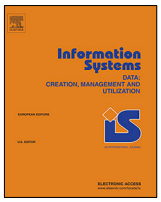
Information Systems 104 (2022) 101894

Contents lists available at ScienceDirect

Information Systems

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



Keeping our rivers clean: Information-theoretic online anomaly detection for streaming business process events

Jonghyeon Ko, Marco Comuzzi *

Department of Industrial Engineering, Ulsan National Institute of Science and Technology, Republic of Korea

ARTICLE INFO

Article history:
Received 10 March 2021
Received in revised form 27 July 2021
Accepted 13 September 2021
Available online 29 September 2021
Recommended by Stefanie Rinderle-Ma

Keywords:
Process mining
Online anomaly detection
Event streams
Information measure
Statistical leverage

ABSTRACT

Event log anomaly detection aims at identifying anomalous information in the logs generated by the execution of business processes. While several techniques for detecting trace-level anomalies in event logs in offline settings, i.e., when event logs are processed as a batch, have appeared recently in the literature, such techniques are currently lacking for online settings, i.e., when events are processed as a stream. Event log anomaly detection in online settings can be crucial for discovering anomalies in process execution as soon as they occur and, consequently, allowing to take early corrective actions. Moreover, it is also crucial for creating models that can adapt to concept drift in the process generating the events. This paper describes a novel approach to event log anomaly detection in process event streams: we define a general framework in which different anomaly detection methods can be plugged in and we propose and evaluate our own method based on statistical leverage. The leverage is an information-theoretic measure that has been used extensively in statistics to identify outliers and it has been adapted in this paper to the specific scenario of event streams. The proposed approach has been evaluated on artificial and real event streams and also on artificial event streams characterised by concept drift.

1. Introduction

Information logged during the execution of business processes is available in so-called event logs [1], which contain events belonging to different process instances (or *cases*). Each event is described by multiple attributes, such as a timestamp and a label capturing the activity in the process that was executed. Event logs can be compiled from the logs of different types of business applications, such as banking systems or medical information systems [2]. The sequence of events regarding one particular execution of a process (a process *case*) is usually called a *trace* [3–5].

Event logs are prone to errors, which can stem from a variety of root causes [6–8], such as system malfunctioning or sub-optimal resource behaviour. For instance, human resources may forget to log the execution of specific activities in a process, or a system reboot may assign a different case identifier to all the new events recorded after rebooting. Errors in event logs hamper the possibility of extracting useful process insights from event log analysis and should therefore be fixed as early as possible [9]. To this end, the research field of event log anomaly detection (or event log cleaning) has emerged recently, providing methods to detect anomalies at the trace level [3,6,7,10,11], i.e., concerning the order and occurrence of activities in a process trace, and at the event level [5,12,13], i.e., concerning the value of attributes of events, using a variety of different approaches. Note that event log anomaly detection is normally unsupervised and (process) model-agnostic, that is, it does not assume the existence of (i) labelled historical cases (anomalous v. normal) and (ii) a model of the business process generating the data or clean traces from which it can be extracted. The latter aspect separates this research field from traditional process mining research on conformance checking [14].

The traditional approach in event log anomaly detection is to consider the event log as a *batch* of data to train an unsupervised model that learns how to discriminate anomalous from normal data. The model may rely on different design paradigms, such as probabilistic, e.g., based on trace frequency [6], distance-based, e.g., based on k-Nearest Neighbour (k-NN) [15], reconstruction-based, based on autoencoders [13], or statistical, e.g., based on information-theoretic measures [1].

Besides the *batch* perspective, a.k.a. the *offline* perspective, an alternative perspective on event log data is the one of a *stream* of events, a.k.a. the *online* perspective. Since events are timestamped, an event log is in practice equivalent to a stream of events, which become available for the analysis *online*, i.e., as soon as they are logged during the process execution. This perspective has been adopted successfully in the research on online process

* Corresponding author.
E-mail address: mcomuzzi@unist.ac.kr (M. Comuzzi).

https://doi.org/10.1016/j.is.2021.101894
0306-4379/

discovery [16,17] and online conformance checking [9,18,19], but its adoption in event log anomaly detection is limited. The advantage of treating event logs as a stream is twofold: first, it enables to develop event log analysis models that can be updated with each individual new event; second, if designed properly, it can naturally adapt to concept drift in the event log [20], such as a change in the process model. If a concept drift in the process generating events occurs, in fact, the stream of events will reflect such a drift as soon as it occurs and the models developed using it will soon adapt to the new process execution conditions naturally.

Event log anomaly detection in online settings can be crucial for discovering anomalies in process execution as soon as they occur and, consequently, allowing to promptly take early corrective actions. The online settings, however, obviously introduce additional challenges to the design of an event log anomaly detection method. In particular, owing to the finite memory assumption [9,18,19,21], only a limited number of (recent) events should be available at any given time for the analysis. This prevents to apply effectively some of the approaches mentioned earlier for event log anomaly detection in offline settings. Probabilistic methods that detect anomalies after having created an intermediate model of frequent process behaviour [6,10,11] are hampered by the fact that only a limited number of events may be available to create such models, particularly at early stages of the process execution. The online settings also prevent the application of machine learning reconstructive techniques for anomaly detection, e.g. [13,22]. These, in fact, normally rely on deep learning models, which require a high number of data points (complete process traces in this scenario) to be trained effectively. Also, any update of these models may require a long training time. In this regard, incremental machine learning approaches to deal with data streams have emerged recently [9,23,24], but have not been considered yet in the event log anomaly detection literature.

The work by Tavares et al. [21] is the only one addressing the issue of anomaly detection over event streams. In particular, the authors apply DenStream [25], a clustering algorithm for event streams, to detect point anomalies specified by Principal Component Analysis (PCA) based on profiles of cases over event streams. These point anomalies, however, normally do not reflect real-life anomaly patterns commonly considered by event log anomaly detection in offline settings, such as inserting, skipping or switching events in a trace. Moreover, to address the finite memory constraint, this approach, following others in online conformance checking [9,18], considers a sliding window of events that samples a fixed number of recent events. Such a design of the sliding window can be counterproductive when detecting trace-level anomalies, since it does not guarantee that all events received belonging to a given trace are retained when taking a decision on whether to label this trace anomalous or normal.

In this paper we propose an information-theoretic approach to online event log anomaly detection at the trace level. Specifically, we propose a trace anomaly score based on statistical leverage [26]. The leverage is a relative measure of the information content of observations in a dataset that has been used extensively in statistics to develop observation distance measures and outlier detection techniques. Since the leverage captures the information content of one observation in respect of all others in a dataset, the anomaly score proposed in this paper is robust, i.e., it can always be calculated reliably based on the information available at any point in time, resulting in an anomaly detection method that does not require extensive amount of data to be implemented effectively.

While an anomaly score based on leverage has been introduced by the authors in a previous publication [1], in this paper we focus on its application in online settings. In particular, the online adaptation that we propose builds on the following three design pillars:

- Non-incremental approach: while other approaches to streaming data analysis often consider incremental models, i.e., they maintain one decision model, e.g., for predictive analytics, updated by each new event in the stream, we propose an approach in which the anomaly detection model is recalculated from scratch with each new event. This choice is dictated by the proposed anomaly score calculation, which is not incremental. While this choice may appear inefficient in terms of calculation cost, the leverage-based anomaly score has been shown to be robust and lightweight in runtime and memory required for its calculation in previous work [1,27]. As we will show, the method proposed in this paper still achieves a better run time performance when compared to baseline anomaly detection techniques for streaming data.
- One-pass approach: while other approaches to streaming data analysis in non-incremental settings often consider multiple passes over the data, particularly when having to deal with concept drift [28,29], for instance separating online and offline steps for anomaly detection and a model-updating step by concept drift detection, the approach that we propose reads the data received in the stream only once while detecting anomalies; the concept drift adaptation is a by-product of the design, since the anomaly scores are recalculated from scratch with each new event;
- Trace-level sliding window: to avoid dropping events from traces still represented in a sliding window, we introduce the concept of trace-level sliding window, in which all the events received belonging to a given number of recently started traces are retained, making it possible to achieve finite memory usage, without restricting the analysis to a fixed number of recently received events.

The work presented here extends the one [27] presented by the authors at the SA4PM workshop held at the International Conference on Process Mining in 2020. Besides a more thorough discussion and presentation of the contribution and related work, the extension concerns the following aspects: (i) to consider the prefix length-based encoding of events in order to improve the anomaly detection accuracy; (ii) to present an extensive evaluation of the proposed approach, which includes additional event logs, a comparison with baseline anomaly detection methods in event stream analysis, and an in-depth analysis of the performance using event logs characterised by different types of concept drift.

After having presented the related work in Section 2, Section 3 introduces the research framework, which is evaluated in Section 4 on both artificial and real event logs injected with trace-level anomalies. Conclusions finally are drawn in Section 5.

2. Related work

Existing unsupervised (or model-agnostic) anomaly detection methods for data streams can be classified into 6 categories: Nearest Neighbour-based (NN-based), clustering-based, statistical, subspace-based, classifier-based, and ensemble-based [30].

NN-based approaches are based on either distance-based algorithms, which filter outliers because they are too distant from other normal observations, or density-based algorithms, which calculate the density in the surroundings of each observation to identify anomalies. Angiulli and Fassetti [28] introduce a novel notion of outlier query on the k-Nearest Neighbour (k-NN) model to detect distance-based anomalies under concept drift, which is a typical issue in data streams. Kontaki et al. [31] propose a more efficient and flexible k-NN model to reduce the storage overhead by integrating microcluster-based algorithm when processing streams. Pokrajac et al. [23] have proposed an incremental

algorithm that calculates the local densities of observations in data streams. Generally, the performance of approaches based on k-NN is strongly influenced by the choice of the value of k. Finding an optimal value of k in unsupervised settings is often a challenge.

Clustering-based approaches divide observations into groups of similar ones, defining outliers as observations either belonging to small clusters or far away from the centre of the closest cluster. Liu et al. [32] have developed an extended DenStream clustering algorithm that allows to distinguish between outliers and new patterns in event streams. Wang et al. [33] have applied the autonomic affinity propagation model [34] to streams in computer networks using a forgetting factor that puts more weights on recent models. Although clustering-based approaches can be efficiently designed in online settings, similarly to NN-based approaches they require a priori knowledge to optimally set the value of parameters, such as the size of the clusters. Feature reduction in the case of high-dimensional data is also a typical issue of this type of approaches.

Statistical methods detect outliers using statistical tests on a defined hypothesis or using an information-theoretic measure, such as likelihood or entropy. Wang et al. [35] propose two distributional approaches to detect outliers in 1-dimensional data streams based on the Tukey and Relative entropy methods. For multi-dimensional data, Samparthi and Verma [36] and Uddin et al. [37] have proposed kernel density estimation-based approaches for sensor networks data and power grids stream data, respectively. Similarly to statistical approaches that identify outliers based on distributional changes, subspace-based approaches detect subspace anomalies deviating from the normal subspaces. For instance, Dos Santos Teixeira and Milidiú [20] have proposed a non-parametric unsupervised subspace-based approach to detect anomalous subspaces in multi-variate numerical data streams by estimating a rank parameter to maintain a wide range of sampled data. On the one hand, statistical and subspace-based approaches are based on strong assumptions about the distribution of normal or anomalous data. On the other hand, they depend less than other methods, such as NN-based ones, on the value assigned to parameters. The combination of these two factors makes them generally robust, but less sensitive.

Classifier-based approaches, which may include deep learning, filter outliers by learning a normal space of data through training and excluding observations that are far from it. Uddin and Kuh [29] utilise the least squares version of the One-Class SVM (OC-SVM) to efficiently approximate the normal space of power grid data instead of solving an optimisation problem analytically. Tuor et al. [38] introduce two deep learning models that use DNN and LSTM for unsupervised anomaly detection of network intrusion. This type of models are a typical example of parametric approaches that learn a normal space of data offline. Since machine learning classifiers can analyse high-dimensional data under the assumption that attributes are independent, these models are also suitable for anomaly detection with high-dimensional data. However, to avoid overfitting, they also require a large number of observations to learn a model of normal data.

Ensemble-based approaches are based on the key idea to combine different models. Since computational run time and memory usage can be a major constraint when processing data streams, the most adopted ensemble methods for data streams are based on tree isolation mechanisms, which consider multiple simple classifiers. In this regard, the isolation Forest (iForest) model [39,40] is the state-of-the-art for anomaly detection over data streams. Similarly to the classifier-based approaches, the performance of iForest depends on proper setting of the parameters, such as the number of trees or the size of the window of recent observations to be considered [40].

While there is only limited literature regarding online event log anomaly detection, a number of recent contributions have focused on online conformance checking, which detects behaviour in an event stream deviating from a given process model. Conformance checking can be seen as model-aware anomaly detection, since process models, given or extracted from clean traces, can be seen as signatures of positive behaviour to detect anomalies. As referred by [19], there are currently two research lines in online conformance checking: the prefix-alignment approach [9] and the model-based approach [18,19].

Conformance checking on streaming events tends to overestimate the computation of optimal alignments since streaming events dismiss the assumption that traces are complete. In order to avoid this issue, Van Zelst et al. [9] provide a novel incremental/online conformance checking technique that uses prefix alignment. However, prefix alignment has to be carefully designed in warm start scenarios, to avoid dropping relevant events at the start of an event stream, and it also shows high computational complexity. Alternatively, the Online Conformance Transition System (OCTS) [18] has been proposed to process incomplete traces by partially checking compliance on regions of a process. This technique also suffers from high complexity on computation of regions as well as the issue with the warm start scenario highlighted earlier. In [19], the first solution to achieve a warm start with streaming events has been proposed, introducing a weak order relations in Petri net unfolding. Generally, besides requiring a process model, model-aware online anomaly detection has a limited capability to deal with concept drift, since the drift can only be handled once it appears in the model of the process behaviour used to detect anomalies.

Regarding model-agnostic or unsupervised anomaly detection in event streams, Tavares et al. [21] detect point anomalies defined by Principal Component Analysis (PCA) [41] applying the DenStream clustering algorithm [25]. Since DenStream is based on frequency histograms, which ignore the sequential feature of events belonging to a trace, this approach cannot detect typical anomaly patterns in business process event streams that *break* the normal process flow, such as skipping an event or moving events earlier/later in a trace.

3. Research framework

This section introduces the research framework and, in particular, the online anomaly detection procedure that we have designed and evaluated. To do so, we first introduce some preliminary notation and a few basic concepts relevant in the online anomaly detection scenario. A separate sub-section (Section 3.1) presents the details of the anomaly detection score used in the proposed framework.

Preliminaries. A process event stream E is an infinite sequence of events $e = \langle c, a, t \rangle$, where c is the id of the process case to which an event belongs, a is the activity label, and t is the timestamp of the event. Note that events in real world event logs may include additional attributes, such as resource id and department, but we only consider the activity as our objective is to detect anomalies at the level of occurrence and order of events in traces. We use the notation $\#_x(e)$, with $x \in \{c, a, t\}$, to refer to an individual attribute of an event and, when identifying the activity label and timestamp of an event is not necessary, we refer to an event as e_c. The events belonging to the same case i form a trace $\sigma_i = \{e_{i,j}\}$, such that, $\forall j$, $\#_t(e_{i,j}) < \#_t(e_{i,j+1})$ and $\nexists k : \#_t(e_{i,j}) < \#_t(e_{i,k}) < \#_t(e_{i,j+1})$. We refer to $\mathcal{E}$ and $\mathcal{S}$ as the universe of events and sequences of events, respectively, and to $A = \{a_1, \ldots, a_K\}$ as the domain of activity labels $\#_a(\cdot)$. The function $l : \mathcal{S} \longrightarrow \mathbb{N}$ returns the current length $l(\sigma_i)$ of a trace, calculated as the number of events received in the stream belonging to σ_i. The prefix function

$pref : \mathcal{S} \times \mathbb{N} \longrightarrow \mathcal{S} \cup \perp$ returns either the first n events received of a trace, that is $pref(\sigma_i, n) = \{e_{i,1}, \ldots, e_{i,n}\}$ if $l(\sigma_i) \geq n$, or $\perp$ (undefined) otherwise. Note that $l(pref(\sigma_i), n) = n$, $\forall i$ and $\forall n : l(\sigma_i) \geq n$.

Grace period. Similarly to other online anomaly detection methods in the literature [21,42], for practical reasons, it is reasonable to begin taking decisions on trace anomaly only after a sufficient number of events have been received. For this purpose, we introduce the parameter Grace Period (*GP*, with $GP \in \mathbb{N}^+$), defined as the number of traces for which at least 2 events have been received, i.e., $GP = |\sigma : pref(\sigma, 2) \neq \perp|$. For example, if GP=100, the online anomaly detection starts from the first event after having received at least the first 2 events of 100 different traces.

Trace-based sliding window. One of the main problems of data analysis in online settings is to design methods that have finite memory usage. To deal with this issue, the non-incremental event stream analysis approaches, e.g., [9,19,21,40,43], adopt a sliding window approach, in which only a number of observations received recently are considered for the analysis.

In the context of process event streams, we introduce a *trace-based* sliding window, which aims at keeping in the memory only the events associated with a finite number of recent traces. Let us refer to this window of traces as $SW = \{\sigma_i\}$ and to $|SW| = |\{\sigma \in SW : l(\sigma) \geq 2\}|$ as the size of the window, i.e., the number of traces represented by at least two events in *SW*. Now, let us define a parameter W, with $W > 0$, setting the upper limit of $|SW|$, i.e., the maximum number of traces with at least two events in the window. When $|SW| = W$ and a new event is received belonging to a trace in *SW* not represented by two events at least, then all the events belonging to the earliest trace in *SW* are removed. The earliest trace in *SW* is the one characterised by the earliest timestamp of the first event. Note that, since *SW* may also include traces represented by only one event, the number of traces represented by at least one event in *SW* can be higher than W (note also, however, that is impossible to take a decision regarding the trace-level anomaly of a trace by looking only at its first event). In the remainder, let us also refer to L^{max} as the length of the longest trace in *SW*, i.e., $L^{max} = max_{\sigma \in SW} l(\sigma)$. Limiting the number of traces in *SW* does not limit the number of total events considered for analysis. Therefore, we introduce an additional parameter W_e, with $W_e > 0$, which sets the maximum number of events in *SW*. Let us define $|SW_e|$ as the total number of events in *SW*; if $|SW_e| > W_e$, then the events belonging to the oldest traces in *SW* are removed until $|SW_e| \leq W_e$.

After having defined event streams and introduced the basic concepts of grace period and (trace-based) sliding window, we now introduce the main procedure for the online anomaly detection in Algorithm 1. The input of this procedure are a process event stream, the parameters *GP*, *W*, and W_e, a flag controlling how the events are encoded to calculate the anomaly score, and the value of the anomaly threshold T.

In Algorithm 1, each time a new event $e_{i,j}$ is received, it is mapped to the corresponding trace σ_i (line 3), which can be either a new one if $e_{i,j}$ is the first event of σ_i or an existing one in *SW*. Then, an initialisation step is required to meet the *warm start* conditions specified by the parameter *GP*. Once the warm start condition is met (line 4), the size of *SW* is checked (lines 5–6) and, if t$|SW|$ is greater than W, then the events belonging to the earliest trace in *SW* (excluding the current one to which $e_{i,j}$ belongs) are deleted. Then, the condition of the total number of events W_e is verified: the oldest traces in *SW* may be removed until the total number of events gets below W_e (lines 7–8).

At this point, it is important to clarify the approach to memory management that we take in this framework. One typical approach in online (streaming) process mining is to consider only a recent number of events for the analysis, which are stored in the memory. This solution allows to limit the amount of memory used, but it also prevents to execute the analysis for many events for which, when received, the *context*, i.e., the event(s) that precede it in a trace, are missing. In our case, this happens for a new event e belonging to a given trace t that is received long after a number of events for t already have arrived, such that the events already received for t have been deleted from the sliding window. Now, such a situation makes it impossible to run, for instance, online conformance checking or to consider the trace t for online process discovery, because the event e is missing its context, i.e., it cannot be matched to the events that precede it in t. Approaches published in the literature, then, would simply ignore this event e [9]. This can be a reasonable choice in online process discovery, since other traces may be available in the sliding window to discover a high quality process anyway, or in conformance checking, in which simply the checking of trace t will not be executed. In anomaly detection, though, any event e can be the one making a trace anomalous. It is important, therefore, to be able to take a decision for each possible event received using its appropriate context.

To do so, in the proposed framework we separate the volatile memory, i.e., the *memory* that we have referred to until now, the use of which must remain limited, from the persistent storage, which we assume to be always available to store the events received in the stream that we do not want to keep in the memory. In Algorithm 1, removing the oldest trace in *SW* means to move all its events to the persistent storage, so that they may be reloaded in the memory in the future should another event belonging to the same trace be received. Note that the logic to recall the events of a trace from the persistent storage into the memory is part of the *update* of the content of *SW* (line 3).

The next step in Algorithm 1 concerns the encoding of traces (lines 9–12). We consider two types of encoding, namely *trace-level* and *prefix-length*, which are explained in depth later in

Algorithm 1: Online trace anomaly detection

```
1  Inputs: E : stream of events
           GP : grace period parameter
           W : maximum number of traces in SW
           W_e : maximum number of events in SW
           T : a threshold on anomaly score (default=0.2)
           ENC : type of encoding flag
2  while new e_{i,j} ∈ E received do
3      map e_{i,j} to trace σ_i and update SW
4      if |SW| ≥ GP                          ▷ Check GP condition
5          if |SW| ≥ W                       ▷ Trace-based Sliding Window
6              Remove the oldest trace from SW
7          while |SW_e| ≤ W_e do
8              Remove the oldest trace from SW
9          if ENC = trace-level
10             M ← traceLevelEncoding(SW)
11         else if ENC = prefix-length
12             M ← prefixLengthEncoding(SW)
13         Scores ← anomalyScore(M)          ▷ Discussed in Section 3.1
14         if Scores(σ_i) ≥ T
15             label σ_i 'anomalous'
           else
16             label σ_i 'normal'
17     else continue
```

this section. In the *trace-level* encoding, each trace is encoded into a single feature vector, whereas the *prefix-length* encoding considers, for a given trace, the set of all encoded prefixes that can be derived from it. The encoding returns a matrix M of encoded prefixes or traces. Specifically, if the *trace-level* encoding is used, then M contains all the encoded traces in SW; if the *prefix-length* encoding is used, assuming that the current event is the pth received for a given trace σ, then M contains all the prefixes of length p.

Finally, the last two steps concern the calculation of the anomaly score and the anomaly detection. First, an anomaly score is calculated for each trace in M (line 13). The information-theoretic anomaly score considered in this paper is based on the previous work of the authors [1] and it is described in depth in Section 3.1. Then, in the anomaly detection step (line 14–16), the objective is to establish whether the trace σ_i modified by the current event $e_{i,j}$ is normal or anomalous. In offline settings, this step may be done manually by decision makers or using a variable threshold based on the fitting of an expected probability distribution of the scores [1,5]. In online settings, it is advisable to have a quicker and fully automated way of detecting anomalies. Owing to this reason, anomalies are determined using a threshold T: if the anomaly score of a trace is greater than T, then a trace is considered anomalous, and normal otherwise. In the experiments, we consider the value $T = 0.2$, which has been shown by previous work to be a reasonable assumption in the case of event log anomaly detection [27].

Encoding of traces. As mentioned above, we consider two different encoding approaches to generate a numeric feature matrix M for a given set of traces in the sliding window SW: (i) *trace-level* and (ii) *prefix-length* (see Fig. 1).

In general, all traces (or prefixes) are encoded using one-hot encoding of activity labels. Let us refer to $A^{SW} \subseteq A$ as the set of activity labels represented by at least one event in SW, that is, $A^{SW} = \{a \in A : \exists \sigma_i \in SW \, s.t. \, e_{i,j} \in \sigma_i \wedge \#_a(e_{i,j}) = a\}$, then each event $e_{i,j} \in \sigma_i$, with $\sigma_i \in SW$ is encoded into a set K dummy attributes $d_{i,j,k}$ such that:

$$d_{i,j,k} = \begin{cases} 1 & \text{if } \#_a(e_{i,j}) = a_k \in A^{SW} \\ 0 & \text{otherwise} \end{cases}$$

In the *trace-level* encoding, we consider one-hot encoded events horizontally concatenated for each trace, which yields a numeric matrix M of $|SW|$ rows (traces) and $J = L^{max} \times K$ columns (attributes). Since traces in SW have different length, the matrix M should include `null` values for the traces that are shorter than the longest one(s) in SW. Similarly to other studies [13,22], we apply zero padding to represent these `null` values. For example, given a case consisting of 4 events and $L^{max} = 5$, the fifth event of this case is zero padded, therefore, $d_{i,5,k} = 0$, $\forall k$. Based on this pre-processing, the trace contained in SW are encoded into a numeric matrix M with $|SW|$ rows (traces) and $J = L^{max} \times K$ columns (attributes).

In the *prefix-length* encoding, a different matrix is created for any given prefix length up to L^{max} containing the encoded prefixes. That is, given the max length L^{max} of traces in SW, a matrix M_p is generated for all prefix length p, with $p = 1, \ldots, L^{max}$. Note that, by construction, matrices M_p do not contain zero padded value. Given the current event $e_{i,j}$, the matrix M returned by the encoding step in Algorithm 1 (line 10), among all possible matrices M_p, is the one modified by the encoding of $e_{i,j}$. For instance, let us assume that the current event $e_{i,j}$ is the 5th one of a trace σ_i, then the matrix returned is M_5.

3.1. Anomaly score

This section discusses the calculation of anomaly scores (line 11 in Algorithm 1), which is based on the statistical leverage measure. The statistical *leverage* [26] is a measure indicating how far away each observation is scattered from other observations in a dataset. It has been used as a key support measure for developing different observation distance metrics, such as Cook's distance, the Welsch–Kuh distance, and the Welsch's distance. As shown in [1], the anomaly detection using statistical *leverage* shows remarkable generalisable performance when applied to event log anomaly detection.

Given a matrix M, with $M \in \mathbb{R}^{I \times J}$, of a dataset with I observations and J numerical attributes (or variables), the leverage of the observations in M are the diagonal elements of the projection matrix $H = M(M^T M)^{-1} M^T$. Specifically, the leverage of the ith observation in M is the diagonal element $h_{i,i} \in H$, which is comprised by definition between 0 and 1. The higher its leverage, the more likely an observation to be an anomaly. In this paper, the matrix M that is considered to calculate H is generated as output of the encoding phase.

Based on these premises, we can now define a first leverage-based anomaly score $\hat{l}(\sigma_i)$ by extracting the diagonal elements of $H = M \cdot (M^T \cdot M)^{-1} \cdot M^T$, that is:

$$\hat{l}(\sigma_i) = h_{i,i}$$

The value of this leverage measure is biased by the presence of zero-padded values in the input matrix M. Normally, in fact, zero-padded values should be treated as `null` values by any statistical method and therefore not considered in the analysis. However, this is not the case when calculating $\hat{l}(\sigma_i)$. The presence of zero-padded values, as shown in Fig. 2, creates a *seesaw* effect that increases the leverage of longer traces and decrease the one of shorter traces. Shorter traces, in fact, are more likely to be considered similar to each other, and therefore not anomalous, because they are encoded into a higher number of zero-padded values. Moreover, the zero-padding artificially assigns the same length to all traces, i.e., no matter their length, all traces are encoded into the same number of features.

In the context of this paper, this seesaw effect on leverage values is critical for the *trace-level* encoding, which can introduce a number of zero-padded values as discussed above. Note that the number of zero-padded values introduced depends on the distribution of the length of the traces currently in the sliding window SW (the number of zero-padded values is lower if all traces have similar length).

In order to counter this issue, we introduce a weighting factor w_i as a function of the trace length to increase/decrease the leverage of shorter/longer traces σ_i. This weighting factor is calculated by first normalising the trace length N_i in the range [0, 1]. This is done by applying the Z transformation to normalise N_i to the average $mean[N_i]$, followed by the application of a sigmoid function. The sigmoid-based normalisation is generally used to improve the fit accuracy and decrease the computational complexity of the fitting model [44]:

$$sig(Z_i) = \frac{1}{1 + e^{-Z_i}}, \text{ with } Z_i = \frac{\{N_i - mean[N]\}}{stdev[N]} \quad (1)$$

The weighting factor w_i is then defined as:

$$w_i = \left[1 - sig(Z_i)\right]^{c(N^{max})} \quad (2)$$

The power coefficient $c(N^{max})$ is required to adjust the strength of the weighting factor for different event logs. Intuitively, if all traces in a log have similar length, then this adjustment factor should be low, approaching 1; if trace length variance is very high, then the adjustment should be higher.

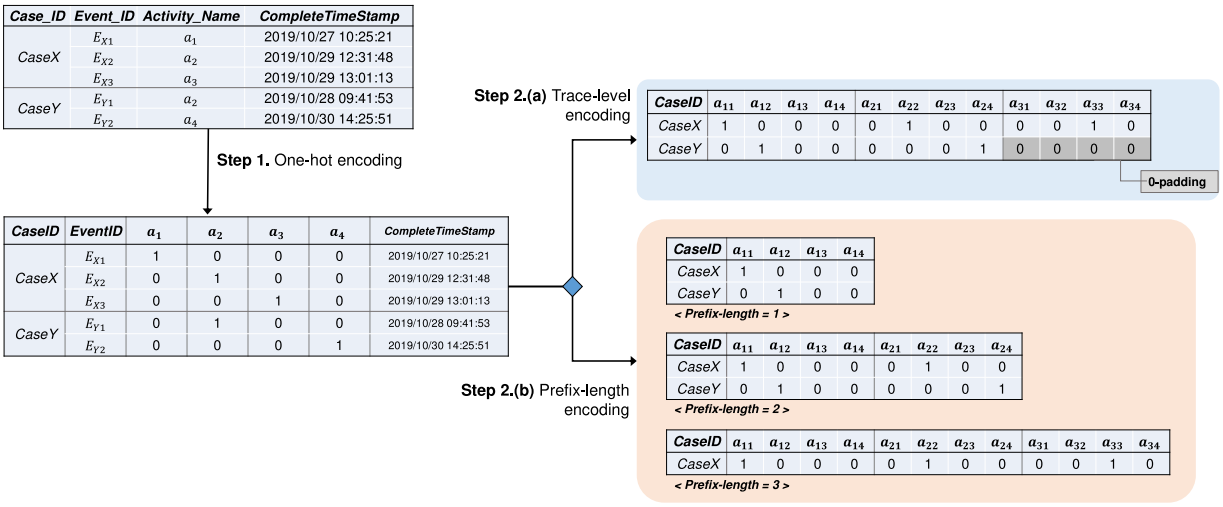
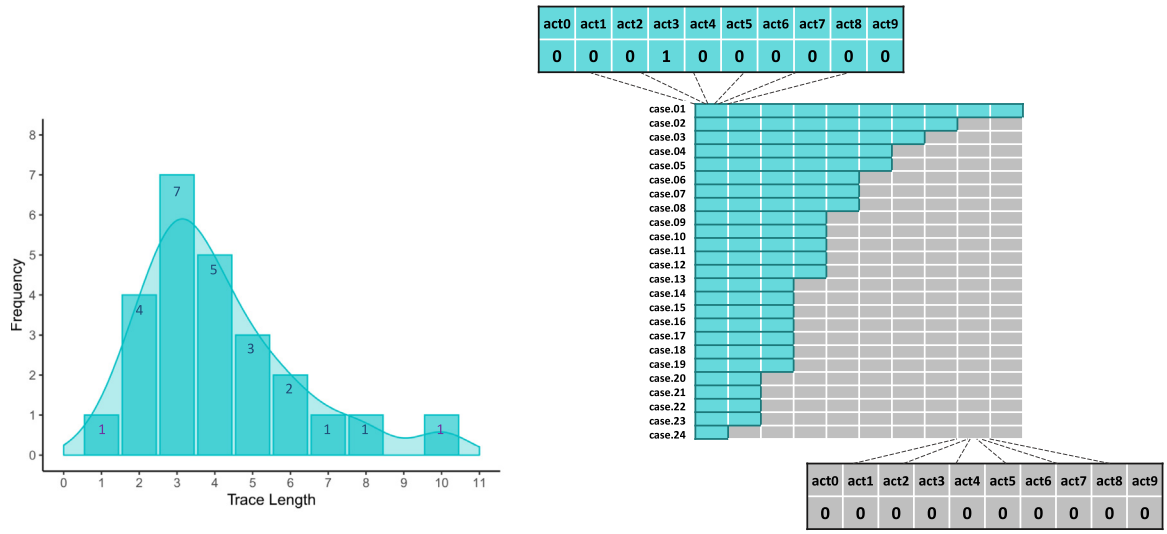


Fig. 1. Encoding of traces for anomaly detection: example.



(a) Histogram for trace length (b) Preprocessed matrix of log

Fig. 2. Seesaw effect on leverage values of zero-padding.
Source: From [1].

To define an appropriate value of the power coefficient $c(N^{max})$, a relation between N^{max} and the anomaly detection performance bias should be first found. However, this relation can only be estimated and not optimised because trace length has no upper bound, which would lead to a non-finite state optimisation problem. Therefore, we have estimated the value of $c(N^{max})$ by fitting a non-linear regression using 6 real-life event logs.[1] To model a non-linear regression function, we use the values of $c(N^{max})$ that achieve the highest F1-score in anomaly detection using the 6 different logs in offline settings, i.e., considering all the traces in an event log at the same time in the observation matrix M. Under significance level 0.01, the non-linear Eq. (3) has been fitted with two coefficient parameters a and b as in Table 1. For validation, we have tested the fitted model on a *validation* set of other 3 logs publicly available (BPIC2014, Sepsis, and Helpdesk[2]). As seen in Fig. 3, the performance achieved by the weighted leverage is always close to the optimal performance that could be achieved and, most importantly, always considerably higher than the one achieved by the non-weighted leverage.

Table 1
Result of fitted non-linear regression model: $f(x) = a + x^b$.

Parameter	Estimate	Standard Error	t value	p-value
a	−2.2822	0.3533	−6.46	0.0030
b	0.3422	0.0191	17.96	0.0001

$$c(N^{max}) = \begin{cases} -2.2822 + (N^{max})^{0.3422} & \text{if } N^{max} > \frac{2.2822}{0.3422} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

[1] These event logs belong to the ones made available by the Business Process Intelligence Challenge in 2012, 2013, and 2017. Note that, to minimise attribution bias, only 1 one of these logs (2013) is used also in the evaluation of the approach proposed in this paper.

[2] Note that only the Helpdesk event log is used in the evaluation of the approach proposed in this paper.

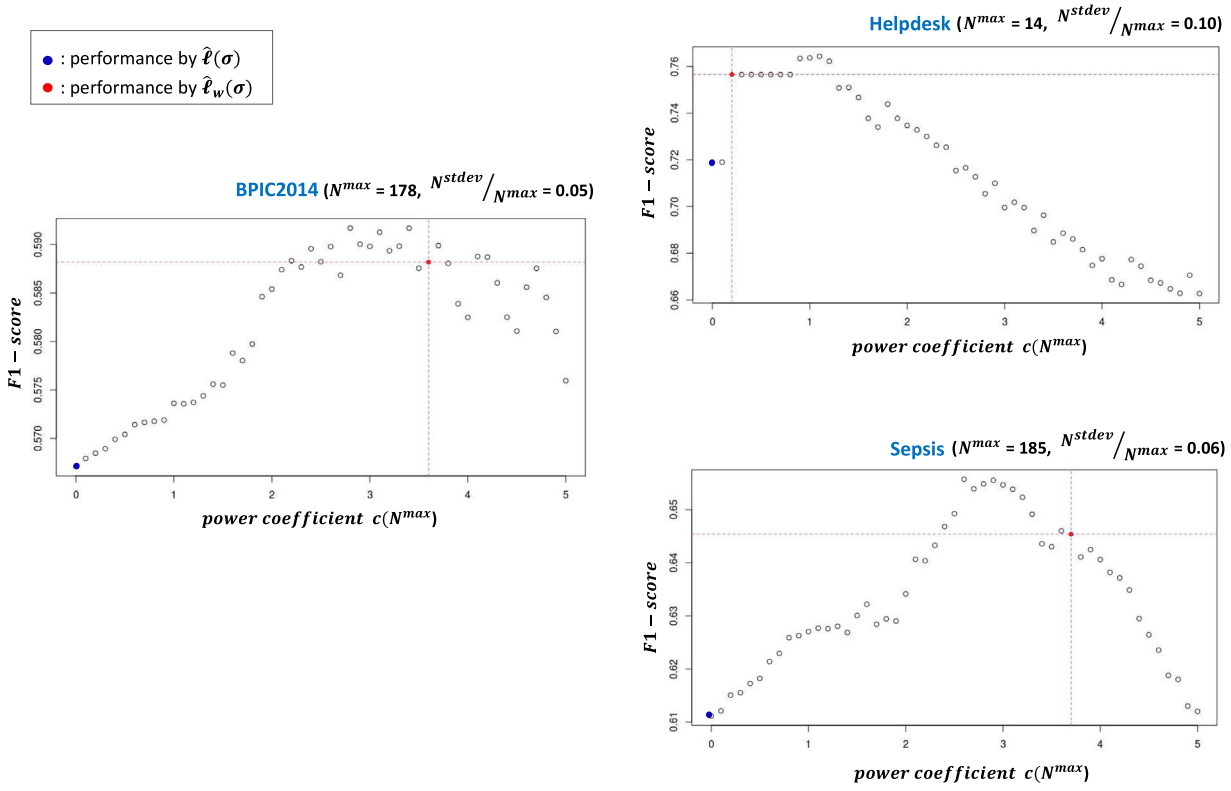


Fig. 3. Validation of the fitted weighting factor using heterogeneous event logs.

In the end, using the estimated power coefficient $c(N^{max})$, we define the weighted *leverage*-based anomaly score as:

$$\hat{l}_w(\sigma_i) = w_i \cdot \hat{l}(\sigma_i). \tag{4}$$

The two defined measures of anomaly score, i.e., $\hat{l}(\sigma_i)$ and $\hat{l}_w(\sigma_i)$, are applied with prefix-length and trace-level encoding, respectively.

The objective of anomaly detection is ultimately to determine whether traces are anomalous or not. Therefore, the problem of anomaly detection can be reduced to a binary classification problem. Based on the anomaly score values, a decision has to be made whether the trace σ_i is anomalous or not. This is normally done by setting the value of an anomaly detection threshold T for $\hat{l}_w(\sigma_i)$, such that a trace is anomalous if $\hat{l}_w(\sigma_i) > T$. In this work, we consider a constant threshold value $T = 0.2$, which is a good choice based on our experience with experiments in online and offline settings. Note that the availability of anomaly score allows to assess the *severity* of an anomaly, giving additional information in respect of solving the anomaly detection problem with a traditional classification technique, which would only output a class for each trace. For example, as often implemented by anomaly detection approaches [34], different threshold values can be considered for different levels of severity of an anomaly: a threshold $T = 0.1$ can be additionally used to define a trace as *suspicious*, which would allow to define three labels for traces, i.e., *normal*, *suspicious*, *anomalous*, turning the classification problem into a multi-class one.

4. Evaluation

We have conducted first a general evaluation of the proposed approach using event logs considered by other approaches to online anomaly detection in the literature (Section 4.1). Then, we have evaluated the performance of the proposed approach with event logs characterised by different types of concept drift (Section 4.2) The experiments discussed in this section are implemented in R on an Intel Xeon Linux machine with 128GB of memory. The source code and datasets to reproduce the experiments are publicly available at https://github.com/paai-lab/Online-Anomaly-Detection-Extension-2021.

4.1. Evaluation with event logs commonly used in the literature and baseline models

We consider three groups of event logs: the five artificial logs (ART-LOGS-1) used by [5], two artificial logs obtained through simulation (ART-LOGS-2), and three real-life log publicly available (REAL-LOGS). The ART-LOGS-1 logs are generated by simulating 5 process models (Small, Medium, Large, Huge, and Wide used in [5]) using the PLG2 tool [45]. Although these logs have been considered by previous work in anomaly detection [27], they present a critical issue regarding the way in which timestamps are generated: the first event of all cases in these logs have the same timestamp. Obviously this situation occurs only in special cases in the real world and, therefore, the results obtained on these logs should not be generalised to the situation in which cases start along a certain temporal time frame.

The ART-LOGS-2 logs are generated by simulating two BPMN processes loosely modelled based on real world processes using the BIMP[3] process simulator. The process Credit refers to the processing of credit card applications at a bank, whereas the Pub log refers to the process of preparing and serving food at a restaurant. Each log contains events belonging to 5000 instances. The inter-arrival time between the start of new instances is drawn from a negative exponential distribution with mean of 6 min. The logs, including the BPMN models annotated with the simulation parameters, are available at the repository referenced earlier.

[3] https://bimp.cs.ut.ee/simulator/.

For the REAL-LOGS group, we consider the *Helpdesk* event log containing events logged by a ticketing management system of the help desk of an Italian software company, the *Hospital Billing* event log containing events about the billing of medical services that have been obtained from the financial modules of the ERP system of a regional hospital, and the *BPIC 2013* event log representing an incident and problem management process from Volvo IT. These logs have been chosen because they differ along three characteristics, as reported by Tama et al. [46]: (i) the size of the sample for larger prefix-length, (ii) the event log variability, defined as the number of most frequent variants that cover at least 80% of the cases in the log, and (iii) the mean/median trace length. Table 2 compares these logs along these three dimensions qualitatively. Readers are referred to [46] for the quantitative comparison. The diversity of the logs increases the reliability of the experiments, preventing the overfitting problem.

Evaluating an unsupervised approach to anomaly detection like the one that we propose requires event logs with labelled traces (normal v. anomalous), which are generally unavailable in practice. Therefore, a common practice in this research field is to inject anomalies using different types of anomaly patterns in event logs and to create labels during the anomaly injection process [1,4–7,13]. We consider the 6 anomaly patterns *Skip*, *Insert*, *Replace*, *Early*, *Late*, and *Rework* as defined in [1,5,13]: in *Skip*, a sequence of events is skipped in some cases; in *Insert*, one or more events are generated in random positions within existing traces; in *Replace*, the activity label in one event is changed to another activity label in some cases; in *Early*/*Late*, timestamps of events are manipulated such that a sequence of events is moved earlier/later in a trace; in *Rework*, a sequence of events is repeated after its occurrence. Anomalies are randomly injected in an event log until 10% of the traces in the log have become anomalous. The parameters used to inject anomalies for each pattern are the same used by the authors in their previous publication [1].

Furthermore, when generating anomalies we discard those cases in which anomaly injection leads to a normal trace. These anomalies obviously cannot be discovered and therefore have been excluded to preserve the reliability of the results obtained. The descriptive statistics of the event logs (after injecting anomalies) are reported in Table 3.

As far as performance measures are concerned, we consider the typical confusion matrix-based metrics (accuracy, precision, recall, and F1-score) for evaluating a classification model for anomaly detection and the average run time (in seconds) to process one new event in the stream. Note that performance measures are evaluated after the GP condition is met.

Regarding the experiment settings, we set the GP to 100 traces, that is, the online anomaly detection starts when the sliding window *SW* contains 100 cases represented by at least 2 events. We consider a variable range of values for W (100, 500, and 1000 traces in the trace-based sliding window) and a fixed maximum number of events $W_e = 15000$. Note that a larger value of GP and W is likely to lead to better and more stable performance, while also implying a higher computational cost (run time).

Regarding baseline models, we consider two alternative approaches: One-Class Support Vector Machine (OC-SVM) and Isolation Forest (iForest). Each of these is used to replace the anomaly scores calculation (line 13) of the procedure depicted in Algorithm 1. OC-SVM is a popular unsupervised binary classifier used for anomaly detection in many scenarios [5], whereas iForest is a state-of-art unsupervised ensemble-based method, which can be used to detect anomalies based on its heuristic degree of anomaly, i.e., the path length measure, in an unsupervised environment [47]. We have implemented OC-SVM using the R package 'e1071' with the default parameter value $nu = 0.5$ to be consistent with the baseline research [5] and iForest using the R package 'IsolationForest' with the default parameters $ntree = 100$ and $nmin = 5$. Note that higher values of *ntree* and *nmin* in iForest result in better performance at the price of larger memory cost. iForest considers an ad-hoc function for setting a threshold on the path length measure over isolation trees (iTrees) for the class allocation, which is not easy to interpret [48], and does not provide the opportunity to use a fixed value. Therefore, we consider a variable threshold $T = mean_{\sigma_i \subseteq E}[Score(\sigma_i)] + \alpha \cdot stdev_{\sigma_i \subseteq E}[Score(\sigma_i)]$, with $\alpha = 1$, i.e., a trace is anomalous if its path length exceeds the mean value of the path lengths of traces in M of at least one standard deviation. This same variable threshold is used in offline trace anomaly detection in [1] and for timestamp anomaly detection in [13]. OC-SVM directly yields a class label for each trace in M – or M_p when using the prefix-length encoding (that is, there is no need to set a classification probability threshold when using OC-SVM).

Finally, we also consider as baselines the corresponding offline implementation of OC-SVM and iForest and also the offline leverage-based anomaly detection framework previously published by the authors [1]. As far as hyperparameter values are concerned, the same values described in the previous paragraph for the online settings are considered. For the leverage offline approach, we consider the weighted version of the anomaly score and the same fixed value $T = 0.2$ of the threshold for anomaly detection used in the online settings.

Table 4 shows the accuracy achieved by the two different encoding options for $GP = 100$ and $W = 1000$ on the 10 event logs considered in this evaluation. The results in Table 4 highlight the structural advantages/disadvantages of each encoding option. The *trace-level* encoding is less accurate than the *prefix-length* one for the ART-LOGS2 and REAL-LOGS. Regarding the ART-LOGS1, the slightly higher accuracy achieved by the trace-level encoding is explained by the nature of timestamps in ART-LOGS-1. As already mentioned, all cases in these logs start at the same time and, therefore, it is unlikely that, at a given time, the cases considered for anomaly detection are characterised by widely different prefix lengths. In other words, during the experiment all traces for the logs ART-LOGS-1 are likely to have a similar length at any time, which reduces the bias introduced by the zero-padding required by the *trace-level* encoding. Overall, these results confirm that the *trace-level* encoding introduces a bias in the calculation of anomaly scores, which can be prevented by considering batches of prefixes at different lengths. As far as run time is concerned, the trace-level level encoding fares worse than the prefix-length one (at least 50% worse in many cases). This is due to the fact that, while in the trace-encoding the size of the matrix H to invert is always proportional to the number of traces in the log, in the prefix-length encoding this matrix is normally smaller (and proportional to the prefix length determined by the current event).

The performance of the *trace-level* encoding for the Pub event log is exceptionally low. This is due to the fact that the weighting factor in $\hat{l}_w(\sigma_i)$ can sharply change the distribution of anomaly scores so that the constant threshold $T = 0.2$ is not appropriate to classify anomalies. While this happens only for the Pub event log in our experiment, this points to the need to analyse more in depth the interaction between the weighting of anomaly scores and the anomaly threshold. A possible solution for this problem is to use a variable anomaly threshold, as introduced by the authors in the offline anomaly detection scenario [1]. This however generates an additional memory cost as well as additional computational time and it is therefore not advisable when dealing with event streams.

To prove the finite memory usage of the proposed approach, Fig. 4 shows the memory usage during the experiment using the

Table 2
Qualitative comparison of event logs REAL-LOGS.

Event logs	Number of samples for larger prefix-length	Event log variability	Mean/Median trace length
Helpdesk	Low	Low	Low
BPIC 2013	High	High	High
Hospital billing	High	Low	Low

Table 3
Descriptive statistics of the event logs.

Group	Event logs	# of activities	# of cases	# of events	# of variants	# of anomalous cases
ART-LOGS-1	Small	20	5K	44.6K	369	467
	Medium	32	5K	29.5K	385	483
	Large	42	12.5K	137.9K	1,188	1,298
	Huge	54	12.5K	100.1K	1,068	1,300
	Wide	34	12.5K	74.9K	838	1304
ART-LOGS-2	Credit	12	5K	44.7K	409	470
	Pub	12	5K	59.4K	610	483
REAL-LOGS	Helpdesk	9	3.8K	13.9K	250	264
	Hospital Billing	16	10K	65.8K	2590	647
	BPIC 2013	25	7.5K	50.2K	651	735

Table 4
Accuracy and average computational time (seconds) of leverage-based anomaly detection with different encoding options. For each log, the best accuracy and shorter run time are highlighted in bold and italic, respectively.

Group	Event logs	*trace-level*		*prefix-length*	
		Accuracy	Time	Accuracy	Time
ART-LOGS-1	Small	**0.885**	0.99	0.880	*0.46*
	Medium	**0.873**	1.38	0.822	*0.66*
	Large	**0.861**	1.82	0.818	*0.85*
	Huge	**0.766**	2.71	0.733	*1.03*
	Wide	**0.731**	1.73	0.689	*0.94*
ART-LOGS-2	Credit	0.818	1.64	**0.911**	*0.86*
	Pub	0.160	2.06	**0.941**	*0.91*
REAL-LOGS	Helpdesk	0.879	1.15	**0.923**	*0.57*
	BPIC 2013	0.771	1.65	**0.778**	*0.46*
	Hospitalbilling	0.842	5.65	**0.892**	*1.92*

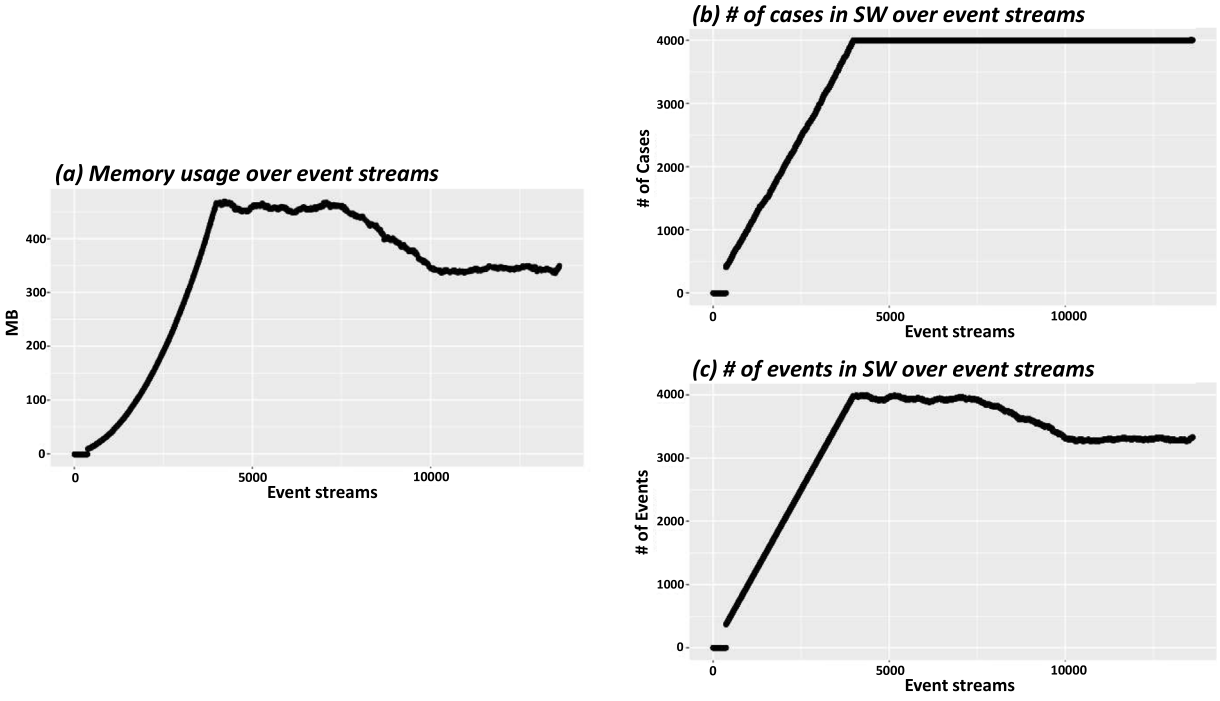


Fig. 4. Finite memory usage of the proposed online setting (Helpdesk event log using leverage-based anomaly detection with prefix-length batch encoding, $GP = 100$ and $W = 1000$).

Table 5
Accuracy and run time (seconds) comparison of leverage-based anomaly detection with baselines; the offline settings are also included as a reference. For each experiment (a row in the table), the highest accuracy and lowest runtime are highlighted in bold and italic, respectively.

Group	Event logs	Window size	OC-SVM		iForest		leverage	
			Accuracy	Time	Accuracy	Time	Accuracy	Time
ART-LOGS-1	Small	100	0.533	0.20	**0.877**	0.74	0.778	*0.13*
		500	0.632	0.29	**0.951**	0.82	0.901	*0.28*
		1000	0.583	0.58	**0.931**	1.05	0.880	*0.47*
		(5000; offline)	(0.537)	–	(0.950)	–	(0.928)	–
	Medium	100	0.291	0.31	**0.707**	1.00	0.707	*0.24*
		500	0.486	0.56	**0.892**	1.22	0.822	*0.44*
		1000	0.502	0.90	**0.882**	1.72	0.822	*0.66*
		(5000; offline)	(0.646)	–	(0.950)	–	(0.936)	–
	Large	100	0.336	*0.35*	0.555	0.82	**0.765**	0.36
		500	0.449	*0.56*	**0.862**	1.06	0.824	0.59
		1000	0.435	*0.83*	**0.885**	1.33	0.818	0.85
		(12500; offline)	(0.535)	–	(0.938)	–	(0.917)	–
	Huge	100	0.336	0.52	0.480	0.78	**0.846**	*0.48*
		500	0.463	*0.70*	**0.874**	1.02	0.737	0.73
		1000	0.412	*1.03*	**0.887**	1.52	0.733	1.03
		(12500; offline)	(0.465)	–	(0.929)	–	(0.923)	–
	Wide	100	0.431	0.64	0.343	0.84	**0.651**	*0.56*
		500	0.396	0.71	**0.774**	1.26	0.649	*0.67*
		1000	0.436	1.00	**0.816**	1.53	0.689	*0.94*
		(12500; offline)	(0.519)	–	(0.936)	–	(0.908)	–
ART-LOGS-2	Credit	100	0.776	0.35	**0.914**	0.58	0.862	*0.22*
		500	0.761	0.57	0.893	1.22	**0.907**	*0.48*
		1000	0.718	1.02	0.879	1.60	**0.911**	*0.86*
		(5000; offline)	(0.713)	–	(0.949)	–	(0.917)	–
	Pub	100	0.681	0.13	**0.841**	0.71	0.768	*0.08*
		500	0.668	0.50	0.865	1.17	**0.915**	*0.39*
		1000	0.659	1.17	0.875	1.79	**0.941**	*0.91*
		(5000; offline)	(0.603)	–	(0.885)	–	(0.915)	–
REAL-LOGS	Helpdesk	100	0.618	0.07	**0.859**	0.69	0.844	*0.03*
		500	0.613	0.25	0.862	0.98	**0.908**	*0.23*
		1000	0.620	0.93	0.871	1.53	**0.923**	*0.57*
		(3804; offline)	(0.431)	–	(0.865)	–	(0.909)	–
	BPIC 2013	100	0.587	0.10	**0.816**	0.48	0.694	*0.06*
		500	0.577	0.27	**0.807**	0.64	0.738	*0.24*
		1000	0.563	0.54	**0.797**	1.22	0.778	*0.46*
		(7554; offline)	(0.566)	–	(0.827)	–	(0.845)	–
	Hospital billing	100	0.697	1.31	**0.893**	1.77	0.892	*1.02*
		500	0.651	1.71	**0.918**	2.16	0.901	*1.38*
		1000	0.632	2.16	**0.913**	2.76	0.892	*1.92*
		(10000; offline)	(0.629)	–	(0.910)	–	(0.911)	–
		Average	0.551	0.67	**0.824**	1.20	0.816	*0.58*

Helpdesk event log. Similar trends in memory usage are registered for all the other event logs. The amount of memory used remains below 1GB at all times and the total number of events remains limited event after the sliding window *SW* reaches the maximum number of allowed traces.

Next, we compare the proposed approach with the baseline methods (online OC-SVM and iForest, and the 3 offline methods: OC-SVM, iForest, leverage). For the proposed approach, we consider the *prefix-length* encoding, which generally outscores the *trace-level* encoding in the results shown in Table 4, both in accuracy and run time.

As far the quantitative comparison is concerned, we run experiments with $GP = 100$ and three different sizes of the sliding window, $W \in [100, 500, 1000]$ for all 10 event logs.

Table 5 compares the accuracy and run time obtained for the different methods, whereas Fig. 5 compares the average and standard deviation of the performance obtained for different values of the maximum size W of the sliding window *SW*. The accuracy of OC-SVM is on average the lowest, consistently across all event logs. This may be explained by a sub-optimal value of the parameter *nu*, which serves as an estimated anomaly ratio in the input dataset. Ideally, OC-SVM performs best if the value of this parameter is known a priori [1]. However, since we assume unsupervised settings where the anomaly ratio is not known, we use OC-SVM with the default value $nu = 0.5$. The iForest method shows on average the best accuracy, but also it incurs the highest average run time. Moreover (see Fig. 5), while the proposed approach shows the same limited standard deviation of accuracy for all window sizes, for the small window size ($W = 100$), iForest shows a very high standard deviation.

As expected, the offline baselines achieve better performance than any online approach. This is due to the fact that an offline method can detect anomaly by considering all the traces at the same time. However, it should be noted that the performance of all online methods normally increases with the size of the window W, which shows that the accuracy of an online anomaly detection framework is likely to converge to the one of the corresponding offline one as the number of recent traces stored in the memory for the analysis increases.

Finally, the Appendix shows the results obtained in this experiment for other performance measures (precision, recall, and F1-score) with the proposed framework and the online baselines. These additional results show the same basic trend already discussed above, with the OC-SVM approach normally performing worse that the other two online approaches.

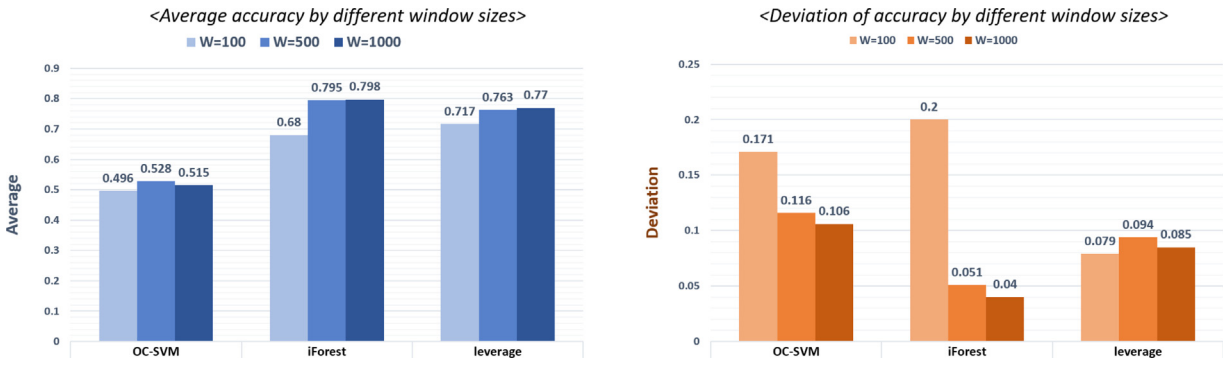
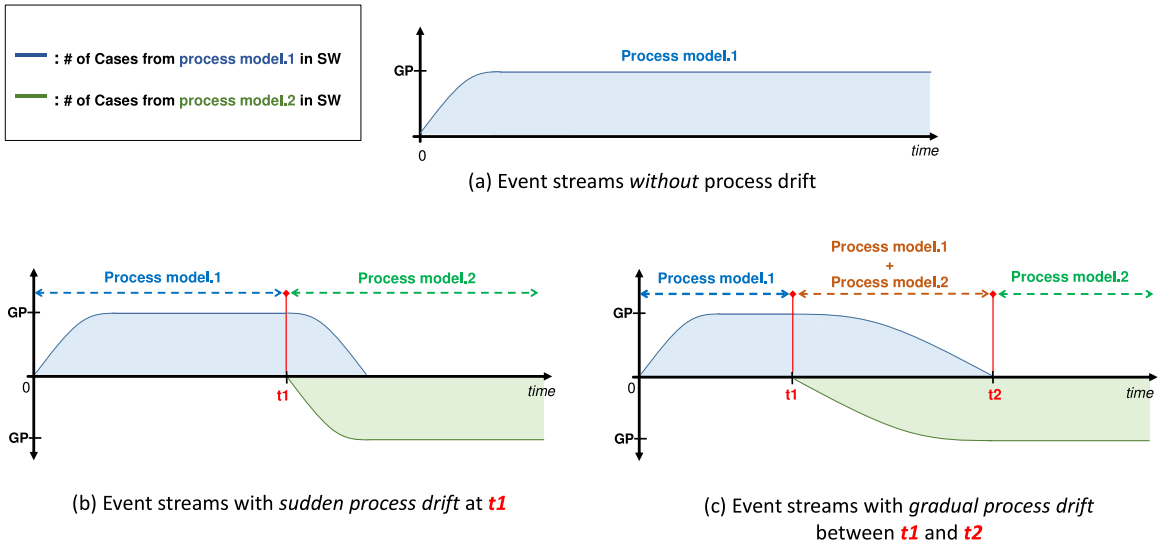


Fig. 5. Mean (left graph) and standard deviation (right graph) of performance for different 10 event logs by window sizes.



(a) Event streams *without* process drift

(b) Event streams with *sudden process drift* at ***t1***

(c) Event streams with *gradual process drift* between ***t1*** and ***t2***

Fig. 6. Qualitative representation of the number of cases (with and without process drift) in the sliding window *SW* as a function of time.

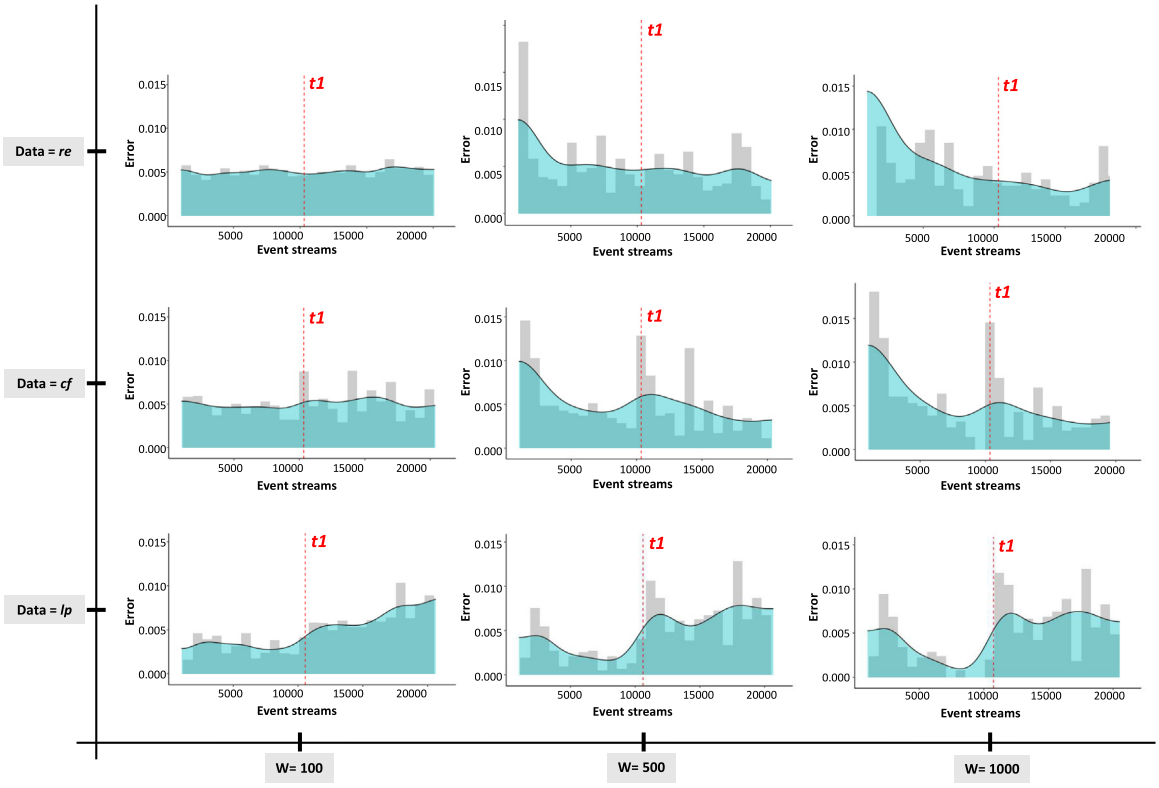
Table 6
Descriptive statistics of event logs with concept drift.

Event logs	Type of concept drift	# of activities	# of cases	# of events	# of cases with concept drift	# of anomalous cases
re	Sudden	19	2,000	20,779	1,000	199
	Gradual	19	2,000	20,779	1,000	199
cf	Sudden	19	2,000	21,250	1,000	182
	Gradual	19	2,000	21,165	1,000	183
lp	Sudden	19	2,000	21,261	1,000	187
	Gradual	19	2,000	21,261	1,000	187

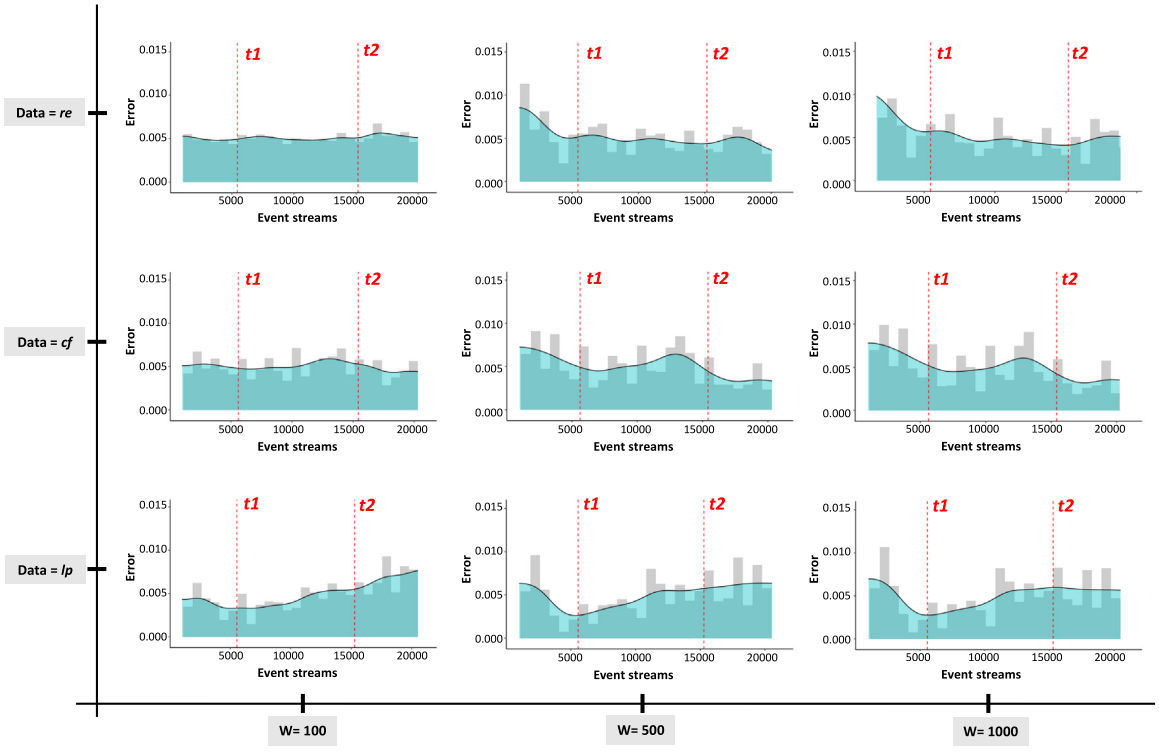
Generally, the proposed approach shows high performance, comparable with the one of iForest, with a lower run time. Moreover, the accuracy achieved by the proposed approach is more reliable, as shown by the lower standard deviation of the results shown in Fig. 5. For all three online models, the accuracy and computational time tend to increase with a larger window size. The proposed model is robust across different event logs and different window sizes. Such a robustness is typical of statistical models used for anomaly detection [49]. Conversely, the iForest method may yield a better performance for larger window size at the cost of higher computational run time.

4.2. Evaluation using event logs with concept drift

In this section, we evaluate how the proposed approach performs with event streams characterised by different types of concept drift. Also here we consider only the *prefix-length* encoding. As reported by Maaradji et al. [51] in the context of process concept drift detection, when concept drift occurs in the middle of an event stream, a sliding window approach should show a trade-off between the performance and the adaptation delay, i.e., the speed at which a model can adapt to the new process conditions after concept drift. We consider the same event logs used by Maaradji et al. [51], that is, 6 event logs resulting from combining two concept drift types (sudden v. gradual) and 3 concept drift patterns: (i) add/remove a process fragment (*re*), (ii) make two fragments conditional/sequential (*cf*), and (iii) make a fragment loopable/non-loopable (*lp*). In the sudden concept drift, see Fig. 6(b), the process change is applied from a specific time instant $t1$ which coincides with the middle of the event stream. In the gradual concept drift (see Fig. 6(c)) traces are generated from the old/new process at different frequency within a given timeframe $[t1, t2]$. Specifically, $t1$ and $t2$ are set at the time



(a) Prediction error in *sudden drift*



(b) Prediction error in *gradual drift*

Fig. 7. Density histogram of prediction error under process drift (the smooth line represents the 1-d kernel density estimation of the histogram [50]).

instant of the termination of the first 535 cases (26% of the cases in the log) and of the first 1465 cases (73% of the cases in the log), respectively. Between $t1$ and $t2$, cases generated according to the new process models are interleaved with cases generated by the old process at an increasing frequency, until from $t2$ onwards all cases are generated using the new process model. Similarly to the experiment discussed in Section 4.1, the event logs with concept drift have been injected randomly with the 6 anomaly patterns

Table 7
Performance for adaptability test.

Event logs	Type of process drift	Window size	ACC_{all}	ACC_{new}
re	Sudden	100	0.786	0.803
		500	0.947	0.944
		1000	0.957	0.947
	Gradual	100	0.784	0.787
		500	0.946	0.932
		1000	0.956	0.932
cf	Sudden	100	0.900	0.900
		500	0.949	0.946
		1000	0.954	0.948
	Gradual	100	0.901	0.917
		500	0.949	0.947
		1000	0.954	0.947
lp	Sudden	100	0.819	0.828
		500	0.910	0.907
		1000	0.921	0.911
	Gradual	100	0.841	0.778
		500	0.921	0.893
		1000	0.931	0.893

Skip, Insert, Replace, Early, Late, and Rework until 10% of the traces in the log have become anomalous, using the same parameters. The descriptive statistics of the logs after injecting both process drift and anomaly patterns are reported in Table 6.

The objective of this evaluation is twofold: first, we aim at demonstrating that, after the concept drift occurs, the average anomaly detection accuracy on the traces generated by the old and the new process model, respectively, remain comparable; second, we aim at studying qualitatively how the proposed method adapts to the occurrence of concept drift, i.e., by studying the shape of the error ratio around the time instant(s) in the event stream when the concept drift occurs.

Regarding the first objective, in the case of sudden drift, we compare the average accuracy ACC_{new} on events belonging to the traces generated by the new process model after the concept drift (i.e, the *green* traces in Fig. 6(b)) with the average accuracy ACC_{all} of all events received after the concept drift occurs at $t1$. For the gradual drift, we compare the average accuracy ACC_{new} on events belonging to the traces generated by the new process model after the concept drift has occurred fully (i.e., the green traces after $t2$ in Fig. 6(c)) with the average accuracy ACC_{all} of all the events generated after the concept drift has started at $t1$.

Table 7 compares the values of ACC_{new} and ACC_{all} for different event logs, sliding window size W, and concept drift types. It appears clearly that the difference between ACC_{new} and ACC_{all} is always limited. These results show that, generally, the proposed approach is robust to concept drift, that is, its anomaly detection performance remains comparable after concept drift on events generated by both the old and the new process model.

Regarding the second objective, Fig. 7 shows a histogram of the anomaly detection error for different event logs, window size W, and concept drift type. Note that, for each histogram, we have highlighted the time instant(s) at which concept drift occurs and we have overlaid the fitting of the 1-dimensional kernel density estimation of the error value. An individual bar in a histogram represents the proportion (ratio) of classification errors on a given number of events received in the corresponding interval (note that event streams in the figure are split into 28 consecutive intervals).

Next, we discuss the results of Fig. 7 separately for each of the 3 concept drift pattern.

The add/remove fragment concept drift (*re*) does not significantly affect the performance of the anomaly detection framework, both with sudden and gradual drift. In both cases, in fact, there is no significant increase of the error proportion after concept drift occurs. This can be explained by considering that this type of concept drift has a limited impact on the anomaly patterns injected in an event log. Specifically, out of the 6 patterns injected, it only affects the *skip* and *replace* patterns in the case of removing or adding an activity, respectively.

For the concept drift *cf* (make two fragments conditional/sequential), the sudden concept drift shows a typically expected pattern: the anomaly detection error spikes when the concept drift occurs, and then decreases to the level seen before the concept drift more or less rapidly depending on the size of the sliding window. The impact of this type of concept drift on traces is, in fact, more salient than in the previous case. After this type of concept drift, in fact, normal traces may contain entire new sequences of activities that would have been considered anomalous before the concept drift. In the gradual drift, the rise in errors rate is diluted, which follows from the fact that new normal traces occur gradually.

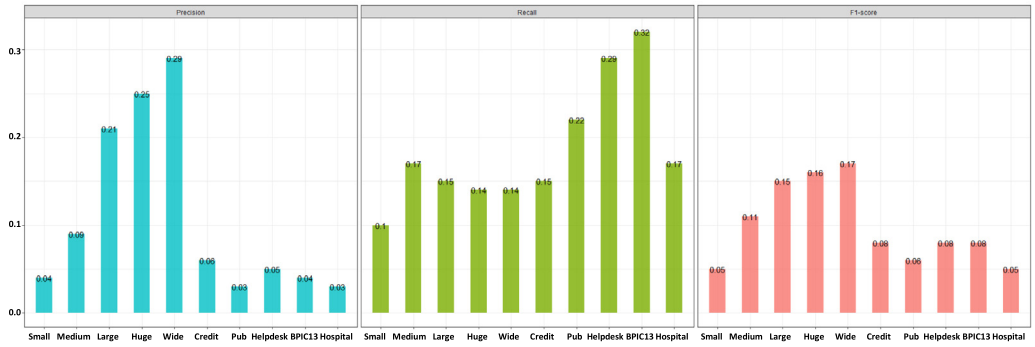
For concept drift *lp* (make fragment loopable/non-loopable), the sudden concept drift behaves similarly to the *c* case, with the exception that, after concept drift, the classification error proportion does not decline to the levels seen before concept drift, but it appears to stabilise at higher levels. This can be explained by considering that this type of concept drift (introducing loops in the process models) makes the anomaly detection more challenging. Therefore, after concept drift occurs, the performance stabilises to slightly worse levels, i.e., higher error ratio. A similar effect can be seen in the case of gradual drift, whereby, after time $t2$, the error levels the performance grow to higher levels.

5. Conclusions

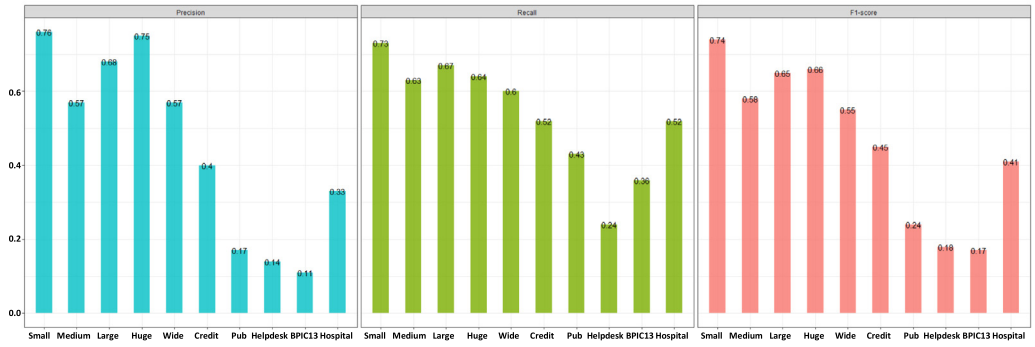
This paper has presented a novel online framework to detect trace level anomalies in business process event streams using statistical leverage. The proposed approach has been evaluated in respect of state-of-the-art classification-based algorithms and also using event logs characterised by different types of concept drift. The results have shown that the proposed approach is robust, i.e., it performs well on average with different types of event logs and different values of the design parameters, e.g., the upper limit of the sliding window, and faster than the other approaches considered, while maintaining an overall level of performance comparable to iForest, a state-of-the-art approach for online anomaly detection in unsupervised settings. As far as concept drift is concerned, we have shown that the proposed approach generally can deal effectively with it, highlighting also that the capability of adapting to concept drift may depend on the type of drift under consideration.

As most enterprise systems, such as network systems, bank transaction systems, or medical information systems, nowadays have the capability to log and make available process data in real time, anomaly detection in event streams is becoming increasingly relevant for practical applications. We argue, in particular, that anomaly detection can play a twofold role in this context: (i) help to isolate the data required to create a positive model of process behaviour, which can then be used to improve the quality of event logs gathered in the future and, therefore, the quality of the results stemming from their analysis, and (ii) identify in real-time cases of negative behaviour, such as fraudulent transactions in financial systems or wrong handling of patient pathways, which helps to improve online the process operations.

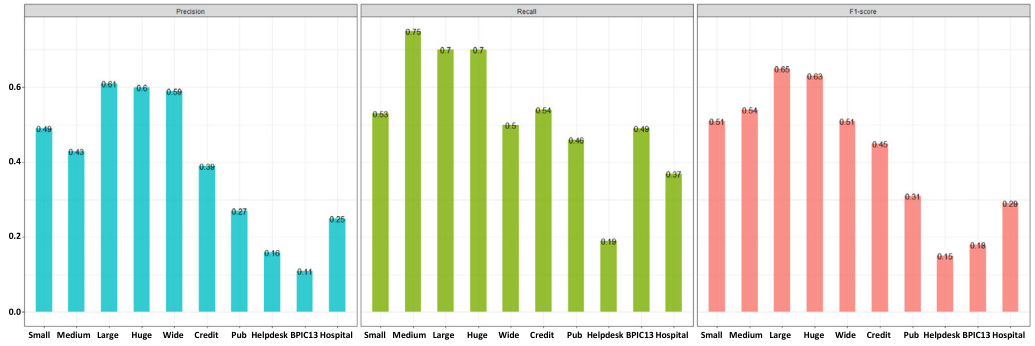
For future work, we first plan to extend the analysis, both in online and offline settings, to trace-level types of event log anomaly better resembling real-world anomalies commonly found in event logs [8], temporal anomalies, and other attribute-level event log anomalies. On of the challenges here would be to



(a) Average precision, recall, and F1 score of **OC-SVM**



(b) Average precision, recall, and F1 score of **iForest**



(c) Average precision, recall, and F1 score of **leverage**

Fig. A.8. Results of comparison with online baselines (Precision, Recall, and F1-score).

adapt the encoding of events to calculate effectively the anomaly scores. We are also working on the event log reconstruction phase, i.e., recognising the type of anomalies occurring in a trace and possibly reconstructing the correct sequence of events or attribute values. We also plan to compare the effectiveness of different types of event log anomaly detection techniques on different types of anomaly patterns, so as to understand whether there is a clear bias of existing methods towards particular types of anomalies. To improve the practical relevance of this line of research, we will also investigate with practitioners the potential benefits and drawbacks of deploying offline and online anomaly detection techniques in real-world process operation settings. Finally, we plan to study the impact of quality issues in the event stream on the anomaly detection process, assessing for instance the impact of events that are delivered in a different order from the one in which they are captured during the process execution, or events that are dropped from a stream.

Declaration of competing interest

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

Acknowledgements

This research was partly funded by the 0000 Project Fund (Project Number 1.210079.01) of UNIST (Ulsan National Institute of Science & Technology), South Korea.

Appendix. Additional results

See Fig. A.8.

References

[1] J. Ko, M. Comuzzi, Detecting anomalies in business process event logs using statistical leverage, Inform. Sci. 549 (2020) 53–67.

[2] K. Böhmer, S. Rinderle-Ma, Anomaly detection in business process runtime behavior–challenges and limitations, 2017, arXiv preprint arXiv:1705.06659.

[3] L. Genga, M. Alizadeh, D. Potena, C. Diamantini, N. Zannone, Discovering anomalous frequent patterns from partially ordered event logs, J. Intell. Inf. Syst. 51 (2) (2018) 257–300.

[4] K. Böhmer, S. Rinderle-Ma, Multi instance anomaly detection in business process executions, in: International Conference on Business Process Management, Springer, 2017, pp. 77–93.

[5] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Binet: Multi-perspective business process anomaly classification, Inf. Syst. (2019) 101458.

[6] F. Bezerra, J. Wainer, Algorithms for anomaly detection of traces in logs of process aware information systems, Inf. Syst. 38 (1) (2013) 33–44.

[7] K. Böhmer, S. Rinderle-Ma, Multi-perspective anomaly detection in business process execution events, in: OTM Confederated International Conferences on the Move To Meaningful Internet Systems, Springer, 2016, pp. 80–98.

[8] J. Ko, J. Lee, M. Comuzzi, Air-BAGEL: An interactive root cause-based anomaly generator for event logs, in: Proceedings of International Conference on Process Mining (ICPM), Demo Track, 2020.

[9] S.J. van Zelst, A. Bolt, M. Hassani, B.F. van Dongen, W.M. van der Aalst, Online conformance checking: relating event streams to process models using prefix-alignments, Int. J. Data Sci. Anal. 8 (3) (2019) 269–284.

[10] L. Ghionna, G. Greco, A. Guzzo, L. Pontieri, Outlier detection techniques for process mining applications, in: International Symposium on Methodologies for Intelligent Systems, Springer, 2008, pp. 150–159.

[11] S.J. Leemans, D. Fahland, W.M. van der Aalst, Discovering block-structured process models from event logs containing infrequent behaviour, in: International Conference on Business Process Management, Springer, 2013, pp. 66–78.

[12] X. Lu, D. Fahland, F.J. van den Biggelaar, W.M. van der Aalst, Detecting deviating behaviors without models, in: International Conference on Business Process Management, Springer, 2016, pp. 126–139.

[13] H.T.C. Nguyen, S. Lee, J. Kim, J. Ko, M. Comuzzi, Autoencoders for improving quality of process event logs, Expert Syst. Appl. 131 (2019) 132–147.

[14] S.J. Leemans, D. Fahland, W.M. Van der Aalst, Scalable process discovery and conformance checking, Softw. Syst. Model. 17 (2) (2018) 599–631.

[15] A. Sureka, Kernel based sequential data anomaly detection in business process event logs, CoRR (2015) 1–4, arXiv:1507.01168.

[16] S.J. van Zelst, B.F. van Dongen, W.M. van der Aalst, Event stream-based process discovery using abstract representations, Knowl. Inf. Syst. 54 (2) (2018) 407–435.

[17] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, F.M. Maggi, Discovering process maps from event streams, in: Proceedings of the 2018 International Conference on Software and System Process, 2018, pp. 86–95.

[18] A. Burattin, J. Carmona, A framework for online conformance checking, in: International Conference on Business Process Management, Springer, 2017, pp. 165–177.

[19] A. Burattin, S.J. van Zelst, A. Armas-Cervantes, B.F. van Dongen, J. Carmona, Online conformance checking using behavioural patterns, in: International Conference on Business Process Management, Springer, 2018, pp. 250–267.

[20] P.H. dos Santos Teixeira, R.L. Milidiú, Data stream anomaly detection through principal subspace tracking, in: Proceedings of the 2010 ACM Symposium on Applied Computing, 2010, pp. 1609–1616.

[21] G.M. Tavares, V.G.T. da Costa, V.E. Martins, P. Ceravolo, S. Barbon Jr., Leveraging anomaly detection in business process with data stream mining, iSys-Revista Brasileira de Sistemas de Informação 12 (1) (2019) 54–75.

[22] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Analyzing business process anomalies using autoencoders, Mach. Learn. 107 (11) (2018) 1875–1893.

[23] D. Pokrajac, A. Lazarevic, L.J. Latecki, Incremental local outlier detection for data streams, in: 2007 IEEE Symposium on Computational Intelligence and Data Mining, IEEE, 2007, pp. 504–515.

[24] S.H. Karimian, M. Kelarestaghi, S. Hashemi, I-inclof: improved incremental local outlier detection for data streams, in: The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012), IEEE, 2012, pp. 023–028.

[25] F. Cao, M. Estert, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: Proceedings of the 2006 SIAM International Conference on Data Mining, SIAM, 2006, pp. 328–339.

[26] D.C. Hoaglin, R.E. Welsch, The hat matrix in regression and ANOVA, Amer. Statist. 32 (1) (1978) 17–22.

[27] J. Ko, M. Comuzzi, Online anomaly detection using statistical leverage for streaming business process events, 2021, arXiv:2103.00831.

[28] F. Angiulli, F. Fassetti, Distance-based outlier queries in data streams: the novel task and algorithms, Data Min. Knowl. Discov. 20 (2) (2010) 290–324.

[29] M.S. Uddin, A. Kuh, Online least-squares one-class support vector machine for outlier detection in power grid data, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 2628–2632.

[30] M. Goldstein, S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, PLoS One 11 (4) (2016) e0152173.

[31] M. Kontaki, A. Gounaris, A.N. Papadopoulos, K. Tsichlas, Y. Manolopoulos, Efficient and flexible algorithms for monitoring distance-based outliers over data streams, Inf. Syst. 55 (2016) 37–53.

[32] L.-x. Liu, Y.-f. Guo, J. Kang, H. Huang, A three-step clustering algorithm over an evolving data stream, in: 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, 1, IEEE, 2009, pp. 160–164.

[33] W. Wang, T. Guyet, R. Quiniou, M.-O. Cordier, F. Masseglia, X. Zhang, Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks, Knowl.-Based Syst. 70 (2014) 103–117.

[34] X. Zhang, C. Furtlehner, M. Sebag, Data streaming with affinity propagation, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2008, pp. 628–643.

[35] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, in: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, IEEE, 2011, pp. 385–392.

[36] V.K. Samparthi, H.K. Verma, Outlier detection of data in wireless sensor networks using kernel density estimation, Int. J. Comput. Appl. 5 (7) (2010) 28–32.

[37] M.S. Uddin, A. Kuh, Y. Weng, M. Ilić, Online bad data detection using kernel density estimation, in: 2015 IEEE Power & Energy Society General Meeting, IEEE, 2015, pp. 1–5.

[38] A.R. Tuor, S.P. Kaplan, B.J. Hutchinson, N.M. Nichols, S.M. Robinson, Deep learning for unsupervised insider threat detection in structured cyber security data streams, Tech. rep., Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2018.

[39] H. Sun, Q. He, K. Liao, T. Sellis, L. Guo, X. Zhang, J. Shen, F. Chen, Fast anomaly detection in multiple multi-dimensional data streams, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 1218–1223.

[40] M.U. Togbe, M. Barry, A. Boly, Y. Chabchoub, R. Chiky, J. Montiel, V.-T. Tran, Anomaly detection for data streams based on isolation forest using scikit-multiflow, in: International Conference on Computational Science and Its Applications, Springer, 2020, pp. 15–30.

[41] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemometr. Intell. Lab. Syst. 2 (1–3) (1987) 37–52.

[42] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 71–80.

[43] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, C. Talhi, An anomaly detection system based on variable N-gram features and one-class SVM, Inf. Softw. Technol. 91 (2017) 186–197.

[44] M. Klimstra, E.P. Zehr, A sigmoid function is the best fit for the ascending limb of the hoffmann reflex recruitment curve, Exp. Brain Res. 186 (1) (2008) 93–105.

[45] A. Burattin, Plg2: Multiperspective process randomization with online and offline simulations., in: BPM (Demos), Citeseer, 2016, pp. 1–6.

[46] B.A. Tama, M. Comuzzi, J. Ko, An empirical investigation of different classifiers, encoding, and ensemble schemes for next event prediction using business process event logs, ACM Trans. Intell. Syst. Technol. (TIST) 11 (6) (2020) 1–34.

[47] S. Aryal, K.M. Ting, J.R. Wells, T. Washio, Improving iforest with relative mass, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2014, pp. 510–521.

[48] C. Dickens, E. Meissner, P.G. Moreno, T. Diethe, Interpretable Anomaly Detection with Mondrian Pólya Forests on Data Streams, 2020, arXiv preprint arXiv:2008.01505.

[49] A.A. Cohen, Q. Li, E. Milot, M. Leroux, S. Faucher, V. Morissette-Thomas, V. Legault, L.P. Fried, L. Ferrucci, Statistical distance as a measure of physiological dysregulation is largely robust to variation in its biomarker composition, PLoS One 10 (4) (2015) e0122541.

[50] R.A. Davis, K.-S. Lii, D.N. Politis, Remarks on some nonparametric estimates of a density function, in: Selected Works of Murray Rosenblatt, Springer, 2011, pp. 95–100.

[51] A. Maaradji, M. Dumas, M. La Rosa, A. Ostovar, Fast and accurate business process drift detection, in: International Conference on Business Process Management, Springer, 2016, pp. 406–422.