A Fast kNN-Based Approach for Time Sensitive Anomaly Detection over Data Streams

Guangjun Wu[1], Zhihui Zhao[1,2], Ge Fu[3](✉), Haiping Wang[1](✉), Yong Wang[1], Zhenyu Wang[1], Junteng Hou[1,2], and Liang Huang[3]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{wuguangjun,zhaozhihui,wanghaiping,wangyong,wangzhenyu, houjunteng}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[3] National Computer Network Emergency Response Technical Team/Coordination, Center of China (CNCERT/CC), Beijing, China
{fg,huangliang}@cert.org.cn

Abstract. Anomaly detection is an important data mining method aiming to discover outliers that show significant diversion from their expected behavior. A widely used criteria for determining outliers is based on the number of their neighboring elements, which are referred to as Nearest Neighbors (NN). Existing kNN-based Anomaly Detection (kNN-AD) algorithms cannot detect streaming outliers, which present time sensitive abnormal behavior characteristics in different time intervals. In this paper, we propose a fast kNN-based approach for Time Sensitive Anomaly Detection (kNN-TSAD), which can find outliers that present different behavior characteristics, including normal and abnormal characteristics, within different time intervals. The core idea of our proposal is that we combine the model of sliding window with Locality Sensitive Hashing (LSH) to monitor streaming elements distribution as well as the number of their Nearest Neighbors as time progresses. We use an ϵ-approximation scheme to implement the model of sliding window to compute Nearest Neighbors on the fly. We conduct widely experiments to examine our approach for time sensitive anomaly detection using three real-world data sets. The results show that our approach can achieve significant improvement on recall and precision for anomaly detection within different time intervals. Especially, our approach achieves two orders of magnitude improvement on time consumption for streaming anomaly detection, when compared with traditional kNN-based anomaly detection algorithms, such as exact-Storm, approx-Storm, MCOD etc, while it only uses 10% of memory consumption.

Keywords: Anomaly detection · Data streams · LSH · Time sensitive

This work was supported by the National Key Research and Development Program of China (2016YFB0801305).

© Springer Nature Switzerland AG 2019

1 Introduction

Anomaly (a.k.a. outlier) detection [4] is an important data mining method in applications, such as DDoS detection [12] and medical health monitoring [23]. The task is to identify instances or patterns that do not conform to their expected behaviors [9]. Anomaly detection over data streams is a popular topic in computational science because of its inherent vagueness in definition of outliers, like how to define regular behavior, to what degree an outlier needs to be inconsistent with the regular behaviors when confronting the one-pass and infinite high-speed data streams [21].

Anomaly detection can be classified into supervised [20] and unsupervised [16] in principle. Supervised scenarios are not appropriate for data steams processing due to lack of label information about outliers for one-pass streaming elements. Unsupervised anomaly detection does not require any label information and it is appropriate for data steams processing. Currently, unsupervised anomaly detection methods over data streams can be classified into three categories: (1) Statistical based, (2) Clustering based, and (3) k Nearest Neighbors (kNN) based [22].

kNN-based approaches have drawn the most attention of researchers to find anomalies over data streams. kNN-based approaches assume that the proximity of outliers to their nearest neighbors deviates significantly from the proximity of normal data to their neighbors. Given a dataset D and a threshold k, if data point x has less than k neighbors in D, x is marked as outlier. This kind of approaches have gained popularity as they are easy to implement on the fly. However, current kNN-based anomaly detection method cannot detect outliers with time sensitive abnormal distribution. For example, in DDoS applications, like port scanning, SATAN scanning or IP half-way scanning, attackers send scanning packets in different time intervals and this kind of pre-attack cannot be detected by current kNN-based methods [24,26].

We briefly describe challenges of anomaly detection over data streams, which are addressed in our work:

(1) *Abnormal distribution detection.* When detecting outliers, kNN-based methods only focus on whether the NN number reaches the threshold but do not care the changes of NN number with time. However, there may be some cases where the NN number is normal but the distribution is significantly uneven during a time period. This kind of anomaly cannot be detected by only counting NN, so that we need time sensitive anomaly detection algorithm.
(2) *Concept drift.* Most existing data stream anomaly detection techniques ignore one important aspect of stream data: arrival of novel anomalies. In dynamically changing and non-stationary environments, the data distribution can change over time yielding the phenomenon of concept drift [13]. Concept drift is aware of different time intervals, but traditional anomaly detection techniques don't take time into consideration, and as a result many novel anomalies cannot be detected.
(3) *Real-time requirements.* Streaming applications require fast-response and real-time processing from volumes of high-speed and multidimensional

dynamic streaming data over time. However, storing all the streaming data requires unbearably large memory so that it is necessary to reduce memory requirements. Additionally, algorithms communicating with DRAM (main memory) are just too slow for the speed at which the data is generated. Ideally, we need ultra-low memory anomaly detection algorithms that can entirely sit in the cache which can be around 2–10 times faster than accessing main memory [11].

The contributions of this paper are as follows:

(1) We propose a kNN-based approach kNN-TSAD for Time Sensitive Anomaly Detection, which combines an ϵ-approximation scheme Deterministic Waves (DW) with LSH to monitor streaming elements distribution as well as the number of their Nearest Neighbors as time progresses. We use LSH to count NN number without massive calculation of distances between data points. We use DW to monitor distribution of streaming elements during different time intervals. Thus, our anomaly detection method is time sensitive.
(2) The time and space efficiency of our approach is very high and it well meets the real-time requirements. We combine DW with LSH by the way that every item of the hash array is implemented as a DW window and data update can be very efficient and cost very low. Furthermore, we use LSH to search NN which does not need massive range queries and distance calculation, and it does not need to store neighbor information for every element.
(3) We provide a comparison of our algorithm with other five traditional kNN-based algorithms based on three real-world datasets. Our experiment shows that kNN-TSAD is two orders of magnitude faster than other algorithms but only uses 10% of memory consumption. Moreover the recall and precision of kNN-TSAD are higher than others over datasets with abnormal distribution.

2 Preliminaries

In the section, we first give the problem definition, and then we will introduce the LSH structure, SRP and the DW model used in this paper.

2.1 Problem Definition

In this paper, we take data distribution into account and we detect kNN-based outliers as well as outliers with abnormal distribution. We give the definition of kNN-based time sensitive outlier in the data streams in Definition 1. The used notation is summarized in Table 1.

Definition 1. *(kNN-based Time Sensitive Outlier in Data Streams). Given a data stream DS, the current window W, a point x_t in DS and two threshold parameters α and β, we aim to decide whether x_t is an outlier or not. We denote the answer by $NN(x_t, W)$ and $V(x_t, W)$. Formally, if $NN(x_t, W) < \alpha\mu$ or $V(x_t, W) > \beta\delta$, x_t is an outlier, otherwise not.*

Table 1. Frequently used notations.

Notation	Description
DS	Data stream
x_t	The data point observed at time t
W	Current sliding window
w	Number of time intervals of every sliding
α, β	Thresholds parameters
$NN(x_t, W)$	Number of neighbors of x_t in the current window
μ	Mean of $NN(x_t, W)$ till current window
$V(x_t, W)$	Variance of the current window
δ	Mean of $V(x_t, W)$ till current window

2.2 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a popular sublinear time algorithm for efficient approximate nearest neighbors search. LSH leverages a family of functions where each function hashes the points in such a way that under the hash mapping, similar points have a high probability of getting the same hash value so that to be mapped into the same bucket. The probability of collision for an LSH hash function is proportional to the similarity distance between the two data vectors, that is $Pr[h(x) = h(y)] \propto sim(x, y)$.

Definition 2. *(LSH Family). A family $\mathcal{H}$ is called $(S_0, cS_0, u_1, u_2) - sensitive$ if for any two data points x,y $\in \mathbb{R}^d$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:*

(1) If $sim(x, y) \geq S_0$ then $Pr[h(x) = h(y)] \geq u_1$.
(2) If $sim(x, y) \leq cS_0$ then $Pr[h(x) = h(y)] \leq u_2$.

2.3 Signed Random Projections

Usually, the family of hash functions of LSH are generated from random projections, the intuition is that points that are nearby in the space will also be nearby in all projections [10,15]. Given a vector x, it utilizes a random vector w with each component generated from i.i.d. normal (i.e., $w_i \sim N(0, 1)$), and only stores the sign of the projection. We define a hash function $h_w(x)$ as follows:

$$h_w(x) = sign(w^T x) \tag{1}$$

And corresponding to this vector w we define the projection function as follows:

$$sign(w^T x) = \begin{cases} 0 & w^T x \geq 0 \\ 1 & w^T x < 0 \end{cases} \tag{2}$$

For vectors x and w, the probability $Pr[h(x) = h(w)] = 1 - \frac{\theta(x,w)}{\pi}$.

2.4 Deterministic Waves

Deterministic Waves [14] is a ϵ-approximation scheme for the Basic Counting problem for any sliding window up to a prespecified maximum window size W. The most important thing that distinguishes DW from other sliding-window models is that, for the sum of integers in $[0, 1, ..., R]$ it improves the per-item processing time from $O(1)$ amortized ($O(\log N)$ worst case) to $O(1)$ worst case. The space complexity of DW is $O(\frac{1}{\epsilon}\log^2(\epsilon W))$.

Consider a data stream of integers and a desired positive $\epsilon < 1$. The algorithm maintains two counters: pos, which is the current length of the stream, and $rank$, which is the current integers sum in the stream. The procedure of estimating the sum of integers in a window of size $n \leq W$ is as follows. Let $s = max(0, pos - n + 1)$, we estimate the sum of integers in stream positions $[s,\ pos]$. The steps are:

(1) Let p_1 be the maximum position stored in the wave that is less than s, and p_2 be the minimum position stored in the wave that is greater than or equal to s. (If no such p_2 exists, return $\hat{x} := 0$ as the exact answer.) v is the integer value at p_2. Let r_1 and r_2 be the $rank$ of p_1 and p_2 respectively.
(2) Return $\hat{x} := rank - \frac{r_1 + r_2 - v}{2}$.

3 kNN-TSAD Algorithm

In this section we first walk through an overview of the framework, and then describe the procedure in detail.

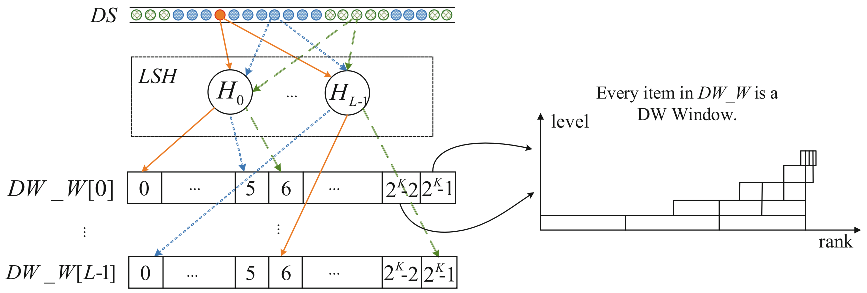


Fig. 1. Framework overview of kNN-TSAD.

3.1 Overview of the Framework

Our kNN-TSAD algorithm combines the functionality of LSH and sliding windows which supports time-based sliding windows queries, so that it can be used for compactly summarizing high-speed and multidimensional streams over sliding windows. It replaces every item of the LSH array with sliding window structures (i.e., DW windows) covering the last W time units of data streams.

Figure 1 illustrates the overview of our proposed framework. DW_W is a two-dimensional LSH array of L rows and 2^K columns. K and L are pre-specified hyper-parameters. $H_i, i \in 0, 1, \ldots, L-1$ are a group of hash functions, and results of H_i are corresponding to the ith row of DW_W. Every item of the LSH array is a DW window. There are two main phases of our kNN-TSAD algorithm.

Insert Phase. The framework performs inserting a data point as follows:

(1) The nearest neighbors are searched by applying LSH on the data point.
(2) The data point is inserted into corresponding neighbourhoods which are implemented as DW windows, and expired data are deleted from DW windows.
(3) The global mean count and the mean variance of active sliding window are updated.

Query Phase. The framework performs an anomaly query as follows:

(1) NN number of current data point and variance of current window are calculated.
(2) NN number and variance of the current data point are compared with mean values, and then an anomaly decision result is given.

3.2 Nearest Neighbors Searching

In kNN-based methods for anomaly detection, NN searching is the most important part. Existing methods search neighbors by computing distances of every two data points, and then compare to a distance threshold. This process is quite time consuming. Instead, utilization of LSH can significantly reduce time overhead and improve efficiency of NN searching.

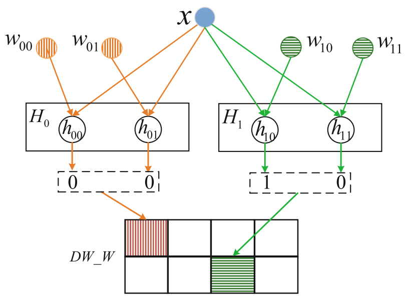


Fig. 2. An example of NN searching using LSH.

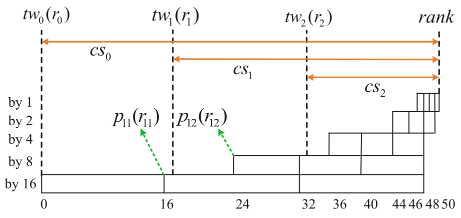


Fig. 3. An example of time sensitive monitoring.

We randomly generate $K \times L$ vectors $w_{ij}, i \in 0, 1, \ldots, L-1, j \in 0, 1, , K-1$, with each component generated from i.i.d. normal (i.e., $w_{ij} \sim N(0, 1)$).

Then using these random vectors we define $K \times L$ independent signed random projections $h_{ij}, i \in 0, 1, \dots, L-1, j \in 0, 1, \dots, K-1$, given in Eqs. 1 and 2 and each of them gives one-bit output. So that we can finally generate L K-bit hash functions $H_i(x) = [h_{i1}(x); h_{i2}(x); ...; h_{iK}(x)], i \in 0, 1, \dots, L-1$, which respectively correspond to the indexes in L rows of hash array DW_W. Final NN number of a data point x is the average of all $H_i(x)$. By this way the collision probability is significantly reduced from $p = 1 - \frac{\theta(x,y)}{\pi}$ to $p = Pr[H(x) = H(y)] = p^K$, if and only if $h_j(x) = h_j(y)$ for all $j \in 0, 1, \dots, K-1$, $H(x) = H(y)$. Figure 2 shows an example of NN searching process of a data point.

3.3 Time Sensitive Monitoring

To monitor the sliding windows in more detail, we preset a time interval whose length is $\frac{1}{w}$ of the window size. Utilizing the recorded position queue in the DW model, we calculate the average counts of each time interval based on different time range within the sliding window and then work out their variance. The variance will significantly deviate from average level if distribution of data in current sliding window is abnormal.

We have illustrated the procedure of estimating the count of DW in Sect. 2.4, we use the same method to calculate the count of every time interval.

Algorithm 1. kNN-based Anomaly Detection.

```
Input: Data stream $DS$, Array parameters $L$ and $K$, Threshold parameter $\alpha$
Hash Initialize: Generate $L \times K$ independent SRP hash functions
$DW\_W$ = new DW[$L$][$2^K$], $\mu = 0$, n=0
As a data point $x_t \in DS$ comes
/*Insert: add the coming data $x_t \in DS$ to the structure.*/
c = 0
for $i = 0$ to $L-1$ do
    Add $x_t$ to $DW\_W[i][H_i(x_t)]$ and delete expired data points
    $c = c + \frac{1}{L} DW\_W[i][H_i(x_t)].count$
end for
$\mu = \frac{1}{n+1}(n\mu + c)$
$n$++
/*Query: determine whether $x_t \in DS$ is an outlier or not.*/
if there are expired data then
    if $c < \alpha\mu$ then
        Report anomaly
    end if
else
    for $i = 0$ to $L-1$ and $j = 0$ to $2^K - 1$ do
        Get the count $c$ of $DW\_W[i][j]$
        if $c < \alpha\mu$ then
            Report anomaly
        end if
    end for
end if
```

Let $tw_i, i \in 0, 1, \ldots, w-1$ be the left bound of every time interval, and tw_0 be the left bound time of the sliding window. Let p_{i1} be the maximum position stored in the wave that is before tw_i, and p_{i2} be the minimum position stored in the wave that is after or at tw_i. Let r_{i1} and r_{i2} be the counts of p_{i1} and p_{i2} respectively. So that $r_i := \frac{r_{i1}+r_{i2}-1}{2}$, $i \in 0, 1, \ldots, w-1$, and $rank$ is the current exact count. Then we use different time range to compute average count of each time interval cs_i and $cs_i = \frac{rank - r_i}{w-i}$. For example, as shown in Fig. 3, p_{11} is the maximum position before tw_1 and p_{12} is the minimum position after tw_1. Counts of the two positions are 16 and 24 respectively, so that we estimate the count at tw_1 which is $r_1 = \frac{16+24-1}{2} = 20$. The current count $rank$ is 50, thus $cs_1 = \frac{rank - r_1}{2} = 15$, $cs_0 = \frac{rank - r_0}{3} = 16$ and $cs_2 = \frac{rank - r_2}{1} = 16$. Finally we get $\bar{cs} = \frac{15+16+16}{3} = 15$, and then the current variance $\sigma^2 = 2$.

3.4 Efficient Update

Updating operation of our algorithm is to straightforward increment (or decrement) the counters when a data point is added (or removed). We only need to add or delete data points in the corresponding DW window and update the global mean values μ and the mean variance $DW\_W[i][H_i(x)].\delta$ of current sliding window. This operation does not need lots of distance computation to update neighbor information like traditional kNN-based algorithms.

As a new data point $x_t \in DS$ arrives, we add it to the DW window at corresponding index $H_i(x)$ of each row so that every DW window keeps the number of hits to this particular window. We get the neighbors count c of x_t, $c = \frac{1}{L}\sum_{i=0}^{L-1} DW\_W[i][H_i(x)]$. Then we calculate average count of every time interval $cs_j, j \in [0, w-1]$ so as to calculate the variance σ^2. We compute the mean count μ,

$$\mu = \frac{1}{n+1}\left(n\mu + \frac{1}{L}\sum_{i=0}^{L-1} DW\_W\left[H_i(x)\right]\right) \tag{3}$$

and the mean variance $DW\_W[i][H_i(x)].\delta$ of past part of $DW\_W[i][H_i(x)]$,

$$DW\_W[i][H_i(x)].\delta = \frac{DW\_W[i][H_i(x)].count \times DW\_W[i][H_i(x)].\delta + \sigma^2}{DW\_W[i][H_i(x)].count + 1}. \tag{4}$$

Deviation from the mean values indicate anomaly. It turns out that we can dynamically update the mean values μ and $DW\_W[i][H_i(x)].\delta$ on the fly, as we observe new coming data.

Algorithm 2. Abnormal Distribution Detection.

```
Input: Data stream $DS$, Array parameters $L$ and $K$, Number of time intervals $w$, $\beta$
Hash Initialize: Generate $L \times K$ independent SRP hash functions
$DW\_W$ = new DW$[L][2^K]$
As a data point $x_t \in DS$ comes
$\sigma^2 = 0$
for $i = 0$ to $L - 1$ do
    /*Insert: add the coming data $x_t \in DS$ to the structure.*/
    Add $x_t$ to $DW\_W[i][H_i(x_t)]$ and delete expired data points
    Get the average counts $cs_0, cs_2, ..., cs_{w-1}$ based on different time range
    $\bar{cs} = \frac{1}{w}\sum_{q=0}^{w-1} cs_q$
    $\sigma^2 = \frac{1}{w}\sum_{q=0}^{w-1}(cs_q - \bar{cs})^2$
    $DW\_W[i][H_i(x)].\delta = \frac{DW\_W[i][H_i(x)].count \times DW\_W[i][H_i(x)].\delta + \sigma^2}{DW\_W[i][H_i(x)].count}$
    /*Query: determine whether there is a distribution anomaly or not.*/
    if $\sigma^2 > \beta DW\_W[i][H_i(x)].\delta$ then
        Report anomaly
    end if
end for
```

3.5 Anomaly Detection

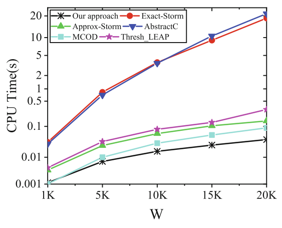
Our algorithm serves for data streams which are high-speed and one-pass. We use two types of queries to accurately conduct anomaly detection.

(1) *kNN-based anomaly detection queries.* kNN-based anomaly detection is a global procedure. A global mean count μ is calculated to get the threshold $\alpha\mu$. In inserting phase we compute current count and compare with the threshold, and if a DW window satisfies $c < \alpha\mu$, all data in that window are marked as outliers. The process is shown in Algorithm 1.

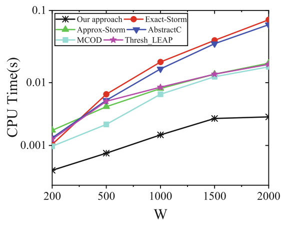
(2) *Distribution anomaly detecion queries.* Distribution anomaly detection is a local procedure. Every DW window keeps a mean variance $DW\_W[i][H_i(x)].\delta$, and with it we get a threshold $\beta DW\_W[i][H_i(x)].\delta$ of every DW window. In inserting phase we compute current variance σ^2 and compare with the threshold, if a DW window satisfies $\sigma^2 > \beta DW\_W[i][H_i(x)].\delta$, there exists a distribution anomaly. The process is shown in Algorithm 2.

3.6 Complexity Analysis

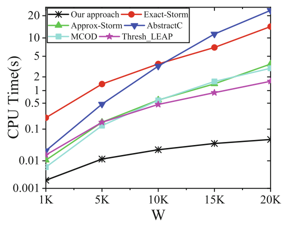
Time Complexity: For a new coming data point, we need to compute KL hashes, Lw DW counts and a variance. For random projections based LSH, signed random projection is a special case. We can calculate KL LSH hashes of the data vector, with dimensions d, in time $O(d \log d + KL)$. Using DW model, each item is processed in $O(1)$ worst case. At each time instant, it can provide an estimate in $O(1)$ time. Finding the time division nodes needs to traverse the



(a) TAO

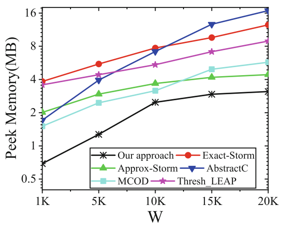


(b) Shuttle

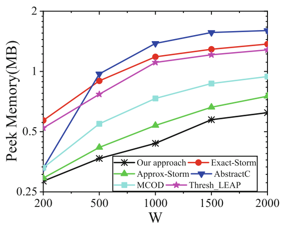


(c) KDD-Cup99

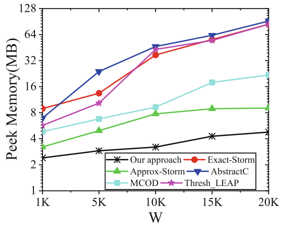
Fig. 4. Time comparison.



(a) TAO



(b) Shuttle



(c) KDD-Cup99

Fig. 5. Memory comparison.

position queue of DW, the time is $O(\frac{1}{\epsilon}(\log(\epsilon W)))$, thus the total time we need is $O(d \log d + \frac{1}{\epsilon}(\log(\epsilon W))$.

Space Complexity: It is presented in [14] that an ϵ-approximation DW window for basic counting uses only $O(\frac{1}{\epsilon}\log^2(\epsilon W))$ memory bits of workspace, and here we need $2^K \times L$ DW windows in the two-dimensional array. K and L are preset constant parameters, and in all our experiments we use $K = 15$ and $L = 50$ for all the three datasets. However, it is unlikely that the DW windows will get too many hits, the number of hits of all our experiments is no more than 3000. Therefore, with these small constant values, the memory cost is extremely low, and the required memory is $O(\frac{1}{\epsilon}\log^2(\epsilon W))$.

4 Experimental Evaluation

4.1 Datasets

We choose the following three real-world and synthetic datasets for our evaluation: (1) TAO, (2) Statlog Shuttle and (3) KDD-Cup99 HTTP. The first dataset contains 575,648 records with 3 attributes. It is available at Tropical Atmosphere Ocean project [3]. The second dataset shuttle dataset [2] describes radiator positions in a NASA space shuttle with 9 attributes. It was designed for supervised anomaly detection. In the original datasets, about 20% of the data are regarded as anomaly. The third dataset is KDD-Cup99 HTTP [1]. KDD-Cup99 HTTP dataset is the largest benchmark for unsupervised anomaly detection evaluation. It contains simulated normal and attack traffic on an IP level in a computer network environment in order to test intrusion detection systems. We use HTTP traffic only and also limit DoS traffic from the dataset. We sample part of it and the features of "protocol" and "port" information is removed. There are totally 395,423 instances with 6,315 labeled anomalies.

4.2 Experimental Methodology

Our experiments are conducted in a 3.30 GHz core Windows machine with 8 GB of RAM. Experiments focus on four aspects: (1) time efficiency; (2) memory consumption; (3) recall and precision; (4) time sensitivity property. We use five kNN-based anomaly detection algorithms to compare with our proposal. We summarize the time and space complexities of each algorithm and provide comparison in Table 2. S denotes the slide size which characterizes the speed of the data stream, k is the neighbor count threshold, and η denotes the fraction of the window stored in micro-clusters in the MCOD algorithm. For fair evaluation and reproducibility, we implement all the algorithms in Java.

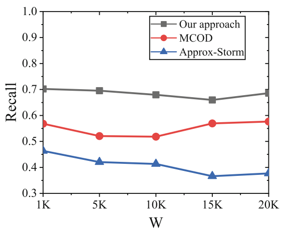
Table 2. Comparison of kNN-based anomaly detection algorithms.

Algorithm	Time complexity	Space complexity
Our proposal	$O(d \log d + \frac{1}{\epsilon}(\log(\epsilon W))$	$O(\frac{1}{\epsilon} \log^2(\epsilon W))$
Exact-Storm	$O(W)$	$O(W)$
Approx-Strom	$O(W^2/S)$	$O(W^2/S + W)$
AbstractC	$O(W \log k)$	$O(kW)$
MCOD	$O(kW \log k + (1-\eta)W \log((1-\eta)W))$	$O(\eta W + (1-\eta)kW)$
Thresh_LEAP	$O(W^2 \log S/S)$	$O(W^2/S)$

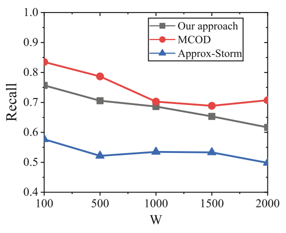
As shown in Fig. 4, when W increases, the CPU time for each algorithm increases as well, due to a larger number of data points to be processed in every window. We can see among the six algorithms kNN-TSAD is consistently efficient. The reason is that adding and removing data points in kNN-TSAD are very efficient as well as neighbor searching. It does not need to carry out range queries and calculate distances to update neighbor information like other kNN-based algorithms, thus kNN-TSAD saves much time.

Figure 5 reports the memory consumption of the evaluated algorithms when varying the window size W. The memory requirement increases with W consistently across different datasets. The five traditional methods store neighborhood information for every data point in current window, which requires large amount of memory. However, every item of the kNN-TSAD array (i.e. every DW window) is a neighborhood with no need to extra store neighborhood information. Thus kNN-TSAD consumes much less memory compared with other methods.

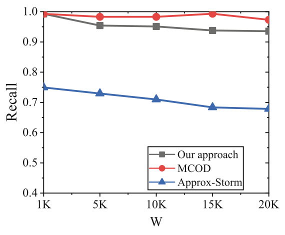
Exact-Storm, AbstractC, MCOD and Thresh_LEAP calculate all the neighbours to decide whether a data point is an outlier. They only differ in neighborhood searching and storing, such that they get almost the same anomaly detection result. So in terms of recall and precision we solely need to test on one of them, we choose MCOD. Approx-Storm adapts exact-Storm with two approximations to reduce the number of data points and the space for neighbor stored in each window. We plot the recall and precision of kNN-TSAD, MCOD and approx-Storm to make a comparison (see Figs. 6 and 7). We can see approx-Storm gets the lowest recall and precision of the three. MCOD calculate exact number of neighbors of every data points so that it gives the highest recall and precision. kNN-TSAD is between the other two algorithms on dataset Shuttle and KDD-Cup99. But on dataset TAO, as seen in Figs. 6(a) and 7(a), kNN-TSAD gets the best anomaly detection result. This is because that part of abnormal data in the TAO dataset appears in a concentrated way during a period of time. In this case the number of neighbors exceeds the threshold due to high frequency. kNN-TSAD divides the sliding window into several time intervals and calculates the variance of their counts to monitor count distribution. Figure 8 shows the variation of variance over the time period when outliers appear. Obviously between 3900 s and 6000 s, the variance experienced a process of sharp increase and decrease indicating that there appears an anomaly. However this cannot be detected by other kNN-based algorithms, and this is why kNN-TSAD get the highest recall and precision over TAO.



(a) TAO

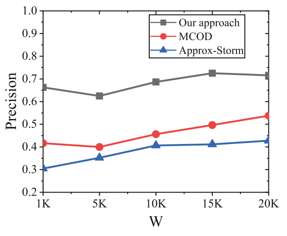


(b) Shuttle

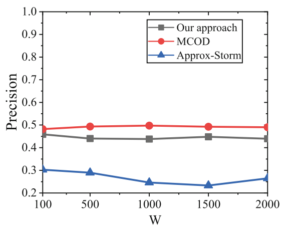


(c) KDD-Cup99

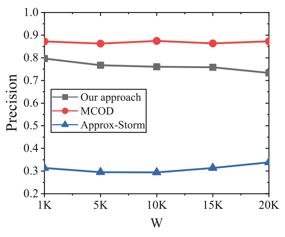
Fig. 6. Recall comparison.



(a) TAO



(b) Shuttle



(c) KDD-Cup99

Fig. 7. Precision comparison.

To summarize, kNN-TSAD provides the highest performance over dataset TAO whose anomaly detection depends on the data distribution, and this indicates the time sensitive property of our approach. Moreover, kNN-TSAD cost the least time and space over all datasets compared with other algorithms and shows its excellent time and space efficiency. LSH and DW introduce calculation error to the algorithm, so that the recall and precision are slightly lower than MCOD which count exact NN number. But when compared with the approximate method Approx-Storm, the recall and precision are significantly higher.

5 Related Work

Anomaly detection has been studied for years in the communities of multidimensional data sets [5] and the metric spaces [25]. Usually, similarity is used to decide whether an object is an outlier or not. Extensively, the problem of anomaly detection has been studied by the statistics community [17], where the objects are modeled as a distribution, and outliers are those whose deviation from this distribution is more than others. However, for high-dimensional data, statistical techniques fail to model the distribution accurately. Additionally, for large databases these techniques fail to scale well either.

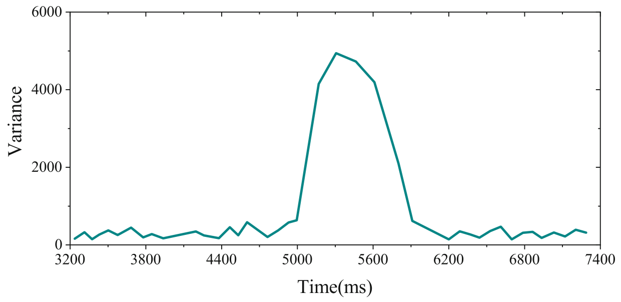


Fig. 8. Variation of variance over time.

The problem of anomaly detection has been also addressed by data mining communities. Techniques that focus on identifying Nearest Neighbors-based patterns are popular in data mining and other related communities. k Nearest Neighbors-based outliers were first introduced by Knorr and Ng [18]. Given parameters k and R, an object is marked an outlier if there are less than k objects in the input data set lying within distance R from it. However, this algorithm operates in a static fashion. This means that if there happen some changes in the data, the algorithm must be executed from scratch, leading to performance degradation while updating frequently. For example in the streaming case, where objects arrive in a streaming fashion [7].

In [6], an exact algorithm exact-Storm was proposed to efficiently detect outliers over data streams. It stores up to k preceding neighbors and the number of succeeding neighbors of x. x is an outlier if x has less than k neighbors. An approximate algorithm approx-Storm is derived from the former one. Up to ρW ($0 < \rho < 1$) safe inliers are preserved for each window, and it only store the ratio between the number of x's preceding neighbors which are safe inliers and the number of safe inliers in the window. Such it can estimate the number of x's neighbours. Kontaki et al. [19] proposed the MCOD algorithm which aims to reduce the number of distance computations. MCOD stores the neighboring data points in micro-clusters, and a micro-cluster is composed of no less than $k+1$ data points. Some data points may not fall into any micro-clusters, if they have less than k neighbors, they are outliers. Cao et al. [8] proposed a framework named LEAP which encompasses two general optimization principles. First, the "minimal probing" principle uses a lightweight probing operation to gather minimal yet sufficient evidence for outlier detection. Second, the "lifespan-aware prioritization" principle leverages the temporal relationships among stream data points to prioritize the processing order among them during the probing process. Yang et al. [27] proposed an algorithm Abstract-C which maintains a compact summary of the neighbourships, namely the count of the neighbors for each data point. Instead of storing a list of preceding neighbors or number of succeeding neighbors for each data point x, a sequence is used to store the number of neighbors in every window that x participates in.

kNN-based anomaly detection methods rely on neighbors search to decide whether a data point is an outlier. And neighbors searching relies on calculating the similarity between data points which is time consuming. In this paper, we use LSH to measure similarity between data points. Data points with high similarity are more likely to enter the same buckets after hashing. There is no need to calculate the similarity of every two data points.

6 Conclusion

Anomaly detection applications need to provide real-time monitoring and the capability of detecting anomalies over continuous data streams using limited resources. In this paper, we leverage the features of LSH and DW and propose a kNN-based time sensitive anomaly detection approach kNN-TSAD over data streams. Our approach can detect abnormal data distribution over different time intervals which cannot be found by traditional kNN-based methods. Moreover, our approach demonstrates high time and space efficiency for solving the problem of data stream processing. However, we can see from the experiments that the recall and precision of our approach are slightly lower than exact algorithms in some cases. In the future, we plan to design a more efficient monitoring strategy to improve and optimize our approach to get higher recall and precision.

References

1. KDD-Cup99 HTTP. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
2. Statlog Shuttle. https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)
3. TAO. http://www.pmel.noaa.gov
4. Aggarwal, C.C.: Data Streams: Models and Algorithms, vol. 31. Springer, London (2007). https://doi.org/10.1007/978-0-387-47534-9
5. Aggarwal, C.C., Yu, P.S.: Outlier detection for high dimensional data. In: ACM Sigmod Record, vol. 30, pp. 37–46. ACM (2001)
6. Angiulli, F., Fassetti, F.: Detecting distance-based outliers in streams of data. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, pp. 811–820. ACM (2007)
7. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 1–16. ACM (2002)
8. Cao, L., Yang, D., Wang, Q., Yu, Y., Wang, J., Rundensteiner, E.A.: Scalable distance-based outlier detection over high-volume data streams. In: IEEE 30th International Conference on Data Engineering (ICDE), pp. 76–87. IEEE (2014)
9. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: a survey. ACM Comput. Surv. (CSUR) **41**(3), 15 (2009)
10. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 380–388. ACM (2002)
11. Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K., Hughes, B.: Cache hierarchy and memory subsystem of the AMD Opteron processor. IEEE Micro **30**(2), 16–29 (2010)
12. Dang, Q.V.: Outlier detection on network flow analysis. arXiv preprint arXiv:1808.02024 (2018)
13. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. (CSUR) **46**(4), 44 (2014)
14. Gibbons, P.B., Tirthapura, S.: Distributed streams algorithms for sliding windows. In: Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 63–72. ACM (2002)
15. Goemans, M.X., Williamson, D.P.: 879-approximation algorithms for MAX CUT and MAX 2SAT. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, pp. 422–431. ACM (1994)
16. Görnitz, N., Kloft, M., Rieck, K., Brefeld, U.: Toward supervised anomaly detection. J. Artif. Intell. Res. **46**, 235–262 (2013)
17. Härdle, W., Simar, L.: Applied Multivariate Statistical Analysis. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-662-45171-7
18. Knox, E.M., Ng, R.T.: Algorithms for mining distance based outliers in large datasets. In: Proceedings of the International Conference on Very Large Data Bases, pp. 392–403. Citeseer (1998)
19. Kontaki, M., Gounaris, A., Papadopoulos, A.N., Tsichlas, K., Manolopoulos, Y.: Continuous monitoring of distance-based outliers over data streams. In: IEEE 27th International Conference on Data Engineering (ICDE), pp. 135–146. IEEE (2011)
20. Leung, K., Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the Twenty-Eighth Australasian Conference on Computer Science, vol. 38, pp. 333–342. Australian Computer Society, Inc. (2005)

21. Sadik, M.S., Gruenwald, L.: DBOD-DS: distance based outlier detection for data streams. In: Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) DEXA 2010. LNCS, vol. 6261, pp. 122–136. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15364-8_9
22. Salehi, M., Rashidi, L.: A survey on anomaly detection in evolving data: [with application to forest fire risk prediction]. ACM SIGKDD Explor. Newslett. **20**(1), 13–23 (2018)
23. Saneja, B., Rani, R.: An efficient approach for outlier detection in big sensor data of health care. Int. J. Commun. Syst. **30**(17), e3352 (2017)
24. Sezari, B., Möller, D.P., Deutschmann, A.: Anomaly-based network intrusion detection model using deep learning in airports. In: 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 1725–1729. IEEE (2018)
25. Tao, Y., Xiao, X., Zhou, S.: Mining distance-based outliers from large databases in any metric space. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 394–403. ACM (2006)
26. Viet, H.N., Van, Q.N., Trang, L.L.T., Nathan, S.: Using deep learning model for network scanning detection. In: Proceedings of the 4th International Conference on Frontiers of Educational Technologies, pp. 117–121. ACM (2018)
27. Yang, D., Rundensteiner, E.A., Ward, M.O.: Neighbor-based pattern detection for windows over streaming data. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp. 529–540. ACM (2009)