EXOS: Explaining Outliers in Data Streams

Egawati Panjei(✉) and Le Gruenwald

School of Computer Science, The University of Oklahoma, Norman, OK, USA
{egawati.panjei,ggruenwald}@ou.edu

Abstract. Real-time outlier detection is important in many data stream applications. To help analysts understand the detected outliers better, the outliers should be presented with their explanations. One type of explanations for an outlier is its set of outlying attributes which is a subset of features responsible for the outlier's abnormality. There exist techniques that generate outlying attributes in data streams; however, none simultaneously considers the cross-correlation among data streams, the unbounded volume of data, and concept drift. To fill this gap, we propose EXOS, a framework that generates outlying attributes in multi-dimensional data streams. For each outlier, it incrementally finds a local context to determine the decision boundary that separates the outlier from the normal data while handling both the unbounded volume of data and concept drift. It considers the potential data correlation within a data stream and across data streams to estimate the local context. The experiments using three real and two synthetic datasets show that, on average, EXOS achieves up to 49% higher F1 score and 29.6 times lower explanation time than existing algorithms.

Keywords: Outlier Explanation · Outlying Attributes · Data Streams

1 Introduction

An outlier in a dataset is a data point that has a significantly different value compared to other data points in the dataset [1]. The increasing demand for real-time analytics has created a need to apply real-time outlier detection over data streams [2, 3]. Unfortunately, these techniques do not provide explanations of why some data points are deemed to be anomalous, leaving analysts with no guidance to decide whether those objects require further actions. for critical applications like structural health monitoring (SHM) and intrusion detection, the investigation of whether the outliers are subjects of interest should be done fast. However, analysts' effort to investigate an outlier roughly corresponds to the number of attributes associated with the outlier [4]. The investigation takes time, but it can be done faster if each detected outlier is presented with a subset of attributes responsible for its abnormality. This type of explanation is known as *outlying attributes* [5].

A data stream is an infinite sequence of data points with explicit or implicit timestamps [2, 5]. Monitoring and processing data streams in real-time are bound to time and memory constraints as data arrive continuously. They require that every data point be processed online and incrementally [5]. Furthermore, data streams are known for

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

concept drift, where the data distribution changes over time [6]. In data streams, data attributes may be correlated not only within a data stream but also across data streams (cross-correlation [5]) which can then be used to improve outlier explanations. The existing outlying attribute algorithms do not simultaneously consider the unbounded volume of data, concept drift, and cross-correlation in data streams [5]. Thus, to fill this gap, we propose EXOS, an algorithm to generate outlying attributes of each point outlier in data streams in real time that addresses all the three characteristics.

EXOS provides outlying attributes for each point outlier detected by any outlier detectors in data streams. We assume that EXOS and outlier detectors are independent processes that communicate through queues. To deal with the unbounded data volume, outlying attributes are generated based on a time-based tumbling window where a window is used to store a sequence of data in the main memory, and when a specified time period expires, i.e., when the window slides, all the data stored in the window are replaced with the new arriving sequence of data [11]. For each stream, when its window slides, EXOS will read the queue that stores the stream's outliers detected by an outlier detector and apply single-pass incremental computation on the data in the window to generate the outliers' outlying attributes.

EXOS is a local neighborhood-based outlier explanation technique. For each outlier, it defines a local context, which is a set of inlier neighbors of the outlier and uses the local context to find the outlier's outlying attributes. The local context is formed by considering the cross-correlations among data streams. The eigenvectors, used in forming the local context, are initially generated using offline data, yet since data streams are subject to concept drift, the eigenvectors are updated whenever the window slides using a single-pass eigendecomposition.

Our contributions are as follows: 1) we develop an algorithm that generates outlier explanations in terms of outlying attributes that considers the possible data attribute correlation within a data stream and across data streams, while simultaneously addressing the unbounded data volume and concept drift in data streams; and 2) we perform a comprehensive experimental analysis comparing EXOS with existing algorithms in terms of average precision, recall, F1 score, and execution time using real and synthetic datasets.

2 Related Work

In the recent survey of outlier explanations [5], there exist techniques that generate outlying attributes. For example, SFE [4] Applies subspace search with heuristics to find a subset of attributes where a detected outlier has the highest outlier score. Micenková Et Al.[7] and COIN [8] utilize inlier neighbors of an outlier to find its outlying attributes. They use a linear classifier to find a decision boundary between inlier and outlier classes. Attributes whose corresponding weights in the hyperplanes are higher than a threshold are considered outlying attributes. However, all these techniques are designed for static or non-stream data where they assume a finite amount of data and require multiple passes on the dataset; thus, they are unsuitable for dealing with the unbounded volume of data streams in real-time.

There are outlying attribute techniques proposed for data streams, for example, EXAD [9], MICOES [10], MacroBase [15, 16], and EXstream [16, 17]. EXAD employs a decision tree estimated from a neural network trained offline to generate the outlying attributes of each detected outlier in an online manner. However, the decision tree is never updated to address concept drift. MICOES absorbs arriving data points into micro-clusters and uses them to form an inlier class and an outlier class associated with each detected outlier. The inlier and outlier classes are used to generate outlying attributes. Since micro-clusters adapt to the changes in data distribution, the explanations also adapt. MacroBase produces outlier explanations using a prefix tree of frequent itemsets updated periodically, allowing it to handle concept drifts. EXstream generates outlier explanations based on the temporal context of the outliers, which is updated over time, making it hardly unaffected by concept drifts. Still, none of these algorithms consider the relationships among data streams. Thus, in this work, we develop a local context-based outlier explanation for data streams that consider the potential cross-correlations among data streams to improve outlying attribute generation while dealing with the unbounded volume and concept drift.

3 Preliminaries

This section formally defines the problem of finding outlying attributes of an outlier in data streams.

Definition 3.1 (Data Stream). A data stream S is an infinite sequence of data points $\{\mathbb{X}_i | i \geq 0\}$. Each $\mathbb{X}_i$ is a tuple of length $d + 1$ denoted as $\mathbb{X}_i = a_1, a_2, \ldots, a_d, t$ where d is the number of attributes, a_k is the value of the k-th attribute, and t is the associated timestamp when the tuple is recorded or collected.

We consider a set of m concurrent data streams $\mathbb{S} = \{S_1, \ldots, S_m\}$ where $m \geq 1$. The i-th data point in a data stream S_j is denoted as $\mathbb{X}_i^j$. The attributes in S_j can differ from those in S_k. We denote $\mathbb{X}_i^j$ as $\mathbb{O}_i^j$ when $\mathbb{X}_i^j$ is detected as an outlier. We assume that all data streams have the same arrival rate (synchronous data streams) and each data stream has a corresponding outlier detector.

Data streams arrive continuously and are inherently unbounded; hence, keeping all data in the memory for real-time processing is impossible. EXOS handles this by processing data on a time-based tumbling window residing in the memory as defined in Definition 3.2.

Definition 3.2 (Tumbling Window). Given a time interval $\mathbb{T}$ that consists of start and end timestamps, a tumbling window W is a finite sequence of data points or tuples $(\mathbb{X}_n, \mathbb{X}_{n+1}, \ldots, \mathbb{X}_N)$ where $\mathbb{X}_N.t - \mathbb{X}_n.t \leq \mathbb{T}$. All the data points in the window will expire when $\mathbb{T}$ expires, i.e., when the window slides.

The outlying attributes of an outlier are the subset of attributes responsible for the abnormality of the outlier. They are formally defined in Definition 3.3.

Definition 3.3 (Outlying Attributes). [5]. Given an outlier $\mathbb{O}$, a set of d dimensions $\mathcal{D} = \{A_1, A_2, \ldots, A_d\}$ where $\mathbb{O} \in A_1 \times A_2 \times \cdots \times A_d$, an outlier attribute contribution score function $h : \mathcal{D} \to \mathbb{R}$ that generates a real-value quantifying the contribution of each attribute to the abnormality of $\mathbb{O}$, and an outlying contribution score threshold $\gamma \geq 0$, the outlying attributes of $\mathbb{O}$ is a subspace $F \subseteq \mathcal{D}$ such that $\forall A_i \in F, h(A_i) > \gamma$.

Problem Definition: Given a set of m synchronous multi-dimensional data streams $\mathbb{S} = \{S_1, S_2, \ldots, S_m\}$, an outlier $\mathbb{O}_i^j$ identified by an arbitrary outlier detector, and an outlying contribution score threshold $\gamma \geq 0$, the problem is to find all the outlying attributes of $\mathbb{O}_i^j$ accurately and efficiently.

4 The Proposed Algorithm: EXOS

Algorithm 1: EXOS ($\mathbb{S}$, k, D, γ, $\mathbb{A}$, $\mathbb{T}$, *init_data*, *n_clusters*, *outlier_queues*)

Input: $\mathbb{S}$ the set of m concurrent data streams, k the number of eigenvectors, D the total number of attributes, γ the attribute contribution threshold, $\mathbb{A}$ the list of the attribute names in the streams, $\mathbb{T}$ the window size represented as a time interval, *init_data* the offline data, *n_clusters* the list of the IDs of the clusters for each data stream, *outlier_queues* the list of m queues storing the outliers of the m streams.
Output: $\mathbb{F}_j$ the set of outlying attributes of each outlier in each stream j

```
1   $Q$ := initialize_eigen_vectors (init_data)
2   $C_a$ := initialize_set_of_clusters(init_data, n_clusters)
3   end_ts := current_time ()
4   start_ts := end_ts – $\mathbb{T}$
5   while (true): ## repeat indefinitely
6       $L_{\mathbb{O}}$ := get_outliers(outlier_queues, start_ts, end_ts)
7       N := the number of data points in each window
8       do in parallel
9           Q, est_list := Estimator ($\mathbb{S}, L_{\mathbb{O}}, k, D, Q$, start_ts, end_ts, N) ##Component 1
10          parallel for j in [1: m]
11              $W_j$ := $S_j$[start_ts, end_ts] # get data in the current window
12              $\mathbb{C}_j$ := TemporalNeighborClustering($W_j$, $C_a$[j]) ## Component 2
13              $C_a$[j] := the latest centroids in $\mathbb{C}_j$
14          end parallel for
15      end do in parallel
16      parallel for j in [1: m]
17          $\widehat{\mathbb{O}}_j$ := est_list[j] ## get estimated normal values of the outliers in stream j
18          $\mathbb{O}_j$ := $W_j[L_{\mathbb{O}}[j]]$ ## get outlier values in stream j
19          $\mathbb{F}_j$ := OutlyingAttributesGenerator($\widehat{\mathbb{O}}_j$, $\mathbb{O}_j, \mathbb{C}_j, \gamma$, $\mathbb{A}[j]$) ) ## Component 3
20        yields($\mathbb{F}_j$) ## return the outlying attributes of the outliers in the current window of stream j
21      end parallel for
22      while (end_ts + $\mathbb{T}$ > current_time()) : wait() end while
23      end_ts = current_time (); start_ts = end_ts – $\mathbb{T}$
24  end while
```

To determine the outlying attributes of the outliers detected in a recent window, EXOS goes through *six steps*: (1) combine the windows from all streams and use the data attribute correlation within each stream and across streams to estimate the normal value of each outlier $\mathbb{O}_i^j$; (2) in parallel with Step (1), incrementally group all the data points in the window into clusters in each data stream S_j; (3) use the estimated normal value of each $\mathbb{O}_i^j$ and its closest cluster to form a local context of the outlier (the inlier class); (4) for each $\mathbb{O}_i^j$, form an outlier class by randomly generating the auxiliary outlier data points that have uncorrelated multivariate normal distribution centered at $\mathbb{O}_i^j$; (5) find a decision boundary that separates the inlier class from the outlier class using a linear classifier; and (6) use the weights of the decision boundary to determine the outlying contribution scores of the attributes of $\mathbb{O}_i^j$. The attributes having the outlying contribution scores greater than the threshold γ are the outlying attributes of $\mathbb{O}_i^j$.

Algorithm 1 shows the overall EXOS algorithm. It consists of the initialization and online phases. Using offline data, EXOS initializes Q, the estimated eigenvectors (Line 1), and $\mathrm{C_a}$ the list of the initial clusters' centroids in each stream (Line 2). The online phase indicated in Lines 3–24 has three key components: (i) Estimator *Est* (Line 9), (ii) Temporal Neighbor Clustering $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ (Line 12) and (iii) Outlying Attribute Generators $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m$ (Line 19). The *Est* component corresponding with *Step (1)* uses the eigenvectors to estimate the normal values of outliers. The eigenvectors capture the correlation among attributes from the same data stream and attributes from different streams (cross-correlation). They are incrementally updated whenever the window slides to ensure they deal with concept drift. The C_j component, which handles *Step (2)*, groups the data points in the window of S_j, based on a symmetric distance function. The clusters serve as a temporal context which is the neighborhood that will be used by $\mathcal{G}_j$ to find the outlying attributes of $\mathbb{O}_i^j$. The $\mathcal{G}_j$ component manages *Steps (3)–(6).*

We now describe the details of the three key components of EXOS.

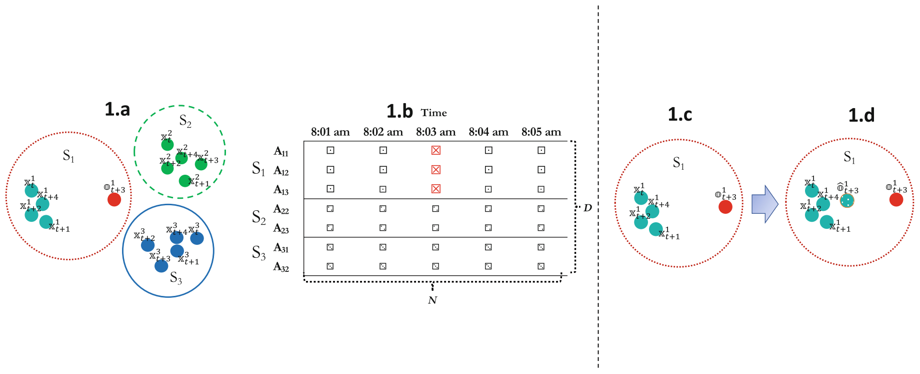

Fig. 1. Illustration of finding a local context of an outlier in concurrent data streams

4.1 Estimator

Estimator, *Est,* approximates the normal value $\hat{\mathbb{O}}_i^j$ of each outlier $\mathbb{O}_i^j$ by using the information from other data streams. Let us consider an example of 3-source data streams depicted in Figs. 1a and b. Between 8:01 am and 8:05 am, the outlier detector at S_1 detects $\mathbb{X}_{t+3}^1$ as an outlier (denoted as $\mathbb{O}_{t+3}^1$). Its normal value estimation is denoted as $\mathbb{O}_{t+3}^1$ in Fig. 1d. Suppose S_1, S_2, and S_3 in Fig. 1a have three, two, and two attributes, respectively. The data are observed between 8:01 am and 8:05 am where an outlier is detected in S_1 at 8:03 am. We can represent the data points from those streams as a 7×5 matrix as shown in Fig. 1b.

```
Algorithm 2: Estimator ($\mathbb{S}, L_{\mathbb{O}}, k, D, Q$, start_ts, end_t, N)
Input: $\mathbb{S}$ the set of $m$ concurrent data streams in the current window, $L_{\mathbb{O}}$ the list of the indices of the outliers in the window, $k$ the number of eigenvectors, $D$ the total number of attributes, $Q$ the estimated eigenvectors, start_ts the starting timestamp of the window, end_ts the end timestamp of the window, and N the number of data points in the window,
Output: eigenvector matrix $Q$, list of estimated outliers $\hat{\mathbb{O}}_j$ est_list
1    set O as an empty list
2    create a set of outlier indices from $L_{\mathbb{O}}$, $I_{\mathbb{O}} = set(L_{\mathbb{O}})$
3    Initialize a zero matrix Z of size $D \times k$
4    for each t between start_ts and end_ts do
5        if $|\mathbb{S}| > 1$ do ## when m > 1
6            x = combine tuples in $\mathbb{S}$ that share the same t into $x \in \mathbb{R}^D$
7        else
8            x = tuple at t
9        end if
10       $Z := Z + \frac{1}{N} x x^T Q$
11       if t is in $I_{\mathbb{O}}$ do
12           $O$.append($x$)
13       end if
14   end for
15   $Q$ := QR-decomposition of $Z$
16   Set $O$ as a matrix $\mathbb{O}$
17   $\hat{\mathbb{O}} = \mathbb{O} Q Q^T$
18   Set est_list as a list of length $|\mathbb{S}|$
19   for $0 \le j < |\mathbb{S}|$ do
20       $\hat{\mathbb{O}}_j :=$ slice_matrix($\mathbb{O}, L_{\mathbb{O}}[j]$)
21       est_list[j] = $\hat{\mathbb{O}}_j$
22   end for
23   return Q, est_list
```

To find $\hat{\mathbb{O}}_i^j$, *Est* uses the PCA-based approach that captures the correlations among the observed attributes to derive the k eigenvectors (principal components) and stores them in the matrix Q. In each window, *Est* combines the data points that share the same timestamp from all the data streams. If there is only one data stream, Est will use only the data points in that stream. The combined tuples, after their timestamp is omitted, are used to compute the eigenvectors in the matrix Q. When there are outliers $\mathbb{O}$ in that window, Q is then used to estimate the normal values of those outliers as follows:

$$\widehat{\mathbb{O}} = \mathbb{O}QQ^T \tag{1}$$

where the matrix $\mathbb{O} \in \mathbb{R}^{|I_{\mathbb{O}}| \times D}$, $|I_{\mathbb{O}}|$ is the number of unique timestamps when the outliers are detected in the current window, D is the total number of attributes in $\mathbb{S}$, and $Q \in \mathbb{R}^{D \times k}$, $k < D$. Referring to the transpose of the matrix in Fig. 1b), $\mathbb{O}$ is a 1×7 matrix because there is one outlier detected between 8:01 am and 8:05 am.

The incremental update of the eigenvectors or principal components allows *Est* to adapt to concept drift. When the data distribution changes, the eigenvectors adjust to that change. While the naïve PCA algorithm can be used to build new eigenvectors in every window, it ignores the data seen in the previous windows and requires multi-passes on the current window. Hence to update the eigenvector Q, we adopt DBPCA, a single-pass eigendecomposition algorithm described in [25].

Algorithm 2 explains how *Est* works. In Line 1, the algorithm initializes an empty list O to store the outliers detected in the current window. Line 2 ensures that there are no duplicate outlier timestamps if the outliers are detected in two or more streams at the same timestamp. Continuing our example in Fig. 1, $L_{\mathbb{O}} = [\{8:03\}, \{\}, \{\}]$ and $I_{\mathbb{O}} = \{8:03\}$. Line 3 gets the eigenvector matrix Q generated from the previous window. Lines 4–15, which are the DBPCA approach, are responsible for updating Q of the current window by first absorbing each combined vector x into a Z matrix and then then conduct QR decomposition on Z to get the updated eigenvectors Q. Lines 16–17 estimate the normal values of the outliers as one matrix $\widehat{\mathbb{O}} \in \mathbb{R}^{D \times |I_{\mathbb{O}}|}$. Finally, Lines 18–22 break down $\widehat{\mathbb{O}}$ so that each stream gets the estimated normal values of its outliers. The algorithm returns the updated estimated eigenvector matrix and the list of estimated normal values of outliers.

4.2 Temporal Neighbor Clustering

The temporal neighbor clustering component is a set of m independent functions $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ that runs in parallel with the estimator component. Each $\mathcal{C}_j$ receives data points from the data stream S_j and forms the temporal context that will be used in generating the outlying attributes of each $\mathbb{O}_i^j \in W_j^{\mathbb{T}}$.

As depicted in Fig. 2, the temporal context of a data stream can be further grouped into a set of clusters $\mathbb{C} = \{C_1, \ldots, C_l\}$ such that $C_a \cap C_b = \varnothing$ for all $a \neq b$ and $W_j^{\mathbb{T}} = \bigcup_{a=1}^{l} C_a$. $\mathbb{C}$ is considered the neighborhood outliers in $W_j^{\mathbb{T}}$. Each cluster has a center $c_a \in \mathbb{R}^d$ and has the total number of points assigned to the cluster. A point $\mathbb{X} \in W_j^{\mathbb{T}}$ is assigned to the cluster C_a if the distance between $\mathbb{X}$ and c_a is the closest. By excluding the timestamp attribute from $\mathbb{X}$, the distance function denoted as $d(\mathbb{X}, c_a)$ is formulated as $d(\mathbb{X}, c_a) = ||\mathbb{X} - c_a||$.

Recall that the continuous arrival of data points demands a single-pass computation; thus, forming clusters of data points in the window should be done incrementally. We use sequential k-means [12] to form $\mathbb{C}$. Due to the unbounded amount of data points in data streams, we cannot store all the data points in all the clusters in the main memory. To handle this problem, for each cluster $C_a \in \mathbb{C}$ we only keep n_a, the number of points in the cluster, and c_a, the centroid of the cluster.

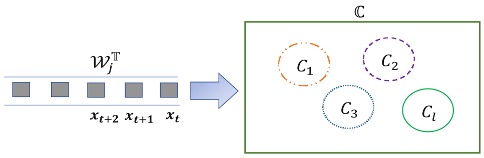

Fig. 2. Grouping Data Points of a Stream Sj

4.3 Outlying Attribute Generators

The outlying attribute generator component is a set of m independent functions $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_m\}$. Each $\mathcal{G}_j$ receives the inputs produced by the estimator component *Est* and the temporal clustering component $\mathcal{C}_j$. For each $\mathbb{O}_i^j \in \mathbb{O}_j^{\mathbb{T}}$, $\mathcal{G}_i$ will form an inlier class and an outlier class. The inlier class is generated by finding a cluster $C_a \in \mathbb{C}_j$ whose centroid c_a is the closest to $\hat{\mathbb{O}}_i^j$. We denote by dc_a the distance between c_a and $\mathbb{O}_i^j$. Recall that the cluster no longer keeps the data points it has absorbed, but it has the information about n_a the number of data points. Initially, the inlier class only has two members: c_a and $\hat{\mathbb{O}}_i^j$. The additional members are then added by generating n_a auxiliary data points whose distances to c_a is less than dc_a. All members of the inlier class are labeled 0. Figure 3a describes how the inlier class is formed.

The outlier class is formed by generating multivariate normal data centered around $\mathbb{O}_i^j$. To ensure that the members of the outlier class do not overlap with those of the inlier class, the standard deviation is set to be less than one-third of the distance between $\mathbb{O}_i^j$ and $\mathbb{O}_i^j$. All objects in the outlier class are labeled 1. Figure 3b illustrates how the outlier class is formed. After forming the inlier and outlier classes, the next step is to find a decision boundary that separates both groups using a linear support vector machine (SVM) [13]. The classifier applies lasso regularization [14] such that the resulting hyperplane has zero weights on any attributes that are not relevant in forming the boundary. The absolute values of the hyperplane's weights, $H_{\mathbb{O}_i^j} \in \mathbb{R}^{d_j}$ are used to calculate the outlying contribution scores stored in the vector $f_{\mathbb{O}_i^j} \in \mathbb{R}^{d_j}$ for the attributes of the outlier $\mathbb{O}_i^j$. This vector is computed using Eq. (2). Each attribute corresponds to a score in f_α. The attributes having a score higher than a given threshold are the outlying attributes of $\mathbb{O}_i^j$.

$$f_{\mathbb{O}_i^j} = \frac{H_{\mathbb{O}_i^j}}{\sum_{b=0}^{d_j-1} H_{\mathbb{O}_i^j}[b]} \tag{2}$$

5 Evaluation

In this section, we discuss our experimental setup and results comparing EXOS with four existing algorithms: COIN [8], MICOES [10], MacroBase [15, 16], and EXstream [9, 16, 17]. Like EXOS, all these algorithms are agnostic to the algorithm used for outlier detection. COIN is one of the algorithms that use inlier neighbors of an outlier as a local

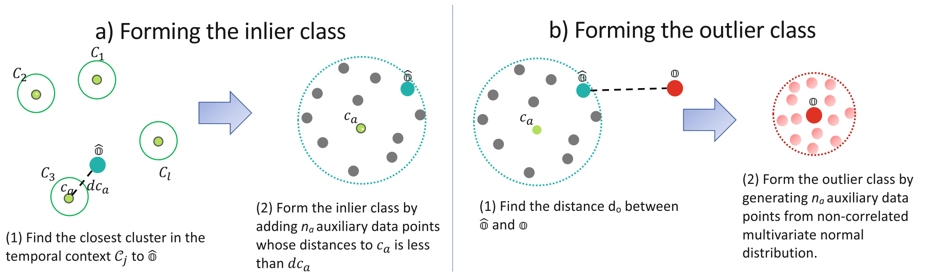
a) Forming the inlier class
C_1
C_2
C_3
C_l
c_a
dc_a
(1) Find the closest cluster in the temporal context C_j to $\hat{o}$
(2) Form the inlier class by adding n_a auxiliary data points whose distances to c_a is less than dc_a
b) Forming the outlier class
(1) Find the distance d_o between $\hat{o}$ and o
(2) Form the outlier class by generating n_a auxiliary data points from non-correlated multivariate normal distribution.

Fig. 3. Forming a) the inlier class and b) the outlier class

context to find the outlier's outlying attributes. It is designed for static data, yet we run it in batches to compare it with EXOS. The other three algorithms are proposed for data streams. EXstream is intended for interval or collective outliers, yet it can also be applied for point outliers. The source code of each algorithm is publicly available.

5.1 Experimental Setup

Dataset. We use three real datasets that reflect multiple-source data streams: Intel, Microtremor, and AMPds2. The Intel dataset [18] is multiple-sensor data where each sensor sent multi-dimensional data with the attributes of humidity, temperature, light, and voltages every 31 s. Since the sensors started sending measurements on different dates, we simulate data gathered between March 1^{st} –31^{st}, 2004 from Sensors 7, 9, 23, 25, 26, 29, 36, 38, and 44. The Microtremor dataset [19] was collected using temporary broadband seismometers over continuous time intervals ranging between 2 and 2.8 h. After examining the metadata, we decide to use the data collected from 4 seismic recording stations, 2030, 2031, 2032, and 2033, because they were collected on the same day. The AMPds dataset [20] contains the measurements of 21 electricity power, 3 water, and 2 natural gas meters at one-minute intervals of a residential house. Each meter sent a total of 1, 051, 200 readings for 2 years (April 2012 to March 2014). Power, water, and gas have 11, 2, and 3 attributes, respectively. These datasets do not have ground truth information for outliers and their outlying attributes. For performance evaluation purposes, we synthetically generate that information. Specifically, for each dataset, we inject outliers into the dataset by randomly selecting around 1% of the original data points to be outliers. For each selected outlier, we randomly select a number of attributes from the dataset to be the outlying attributes and set the value of each of those attributes to be far away from its mean value. For example, suppose in the Intel dataset, the data point #100 is chosen as an outlier and the temperature as the outlying attribute. The temperature value of the data point #100 is then set to be either *max (temperature)* + *delta* * *standard deviation(temperature)* or *min(temperature) – delta* * *standard deviation(temperature),* where *delta* >*3*. The summary of the datasets used for the evaluation is provided in Table 1.

In addition, we create synthetic datasets that consist of five and four groups of data whose attributes are correlated. Each group represents a data stream and has 10 attributes. To ensure that the data streams are correlated, we first generate a correlation matrix using a technique described in [21]. This technique allows us to specify whether the data attributes within and across data streams are highly or weakly correlated. The correlation matrix is then used to generate multivariate normal data. We inject outliers into the dataset in the same way as we do for the real datasets.

Table 1. Summary of the Datasets

Datasets	# Streams	Total number of data points	# Outliers injected	# Attributes per stream
Microtremor	4	7 M	70 K (1%)	3
AMPds2	25	26 M	260 K (1%)	2, 3, 11
Intel	9	401 K	4.5 K (1.12%)	4
Synthetic 1	5	50 K	500 (1%)	10
Synthetic 2	4	40 K	400 (1%)	10

Evaluation Metrics. We measure the effectiveness of the algorithms in finding outlying attributes using average precision, average recall, and average F1 score. For each outlier, we compute the numbers of True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) of its outlying attributes. For each outlier, we compute the numbers of True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN) of its outlying attributes. We then use the information to compute Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, F1 Score $= \frac{2*Recall*Precision}{Recall+Precision}$. In addition to measuring the accuracy of the algorithms, we also consider their efficiency. Given that we are dealing with a continuous arrival of data, we need to ensure that the algorithms can keep up with the incoming data in a timely manner. Therefore, we also measure the average explanation time of the algorithms as part of our evaluation process.

Software and Hardware. We implement EXOS in Python 3 and simulate the multi-source data streams in parallel using the multiprocessing package. Our source code is available on GitHub https://github.com/egawati/exos. We use the Python implementations of MacroBase and EXstream that are provided for the Exathlon benchmark [16]. We add some helper functions to the competitive algorithms so that they can run in parallel as we simulate synchronous data streams. The experiments were conducted on a MacBook Pro: macOS Catalina, Processor 2.7 GHz Quad-Core Intel Core i7, Memory 16 GB 1600 MHz DDR3.

5.2 Results and Analysis

We evaluate EXOS by simulating each dataset described in Sect. 5.1 as data streams. Since EXOS generates outlying attributes in the tumbling window where the number of data points inside the window depends on the data point arrival rate; we set the window

Table 2. Algorithm Performance Evaluation

Datasets	Algorithms	Avg precision	Avg recall	Avg F1 Score	Avg Execution Time (second)
Intel	COIN	0.49	**0.99**	0.63	0.042
	MICOES	0.5	0.91	0.61	0.020
	MacroBase	0.49	0.92	0.59	0.016
	EXstream	0.85	0.59	0.66	0.011
	EXOS	**0.95**	0.84	**0.85**	**0.010**
Microtremor	COIN	0.33	**0.99**	0.49	0.045
	MICOES	0.33	**0.99**	0.50	0.089
	MacroBase	0.34	**0.99**	0.51	0.007
	Exstream	**0.87**	0.87	0.87	**0.003**
	EXOS	0.83	**0.99**	**0.89**	**0.003**
AMPds2	COIN	0.43	0.86	0.54	0.045
	MICOES	0.39	0.80	0.48	0.020
	MacroBase	0.56	**0.89**	**0.64**	0.097
	Exstream	0.91	0.42	0.52	**0.007**
	EXOS	**0.94**	0.48	0.58	**0.007**
Synthetic 1	COIN	**1.00**	0.52	0.64	0.072
	MICOES	0.47	0.79	0.54	0.055
	MacroBase	0.3	**1.00**	0.44	0.599
	Exstream	0.98	0.45	0.56	0.050
	EXOS	0.84	0.87	**0.82**	**0.047**
Synthetic 2	COIN	0.42	0.86	0.54	0.076
	MICOES	0.51	0.58	0.51	0.234
	MacroBase	0.34	**1.00**	0.50	0.629
	Exstream	**0.99**	0.44	0.58	0.093
	EXOS	0.88	0.97	**0.91**	**0.058**

size to be 1,440 data points (1 day) for Intel and AMPds2 and 1,000 data points for Microtremor and Synthetic. Table 2 shows that EXOS has the highest average precision for Intel and AMPds2. Exstream get the best average precision for Microtremor and Synthetic 2 while COIN wins on Synthetic 1. When it comes to average recall, MacroBase wins for AMPds2, Synthetic 1 and 2, while COIN dominates for Intel. For Microtremor, the four algorithms achieve an average recall of 0.99, while Exstream 0.87.

Obviously, there is a tradeoff between precision and recall among these algorithms. Precision measures the extent of error by False Positives, while recall deals with the error caused by False Negatives. F1 score balances those scores. EXOS achieves the best F1 score in all the datasets except for AMPds2. The F1 scores of EXOS are 22%, 40%, 4%, 18%, and 37% better than those of COIN, 24%, 49%, 10%, 28%, and 40% better than those of MICOES, 26%, 38%, −6%, 38%, 41% better than those of MacroBase, and 19%, 2%, 6%, 26%, and 33% better than those of Exstream for the Intel, Microtremor, AMPds2, Synthetic 1, and Synthetic 2 datasets, respectively.

Table 2 shows that for all the datasets, EXOS has the fastest average execution time for generating outlying attributes for outliers. EXOS is 4.2, 15, 6.4, 1.5, and 1.3 times faster than COIN, 2, 29.6, 2.8, 1.1, and 4 times faster than MICOES, and 1.6, 2.3, 13.8, 12.7, and 10.8 times faster than MacroBase for the Intel, Microtremor, AMPds2, Synthetic 1, and Synthetic 2 datasets, respectively. Even though EXOS and Exstream have similar average execution times, EXOS outperforms Exstream in the average F1 scores for all the datasets.

The Impact of Concept Drift. We investigate the impact of data distribution changes, known as concept drift, on the performance of the algorithms. We simulate four data streams, each having ten attributes, eleven windows, and 1K data points per window. Our experiments focus on two key questions: (Q1) Does the performance of the algorithms alter when a concept drift occurs in a specific window?, and (Q2) How does the frequency of concept drift occurrences affect the algorithms' performance?

To address Question (Q1), we simulate the synthetic datasets with varying locations of concept drift occurrences, i.e., varying the IDs of the windows where concept drifts occur. Following the similar approaches used to verify that concept drifts indeed occur in a dataset [22, 23], we confirm the presence of concept drift by adding binary classification labels to the datasets and then training a logistic regression classifier using the data in Window 0 before testing it using the data in Windows 1 through 10. It is worth noting that this classification model is designed for static data and thus is expected to perform similarly across all the windows in the absence of concept drifts. However, as shown in Fig. 4, once there is a change in data distribution at Window *i*, the classifier performance begins to decline, indicating the presence of concept drifts in the datasets.

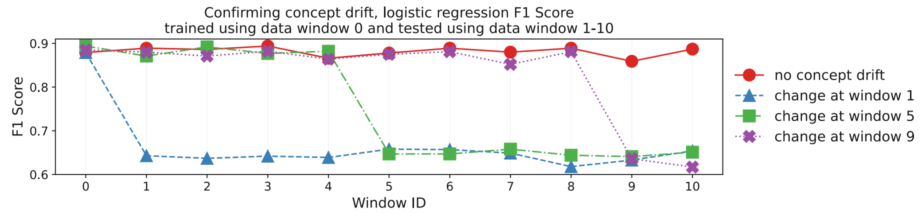

Fig. 4. Using a static data-based classifier to confirm whether a dataset indeed has concept drift

Figure 5 demonstrates the performance of the outlying attribute algorithms when the location of a concept drift occurrence is varied from Window 0 to Window 10. EXOS achieves the highest average F1 scores, followed by Exstream, MacroBase, MICOES, and COIN. However, MacroBase has the highest average explanation time, while other algorithms performed similarly. The consistent performance is maintained by all the algorithms regardless of the concept drift locations as their model and/or temporal context are updated in each window. Thus, we can conclude that the performance of the algorithms is unaffected by the locations of concept drifts.

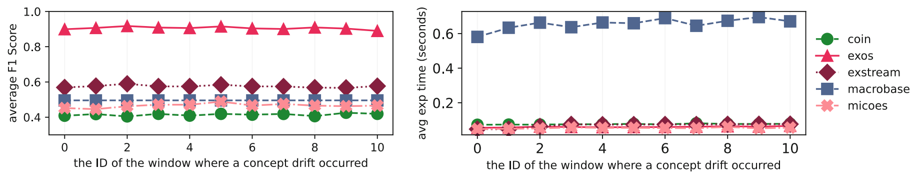

Fig. 5. The impact of the location of concept drift on the algorithms' average F1 Score and explanation time

To answer Question (Q2), we vary the occurrences of concept drifts across eleven windows ranging from 0% to 100%. A 0% concept drift indicates that Windows 1–10 have the same data distribution as Window 0. A 50% concept drift means from Window 1 through Window 10; there are five windows where data distribution changes. A 100% concept drift means the data distribution changes every time the window slides. We run experiments on four synthetic datasets. Sets 1, 2, and 3 are composed of multivariate Gaussian data. In Set 1, a change in the mean vectors occurs during specific windows while the covariance matrices remain unchanged. In contrast, in Set 2, only the covariance matrices change while the mean vectors stay constant. Finally, both the mean vectors and covariance matrices change simultaneously in Set 3.

Set 4 has mixed data distributions consisting of randomly chosen multivariate Gaussian, Beta, Gamma, and Exponential distributions. For instance, when we set the percentage concept drifts to be 50%; Windows 0–2 can have a Gaussian distribution, while Window 3 may use a gamma distribution. Likewise, Windows 4–7 could be exponential, Window 8 beta and finally, Windows 9–10 are gamma distributed.

Figure 6 displays the performance of the algorithms on Sets 1–4, with EXOS having the best average F1 score. All the algorithms have similar average explanation times while MacroBase has the highest one. The frequency of concept drifts hardly has any effect on the performance of EXOS, EXstream, Coin, and MacroBase. With EXOS, the eigenvectors used to estimate the normal values of the outliers in a window are updated when the window slides. The same update applies to its temporal context component, which, together with the estimated normal values of the outliers, forms the inlier and outlier classes. Therefore, even when a window data distribution differs from that in the previous window, the EXOS components are constantly adjusted, allowing it to maintain its performance. A similar window-sliding update on the explanation model also applies to the other algorithms. Even though COIN is intended for static data, we run it in batches such that the subsets of data points used to build its outlier explanation model are continually updated as the window slides. The average F1 score slightly fluctuates on MICOES, indicating that the temporal context formed using denstream-based micro-clusters in a window still mixes with some micro-clusters from the previous window. MICOES's average explanation time also slightly increases in Sets 2–4 when the frequency of concept drifts is getting higher. This explanation time increase relates to the maintenance of its micro-clusters. When the data distribution changes more often, the state of its micro-clusters also changes, requiring more time to maintain/update.

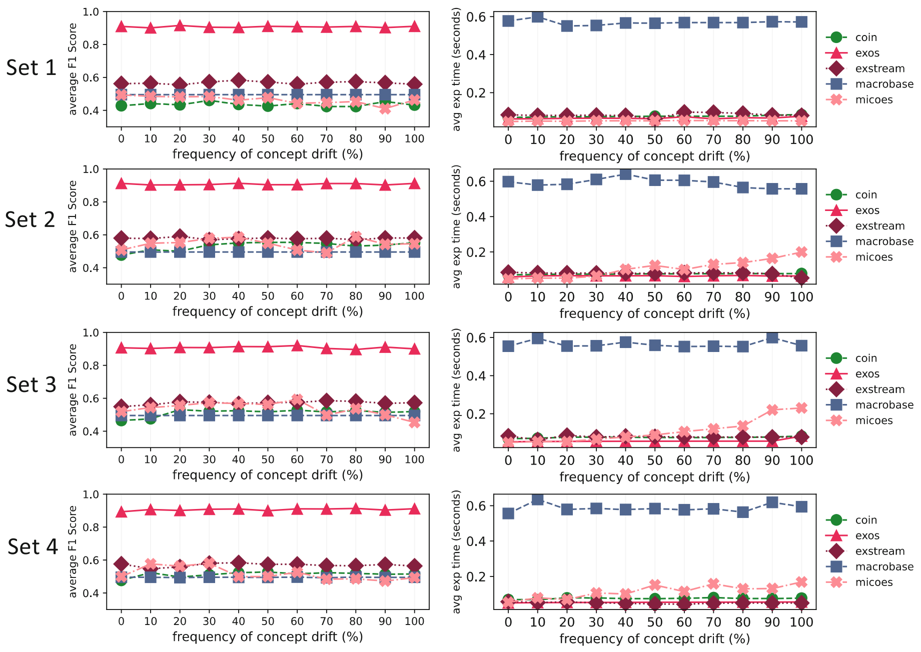

Fig. 6. The Impact of the frequency of concept drifts on average F1 score and explanation time

EXOS' Parameter Study. We also studied the impact of three EXOS user-defined parameters using the synthetic dataset, which we report below.

1) **Impact of the number of eigenvectors (k)**

In order to generate the estimated normal value of each outlier found in each window, EXOS needs the estimated eigenvectors (principal components) whose size depends on k. k can be varied from 1 to D (the total number of attributes). In our study, we vary k from 1 to 10. Figure 7 tells us that k has an impact on the average precision, recall, and F1 score. As k increases, the average recall also increases but the average precision and F1 score decrease. However, the average F1 score remains almost constant. It is not surprising that as k gets larger, the average explanation time gets longer as the EXOS' estimation component uses more eigenvectors. Therefore, to provide the explanation quickly, a small value of k such as $k = 1$ would be preferred.

2) **Impact of the outlier attribute contribution threshold (γ)**

The threshold γ tells EXOS which attributes to consider as the outlying attributes of an outlier after finding the decision boundary that separates the inlier class from the outlier class for that outlier. $\gamma = 0.01$ indicates that only attributes that have the outlying contribution scores greater than 1% are considered the outlying attributes. The study on the synthetic dataset reveals that γ has a small impact on the algorithm's performance as shown in Fig. 7.

3) **Impact of the window size**

In generating outlier explanations, EXOS depends on the time period that determines the size of the tumbling window used by its components. For the synthetic data, we varied the window size from 1,000 to 10,000 data points. Figure 7 shows that when the window size increases, EXOS' average recall increases but average precision and F1 score decrease. Thus, more inliers compared with the outlier does not mean a more accurate explanation of outlying attributes. The larger the window size also means the larger the average running time. The running time using the window size of 10K is about three times of that using the window size of 1K.

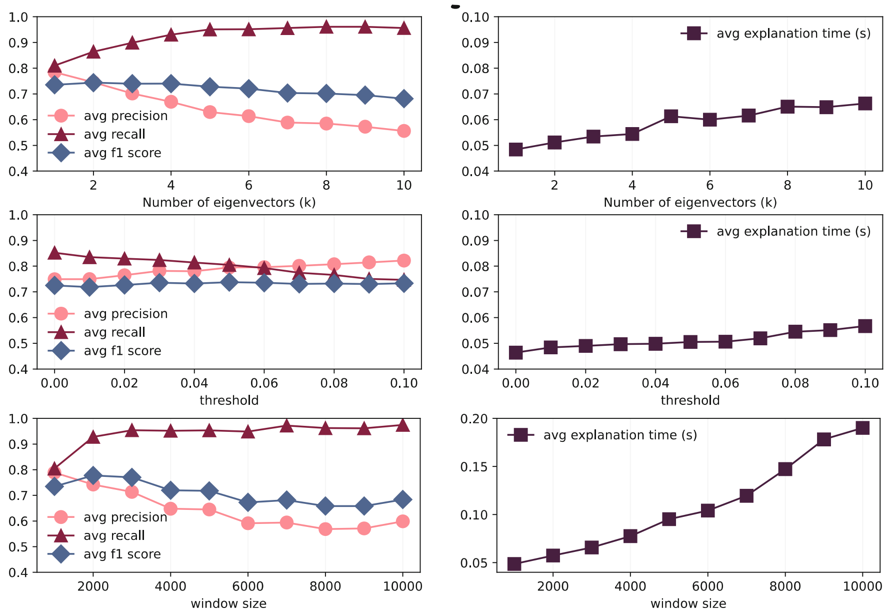

Fig. 7. EXOS parameter study

6 Conclusions

Our experiments on the three real datasets and two synthetic datasets show that EXOS is promising in generating outlying attributes for each outlier in data streams. It addresses the data streams' characteristics of the unbounded data volume, concept drift, and cross-correlation. For all the studied datasets, it achieves the best average F1 score (except for one dataset) and the fastest average running time compared with the existing algorithms. For future work, we plan to extend the estimator component to deal with asynchronous data streams.

References

1. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. **41**, 15:1–15:58 (2009)
2. Tran, L., Mun, M.Y., Shahabi, C.: Real-time distance-based outlier detection in data streams. In: Proceedings of the VLDB Endowment (2020)
3. Yoon, S., Lee, J.G., Lee, B.S.: NETS: Extremely fast outlier detection from a data stream via set-based processing. In: Proceedings of the VLDB Endowment (2018)
4. Siddiqui, M.A., Fern, A., Dietterich, T.G., Wong, W.-K.: Sequential feature explanations for anomaly detection. ACM Trans. Knowl. Discov. Data **13**, 1–22 (2019)
5. Panjei, E., Gruenwald, L., Leal, E., Nguyen, C., Silvia, S.: A survey on outlier explanations. VLDB J. **31**, 977–1008 (2022)
6. Sadik, M., Gruenwald, L.: Research issues in outlier detection for data streams. SIGKDD Explor. **15**, 33–40 (2014)
7. Micenková, B., Ng, R.T., Dang, X.-H., Assent, I.: Explaining outliers by subspace separability. In: 2013 IEEE 13th International Conference on Data Mining, pp. 518–527 (2013)
8. Liu, N., Shin, D., Hu, X.: Contextual outlier interpretation. In: IJCAI (2018)
9. Song, F., Diao, Y., Read, J., Stiegler, A., Bifet, A.: EXAD: a system for explainable anomaly detection on big data traces. In: IEEE International Conference on Data Mining Workshops, pp. 1435–1440 (2018)
10. Panjei, E., Gruenwald, L., Leal, E., Nguyen, C.: Micro-clusters-based outlier explanations for data (2021) Streams. https://sites.google.com/view/andea2021/accepted-papers
11. Li, C.L., Lin, H. ten, Lu, C.J.: Rivalry of two families of algorithms for memory-restricted streaming PCA. In: Proceedings of International Conference on Artificial Intelligence and Statistics (2016)
12. Ackerman, M., Dasgupta, S.: Incremental clustering: the case for extra clusters. In: Advances in Neural Information Processing Systems (2014)
13. Boser, B.E., Guyon, I.M., Vapnik, V.N.: Training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (1992)
14. Tibshirani, R.: Regression shrinkage and selection via the lasso: a retrospective. J. R. Stat. Soc. Ser. B Stat. Methodol. **73** (2011)
15. Bailis, P., Gan, E., Madden, S., Narayanan, D., Rong, K., Suri, S.: MacroBase: prioritizing attention in fast data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (2017)
16. Jacob, V., Song, F., Stiegler, A., Rad, B., Diao, Y., Tatbul, N.: Exathlon: A benchmark for explainable anomaly detection over time series. In: Proceedings of the VLDB Endowment (2021)
17. Zhang, H., Diao, Y., Meliou, A.: EXstream: explaining anomalies in event stream monitoring. In: International Conference on Extending Database Technology (2017)
18. Bodik, P., Hong, W., Guestrin, C., Madden, S., Paskin, M., Thibaux, R.: Intel Lab Data (2004). http://db.csail.mit.edu/labdata/labdata.html
19. Buckreis, T., Winders, A., Wang, P., Brandenberg, S., Stewart, J.: Microtremor Data Collected in Sacramento-San Joaquin Delta Region of California (2021). https://doi.org/10.17603/ds2-dk6t-8610

20. Makonin, S.: AMPds2: The Almanac of Minutely Power Dataset (Version 2) (2016). https://doi.org/10.7910/DVN/FIE0S4
21. Hardin, J., Garcia, S.R., Golan, D.: A method for generating realistic correlation matrices. Ann Appl Stat. **7**, 1733–1762 (2013)
22. Gu, F.: Concept Drift Detection for Machine Learning with Stream Data (2019). https://opus.lib.uts.edu.au/bitstream/10453/140165/2/02whole.pdf
23. Das, S.: Best Practices for Dealing with Concept Drift. https://neptune.ai/blog/concept-drift-best-practices. Accessed 03 Apr 2023