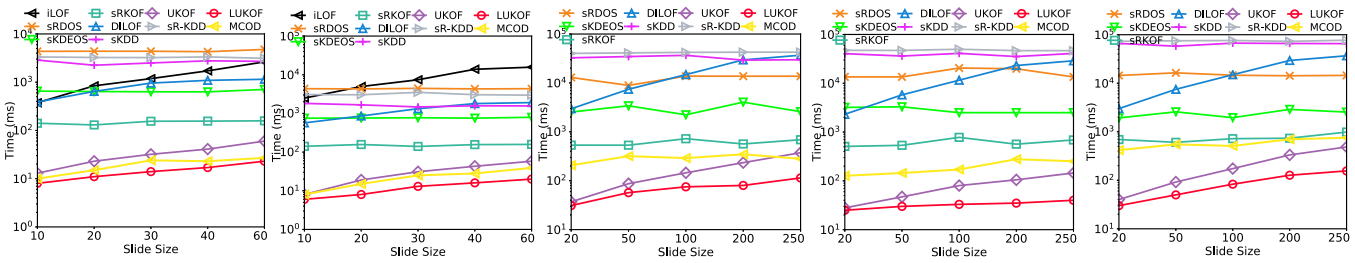
In addition, we can see that distance-based method MCOD is similar to our UKOF and LUKOF in speed on two small datasets i.e., Vowels and Subhttp. This is because MCOD is a distance-based detection method that uses global abnormal standard to detect outliers, that is, it does not need to detect outliers based on the density of points' neighbors, thus it does not need to search kNNs of kNNs of data points. However, LUKOF is 68.5× faster than MCOD on the large dataset, i.e., KDDCup. In order to avoid directly finding the neighbors of the data points, MCOD combines micro-clusters to detect outliers, that is, if a data point falls into a micro-cluster, we do not need to search its neighbors to determine its abnormal state. However, as the window size increases, the process of building micro-clusters takes more time.

As can be observed, as the window size increases, the running time per window utilized by all methods increases. However, the proposed UKOF and LUKOF save additional time compared to all the baselines. This is because the proposed method effectively prunes more inlier points using the proposed upper-bound-based pruning without executing expensive KOF computation, whereas the other baselines must compute the KDE and/or outlierness scores for each data point.

5.7.2. Efficiency w.r.t. slide size s

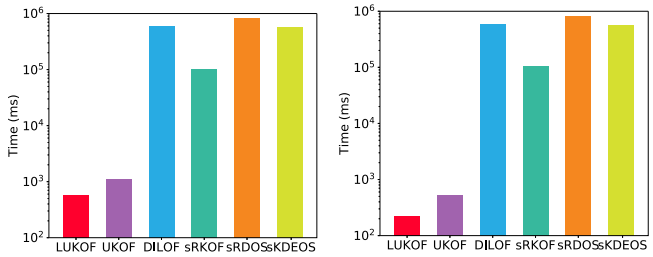
We next study the efficiency of the methods on five datasets by varying the slide size s from 10 to 250. We set the window size w to 600 on two small datasets (Vowels and Subhttp) and 2000 on three larger datasets (Smtp, ForestCover and KDDCup); all other parameters are the same as in the last experiment. Fig. 11 displays the running time results of all the methods. Note that, for a given fixed window size, as the slide size increases, the overlap in two consecutive windows decreases.

As indicated in Fig. 11, LUKOF and UKOF are consistently faster than all baselines by tens- to hundreds-fold. As the slide size s increases, LUKOF and UKOF are required to update the upper bounds and/or KOF scores for more data points in each new window, conversely, static KDE-based methods (sKDD, sR-KDD, sRKOF, sRDOS, and sKDEOS) compute outlierness scores for the same number of data points in each window. Therefore, the KDE-based methods indicate stable running time with respect to slide size. However, the proposed LUKOF and UKOF are still faster than all baselines by ten times on the ForestCover data when the slide size $s = 250$. Since DILOF once samples s expired points from the oldest $w/2$ window using its nonparametric density summarization, its running time increases as slide size increases. It is worth noting that as the slide size s increase, the proposed LUKOF becomes better than UKOF in terms of running time. This is because LUKOF uses a lazy update strategy for newly



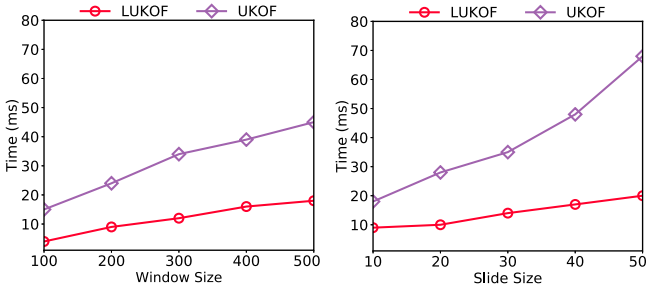
(a) Vowels (b) Subhttp (c) Smtp (d) ForestCover (e) KDDCup

Fig. 11. Efficiency comparison w.r.t. slide size.



(a) Mobike (b) Geolife

Fig. 12. Evaluation of processing time on large datasets.



(a) Window size (b) Slide size

Fig. 13. Efficiency validation on synthetic dataset.

added and expired points in the same slide; that is, we update the outlierness statuses for the neighboring new and expired points once, rather than updating them individually, which saves considerable redundant computation.

5.7.3. Efficiency w.r.t. large-scale streaming data

We next evaluate the efficiency of all the methods with respect to large-scale streaming data. In this experiment, we use two large datasets (Mobike and GeoLife) to generate the large-scale streaming data. We fix the window size w to 50,000, slide size s to 1000, and k to 10. The results are presented in Fig. 12.

As can be observed, LUKOF and UKOF significantly outperform all baselines by thousands of times on the two large-scale streaming data. In particular, the proposed LUKOF achieves 2596×, 3600×, and 2492× speedup compared to DILOF, sRDOS, and sKDEOS on the GeoLife data, respectively. The reason is the same as we explained in above experiments. When the window size becomes very large, DILOF requires additional time to sample the expired points in the nonparametric density summarization, and consumes considerably more time to update the LOF scores for the affected data points. For sRKOF, sRDOS and sKDEOS, it is clear that they significantly increase the time consumption to compute the outlierness score for each data point in such large-scale windows. Conversely, the proposed method only updates the upper bounds of the relatively small number of data points in such large windows, and significantly more inlier points are quickly filtered by the proposed pruning strategy instead of the expensive KOF computation.

In summary, this experiment confirms that the proposed LUKOF and UKOF are more efficient in detecting local outliers over high-speed large-scale data streams, rendering practical local outlier detection in real-time applications.

5.7.4. Complexity validation of proposed method

We now validate the time complexity of our proposed method on the synthetic Mixture RBF dataset. The experimental results are shown in Fig. 13. We vary window size w from 100 to 500 when fixing s to 50, and vary slide size s from 10 to 50 when fixing w to 500. k is fixed to 10.

As we can see, the running time of UKOF method increases linearly with the increase of both window size w and sliding size s, and the linear increase with s is slightly larger than that with w. For example, the running time of UKOF is increased by 2.5 times when w is increased by 5 times, while the running time is increased by 3.5 times when s is increased by 5 times. This is completely consistent with the time complexity of UKOF ($O(s(\psi+\phi)(k\frac{w}{s}(s+log_M s)+k^2))$) analyzed in Section 4.3.5. LUKOF also increases linearly with window size w similar with UKOF, but the proportion of growth with sliding size s is much smaller than that of UKOF. For example, LUKOF only increases by 10 ms as s increases from 10 to 50, while UKOF increases by 50 ms. This is also consistent with the time complexity $O(\alpha w(k\frac{w}{s}(s+log_M s)+k^2))$ of LUKOF, where α significantly decreases as s increases.

Table 3
Effectiveness of the upper-bound based pruning (Time/ms).

Datasets	LUKOF	LUKOF/UBP	UKOF	UKOF/UBP
Mobike	582	7079	1095	11,425
GeoLife	227	3723	424	6267

5.8. Effectiveness evaluation of upper-bound-based pruning

In this subsection ,we evaluate the effectiveness of the upper-bound pruning on Mobike and GeoLife. All parameter settings are the same as in the last experiment.

As indicated in Table 3, the proposed methods using upper-bound-based pruning are superior to their counterparts without upper-bound based pruning (i.e., LUKOF vs. LUKOF/UBP, and UKOF vs. UKOF/UBP) on two datasets. Specifically, LUKOF is 12.2 and 16.4 times faster than LUKOF/UBP on Mobike and GeoLife, respectively. As discussed in Section 4.2, computing the KOF score requires considerably more executing time than computing its upper bounds. The results indicate that the number of inlier

points in high density regions are examined their abnormalities using their KOF upper bounds in the proposed UKOF and LUKOF method. Therefore, this experiment also confirms that the proposed upper-bound-based pruning strategy is powerfully effective in improving detection efficiency for outlier detection over data streams.

6. Conclusion

In this work, we propose an efficient approach for supporting KDE-based top-n local outlier detection over large-scale data streams. We first define a new outlierness measure based on an adaptive KDE, KOF, to detect the local outliers. To avoid directly computing the expensive KOF score, we propose the upper bounds of KOF and use the KOF upper bounds to prune the inlier points with a lower time complexity. Based on the upper-bound-based pruning, we next propose an efficient top-n KOF detection method for data streams, called UKOF. Furthermore, we propose a lazy update version of UKOF, LUKOF, for bulk updates in large-scale data streams. Extensive evaluations on ten real-world and synthetic datasets demonstrate that the proposed method achieves the best performance in detecting local outliers over data streams, and consistently outperforms the state-of-the-art methods by up to thousands of times in speed.

CRediT authorship contribution statement

Fang Liu: Methodology, Software, Data curation, Visualization, Writing - original draft. **Yanwei Yu:** Conceptualization, Supervision, Writing - review & editing, Funding acquisition. **Peng Song:** Data curation, Investigation. **Yangyang Fan:** Writing - review & editing. **Xiangrong Tong:** Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is partially supported by the National Natural Science Foundation of China under Grant Nos.: 61773331, 61703360, 61572418 and 61403328. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

References

[1] Manish Gupta, Outlier detection for temporal data, IEEE Trans. Knowl. Data Eng. 26 (9) (2014) 2250–2267.
[2] Yanwei Yu, Cao Lei, Elke A. Rundensteiner, Wang Qin, Detecting moving object outliers in massive-scale trajectory streams, in: Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 2014.
[3] Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, Yannis Manolopoulos, Efficient and flexible algorithms for monitoring distance-based outliers over data streams, Inf. Syst. 55 (2016) 37–53.
[4] Charu C. Aggarwal, Outlier analysis, in: Data Mining, Springer, 2015, pp. 237–263.
[5] Rémi Domingues, Maurizio Filippone, Pietro Michiardi, Jihane Zouaoui, A comparative evaluation of outlier detection algorithms: Experiments and analyses, Pattern Recognit. 74 (2018) 406–421.
[6] Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, Yannis Manolopoulos, Continuous monitoring of distance-based outliers over data streams, in: 2011 IEEE 27th International Conference on Data Engineering, IEEE, 2011, pp. 135–146.
[7] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, Elke A. Rundensteiner, Scalable distance-based outlier detection over high-volume data streams, in: 2014 IEEE 30th International Conference on Data Engineering, IEEE, 2014, pp. 76–87.
[8] Guanzhe Zhao, Yanwei Yu, Peng Song, Geng Zhao, Zhe Ji, A parameter space framework for online outlier detection over high-volume data streams, IEEE Access 6 (2018) 38124–38136.
[9] Stephen D. Bay, Mark Schwabacher, Mining distance-based outliers in near linear time with randomization and a simple pruning rule, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 29–38.
[10] Fabrizio Angiulli, Clara Pizzuti, Fast outlier detection in high dimensional spaces, in: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2002, pp. 15–27.
[11] Helge Erik Solberg, Ari Lahti, Detection of outliers in reference distributions: performance of Horn's algorithm, Clin. Chem. 51 (12) (2005) 2326–2332.
[12] Charu C. Aggarwal, Philip S. Yu, Outlier detection with uncertain data, in: Proceedings of the 2008 SIAM International Conference on Data Mining, SIAM, 2008, pp. 483–493.
[13] Manzoor Elahi, Kun Li, Wasif Nisar, Xinjie Lv, Hongan Wang, Efficient clustering-based outlier detection algorithm for dynamic data stream, in: 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, Vol. 5, IEEE, 2008, pp. 298–304.
[14] Zengyou He, Xiaofei Xu, Shengchun Deng, Discovering cluster-based local outliers, Pattern Recognit. Lett. 24 (9–10) (2003) 1641–1650.
[15] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jörg Sander, LOF: identifying density-based local outliers, in: ACM Sigmod Record, Vol. 29, ACM, 2000, pp. 93–104.
[16] Edwin M. Knorr, Raymond T. Ng, Vladimir Tucakov, Distance-based outliers: algorithms and applications, VLDB J.—Int. J. Very Large Data Bases 8 (3–4) (2000) 237–253.
[17] Dragoljub Pokrajac, Aleksandar Lazarevic, Longin Jan Latecki, Incremental local outlier detection for data streams, in: 2007 IEEE Symposium on Computational Intelligence and Data Mining, IEEE, 2007, pp. 504–515.
[18] Mahsa Salehi, Christopher Leckie, James Bezdek, Tharshan Vaithianathan, Xuyun Zhang, Fast memory efficient local outlier detection in data streams, IEEE Trans. Knowl. Data Eng. 28 (12) (2016) 3246–3260.
[19] Gyoung S. Na, Donghyun Kim, Hwanjo Yu, DILOF: Effective and memory efficient local outlier detection in data streams, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2018, pp. 1993–2002.
[20] Erich Schubert, Arthur Zimek, Hans-Peter Kriegel, Generalized outlier detection with flexible kernel density estimates, in: Proceedings of the 2014 SIAM International Conference on Data Mining, SIAM, 2014, pp. 542–550.
[21] Longin Jan Latecki, Aleksandar Lazarevic, Dragoljub Pokrajac, Outlier detection with kernel density functions, in: International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, 2007, pp. 61–75.
[22] Bo Tang, Haibo He, A local density-based approach for outlier detection, Neurocomputing 241 (2017) 171–180.
[23] Edwin M. Knorr, Raymond T. Ng, Algorithms for mining distance based outliers in large datasets, in: Proceedings of the International Conference on Very Large Data Bases, Citeseer, 1998, pp. 392–403.
[24] Sridhar Ramaswamy, Rajeev Rastogi, Kyuseok Shim, Efficient algorithms for mining outliers from large data sets, in: ACM Sigmod Record, Vol. 29, ACM, 2000, pp. 427–438.
[25] Wen Jin, Anthony K.H. Tung, Jiawei Han, Mining top-n local outliers in large databases, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 293–298.
[26] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, Christos Faloutsos, Loci: Fast outlier detection using the local correlation integral, in: Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), IEEE, 2003, pp. 315–326.
[27] Pei Sun, Sanjay Chawla, On local spatial outliers, in: Fourth IEEE International Conference on Data Mining (ICDM'04), IEEE, 2004, pp. 209–216.
[28] Wen Jin, Anthony K.H. Tung, Jiawei Han, Wei Wang, Ranking outliers using symmetric neighborhood relationship, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2006, pp. 577–593.
[29] Erich Schubert, Arthur Zimek, Hans-Peter Kriegel, Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection, Data Min. Knowl. Discov. 28 (1) (2014) 190–237.
[30] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, David W. Cheung, Enhancing effectiveness of outlier detections for low density patterns, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2002, pp. 535–548.

[31] Ke Zhang, Marcus Hutter, Huidong Jin, A new local distance-based outlier detection approach for scattered real-world data, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2009, pp. 813–822.
[32] Anil K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognit. Lett. 31 (8) (2010) 651–666.
[33] Weiming Hu, Jun Gao, Bing Li, Ou Wu, Junping Du, Stephen John Maybank, Anomaly detection using local kernel density estimation and context-based regression, IEEE Trans. Knowl. Data Eng. (2018).
[34] Liangwei Zhang, Jing Lin, Ramin Karim, Adaptive kernel density-based anomaly detection for nonlinear systems, Knowl.-Based Syst. 139 (2018) 50–63.
[35] Jun Gao, Weiming Hu, Zhongfei Mark Zhang, Xiaoqin Zhang, Ou Wu, RKOF: robust kernel-based local outlier detection, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2011, pp. 270–283.
[36] M. Pavlidou, G. Zioutas, Kernel density outlier detector, in: Topics in Nonparametric Statistics, Springer, 2014, pp. 241–250.
[37] Pavlidou Meropi, Christoforos Bikos, Zioutas George, Outlier detection in skewed data, Simul. Model. Pract. Theory 87 (2018) 191–209.
[38] Arvind Arasu, Shivnath Babu, Jennifer Widom, The CQL continuous query language: semantic foundations and query execution, VLDB J. 15 (2) (2006) 121–142.
[39] Mineichi Kudo, Jun Toyama, Masaru Shimbo, Multidimensional curve classification using passing-through regions, Pattern Recognit. Lett. 20 (11–13) (1999) 1103–1111.
[40] Charu C. Aggarwal, Saket Sathe, Theoretical foundations and algorithms for outlier ensembles, ACM Sigkdd Explor. Newslett. 17 (1) (2015) 24–47.
[41] Saket Sathe, Charu Aggarwal, LODES: Local density meets spectral outlier detection, in: Proceedings of the 2016 SIAM International Conference on Data Mining, SIAM, 2016, pp. 171–179.
[42] K.D.D. Cup, Dataset. available at the following website http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html 72 (1999).
[43] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, Lifang Gu, A comparative study of RNN for outlier detection in data mining, in: 2002 IEEE International Conference on Data Mining, 2002. Proceedings, IEEE, 2002, pp. 709–712.
[44] Fei Tony Liu, Kai Ming Ting, Zhi-Hua Zhou, Isolation forest, in: 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 413–422.
[45] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J.G.B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, Michael E. Houle, On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study, Data Min. Knowl. Discov. 30 (4) (2016) 891–927.
[46] Jock A. Blackard, Denis J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Comput. Electron. Agric. 24 (3) (1999) 131–151.
[47] Swee Chuan Tan, Kai Ming Ting, Tony Fei Liu, Fast anomaly detection for streaming data, in: Twenty-Second International Joint Conference on Artificial Intelligence, 2011.
[48] Yu Zheng, Xing Xie, Wei-Ying Ma, et al., Geolife: A collaborative social networking service among user, location and trajectory., IEEE Data Eng. Bull. 33 (2) (2010) 32–39.
[49] Viktor Losing, Barbara Hammer, Heiko Wersing, KNN classifier with self adjusting memory for heterogeneous concept drift, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 291–300.
[50] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, Fazli Can, Unsupervised concept drift detection with a discriminative classifier, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 2365–2368.