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 of the detection algorithms and their volatility to data updates. To tackle these shortcomings, we propose local outlier semantics that operate at an abstraction level by leveraging 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 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) strategy. *a*KDE accurately yet efficiently estimates the data density at each point – essential 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 their data density and of 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 6 orders of magnitude faster than existing solutions, while 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 relative to each other as it changes over time.

State-of-the-Art. To satisfy this need, several methods [20, 21] 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 with the points in its local neighborhood as opposed to using a global density [9] or frequency [5] cutoff threshold to detect outliers [11]. 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 [20, 21] 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 [20]. 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 [20, 21] 10 minutes to process just 100k tuples as shown by their experiments. Intuitively, kernel density estimation (KDE) [26], an established probability density approximation method, could be leveraged for estimating the data density at each point [16, 23, 27]. 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 [26]. While the computational costs can be reduced by running the density estimation on 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

*This work was partly supported by the National Science Foundation under grants IIS-1815866 and IIS-1560229.

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.

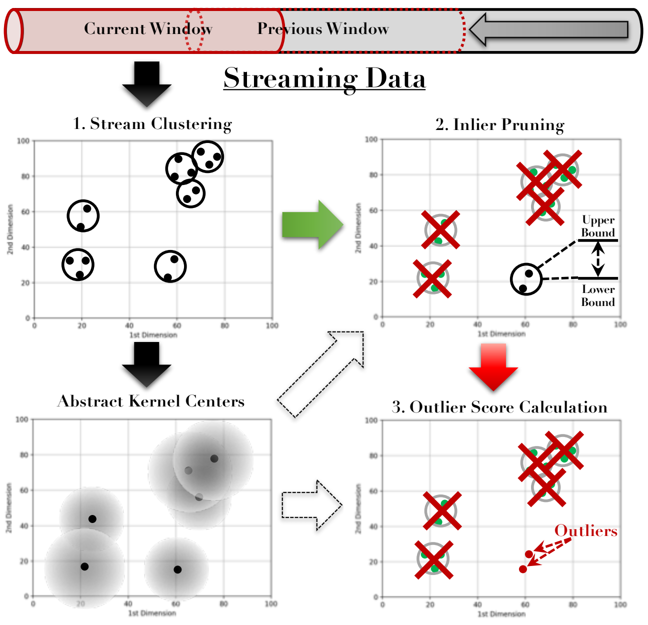


Figure 1: An illustration of KELOS approach.

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 accurate 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.

• Unlike existing techniques [20, 21], 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 6 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.

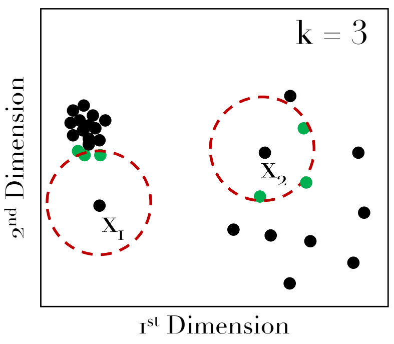
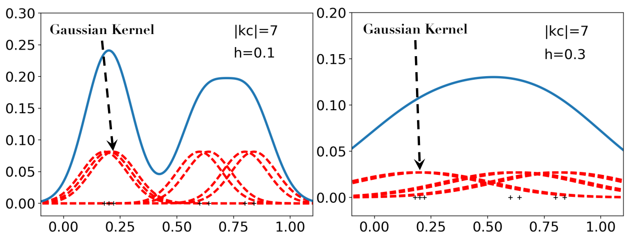


Figure 2: Local outlier detection using local densities.

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



(a) Bandwidth h = 0.1 **(b) Bandwidth h = 0.3**

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:

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

Kernel Centers. $kc_j \in \mathbb{KC}$ where $1 \leq j \leq m$ 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 [26]. 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 [26]. The most commonly used ones are the *Gaussian* and *Epanechnikov* kernel functions [11]. In this study, we adopt the *Gaussian* kernel:

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

$$K_{epanechnikov}(u) = \frac{3}{4h}\left(1 - \frac{u^2}{h^2}\right), \tag{3}$$

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* h 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* [30], 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 [23]:

$$\tilde{f}(x_i) = \frac{1}{k}\sum_{j=1}^{k} K_h(|x_i - kc_j|) \text{ where } kc_j \in k\text{NN}(x_i, \mathbb{KC}). \tag{4}$$

Multi-dimensional Kernel. We adopt the *product kernel* [24] as the form of the kernel function. The product kernel is typical in density estimation for multi-dimensional data. Given a d dimensional target point x_i and a kernel center kc_j, for each dimension l the product kernel (Eq. 5) first computes the density contribution of kc_j to x_i based on the distance on dimension l. The final density contribution by kc_j to x_i is the product of the density contributions by kc_j on all dimensions, so called product kernel. As we will show in Sec. 7, this creates opportunities for the design of constant time density update operation in the streaming context. Moreover, product kernel allows the bandwidth to be customized for each dimension, resulting in more accurate estimation [24].

$$\tilde{f}(x_i) = \sum_{j=1}^{k}\prod_{l=1}^{d} K_{h^l}(|x_i^l - kc_j^l|) \text{ where } kc_j \in k\text{NN}(x_i, \mathbb{KC}). \tag{5}$$

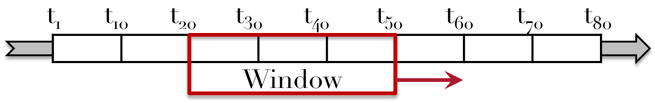


Figure 4: A data stream in the form of sliding windows.

3 PROPOSED OUTLIER SEMANTICS

Next, we propose our 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 a new outlierness measure, KDE-based local outlierness measure (KLOME). The goal is to reduce the computation costs of the existing KDE-based outlier semantics [23], while still effective in detecting outliers.

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-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-score*$(s, S) = (s - \bar{S})/\sigma_S$ [33] indicates how many standard deviations a 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. This in turn significantly reduces the computational complexity compared to working with x_i's kNNs in the much larger input dataset X.

Next, we define the 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) $\leq$ KLOME(x_j).

3.2 Local Outlier Detection in Sliding Window

We work with periodic sliding window semantics, illustrated in Fig. 4, 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 framework for supporting continuous local outlier detection over windows stream with our proposed KLOME semantics.

4 THE KELOS FRAMEWORK

KELOS framework, depicted in Fig. 5, consists of three main components, namely, **stream data abstractor**, **density estimator** and **outlier detector**.

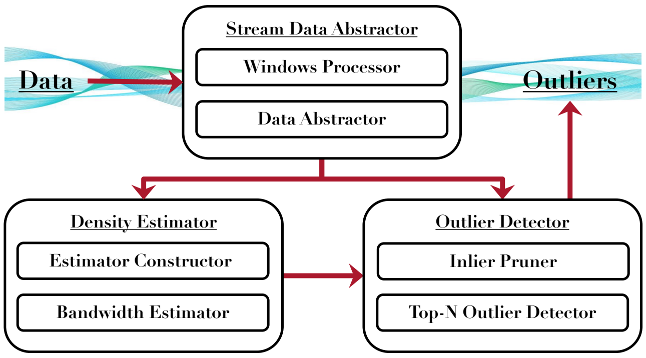


Figure 5: The KELOS framework.

The **stream data extractor** is composed of the *window processor* and the *data abstractor*. The *window processor* feeds the latest data that falls into the current stream window to the system. By leveraging a lightweight stream clustering approach, the *data abstractor* dynamically organizes the data points in each window into evolving clusters based on their affinity. It then generates and maintains statistics that reflect the key distribution properties of each cluster. These statistics are essential to performing density estimation and the inlier pruning during outlier detection.

The *estimator constructor* in the **density estimator** builds kernel density estimators utilizing *abstract kernel centers* where each abstract kernel center represents one data cluster. The *bandwidth estimator* leverages the statistics associated with each cluster to approximate an optimal bandwidth for each density estimator customized for each target. The constant time complexity of the bandwidth estimator ensures that the bandwidth can be continuously updated online to best fit the data.

The **outlier detector** continuously computes the top-N outliers, that is, the N points with the highest outlierness scores. It avoids having to compute the density and the outlierness score for each and every point by pruning clusters that as a whole do not have a chance to contain any outlier. This leverages the *stable density* property of a tight cluster and the characteristics of local outliers (Sec. 6.1).

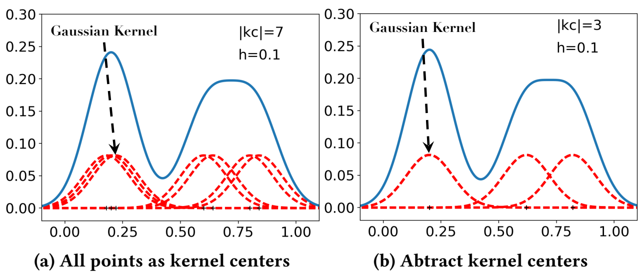
In Sec. 5, we present the key strategies of our **density estimator**, namely abstract kernel center-based KDE. In Sec. 6, we introduce the techniques core to our **outlier detector**. It efficiently identifies and prunes the points that are guaranteed not to be outliers. Finally, in Sec. 7, we introduce our **stream data abstractor**. It features a low complexity dual-purpose clustering algorithm that continuously constructs data clusters, while at the same time generating the statistics needed to support the density estimator and outlier detector.

5 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 [26], 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 groups 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. 6(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. 6(b) with 3 centers and Fig. 6(a) using all 7 points as kernel centers are similar.



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

Figure 6: 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 because of the reduced range of possible k.

Definition 5.1. 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 $\forall i, j, i \ne j \; c_i \cap c_j = \emptyset$.

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 [13], 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:

$$\tilde{f}_{\mathbb{AKC}(S^{W_c})}(x_i) = \sum_{j=1}^{k} \omega(c_{c_j}) \prod_{l=1}^{d} K_{h^l}(|x_i^l - c_{c_j}^l|) \tag{6}$$

where

$$\omega(c_{c_j}) = \frac{|c_j|}{\sum_{m=1}^{k} |c_m|}, \tag{7}$$

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. 6 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 kernel on each dimension. Here we show that the data driven strategy introduced in [25] (Eq. 8) can be efficiently applied here by leveraging the abstract kernel centers.

$$h^l = 1.06\sigma^l k^{-1/(d+1)}. \tag{8}$$

In Eq. 8, d denotes the data dimension. σ^l denotes the weighted standard deviation of the kernel centers on the lth dimension computed by:

$$\sigma^l = \sqrt{\sum_{m=1}^{k} \omega(c_{c_m})(c_{c_m}^l - \mu^l)^2}, \tag{9}$$

where

$$\mu^l = \frac{\sum_{m=1}^{k} \omega(c_{c_m}) c_{c_m}^l}{k}, \tag{10}$$

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

Efficiency. The time complexity of KDE is $O(nm)$, where 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. 8.2.

6 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 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.

6.1 Bounding the KLOME Scores

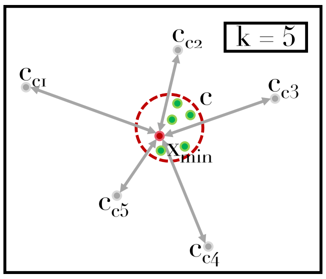
Next, we present an efficient strategy to establish the upper and lower KLOME bounds for each given data cluster.

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}.

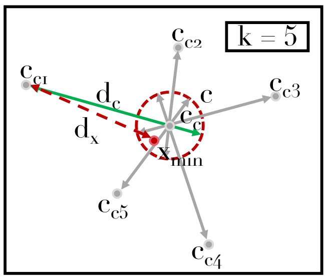
Algorithm 1 : Top-N Outlier Computation.

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
    ascending 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 $R$
```



(a) Lowest KLOME in c



(b) Lower bound of c

Figure 7: An example of lower KLOME bound.

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. 7).

Lemma 6.1. *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, $\tilde{f}_{min}(c_i) \le \tilde{f}(x_{min})$, 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 $d_x = |c_{c_j} - x_{min}|$. $d_c \ge d_x$ by the triangle inequality. Therefore $K_h(d_x) \le K_h(d_c)$. This holds for any kernel center c_{c_j}. Therefore $\tilde{f}_{min}(c_i) = \sum_{j=1}^{k} \omega(c_{c_j})K_h(|c_{c_j} - c_{c_i}| + r) \le \tilde{f}(x_{min})$. □

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 6.1, 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:

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

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.

$$\tilde{f}_{max}(c_i) = \sum_{j=1}^{k} \omega(c_{c_j})K_h(|c_{c_j} - c_{c_i}| - r). \quad (12)$$

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

$$KLOME_{up}(c_i) = z\text{-}score(\tilde{f}_{max}(c_i), \{\tilde{f}(c_{c_1}) \cdots \tilde{f}(c_{c_k})\}). \quad (13)$$

7 THE EFFICIENT STREAM DATA ABSTRACTOR

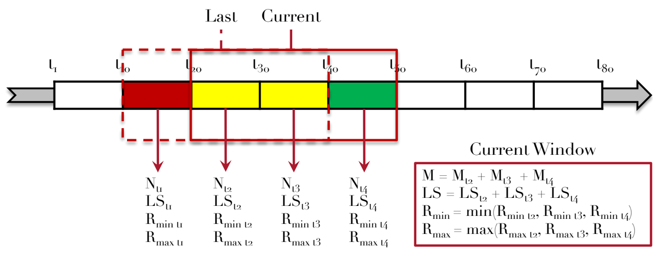
The stream data abstractor adapts a lightweight stream clustering algorithm similar to [2, 8] that clusters the extremely close data points together. As the clusters are continuously constructed and incrementally maintained, the statistics needed by both *a*KDE and inlier pruning, namely the *cardinality*, the *centroid*, and the *radius* of the cluster, must also be continuously generated. We thus refer to this as dual-purpose clustering. The dual-purpose clustering is based on two key ideas: *additive meta data* and *pane-based meta data maintenance.*

The **additive meta data** is inspired by *micro-clustering* [2] – a popular stream clustering approach. The idea is that by maintaining meta data that satisfies the additive properties, the statistics required by both the density estimator and the outlier detector can be computed in constant time whenever the window evolves.

Definition 7.1. A **cluster** c_i in a d-dimensional data set $S_{W_c} = \{x_1, \cdots, x_m\}$ corresponding to the data in the current window W_c of stream S is represented as a 4-tuple set $\{M, LS, R_{min}, R_{max}\}$ where M denotes the cardinality of the cluster, $LS = < \sum_{i=1}^{m} x_i^1, \cdots, \sum_{i=1}^{m} x_i^d >$ is the linear sum of the points by dimension, $R_{min} = < x_{min}^1, \cdots, x_{min}^d >$ and $R_{max} = < x_{max}^1, \cdots, x_{max}^d >$ are the minimum and maximum values of the points in each dimension.

Cardinality and Centroids for *a*KDE. In Def. 7.1, M refers to data cardinality of cluster c_i. M is used to compute the *weight* (Eq. 7) and the *centroid* of the abstract kernel center. The linear sum LS is used to compute the *centroid* of cluster $c_i = \frac{LS}{M}$.
The Radius for Inlier Pruning. R_{min} and R_{max} representing the minimal and maximal values in each dimension are utilized to compute the radius of cluster c_i. Radius is a key statistic needed by our outlier detector to quickly prune the clusters from the outlier candidate set.

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



Last
Current
t_1
t_{10}
t_{20}
t_{30}
t_{40}
t_{50}
t_{60}
t_{70}
t_{80}
N_{t1} LS_{t1} $R_{min\ t1}$ $R_{max\ t1}$
N_{t2} LS_{t2} $R_{min\ t2}$ $R_{max\ t2}$
N_{t3} LS_{t3} $R_{min\ t3}$ $R_{max\ t3}$
N_{t4} LS_{t4} $R_{min\ t4}$ $R_{max\ t4}$
Current Window
$M = M_{t2} + M_{t3} + M_{t4}$
$LS = LS_{t2} + LS_{t3} + LS_{t4}$
$R_{min} = \min(R_{min\ t2}, R_{min\ t3}, R_{min\ t4})$
$R_{max} = \max(R_{max\ t2}, R_{max\ t3}, R_{max\ t4})$

Figure 8: An example of an evolving cluster.

to find the point "furthest" from the new centroid. This, being computational expensive, is not acceptable in online applications.

The remedy comes from our carefully selected product kernel function (Eq. 5). In the product kernel, each dimension has its own customized bandwidth. Accordingly, we only need the radius on each single dimension to estimate the bandwidth instead of the radius over the multi-dimensional data space. Since updating the minimum or maximum value per insertion or deletion to an array is constant cost, the cost of radius maintenance for k separated dimensions (R_{min} and R_{max}) is constant as well.

Pane-based meta data maintenance. The pane-based meta data maintenance strategy [17] 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. Since the data points in the same pane arrive and expire at the same slide pace, the meta data can be quickly computed by aggregating the meta data structures maintained for the unexpired panes as the window moves. This process is illustrated in Fig. 8. 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. Therefore, our stream clustering algorithm only needs to exclusively deal with the new arrivals.

The Dual-Purpose Stream 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. The corresponding *4-tuple* meta data is updated accordingly. On the other hand, if $dist(x, c_{c_i}) > \theta$, a new cluster will be created. This logic is described in Alg. 2.

7.1 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

Algorithm 2 : Dual-Purpose Stream Clustering

Input: Data batch $\mathcal{D}$ at time t_i, window size $S.win$ and threshold θ.
Output: Clusters C.

```
1:  $Array\langle Cluster\rangle\ C$
2:  for every data $d$ in $\mathcal{D}$ do
3:      $dist_{min} \leftarrow +\infty$
4:      $Cluster\ c_n$
5:      for every cluster $c$ in $C$ do
6:          $dist \leftarrow distance(d, c.centroid_{t_{i-S.win+1} \to t_i})$
7:          if dist < $dist_{min}$ then
8:              $dist_{min} \leftarrow dist$
9:              $c_n \leftarrow c$
10:     if $dist_{min}$ < threshold $\theta$ then
11:         $c_n$.insert($d$,$t_i$)
12:     else
13:         Cluster $c_{new}$
14:         $c_{new}$.insert($d$,$t_i$)
15:         $C$.add($c_{new}$)
16: return $C$
```

$O(\log N)$ 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.

8 EXPERIMENTAL EVALUATION

8.1 Experimental Setup & Methodologies

In this section, we compare the efficiency and effectiveness of KELOS against the state-of-art local outlier detection approaches for data streams. All experiments are conducted on an Ubuntu server with 56 Intel(R) Xeon(R) 2.60GHz cores and 512GB memory. Overall, our KELOS is 1-6 orders of magnitude faster, while still at least as accurate as the alternative approaches.

We utilize the public as well as synthetic datasets generated using the data mining framework ELKI [1] to measure the efficiency of KELOS. We also work with real labeled datasets so that we can evaluate the effectiveness of KELOS.

Real Datasets. We work with 3 labeled public datasets. The **HTTP** dataset [10] contains in total 567,479 network connection records with 2,211 among them being outliers. 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. Therefore we can generate a data stream simply by enforcing a sliding window on it.

The Yahoo! Anomaly Dataset (**YAD**) [15] 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 points with 1,669 outliers and 142,101 points with 312 outliers respectively. Each data point has three attributes: timestamp, value, and label.

Synthetic Datasets. We generate synthetic datasets to measure the efficiency of KELOS under various scenarios with different distribution properties. We first create static datasets containing different number of data clusters and then utilize these datasets to simulate windowed streams. For example, to simulate a windowed stream with three clusters and some outliers, we first create a static dataset containing three clusters. The size and shape of the clusters are controlled by different parameter settings in ELKI such as type of distribution, standard deviation, etc. When generating a sliding window data stream, different windows correspond to different datasets with different cluster densities.

Comparative Methods. We compare KELOS against five baselines, namely sLOF, sKDEOS, pKDEOS, iLOF [20] and MiLOF [21]. LOF [6] is the seminal and most popular local outlier detection method. KDEOS [23] leverages KDE in local density estimation which is then used to compute an outlierness score for each point. Since their performance bottleneck is the *k*NN search, we implemented the skyband stream *k*NN search algorithm [29] to speed up their outlier detection process for windowed streams and named the modified methods as sLOF and sKDEOS. iLOF [20] and MiLOF [21] are two incremental LOF algorithms specifically designed for *landmark* windows. iLOF computes the LOF score for each inserted data point and update the LOF score of the affected data points following the reverse *k*NN relationships. MiLOF improves iLOF in a scenario where the memory is limited. When the memory limit is reached, part of the existing data points are summarized into small clusters as references for the future LOF approximation. Since iLOF and MiLOF do not handle data expiration, they have to start from scratch for each new window. In the original KDEOS, every data point in the input dataset is used as kernel center. We also implemented a sampling-based sKDEOS called pKDEOS, since it is the common practice for KDE to use only the data points uniformly sampled from the input dataset as kernel centers. All methods continuously return the N points with the highest outlierness scores as outliers in each window.

Efficiency Measures. We measure 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].

$$P@N = \frac{\text{\# of True Outliers}}{N}. \tag{14}$$

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. Following [7], 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:

$$P@|O| = \frac{\sum_{i=1}^{n} \text{\# of True Outliers in top-}|O|_i}{\sum_{i=1}^{n} |O|_i}, \tag{15}$$

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

To investigate the quality of the ranking in the Top-$|O|$ outlier list, we also measure the *average precision* (AP) [31]:

$$Average\,Precision(AP) = \frac{1}{|O|}\sum_{o \in O} P@rank(o). \tag{16}$$

Here P@rank(o) measures the $P@N$, where N is set as rank(o) representing the position at which a true outlier o appears. AP measures how well the true outliers are ranked among all data points.

8.2 Efficiency Evaluation

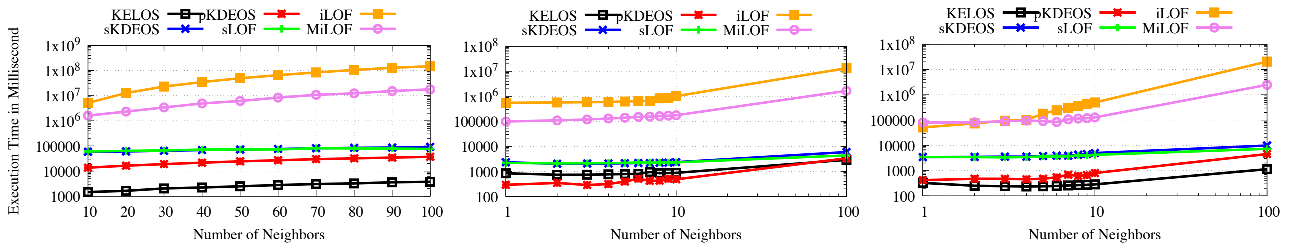
We evaluate the end-to-end execution time and memory consumption by varying the number of the neighbors k, the window size and the window overlap rate.

Number of Neighbors k. The k parameter defines the number of the neighbors to be considered in the computation of outlierness score for each point. We first report the execution time of all methods on real datasets with varied k from 10 to 100. The radius threshold θ of KELOS appropriate for HTTP, Yahoo! A1, and Yahoo! A2 are set as 0.095, 0.1 and 40 (See Sec.8.4 for parameter tuning). The window size of HTTP is set to 6,000 and window sizes of Yahoo! A1 and Yahoo! A2 are set as 1,415, and 1,412 based on the instruction of the data provider for the effectiveness of outlier detection. The sampling rates of pKDEOS is set as 10% which is a relatively high sampling rate ensuring that pKDEOS always has more than k kernel centers to use as k increases. For MiLOF, we configured it to keep 10% of the data in memory for LOF score approximation.

As shown in Fig. 9(a), KELOS is about 2-6 orders of magnitude faster than the alternatives. Among all alternative methods, iLOF and MiLOF are the slowest. To reduce the influence of the new arrivals they use a *point-by-point* processing strategy that computes the LOF score for each new point and update the LOF score of the affected data points immediately after a single insertion. It wastes significant CPU time on unnecessarily updating the LOF scores of some points that are modified again later due to the insertion of other new arrivals. sLOF and sKDEOS are 1-2 orders of magnitude faster than the previous two, as they compute the outlierness score only once for each data point in the entire window. pKDEOS is faster than sKDEOS and sLOF, because pKDEOS 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, pKDEOS 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 *a*KDE 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, others have to compute the outlierness score for each and every data point.

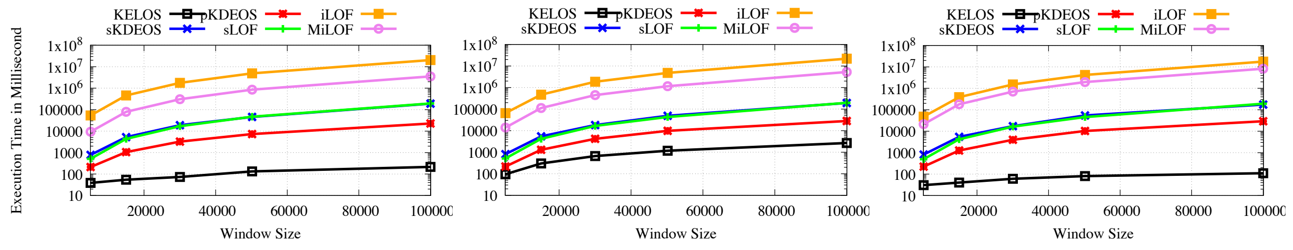
As shown in Fig. 9(b), although pKDEOS is faster than KELOS on Yahoo! A1 due to the smaller population of the sampled kernel centers, KELOS outperforms pKDEOS in the effectiveness measurements (Tab. 1). On average, KELOS keeps slightly more kernel centers in the memory than pKDEOS for Yahoo! A2 (Fig. 12(c)). However, KELOS still outperforms pKDEOS on execution time because of our inlier pruning strategy.

Window Size. To evaluate how window size affects the efficiency of KELOS, we measure and compare the average window



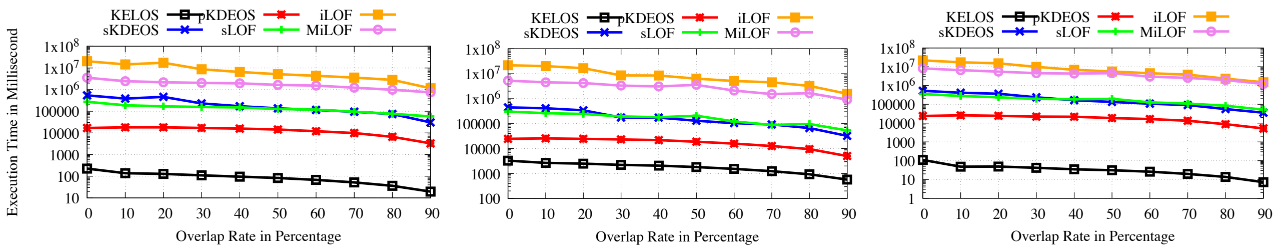
(a) HTTP (b) Yahoo! A1 (c) Yahoo! A2

Figure 9: Execution time of varied number of neighbors k. Note the maximum k that each method can reach is different. For LOF-based methods, it depends on the total number of data points. For KDE-based methods, it depends on the number of the kernel centers available.



(a) One-Cluster Stream (b) Three-Cluster Stream (c) Six-Cluster Stream

Figure 10: Execution time on synthetic datasets of varied window size.



(a) One-Cluster Stream (b) Three-Cluster Stream (c) Six- Cluster Stream

Figure 11: Execution time on synthetic datasets of varied overlap rate.

processing time of each method by varying the window size of synthetic datasets from 5,000 to 100,000. For these experiments, we created three data streams with one, three and six Gaussian clusters with various parameter configurations (mean, variance and etc.) across the timeline and 10 outliers for each window. The parameters of different all the methods are configured such as they achieve the same accuracy ($P@|O| > 0.95$) with as little memory consumption as possible while k=10. As shown in Fig. 10, in general, the execution time increases with the increase of the window size (average data points within a single window). KELOS constantly outperforms all other baselines by 1-5 orders of magnitude.

Window Overlap Rate. We evaluate the efficiency of KELOS by varying synthetic streams' window overlap rate from 0% to 90%. The synthetic streams are created in the same fashion to the previous experiments. The parameters of different all the methods are configured such as they achieve the same accuracy ($P@|O| > 0.95$) with as little memory consumption as possible and k=10. Based on this controlled accuracy, we evaluate the average window processing time.

When the overlap rate increases, the execution time of all six approaches decrease as demonstrated in Fig. 11. This is because iLOF and MiLOF are designed to process the data incrementally. Although iLOF and MiLOF are not capable of handling data pruning, in these experiments, we simply assume that the overlapped data are already processed and we report the their processing times for the rest of the window. sKDEOS, pKDEOS and sLOF all leverage the skyband stream kNN search that incrementally computes the kNN for each point as window slides, while the clustering algorithm used by KELOS is incremental by nature as shown in Sec. 7. As shown in Fig. 11, KELOS is 2-5 orders of magnitude faster than iLOF, MiLOF, sKDEOS, sLOF and pKDEOS in all settings on all three streams. iLOF and MiLOF are the slowest. sLOF and sKDEOS are faster than the previous two but slower than pKDEOS. Although they all can save computation by avoiding some unnecessary reprocessing of the existing data, the reason of the performance difference is the same as we already explained in the previous experiments.

Memory Consumption. The memory consumption shown in Fig. 12 is evaluated by counting the number of the kernel centers and data points kept in the memory by each approach. sKDEOS, iLOF and sLOF (represented by sLOF in Fig. 12) utilizes all points as kernel centers or reference points, while pKDEOS and MiLOF (represented by pKDEOS in Fig. 12) dramatically reduces the number of the kernel centers by sampling and the number of the references by clustering. KELOS uses the smaller number of kernel centers as compared to sLOF which facilitate more accuracte density estimation. The number of the kernel centers is equivalent to the number of the clusters that tends to be small. sLOF and iLOF measures the local density at each point by computing the local reachability density (LRD). Since the LRD computation requires the access to all points falling in each window, it is equivalent to using all points as kernel centers similar to sKDEOS. On synthetic data stream, Similar to the memory consumption on the real HTTP dataset, KELOS uses the least number of kernel centers. pKDEOS and MiLOF uses fewer kernel centers or references than sKDEOS, iLOF and sLOF because of the sampling and clustering.

8.3 Effectiveness Evaluation

We report the accuracies of our KELOS, sKDEOS (≈ pKDEOS) and sLOF methods on the real data streams. Tab. 1 (with pKDEOS included) shows the peak $P@|O|$ and AP for each approach on each dataset. KELOS outperforms all other approaches in all cases. We are not reporting the results of iLOF and MiLOF, since they perform equally to sLOF in effectiveness. Similarly, for the KDE-based methods we only report the results on sKDEOS for various configurations.

Number of Neighbors k. The number of the neighbors k is the most influential factor that affects the detection accuracies of all methods. The parameter settings are the same to the efficiency evaluation when varying k.

Fig. 13& 14 demonstrate the trend of $P@|O|$ and AP as k varies. The line of KELOS stops at 800 in Fig. 13(a)&14(a), because KELOS uses cluster-based *a*KDE approach. The number of the kernel centers is restricted by the number of the clusters. Fig. 13(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 sKDEOS and sLOF show the similar trend. Compared to KELOS they have to use a much larger k to get relative high accuracy. The trends on the Yahoo A1 and A2 datasets are different from that on the HTTP dataset as shown in Fig. 13(b)% 13(c). The $P@|O|$ continuously increases and gets stable after k reaches certain value. This confirms our observation that using as many as possible kernel centers in the density estimator does not always lead to more accurate density estimation. This justifies our decision of adopting the balloon kernel that only takes the close kernels into consideration when estimating the density at a point x.

The trends of AP are similar to the trends of $P@|O|$ on all datasets as shown in Fig. 14. Overall, KELOS is as accurate or more accurate than alternative approaches. Furthermore, compared to the alternatives, KELOS uses a smaller k to achieve high accuracy. This also contributes to the performance gain of KELOS in execution time.

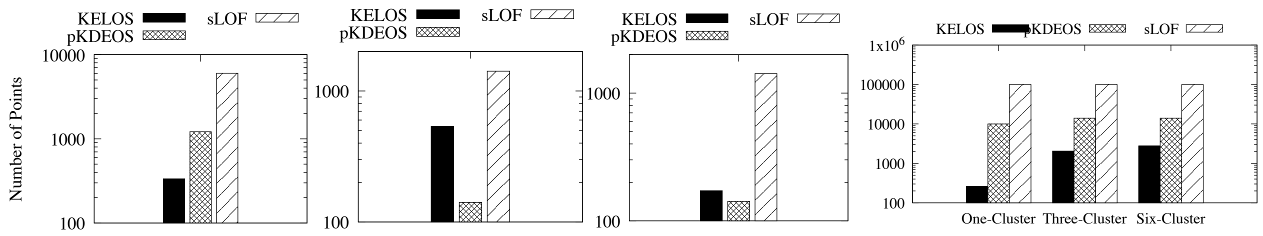
8.4 Hyper Parameter Tuning

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 *a*KDE approach (Sec. 5) and the pruning strategy (Sec. 5) 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. 15(a), when θ is at 0.1, the $P@|O|$ is at the peak. Then $P@|O|$ and AP 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 *a*KDE method as shown in Fig. 15(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.

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. 15(a), 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 based on the user feedback. As concept drift occurs when stream data evolves, the θ is adjusted again using the same principle.

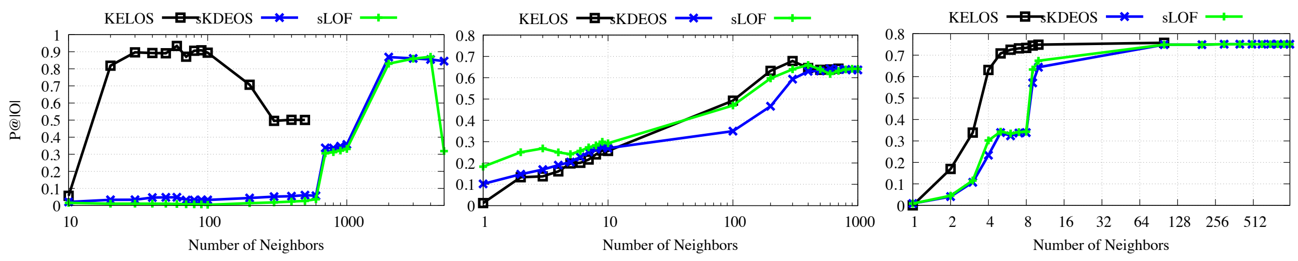
9 RELATED WORK

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 applied in many applications [7]. However, LOF requires kNN search for 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, iLOF was proposed [20] to quickly find the points whose LOF scores are influenced by new arrivals. This avoids 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 [21] an approximation approach MiLOF was designed to support LOF in streaming data that focuses on the memory efficiency. However, MiLOF only considers new incoming data. It does not offer any efficient strategy to handle the update caused by data expiration. Therefore, it is not efficient when handing windowed streams. As evaluated



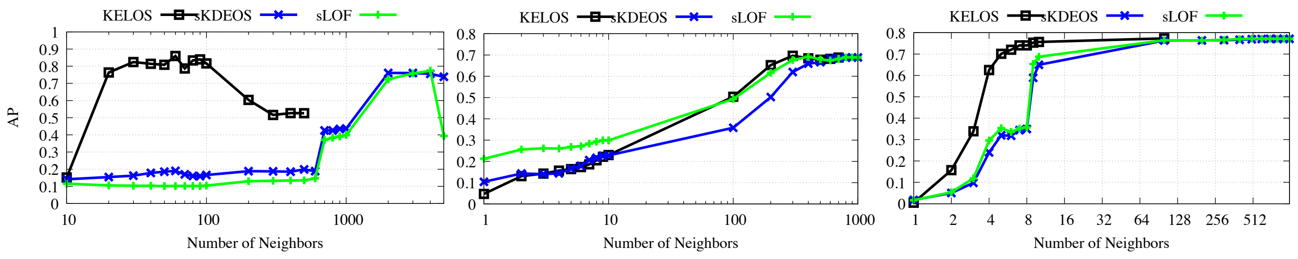
(a) HTTP **(b) Yahoo! A1** **(c) Yahoo! A2** **(d) Synthetic**

Figure 12: Memory consumption for real and synthetic data streams (window size as 100,000).



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

Figure 13: $P@|O|$ of varied number of neighbors k.

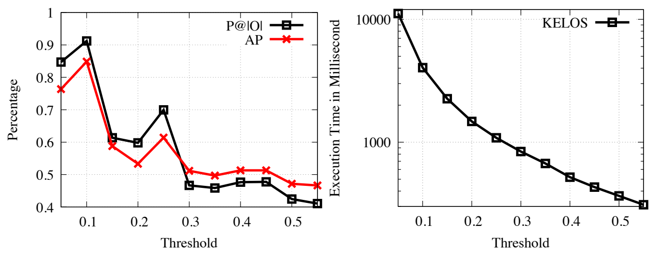


(a) HTTP **(b) Yahoo A1** **(c) Yahoo A2**

Figure 14: AP of varied number of neighbors k.

Table 1: Peak accuracies among various k.

	P@|O|			AP		
	HTTP	Yahoo! A1	Yahoo! A2	HTTP	Yahoo! A1	Yahoo! A2
sLOF	87.06%	65.97%	75.11%	77.34%	69.16%	77.19%
sKDEOS	86.88%	64.17%	75.11%	76.06%	68.84%	76.95%
pKDEOS	87.43%	37.39%	74.89%	77.54%	36.43%	77.10%
KELOS	93.40%	67.83%	75.75%	85.92%	69.64%	77.30%



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

Figure 15: Varying radius threshold θ.

in our experiments, iLOF and MiLOF are much slower than our skyband-based streaming LOF implementation sLOF.

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 aimed to accelerate this process while still providing accurate estimation, such as utilizing sampling [26]. [14, 32] designed a method that incrementally maintains a small, fixed size of kernel centers to perform density estimation over data streams. However, to ensure the accuracy of density estimation over skewed datasets, the sample size has to be large. Therefore it cannot solve the efficiency problem of KDE in our context. [12] 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, [27] 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 [19]. It directly applies off-the-shelf KDE method on each window. No optimization technique is proposed to speed up KDE in the streaming context. [16] is the first work that studied how to utilize KDE to detect local outliers in static datasets. This later was improved by [23] to be better aligned with LOF semantics. 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 improve the efficiency of KDE. Nor does it consider streaming data. As confirmed in our experiment (Sec. 8.2), it is indeed orders of magnitude slower than our KELOS.

Other Streaming Outlier Detection Approaches. LEAP [9] 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 with the local outlier method which determines whether a point is an outlier based on the density relative to its neighbors. It tends to be more effective than the absolute density-based methods [11]. Streaming HS-Trees [28] detects outliers by using a classification forest containing a set of randomly constructed trees. The points falling in the leafs that contain a small number of points are considered as outliers. Similar to [5, 9], this method also relies on absolute density of each point to detect outliers. RS-Hash [22] proposed an efficient outlier detection approach using sample-based hashing and ensemble. However, different from our local outlier mining problem, it focuses on subspace outlier detection, that is, detecting outliers hidden in different subspaces of a high dimensional dataset. Similar to iLOF [20] and MiLOF [21], DILOF [18] processes the data stream in a point-by-point fashion and incrementally detects outliers and sequential outliers from *landmark* windows. It does not handle data expiration which is a required operation in sliding windows scenario.

10 CONCLUSION

We present KELOS – the first solution for continuously monitoring top-N KDE-based local outliers over sliding window streams. First, we propose the KLOME semantics to continuously capture the n points that have the highest outlierness scores in the streaming data. Second, a continuous detection strategy is designed that efficiently supports the KLOME semantics by leveraging the key properties of KDE. Using real world datasets we demonstrate that KELOS is 2-6 orders of magnitude faster than the baselines, while being highly effective in detecting outliers from data streams.

REFERENCES

[1] Elke Achtert, Hans-Peter Kriegel, and Arthur Zimek. 2008. ELKI: A Software System for Evaluation of Subspace Clustering Algorithms. In *Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008, Hong Kong, China, July 9-11, 2008, Proceedings*. 580–585.

[2] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. 2003. A framework for clustering evolving data streams. In *VLDB*. VLDB Endowment, 81–92.

[3] Sarah Miller Al Pascual, Kyle Marchini. 2017. Identity Fraud: Securing the Connected Life. (2017).

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *PODS*. 1–16.

[5] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*. 541–556.

[6] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*. 93–104.

[7] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery* 4, 30 (2016), 891–927.

[8] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *SDM*. SIAM, 328–339.

[9] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A Rundensteiner. 2014. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*. IEEE, 76–87.

[10] The Third International Knowledge Discovery and Data Mining Tools Competition. 1999. KDD Cup Dataset. *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html* (1999).

[11] Charu C. Aggarwal: Second Edition. 2017. *Outlier Analysis*. Springer.

[12] Edward Gan and Peter Bailis. 2017. Scalable Kernel Density Classification via Threshold-Based Pruning. In *SIGMOD*. 945–959.

[13] Francisco J Goerlich Gisbert. 2003. Weighted samples, kernel density estimators and convergence. *Empirical Economics* 28, 2 (2003), 335–351.

[14] C. Heinz and B. Seeger. 2008. Cluster Kernels: Resource-Aware Kernel Density Estimators over Streaming Data. *TKDE* 20, 7 (July 2008), 880–893.

[15] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *SIGKDD*. 1939–1947.

[16] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. 2007. Outlier detection with kernel density functions. In *MLDM*. Springer, 61–75.

[17] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. 2005. 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)*. 311–322.

[18] Gyoung S Na, Donghyun Kim, and Hwanjo Yu. 2018. DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1993–2002.

[19] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. 2003. LOCI: Fast Outlier Detection Using the Local Corr. Integral. In *ICDE*. 315–326.

[20] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. 2007. Incremental local outlier detection for data streams. In *CIDM*. IEEE, 504–515.

[21] Mahsa Salehi, Christopher Leckie, James C. Bezdek, Tharshan Vaithianathan, and Xuyun Zhang. 2016. Fast Memory Efficient Local Outlier Detection in Data Streams. *TKDE* 28, 12 (2016), 3246–3260.

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

[23] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. 2014. Generalized outlier detection with flexible kernel density estimates. In *SDM*. SIAM, 542–550.

[24] David W Scott. 2015. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons.

[25] Simon J Sheather and Michael C Jones. 1991. A reliable data-based bandwidth selection method for kernel density estimation. *J. Royal Stat. Soc.* (1991), 683–690.

[26] Bernard W Silverman. 1986. *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

[27] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. 2006. Online Outlier Detection in Sensor Data Using Non-parametric Models. In *VLDB*. VLDB Endowment, 187–198.

[28] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast anomaly detection for streaming data. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22. 1511.

[29] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. 2002. Continuous Nearest Neighbor Search. In *VLDB*. 287–298.

[30] George R Terrell and David W Scott. 1992. Variable kernel density estimation. *The Annals of Statistics* (1992), 1236–1265.

[31] Ethan Zhang and Yi Zhang. 2009. Average precision. In *Encyclopedia of database systems*. Springer, 192–193.

[32] Aoying Zhou, Zhiyuan Cai, Li Wei, and Weining Qian. 2003. M-kernel merging: Towards density estimation over data streams. In *DASFAA*. IEEE, 285–292.

[33] Dennis Zill, Warren S Wright, and Michael R Cullen. 2011. *Advanced engineering mathematics*. Jones & Bartlett Learning.