Scalable Kernel Density Estimation-based Local Outlier Detection over Large Data Streams

Xiao Qin[1], Lei Cao[2], Elke A. Rundensteiner[1] and Samuel Madden[2]
[1]Department of Computer Science, Worcester Polytechnic Institute
[2]CSAIL, Massachusetts Institute of Technology
[1]{xqin,rundenst}@cs.wpi.edu [2]{lcao,madden}@csail.mit.edu

ABSTRACT

Local outlier techniques are known to be effective for detecting outliers in skewed data, where subsets of the data exhibit diverse distribution properties. However, existing methods are not well equipped to support modern high-velocity data streams due to the high complexity detection algorithms and their volatility to data updates. To tackle these shortcomings, we propose new local outlier semantics that leverage kernel density estimation (KDE) to effectively detect local outliers from streaming data. A strategy to continuously detect top-N KDE-based local outliers over streams is also designed, called **KELOS** – the first linear time complexity streaming local outlier detection approach. The first innovation of KELOS is the abstract kernel center-based KDE (*a*KDE) model. *a*KDE accurately while efficiently estimates the data density at each point – essential ingredient for local outlier detection. This is based on the observation that a cluster of points close to each other tend to have a similar influence on a target point's density estimation when used as kernel centers. These points thus can be represented by one abstract kernel center. Next, the KELOS's inlier pruning strategy early prunes points that have no chance to become top-N outliers. This empowers KELOS to skip the computation of the data density and then the outlier status for every data point. Together *a*KDE and the inlier pruning strategy eliminate the performance bottleneck of streaming local outlier detection. The experimental evaluation demonstrates that KELOS is up to 3 orders of magnitude faster than existing solutions, while still being highly effective in detecting local outliers from streaming data.

1 INTRODUCTION

Motivation. The growth of digital devices coupled with their ever-increasing capabilities to generate and transmit live data presents an exciting new opportunity for real time data analytics. As the volume and velocity of data streams continue to grow, automated discovery of insights in such streaming data is critical. In particular, finding *outliers* in streaming data is a fundamental task in many online applications ranging from fraud detection, network intrusion monitoring to system fault analysis. In general, outliers are data points situated away from the majority of the points in the data space. For example, a transaction of a credit card in a physical location far away from where it has normally been used may indicate fraud. Over 15.4 million U.S residents were victims of such fraud in 2016 according to [3]. On the other hand, as more transactions take place in this new location, the previous transaction may appear legitimate as it begins to conform to the increasingly expected behavior exemplified by the new data. Thus, in streaming environments, it is critical to design a mechanism to efficiently identify outliers by monitoring the statistical properties of the data as it changes over time.

State-of-the-Art. To satisfy this need, several methods [17, 18] have been proposed in recent years that leverage the concept of local outlier [6] to detect outliers from data streams. The local outlier notion is based on the observation that real world datasets tend to be skewed, where different subspaces of the data exhibit different distribution properties. It is thus often more meaningful to decide on the outlier status of a point based on its difference from the points in its *local neighborhood* as opposed to using a global density [8] or frequency [5] cutoff threshold to detect outliers [9]. More specifically, a point x is considered to be a *local outlier* if the *data density* at x is low *relative* to that at the points in x's local neighborhood.

Unfortunately, existing streaming local outlier solutions [17, 18] are not scalable to high volume data streams. The root cause is that they measure the data density at each point x based on the point's distance to its k nearest neighbors (kNN). Unfortunately, kNN is very sensitive to data updates, meaning that the insertion or removal of even a small number of points can cause the kNN of many points in the dataset to be updated. Since the complexity of the kNN search [6] is quadratic in the number of the points, significant resources may be wasted on a large number of unnecessary kNN re-computations. Therefore, those approaches suffer from a high response time when handling high-speed streams. For example, it takes [17, 18] 10 minutes to process just 100k tuples as shown by their experiments.

Intuitively, kernel density estimation (KDE) [22], an established probability density approximation method, could be leveraged for estimating the data density at each point [14, 20, 23]. Unlike kNN-based density estimation that is sensitive to data changes, KDE estimates data density based on the statistical properties of the dataset. Therefore, it tends to be more robust to gradual data changes and thus a better fit for streaming environments. However, surprisingly, to date no method has been proposed that utilizes KDE to tackle the local outlier detection problem for data streams.

Challenges. Effectively leveraging KDE in the streaming context comes with challenges. Similar to kNN search, the complexity of KDE is quadratic in the number of points [22]. While the computational costs can be reduced by running the density estimation on

KDD'18, London, United Kingdom
© 2018 ACM. 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

kernel centers sampled from the input dataset, sampling leads to a trade-off between accuracy and efficiency. Although a low sampling rate can dramatically reduce the computational complexity, one must be cautious because the estimated data density at each point may be inaccurate due to an insufficient number of kernel centers. On the other hand, a higher sampling rate will certainly lead to a better estimation of the data density. However, the computational costs of KDE increase quadratically with more kernel centers. With a large number of kernel centers, KDE would be at risk of becoming too costly to satisfy the stringent response time requirements of streaming applications. Due to this accuracy versus efficiency trade-off, to the best of our knowledge, no method has successfully adapted KDE to function efficiently on streaming data to date.

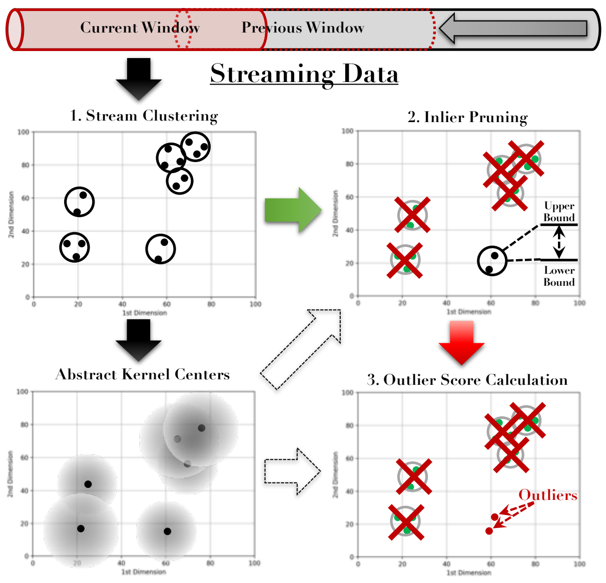


Figure 1: An illustration of KELOS approach.

Proposed Solution. In this work, we propose a scalable KDE-based strategy (Fig. 1) for detecting top-N local outliers over streams, or in short **KELOS**. KELOS provides the first practical solution for local outlier detection on streaming data. Our key contributions are given below.

• New KDE-based semantics are proposed for the continuous detection of the top promising outliers from data streams. This establishes a foundation for the design of a scalable streaming local outlier detection method.

• A notion of the *abstract kernel center* is introduced to solve the accuracy versus efficiency trade-off of KDE. This leverages the observation that kernel centers close to each other tend to have a similar strength of influence on the densities at other points. These nearby points thus can be clustered together and considered as one abstract kernel center weighted by the amount of data it represents. Compared to the traditional sampling-based KDE, our strategy achieves higher accuracy in density estimation using *much fewer kernel centers*. This in turn speeds up the quadratic complexity process of local density estimation. This notion of abstract kernel centers by itself could be applied to a much broader class of density estimation related stream mining tasks beyond local outlier detection.

• An inlier pruning strategy. Unlike existing techniques [17, 18], which detect outliers by computing the data density and then the outlierness score for every data point, KELOS quickly prunes the vast majority of the data points that have no chance to become outliers. The more expensive KDE method itself is only used thereafter to evaluate the remaining much smaller number of potential outlier candidates.

• Putting these optimizations together, we obtain the first linear time complexity streaming local outlier detection approach that outperforms the state-of-the-arts by up to 3 orders of magnitudes in speed confirmed by our experiments on real world datasets.

2 PRELIMINARIES

2.1 Local Outlier

Given a point x_i, it is a local outlier if the *data density* at x_i (e.g. inverse of average distances to its kNN) is significantly lower than the densities at x_i's neighbors.

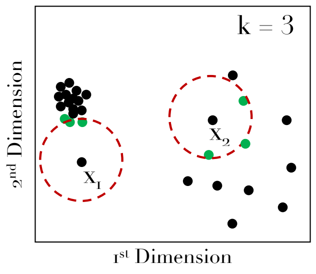


Figure 2: Local outlier detection using local densities.

As illustrated in Fig. 2, although the densities at x_1 and x_2 are both low, the density at x_1 is quite different than the densities at the locations of its neighbors. However, the densities at the neighbors of x_2 is similar to x_2. Therefore, x_1 is more likely to be an outlier than x_2 due to its *relatively low density* in contrast to those at its neighbors. Therefore, conceptually measuring a point x_i's status of being a **local outlier** corresponds to the two-steps:

(1) Estimate the density at x_i and the densities at its neighbors;
(2) Compute the outlierness score of x_i based on the deviation of the density at x_i in contrast to those at its neighbors.

2.2 Kernel Density Estimation

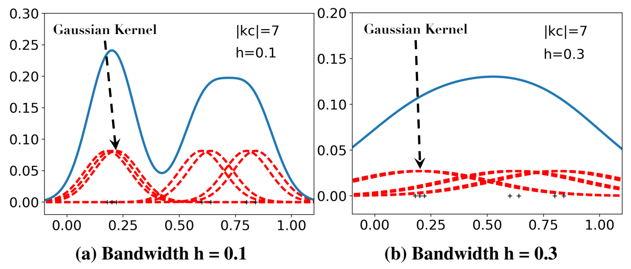


Figure 3: An example of univariate kernel density estimator using Gaussian kernel with different bandwidth.

Kernel density estimation (KDE) is a non-parametric method to estimate the probability density function (PDF) of a dataset $X = \{x_1, \cdots, x_n\}$. Given a point x_i, the kernel density estimator of X computes how likely x_i is drawn from X. This computed probability can be interpreted as the "data density" at x_i in X.

The density at x_i in X is:

$$\tilde{f}(x_i) = \frac{1}{m}\sum_{j=1}^{m} K_h(|x_i - kc_j|). \tag{1}$$

Kernel Centers. $kc_j \in \mathbb{KC}$ where 1 ≤ j ≤ m are called the kernel centers in the estimator. Typically, kc_j is a point sampled from X. The selected set of kernel centers must be sufficient to represent the data distribution of X [22]. Each kernel center kc_j carries a kernel function K_h. The *density contribution* by a kernel center kc_j is calculated based upon the distance from kc_j to the target point x_i. The density at x_i is estimated by the average density contribution by all kernel centers. For example, in Fig. 3(a), there are 7 kernel centers. Each of them carries a kernel function (red dashed curve). The shape of the overall density function across all kernels is represented by the blue solid line. Given a dataset X with n points and m kernel centers, the time complexity of computing the densities of all $x_i \in X$ is $O(nm)$.

Kernel Function. A wide range of kernel functions can be used in kernel density estimation [22]. The most commonly used ones are the *Gaussian* and *Epanechnikov* kernel functions [9]. In this study, we adopt the *Gaussian* kernel:

$$K_{gauss}(u) = \frac{1}{(\sqrt{2\pi})h} e^{(-\frac{1}{2}\frac{u^2}{h^2})}, \tag{2}$$

where u represents the distance from a kernel center kc_j to the target point x_i and h is an important smoothing factor, called *bandwidth*. The bandwidth controls the smoothness of the shape of the estimated density function. The greater the value h, the smoother the shape of the density function $\tilde{f}$. As shown in Figs. 3(a) and (b), using the same set of kernel centers but different bandwidth values, the estimated PDFs (the blue lines) are significant different from each other. Therefore, an appropriate bandwidth is critical to the accuracy of the density estimation.

Balloon Kernel. In Balloon kernel, when estimating the density at a target point x_i, only the k nearest kernel centers of x_i denoted as kNN(x_i,$\mathbb{KC}$) are utilized in the estimator. This provides each point x_i a customized kernel density estimator that adapts to the distribution characteristics of x_i's surrounding area, hence also called *local density*. Therefore, Balloon kernel fits the local outlier that detects outliers based on the local distribution properties as shown in [20].

$$\tilde{f}_b(x_i) = \frac{1}{k}\sum_{j=1}^{k} K_h(|x_i - kc_j|) \text{ where } kc_j \in k\text{NN}(x_i, \mathbb{KC}). \tag{3}$$

3 PROPOSED OUTLIER SEMANTICS

Next we propose semantics of KDE-based streaming local outliers. We first introduce the notion of top-N local outliers that captures the most extreme outliers in the input dataset. We then apply this concept to sliding windows to characterize outliers in data streams.

3.1 Top-N KDE-based Local Outliers

We first define our new outlierness measure, KDE-based local outlierness measure (KLOME).

Definition 3.1. **KLOME.** Given a set of data points $X = \{x_1, \cdots, x_n\}$ and a set of kernel centers $\mathbb{KC} = \{kc_1, \cdots, kc_m\}$, the KLOME score of a target point $x_i \in X$ is defined as KLOME(x_i) = $z\text{-}score(\tilde{f}_b(x_i), \{\tilde{f}_b(kc_j) \mid \forall kc_j \in k\text{NN}(x_i, \mathbb{KC})\})$.

Here $z\text{-}score(s, S) = (s - S)/\sigma_S$ [27] indicates how many standard deviations an value s is above or below the mean of a set of values S. In this definition, KLOME(x_i) measures how different the local density at x_i is from the average local density at x_i's nearest kernel centers denoted as kNN($x_i, \mathbb{KC}$). A negative KLOME score of a target point x_i indicates that the local density at x_i is smaller than the local densities at its neighbors' locations. The *smaller* the KLOME score of a point x_i is, the *larger* the possibility that x_i is an outlier.

The key property of our KLOME semantics is that the density at x_i is compared against the densities at its kNN in the kernel center set $\mathbb{KC}$ (kNN($x_i, \mathbb{KC}$)) instead of its actual kNN in the dataset. The intuition is as below. The kernel centers sufficient to recover the distribution of the original dataset can well capture every local phenomenon. The density at x_i is estimated based on its location *relative to* the selected kernel centers $kc_j \in k$NN($x_i, \mathbb{KC}$). Naturally kc_j can serve as the local neighbor of x_i in the density deviation computation of x_i (*z-score*). In other words, kNN($x_i, \mathbb{KC}$) effectively models the local neighborhood of a point x_i.

Next, we define the concept of top-N KDE-based local outlier:

Definition 3.2. Given a set of data points $X = \{x_1, \cdots, x_n\}$ and a count threshold N. The **top-N KDE-based local outliers** are a set of N data points, denoted by Top-KLOME(X, N) such that $\forall x_i \in Top$-KLOME(X, N) and $\forall x_j \in X \setminus Top$-KLOME($X, N$), KLOME($x_i$) ≤ KLOME($x_j$).

3.2 Local Outlier Detection in Sliding Window

We work with periodic sliding window semantics commonly adopted to model a finite substream of interest from the otherwise infinite data stream [4]. Such semantics can be either time or count-based. Each data point x_i has an associated time stamp denoted by $x_i.time$. The window size and slide size of a stream S are denoted as $S.win$ and $S.slide$ correspondingly. Each window W_c has a starting time $W_c.T_{start}$ and an ending time $W_c.T_{end} = W_c.T_{start} + S.win$. Periodically the current window W_c slides, causing $W_c.T_{start}$ and $W_c.T_{end}$ to increase by $S.slide$ respectively. For count-based windows, a fixed number (count) of data points corresponds to the window size $S.win$. Accordingly $S.slide$ is also measured by number of data points. $\mathcal{S}^{W_c}$ denotes the set of data points falling into the current window W_c. The local outliers are then always detected in the current active window W_c. An outlier in the current window might turn into an inlier in the next window.

Next, we define the problem of continuously detecting Top-KLOME(X, N) over sliding windows:

Definition 3.3. Given a stream S, a window size $S.win$, a slide size $S.slide$, and an integer N, continuously compute Top-KLOME($\mathcal{S}^{W_c}, N$) over sliding windows.

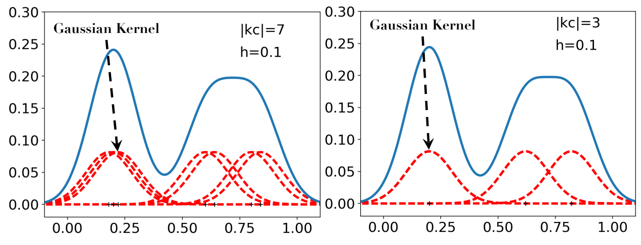
Next, we introduce our **KELOS** approach for supporting the new semantics. KELOS contains two components: density estimator and outlier detector. Given a point x_i, the density estimator efficiently computes the density at x_i using the abstract kernel center concept (Sec. 4). The outlier detector first uses an inlier pruning strategy to prune the points that are guaranteed to be not outliers. Then it detects outliers from the remaining outlier candidates using their densities computed by the density estimator (Sec. 5).

4 DENSITY ESTIMATOR

In this section, we propose our abstract kernel center-based KDE strategy (*a*KDE). It solves the problem of accurately yet efficiently estimating the density at a given point. In contrast to the traditional sampling-based KDE approach [22], our density estimation is performed on top of a set of clusters (Fig. 1) that succinctly summarize the distribution characteristics of the dataset. This approach is inspired by our *abstract kernel center* observation below.

Abstract Kernel Center Observation. In KDE, the density at a given point x_i is determined by the *additive influences* of the kernel centers, while the influence from one center kc_j is determined by the distance between kc_j and x_i. The centers close to each other tend to have similar influence on the target point x_i. Using them redundantly instead of representing them as a whole perplexes the density estimation by unnecessarily enlarging the center space. To obtain succinct while informative representatives as kernel centers, KELOS first performs a lightweight clustering that only groups extremely close points together. The centroid of the cluster weighted by the cluster's data cardinality, called abstract kernel center (AKC) is then selected as a kernel center to perform density estimation.

Fig. 4(b) shows an example estimation using the abstract kernel centers. The original 7 points in Fig. 3(a) are abstracted into three clusters. The estimations (blue line) in Fig. 4(b) with 3 centers and Fig. 4(a) using all 7 points as kernel centers are similar.



(a) All points as kernel centers **(b) Abstract kernel centers**

Figure 4: Local kernel density estimator.

On the performance side, real world data sets tend to be skewed. Therefore, typically most points can be clustered into a small number of tight clusters. Correspondingly, the number of the abstract kernel centers tends to be much smaller than the number of sampled kernel centers that would be sufficient to represent the overall data distribution of the dataset. Since the bottleneck of local density estimation is on the computation of the k nearest kernel centers for each to be estimated point x_i, the small number of abstract kernel centers promises to reduce the complexity of the successive density estimation process.

Furthermore, the abstract kernel centers allow us to use a small k while establishing a diversified neighborhood – hence a comprehensive density estimator for each point. This not only reduces the complexity of the kNN search and kernel density computation, but also alleviates the problem of selecting an appropriate k. Since the abstract kernel centers representing data clusters are more stable than sampled individual points in terms of their statistical properties, our selected k by such method would be more robust to the continuously changing stream data. Next, we formally define the concept of abstract kernel centers.

Definition 4.1. Given a stream window $S^{W_c} = \{x_1, \cdots, x_n\}$, the abstract kernel centers of S^{W_c} are a set of pairs $\mathbb{AKC}(S^{W_c}) = \{\langle c_{c_1}, |c_1|\rangle, \cdots, \langle c_{c_m}, |c_m|\rangle\}$, where c_{c_i} $(1 \le i \le m)$ corresponds to the centroid of the respective data cluster c_i and $|c_i|$ the number of points in c_i. Here $\bigcup_{i=1}^{m} c_i = S^{W_c}$ and $\forall i, j, i \neq j\ c_i \cap c_j = \emptyset$.

Weighted Kernel Density Estimator. Intuitively, each abstract kernel center represents the centroid of a cluster of points close to each other along with the data cardinality of this cluster. Utilizing these abstract kernel centers, we construct a weighted kernel density estimator [11], where the kernel centers correspond to the centroids in $\mathbb{AKC}(S^{W_c})$ (the first component of $\mathbb{AKC}$) and the weight corresponds to the cardinality of the data cluster represented by the centroid (the second component). Therefore, the weighted kernel density estimator reflects the distribution characteristics of the entire dataset by utilizing only a small number of kernel centers. The formula is shown below:

$$\tilde{f}_{\mathbb{AKC}(S^{W_c})}(x_i) = \sum_{j=1}^{k} \omega(c_{c_j}) K_h(|x_i - c_{c_j}|), \tag{4}$$

where

$$\omega(c_{c_j}) = \frac{|c_j|}{\sum_{m=1}^{k} |c_m|}, \tag{5}$$

and $c_{c_m} \in k\text{NN}(x_i, \mathbb{AKC}(S^{W_c}))$. Here $\tilde{f}_{\mathbb{AKC}(S^{W_c})}(x_i)$ in Eq. 4 corresponds to a weighted product kernel estimator that computes the local density at x_i and $k\text{NN}(x_i, \mathbb{AKC}(S^{W_c}))$ corresponds to the k nearest centroids of x_i in the abstract kernel centers.

Bandwidth Estimation. One additional step required to make the weighted kernel density estimator work is to establish an appropriate bandwidth for the product kernel. Here we show that the *rule-of-thumb* strategy [21] (Eq. 6) can be efficiently applied here by leveraging the abstract kernel centers.

$$h = 1.06\sigma k^{-1/(d+1)}. \tag{6}$$

In Eq. 6, d denotes the data dimension. σ denotes the weighted standard deviation of the kernel centers computed by:

$$\sigma = \sqrt{\sum_{m=1}^{k} \omega(c_{c_m})(c_{c_m} - \mu)^2}, \tag{7}$$

where

$$\mu = \frac{\sum_{m=1}^{k} \omega(c_{c_m}) c_{c_m}}{k}, \tag{8}$$

and $c_{c_m} \in k\text{NN}(x_i, \mathbb{AKC}(S^{W_c}))$.

Effectiveness. Our *a*KDE builds robust density estimator based on the observation that real world datasets typically can be represented by tight data clusters because of the skewness of the data distribution. This is confirmed by our experiments. If a dataset is uniformly distributed, sampling-based KDE is effective as well.

Efficiency. The time complexity of KDE is $O(nm)$, with n is number of data points and m is the number of kernel centers. Since *a*KDE dramatically reduces the number of kernel centers, it significantly speeds up the KDE computation. On the other hand, data clustering introduces extra computation overhead. In this work we apply the low complexity Micro clustering [2] strategy that processes each point only once. This overhead is significantly outweighed by the saved KDE computation costs. Therefore, overall *a*KDE is much faster than the traditional KDE – as shown in Sec. 6.2.

Stream Clustering. Next we briefly introduce the Micro clustering algorithm. Once a new data point x arrives, the algorithm first finds its nearest cluster according to the distance of x to all the centroids. If the distance from x to its nearest cluster c_i denoted as $dist(x, c_{c_i})$ is smaller than a *radius threshold* θ, x is inserted into c_i. On the other hand, if $dist(x, c_{c_i}) > \theta$, a new cluster will be created. During this process, the *centroid* of each cluster which is needed by *a*KDE, is also efficiently generated by continuously maintaining the *linear sum* of the points by dimensions. For the details please refer to [2].

Tuning Radius Threshold θ. Micro-clustering utilizes the radius threshold θ to make sure only the extremely similar points fall into the same cluster and produce tight clusters. The smaller the θ is, the tighter the formed clusters are. However, as shown in Fig. 9 of Sec. 6.4, although small θ achieves high accuracy in density estimation, it also slows down the overall speed of KELOS. Therefore, it is important to find an appropriate θ threshold that can balance the speed and the accuracy. Instead of trying to acquire an optimal θ value at the beginning by exploring some expensive preprocessing, in streaming context we recommend that the θ threshold could be dynamically adapted to the optimal value. More specifically, one can start by initializing the system using a relatively small θ value to ensure the accuracy of the results. Then the θ value can be gradually adjusted to larger values as the data stream evolves as long as the accuracy is still reasonably good. If concept drift occurs when stream data evolves, the θ is adjusted again using the same principle.

5 OUTLIER DETECTOR

Our outlier detector fully utilizes the data clusters produced for *a*KDE by leveraging our *stable density* observation described below.

Stable Density Observation. Data points in a tight cluster are close to each other. Therefore, they tend to share the same kernel centers and have similar local densities. By the definition of local outliers, the outlierness score of a point x depends on the relative density at x in contrast to those at its neighbors. Therefore, these points tend to have similar outlierness scores. Since outliers only correspond to small subset of points with the highest outlierness scores, it is likely that most of the data clusters do not contain any outlier.

Assume we have a method to approximate the highest (upper bound) and lowest (lower bound) outlierness scores for the points in each data cluster. Using these bounds, the data clusters that have no chance to contain any outlier can be quickly identified and pruned from outlier candidate set without any further investigation. More specifically, if the upper bound outlierness score of a data cluster c_i is smaller than the lower bound outlierness score of a data cluster c_j, then the whole c_i can be pruned (under the trivial premise that c_j has at least N points). This is so because there are at least N points in the dataset whose outlierness scores are larger than any point in c_i.

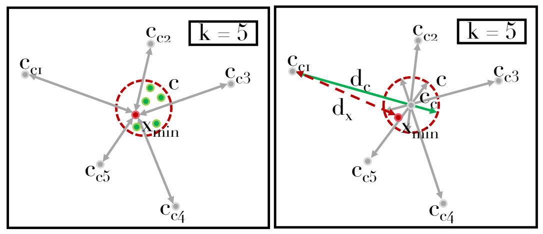
Leveraging this observation, we now design an efficient local outlier detection strategy. The overall process is given in Alg. 1. We first rank and then prune data clusters based on their upper and lower KLOME score bounds. As shown in Sec. 3.1, a small KLOME score indicates large outlier possibility. Therefore, the upper KLOME bound corresponds to the lower outlierness score bound. Similarly, the lower KLOME bound corresponds to the upper outlierness score bound. Therefore, if the lower KLOME bound of a cluster c_i is higher than the upper KLOME bound of another cluster c_j, all points in c_i can be pruned immediately. Only the clusters with a small lower KLOME bound (large outlierness score upper bound) are subject to further investigation. The densities and KLOME scores at the data point-level are computed only for the data points in these remaining clusters. Finally, the top-N results are selected among these points by maintaining their KLOME scores in a priority queue.

```
Algorithm 1 : Top-N Outlier Computation.

Input: Clusters $C$.
Output: Top-$N$ Outliers.
 1: $PriorityQueue$<Cluster> $P$ of size $N$ /*by upperbound in
    ascending order*/
 2: $P \leftarrow$ first $N$ in $C$
 3: for rest of the cluster $c$ in $C$ do
 4:     if $KLOME_{low}(c) > KLOME_{up}(P.peek)$ then
 5:         prune $c$
 6:     else if $KLOME_{up}(c) < KLOME_{low}(P.peek)$ then
 7:         $P$.poll & $P$.add($c$)
 8:     else
 9:         $P$.add($c$)
10: $PriorityQueue$<Data> $R$ of size $N$ /*by $KLOME$ in as-
    cending order*/
11: for cluster $c$ in $P$ do
12:     for data $d$ in $c$ do
13:         compute $KLOME$ of $d$
14:         $R$.add($d$)
15: return $R$
```

5.1 Bounding the KLOME Scores

Next, we present an efficient strategy to establish the upper and lower KLOME bounds for each given data cluster.



(a) Lowest KLOME in c **(b) Lower bound of c**

Figure 5: An example of lower KLOME bound.

By Def. 3.1, the KLOME score of a point x_i corresponds to *z-score*($\tilde{f}(x_i)$, S), where S refers to the local densities at x_i's kernel centers. Since the points in the same cluster c_i typically share the same kernel centers, the data point $x_{min} \in c_i$ with the minimal density determines the lower bound KLOME score of the entire cluster c_i. Similarly the upper bound is determined by the point x_{max} with the maximal density. Obviously it is not practical to figure out the lower/upper bound by computing the densities at all points and thereafter finding x_{min} and x_{max}.

Lower bound. We now show that by utilizing the statistical property of each data cluster – more specifically the *radius*, the bounds can be derived in constant time. Here we use the lower bound as example to demonstrate our solution (Fig. 5).

LEMMA 5.1. *Given a data cluster c_i, its k nearest kernel centers $\{c_{c_1}, \cdots, c_{c_k}\}$ and the data point x_{min} which has the minimum* *density among all points in c_i, $\tilde{f}_{min}(c_i) \leq \tilde{f}(x_{min})$, where $\tilde{f}_{min}(c_i) = \sum_{j=1}^{k} \omega(c_{c_j}) K_h(|c_{c_i} - c_{c_j}| + r)$. Here r is the radius of c_i and c_{c_i} is the centroid of c_i.*

PROOF. The density contribution $K_h(|x_i - c_{c_j}|)$ is inversely proportional to the distance between the evaluated point x_i and the kernel center c_{c_j}. The longer the distance, the smaller the density contribution is from the kernel center. The radius r of a cluster c_i is the distance from c_i's centroid c_{c_i} to the furthest possible points in c_i. The longest possible distance from a kernel center c_{c_j} to any point in c_i is denoted as $d_c = |c_{c_i} - c_{c_j}| + r$. The distance from c_{c_1} to x_{min} is denoted as $d_x = |c_{c_j} - x_{min}|$. $d_c \geq d_x$ by the triangle inequality. Therefore $K_h(d_x) \leq K_h(d_c)$. This holds for any kernel center c_{c_j}. Therefore $\tilde{f}_{min}(c_i) = \sum_{j=1}^{k} \omega(c_{c_j}) K_h(|c_{c_j} - c_{c_i}| + r) \leq \tilde{f}(x_{min})$. □

Intuitively, the density at a data point is measured by the summation of the density contributions of all relevant kernel centers. The summation of the density contribution from each kernel center c_{c_j} to the point x_j that is the point furthest to c_{c_j} in c_i is guaranteed to be smaller or equal to the density at point x_{min}. This is so because the distance from x_{min} to each kernel center c_{c_j} cannot be larger than the distance between c_{c_j} and x_j.

According to Lemma 5.1, given the radius of a data cluster c_i and its k nearest kernel centers $c_{c_1} \cdots c_{c_k}$, the **lower KLOME bound** of cluster c_i is computed as:

$$KLOME_{low}(c_i) = z\text{-}score(\tilde{f}_{min}(c_i), \{\tilde{f}(c_{c_1}) \cdots \tilde{f}(c_{c_k})\}). \quad (9)$$

Upper Bound. Similarly, we can show that the maximal local density at a cluster c_i, denoted by $\tilde{f}_{max}(c_i)$, can be obtained based on the shortest distance from each kernel center to the points in c_i.

$$\tilde{f}_{max}(c_i) = \sum_{j=1}^{k} \omega(c_{c_j}) K_h(|c_{c_j} - c_{c_i}| - r). \quad (10)$$

Accordingly, the upper KLOME bound of each cluster c_i $KLOME_{up}(c_i)$ is derived based on $\tilde{f}_{max}(c_i)$.

$$KLOME_{up}(c_i) = z\text{-}score(\tilde{f}_{max}(c_i), \{\tilde{f}(c_{c_1}) \cdots \tilde{f}(c_{c_k})\}). \quad (11)$$

5.2 The Efficient Maintenance of the Meta Data

Radius of the Cluster. As the clusters are continuously constructed along time, the radius of each cluster needed by our inlier pruning strategy must also be continuously generated. Since the radius is defined as the distance from the centroid c_{c_i} to its furthest point in cluster c_i, the radius changes whenever the centroid changes. All points in c_i then have to be re-scanned to find the point "furthest" from the new centroid. This, being computational expensive, is not acceptable in online applications. Next we introduce a strategy to efficiently generate the *radius* based on Lemma 5.2.

LEMMA 5.2. *Given a new centroid c_{c_i}, the radius r of cluster $c_i \leq r_u = \sqrt[2]{\sum_{l=1}^{d} max\{(v^l - x^l_{min})^2, (x^l_{max} - v^l)^2\}}$, where v^l is the value c_{c_i} at dimension l, x^l_{min} and x^l_{max} are the minimum and maximum values of the points in c_i at dimension l.*

By Lemma 5.2 given an cluster c_i, we can compute an upper bound radius r_u by maintaining the minimum (x^l_{min}) and maximum (x^l_{max}) values of the points in c_i at each dimension l. Since updating the minimum or maximum value per insertion or deletion takes constant time, computing r_u is very efficient. By replacing r with r_u, it is still guaranteed that no outlier will be erroneously pruned by our inlier pruning strategy, because the upper and lower KLOME bounds still hold. Due to space constraint, the proof is omitted here.

Pane-based meta data maintenance. The pane-based meta data maintenance strategy [15] is utilized to effectively update the meta data for each cluster as the window slides. Given the window size $S.win$ and slide size $S.slide$, a window can be divided into $\frac{S.win}{gcd(S.win, S.slide)}$ small panes where gcd refers to greatest common divisor. The meta data of a cluster c_i is maintained at the pane granularity instead of maintaining one meta data structure for the whole window. Because the data points in the same pane arrive and expire at the same slide pace, the meta data of the new window can be quickly computed by aggregating the meta data structures maintained for the unexpired panes as the window moves. Since the meta data satisfies the additive property, the computation can be done in constant time. In this way, no explicit operation is required to handle the expiration of outdated data from the current window.

5.3 Time Complexity Analysis of KELOS

The complexity of the clustering comes from the nearest centroid search. The complexity is $O(mc)$ with m the number of new arrivals and c the number of centroids. In the density estimation step, each point has to find its k nearest kernel centers from the c centroids. Therefore, in worst case the complexity is $O(c)$ for each point. In the outlier detection step, the cluster-based pruning takes $O(c^2)$.

One more pass is required for the remaining points to compute the density and the outlierness score. Assuming the number of remaining points is l, the density computation takes $O(lc)$, while the outlierness score computation takes $O(l \log N)$, where $O(\log N)$ comes from the priority queue operation for maintaining the top-N outlierness score points. Therefore, the overall computation costs for a batch of newly arriving data points is $O(mc) + O(c^2) + O(lc) + O(l \log N)$. In summary, the time complexity of our KDE based outlier detection approach is linear in the number of points. Since typically $N \ll c \ll l \ll m$, the complexity is dominated by the clustering step.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup & Methodologies

In this section, we compare the effectiveness and efficiency of KELOS against the state-of-art local outlier detection approaches. All experiments are conducted on an Ubuntu server with 56 Intel(R) Xeon(R) 2.60GHz cores and 512GB memory. Overall, our KELOS is 2-3 orders of magnitude faster, and always as accurate or more accurate than alternative approaches.

Datasets. We work with 3 labeled public datasets. The **HTTP** dataset [1] contains in total 567,479 network connection records. The labeled outliers correspond to different types of network intrusions including DOS, R2L, U2R, etc. Three numerical attributes, namely duration, src_bytes and dst_bytes are utilized in our experiments. The points in the HTTP dataset are ordered by their arrival time. This makes it possible to create a data stream from this data simply by enforcing a sliding window over time.

The Yahoo! Anomaly Dataset (**YAD**) [13] is considered as one of the industry standards for outlier detection evaluation. It is composed of 4 distinct data sets. In this work we utilize Yahoo! A1 and Yahoo! A2. Yahoo! A1 is based on the real production traffic to some of the Yahoo! services. The anomalies are marked by Yahoo! domain experts. Yahoo! A2 is a synthetic data set containing time-series with random seasonality, trend and noise. Yahoo! A1 and Yahoo! A2 contain 94,866 and 142,101 points. Each data point has three attributes: timestamp, value, and label.

Comparative Methods. We compare KELOS against three baselines, namely LOF [6] , KDEOS [20] and KDEOS with sampling (KDEOS_S). LOF is the seminal and most popular local outlier detection method. KDEOS leverages KDE in local density estimation which is then used to compute an outlierness score for each point. In KDEOS every data point in the input dataset is used as kernel center. The sampling-based KDEOS (KDEOS_S) instead only uses the data points uniformly sampled from the input dataset as kernel centers. Since all three baseline methods were designed for static datasets, we adapt them to streaming by running them window by window. All methods continuously return the N points with the highest outlierness scores as outliers in each window. Since their performance bottleneck is the kNN search, we implemented the skyband stream kNN search algorithm [25] to speed up the outlier detection process.

Efficiency Measures. We evaluate the end-to-end execution time.

Effectiveness Measures. We measure the effectiveness using the *Precision@N* ($P@N$) metric typical for the evaluation of outlier detection techniques [7].

$$P@N = \frac{\text{\# of True Outliers}}{N}. \tag{12}$$

Intuitively, $P@N$ measures the true positive of the detected top-N outliers. An ideal $P@N$ equals to 1, where all outliers are found and no inlier is returned as result. Here we measure the $P@N$ metric window by window and report the average $P@N$ over all windows. To account for the skewed distribution of outliers over different windows, we replace N with $|O|$ in the $P@N$ computation for each window, where $|O|$ corresponds to the total number of labeled (ground truth) outliers falling in this window. Only the top-$|O|$ points out of the top-N outlier list are used in the evaluation. Therefore, the $P@|O|$ for n consecutive stream windows is:

$$P@|O| = \frac{\sum_{i=1}^{n} \text{\# of True Outliers in top-}|O|_i}{\sum_{i=1}^{n} |O|_i}, \tag{13}$$

where $|O|_i$ denotes the number of true outliers in the ith window.

6.2 Efficiency Evaluation

We evaluate the end-to-end execution time by varying the number of neighbors k and the window size.

Number of Neighbors k. The k parameter defines the number of neighbors to be considered in the computation of outlierness score for each point. The radius threshold θ of KELOS for HTTP, Yahoo! A1, and Yahoo! A2 are set as 0.095, 0.1 and 40. The window sizes of HTTP, Yahoo! A1, and Yahoo! A2 are set as 6,000, 1,415, and 1,412 respectively. The sampling rates of KDEOS_S is set as 10% which is a relatively high sampling rate ensuring that KDEOS_S always has more than k kernel centers to use as k increases.

As shown in Fig. 6 (a), KELOS is about 2 orders of magnitude faster than the alternatives. The line of KELOS stops at 800, because KELOS uses cluster-based aKDE approach. The number of the kernel centers is restricted by the number of the clusters. Similarly, the line of KDEOS_S stops at 1,000 due to the limited number of samples. Among these algorithms LOF and KDEOS are the slowest and have the similar performance, because their time complexities are both quadratic in the number of points in each window. KDEOS_S is much faster than KDEOS and LOF, because KDEOS_S only utilizes the sampled points as kernel centers. Searching for the k nearest kernel centers from the sampled kernel center set is much faster than searching among all points in each window. However, KDEOS_S is still at least 1 order of magnitude slower than KELOS on HTTP. This is because in order to satisfy the accuracy requirement, the number of the sampled kernel centers has to be large enough to represent the distribution of the data stream. While the aKDE approach of KELOS only uses the centroid of each cluster as abstract kernel center. Therefore, the number of the clusters tends to be much smaller than the number of the sampled kernel centers. Furthermore, KELOS effectively prunes most of the inliers without conducing the expensive density estimation, while in contrast, KDEOS_S has to compute the outlierness score for each and every data point.

As shown in Fig. 6(b), although KDEOS_S is faster than KELOS on Yahoo! A1 due to the smaller population of the sampled kernel centers, KELOS outperforms KDEOS_S in the effectiveness measurements (Fig. 8(b)). On average, KELOS keeps slightly more kernel centers in the memory than KDEOS_S for Yahoo! A2. However, KELOS still outperforms KDEOS_S on execution time because of our inlier pruning strategy. In general, KELOS outperforms KDEOS and LOF by 1-2 orders of magnitude on Yahoo! datasets.

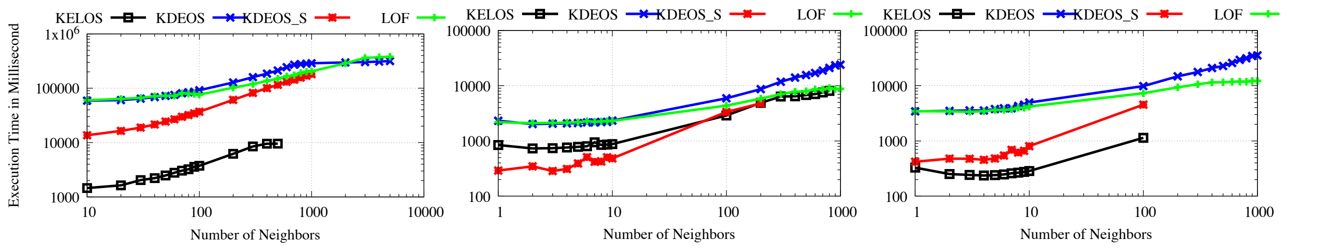
Window Size. The radius threshold θ of KELOS and the sampling rates of KDEOS_S are set the same as the previous experiments. The number of neighbors k is set as 100. As shown in Fig. 7, KELOS consistently outperforms others except on Yahoo! A1 when the window size is extremely small. In that particular case, KELOS is slower than KDEOS_S because there are not many similar objects within each small window resulting in relatively more abstract kernel centers than the sampled kernel centers in KDEOS_S. However, under this setting, KELOS achieves much better accuracy than KDEOS_S. Furthermore, as shown in Figs. 7(a) and (c) the execution time of KELOS is stable as the window size increases. This confirms the scalability of KELOS to large volume stream data.

6.3 Effectiveness Evaluation

Number of Neighbors k. The parameter settings are the same to the efficiency evaluation when varying k. Table 1 shows the peak $P@|O|$ for each approach on each dataset. KELOS outperforms all other approaches in all cases.

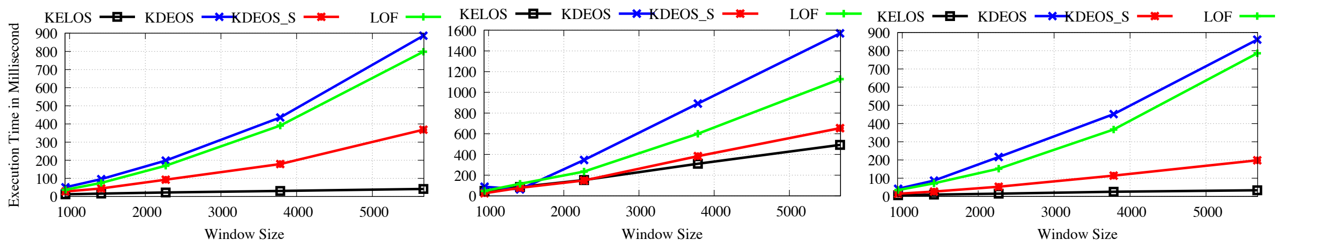
Fig. 8 further demonstrate the trend of $P@|O|$ as k varies. Fig. 8(a) shows the results on the HTTP dataset. For our KELOS, as k increases, the $P@|O|$ increases until k reaches 80. It then starts decreasing after k is larger than 100. Overall KDEOS, KDEOS_S, and LOF show the similar trend. Compared to KELOS they have to use a much larger k to get relative high accuracy. Interestingly, KDEOS_S has similar $P@|O|$ with KDEOS. This shows that sampling-based KDE works well on large datasets.

The trends on the Yahoo! A1 and A2 datasets are different from that on the HTTP dataset as shown in Fig. 8 (b) and (c). The $P@|O|$ continuously increases and gets stable after k reaches certain value. Furthermore, we observe that KDEOS_S works poor on Yahoo! A1 dataset. The reason is that the Yahoo! A1 and A2 datasets are



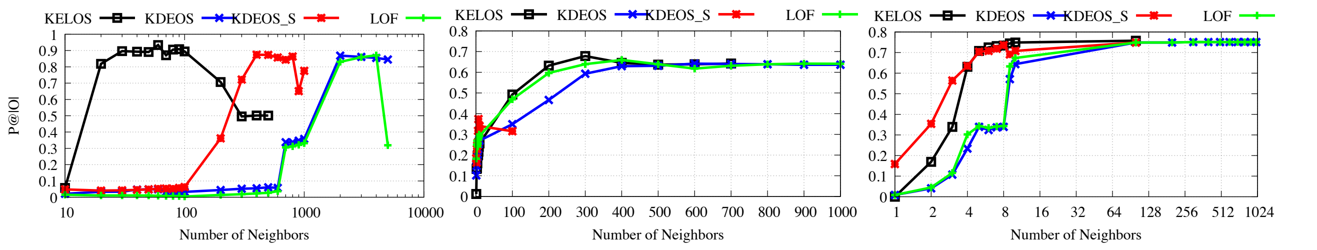
(a) HTTP **(b) Yahoo! A1** **(c) Yahoo! A2**

Figure 6: Execution time of varied number of neighbors k. Note the maximum k that each method can reach is different. For LOF, it depends on the total number of data points. For KDE-based methods, it depends on the number of kernel centers available.



(a) HTTP **(b) Yahoo! A1** **(c) Yahoo! A2**

Figure 7: Execution time on real datasets by varying window size.

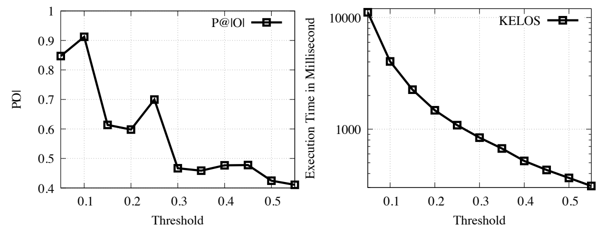


(a) HTTP **(b) Yahoo! A1** **(c) Yahoo! A2**

Figure 8: $P@|O|$ of varied number of neighbors k.

Table 1: Peak accuracies among various k.

	HTTP	Yahoo! A1	Yahoo! A2
LOF	87.06%	65.97%	75.11%
KDEOS	86.88%	64.17%	75.11%
KDEOS_S	87.43%	37.39%	74.89%
KELOS	93.40%	67.83%	75.75%



(a) Accuracy **(b) Execution Time**

Figure 9: Varying radius threshold θ.

relatively small. The samples drawn from small dataset often are not sufficient to represent the distribution of the whole dataset.

6.4 Parameter Tuning

In KELOS, the radius threshold θ defines the maximum possible radius of the formed clusters, that is, the tightness of the clusters. Since the effectiveness of our aKDE approach (Sec. 4) and the pruning strategy (Sec. 4) rely on the tightness of the clusters, θ is important for the accuracy of KELOS. In this set of experiments, we vary θ from small to large on the large HTTP dataset that contains multiple data clusters. As shown in Fig. 9 (a), when θ is at 0.1, the $P@|O|$ is at the peak. Then $P@|O|$ of KELOS starts to decrease gradually as θ increases. Large θ results in a small number of clusters that have large radius. Potentially the centroid of a large radius cluster might not precisely represent all points in the cluster. This leads to inaccurate density estimation. Furthermore, a larger radius causes looser upper and lower KLOME bound. This makes the inlier pruning less effective. However, a smaller radius θ inevitably leads a large number of clusters. This increases the computation costs of both stream clustering and the cluster-based aKDE method as shown in Fig. 9 (b). Therefore, KELOS will achieve the best performance when radius θ is set to the largest value that is still 'small' enough to generate tight data clusters.

7 RELATED WORK

Local Outlier Factor. Local outlier detection has been extensively studied in the literature since the introduction of the Local Outlier Factor (LOF) semantics [6]. A detailed survey of LOF and its variations can be found in [7]. The concept of local outlier, LOF in particular, has been successfully applied in many applications [7]. However, LOF requires kNN search for each and every data point and needs multiple iterations over the entire dataset to compute these LOF values. For this reason, to support continuously evolving streaming data, [17] studied how to quickly find the points whose

LOF scores are influenced by new arrivals or expired data to avoid re-computing the LOF score for each point as the window slides. However as the velocity of the stream increases, most of the points in a window will be influenced. Therefore this approach does not scale to high volume streaming data. In [18] an approximation approach was designed to support LOF in streaming data that focuses on the memory efficiency. However, the more important problem in stream mining, namely the CPU efficiency, was overlooked, which now is instead the focus of our work.

Efficient Kernel Density Estimation. Kernel density estimation is considered as a quadratic process $O(nm)$ with n the total number of data points and m the number of kernel centers. Previous efforts have been made to accelerate this process while still providing accurate estimation, such as utilizing sampling [22]. [12, 26] designed a method that incrementally maintains a small and fixed size of kernel centers to perform density estimation over data streams. However, to ensure the accuracy of density estimation over skewed dataset, the sample size has to be large. Therefore it cannot solve the efficiency problem of KDE in our context. [10] studied the density-based classification problem. It proposed a pruning method that correctly classifies the data without estimating the density for each point by utilizing a user-defined density threshold. However, this pruning method can not be applied to solve our problem, since a point with low density is not necessarily an outlier based on the local outlier semantics we target on.

Outlier Detection using KDE. For each point in the current window of a sliding window stream, [23] utilizes KDE to approximate the number of its neighbors within a certain range. This information is then utilized to support distance-based outlier detection and LOCI [16]. It directly applies off-the-shelf KDE method on each window. No optimization technique is proposed to speed up KDE in the streaming context. [14] is the first work that studied how to utilize KDE to detect local outliers in static dataset. This later was improved by [20] to be better aligned with LOF semantic. Each data point's density is estimated based upon the surrounding kernel centers only, therefore called local density. Instead of considering outliers only based on their density value, data points are measured based on the density in contrast to their neighbors. However, this work does not focus on improving the efficiency of KDE. Nor it considers streaming data. As confirmed in our experiment (Sec. 6.2), it is orders of magnitude slower than our KELOS.

Other Streaming Outlier Detection Approaches. LEAP [8] and Macrobase [5] scale distance-based and statistical-based outlier detection respectively to data streams. They rely on the absolute density at each point to detect outliers, while we work on the local outlier method which determines whether a point is an outlier based on the density relative to its neighbors and tends to be more effective than the absolute density-based methods [9]. Streaming HS-Trees [24] detects outliers by using a classification forest containing a set of randomly constructed trees. The points falling in the leafs that contain small number of points are considered as outliers. Similar to [5, 8], this method also relies on absolute density of each point to detect outliers. RS-Hash [19] proposed an efficient outlier detection approach using sample-based hashing and ensemble. However, different from our local outlier mining problem, it focuses on the problem of subspace outlier detection, that is, detecting outliers hidden in different subspaces of a high dimensional dataset.

8 CONCLUSION

We present the first solution called KELOS for continuously monitoring top-N KDE-based local outliers over sliding window streams. First, we propose the KLOME semantics, effective in measuring the outlierness scores of streaming data. Second, continuous detection strategy is devised that efficiently supports the KLOME semantics by leveraging the key properties of KDE. Using real world datasets we demonstrate that KELOS is 2-3 orders of magnitude faster than the baselines, while being highly effective in detecting outliers from data stream.

REFERENCES

[1] KDD Cup dataset. *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92. VLDB Endowment, 2003.

[3] S. M. Al Pascual, Kyle Marchini. Identity fraud: Securing the connected life. 2017.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[5] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. Macrobase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[6] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000.

[7] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 4(30):891–927, 2016.

[8] 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 *ICDE*, pages 76–87. IEEE, 2014.

[9] C. C. A. S. Edition. *Outlier Analysis*. Springer, 2017.

[10] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *SIGMOD*, pages 945–959, 2017.

[11] F. J. G. Gisbert. Weighted samples, kernel density estimators and convergence. *Empirical Economics*, 28(2):335–351, 2003.

[12] C. Heinz and B. Seeger. Cluster kernels: Resource-aware kernel density estimators over streaming data. *TKDE*, 20(7):880–893, July 2008.

[13] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *SIGKDD*, pages 1939–1947, 2015.

[14] L. J. Latecki, A. Lazarevic, and D. Pokrajac. Outlier detection with kernel density functions. In *MLDM*, pages 61–75. Springer, 2007.

[15] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, SIGMOD 2005, pages 311–322, 2005.

[16] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. LOCI: fast outlier detection using the local corr. integral. In *ICDE*, pages 315–326, 2003.

[17] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *CIDM*, pages 504–515. IEEE, 2007.

[18] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang. Fast memory efficient local outlier detection in data streams. *TKDE*, 28(12):3246–3260, 2016.

[19] S. Sathe and C. C. Aggarwal. Subspace outlier detection in linear time with randomized hashing. In *ICDM*, pages 459–468. IEEE, 2016.

[20] E. Schubert, A. Zimek, and H.-P. Kriegel. Generalized outlier detection with flexible kernel density estimates. In *SDM*, pages 542–550. SIAM, 2014.

[21] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *J. Royal Stat. Soc.*, pages 683–690, 1991.

[22] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[23] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198. VLDB Endowment, 2006.

[24] S. C. Tan, K. M. Ting, and T. F. Liu. Fast anomaly detection for streaming data. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1511, 2011.

[25] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[26] A. Zhou, Z. Cai, L. Wei, and W. Qian. M-kernel merging: Towards density estimation over data streams. In *DASFAA*, pages 285–292. IEEE, 2003.

[27] D. Zill, W. S. Wright, and M. R. Cullen. *Advanced engineering mathematics*. Jones & Bartlett Learning, 2011.