Expert Systems With Applications 131 (2019) 132–147

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Autoencoders for improving quality of process event logs

Hoang Thi Cam Nguyen[a], Suhwan Lee[b], Jongchan Kim[b], Jonghyeon Ko[b], Marco Comuzzi[b,*]

[a] Trusting Social, Vietnam
[b] Ulsan National Institute of Science and Technology, Republic of Korea

ARTICLE INFO

Article history:
Received 20 July 2018
Revised 11 March 2019
Accepted 19 April 2019
Available online 22 April 2019

Keywords:
Autoencoder
Event log
Business process management
Event log cleaning
Event log reconstruction
Event log quality

ABSTRACT

Low quality of business process event logs, as determined by anomalous and missing values, is often unavoidable in practical contexts. The output of process analysis that uses event logs with missing and anomalous values is also likely to be of low quality, thus decreasing the quality of any decisions based on it. While previous work has focused on reconstructing missing events in an event log or removing anomalous traces, in this paper we focus on detecting anomalous values and reconstructing missing values at the level of attributes in event logs. We propose methods based on autoencoders, which are a class of neural networks that can reconstruct their own input and are particularly suitable to learn a model of the complex relationships among attribute values in an event log. These methods do not rely on any a-priori knowledge about the business process that generated an event log and are evaluated using real world and artificially-generated event logs. The paper also discusses a qualitative analysis of the impact of event log cleaning and reconstruction on the output of process discovery. The proposed approach shows remarkable performance regarding activity labels and timestamps in artificial event logs. The performance in the case of real world event logs, in particular timestamp anomaly detection, is lower, which may be due to high variability of attribute values in the chosen event logs. Process models discovered from reconstructed event logs are characterised by lower variability of allowed behaviour and, therefore, are more usable in practice.

1. Introduction

The execution of business processes in modern organisations is supported by different types of enterprise systems, such as ERP, CRM, and workflow management systems. Historical information about execution of business processes can be recorded in so-called *event logs* using data produced by these systems. A record in an event log is an individual event that occurred in a particular process instance, or *case*, and includes attributes such as a case identifier, timestamp of occurrence, activity name, i.e., what was executed, and resource identifier, i.e., who was in charge of execution/supervision. The availability of event logs triggers the opportunity to engage in evidence-based analysis and improvement of business processes (van Eck, Lu, Leemans, & van der Aalst, 2015).

The quality of data-enabled analysis strongly depends on the quality of the underlying data used for it Batini, Cappiello, Francalanci, and Maurino (2009). This holds true also for event log-enabled business process analysis. For instance, inaccurate or missing timestamps hinder process discovery and compliance checking, which require events to be correctly ordered in time. In the monitoring of process KPIs, low quality information about resource identifiers may lead to unreliable values of indicators related with individual resources' efficiency in executing the tasks to which they are assigned.

A certain level of errors in event logs is often unavoidable, particularly when event logs are built by integrating several heterogeneous data sources or where manual logging is involved. Mans, van der Aalst, Vanwersch, and Moleman (2013), for instance, report that errors in event logs of health care processes mainly occur due to manual logging and that, among them, missing or incorrect case id, resource information, and activity labels occur at higher frequency than missing or abnormal timestamps. Therefore, there is a need for research in business process management to address the challenge of improving the quality of event logs, which in turn will enable higher quality analyses of business processes.

Quality of data can be improved by (i) improving the way in which data are captured while they are being generated and (ii) improving the data after they have been acquired (Batini et al., 2009). In this paper, we focus on the latter by considering two stages to improve the quality of event logs, namely data *cleaning* and *reconstruction*. The former refers to identifying abnormal

* Corresponding author.
E-mail addresses: hoang.cam.nguyen@trustingsocial.com (H.T.C. Nguyen), ghksdl6025@unist.ac.kr (S. Lee), jckim@unist.ac.kr (J. Kim), whd1gus2@unist.ac.kr (J. Ko), mcomuzzi@unist.ac.kr (M. Comuzzi).

https://doi.org/10.1016/j.eswa.2019.04.052
0957-4174/

values in a dataset, while the latter refers to the process of replacing, or *imputing*, missing values in a dataset with reliable substituted values.

Previous work in the area of improving quality of event log has interpreted event log cleaning as identifying anomalous (or simply infrequent) traces in a log (Bezerra & Wainer, 2013), and event log reconstruction as reconstructing missing events in a log (Wang, Song, Zhu, & Lin, 2013). Differently from previous work, we focus in this paper on event log cleaning and reconstruction at the level of attribute values, that is, whereby anomalous or missing information concerns individual attributes of given events in a log. Besides anomalous traces and missing events, anomalous or missing attributes are also likely to occur in practice in a random way. For instance, attribute values may be missing when operators forget to manually record the time at which a specific exam for a given patient was administered. Anomalous events may occur in IoT-supported process execution scenarios as a result of malfunctioning sensors registering abnormal values (Meroni, Baresi, Montali, & Plebani, 2018). Particularly regarding event log reconstruction, missing events reconstruction or other approaches reconstructing at the attribute level, e.g., timestamps in Dixit et al. (2018), require a-priori knowledge of the process control flow, for instance in the form of a process model. The approach presented in this paper is fully automated and does not require any a-priori knowledge.

Attribute-level event log cleaning and reconstruction is challenging because an event log has unique characteristics that make it different from other types of datasets, such as the ones traditionally used in health care or social science research. Traditionally, an individual record in a dataset should represent an individual observation of a phenomenon under study, e.g., one record in a medical dataset contains data about an individual patient, the treatment(s) that they have received and information about treatment outcomes. This does not apply to process event logs. While records in an event log represent a single event, process analysis is normally conducted at the level of process cases, e.g., analysing whether a case has complied with a reference process model, or at a global process level, e.g., process model discovery. Cases, rather than individual events, represent the actual observations in an event log. Because of this multi-layered structure of event logs, combined with temporal relations among events determined by timestamps, traditional data cleaning and reconstruction, such as replacing anomalous values with the mean or median values or with the most frequent label, are likely to perform poorly on event logs. Moreover, simply removing events with missing or anomalous values is likely to hinder the opportunity for process analysis.

In this paper, we address the challenges of event log cleaning and reconstruction using machine learning techniques and, in particular, neural networks. The application of machine learning techniques to event log analysis has shown to be successful in the use case of predictive monitoring (Marquez-Chamorro, Resinas, & Ruiz-Cortés, 2017). Machine learning techniques, in fact, and neural networks in particular (Tax, Verenich, La Rosa, & Dumas, 2017), can be used to learn non-linear models of data, that can take into account the complex relations between events and cases in an event log highlighted before. In this paper, we use autoencoders to clean and reconstruct event logs. Autoencoders are a particular class of neural networks that comprise two steps: an encoding step, in which a hidden distribution of input data is learnt, and a decoding step, in which the input data can be re-obtained from the parameters discovered during the encoding. In event log cleaning, anomalous values are those with a high reconstruction error in the decoding step. In event log reconstruction, autoencoders are used to reconstruct values for the values missing in the input dataset.

From an expert systems perspective, event logs represent a *knowledge base* (Attaran, 2004) that is exploited by different tools that support decision makers in different phases of the business process management lifecycle, such as process discovery, process compliance verification, or process predictive monitoring. Therefore, the techniques proposed in this paper represent a basic technology that improves the quality of the event log knowledge base to support a number of possible decision making use cases in business process management. In particular, in this paper we focus on the process discovery use case, by qualitatively assessing the impact of event log anomaly detection and reconstruction on the quality of the models that can be discovered from event logs.

In summary, the contribution of this paper are methods for cleaning and reconstructing event logs, which, differently from available literature, focus on the *attribute* level and do not require any particular a-priori domain specific knowledge about the process generating the log, particularly as far as the event log reconstruction phase is concerned.

The proposed methods are evaluated on both artificially-generated and real world event logs. Besides a performance analysis of event log cleaning and reconstruction, we also present a qualitative evaluation of the effects of event log cleaning and reconstruction on the typical process mining use case of process discovery. In particular, we show that anomalous and missing attribute values increase the variability in event logs and, therefore, the complexity of discovered process models. In the context of the framework presented in this paper, event log cleaning and reconstruction help to discover process models with smaller variability, which are also visually closer to the ones discovered from clean and complete event logs.

The paper is organised as follows. Section 2 provides required background on neural computing and discusses related work on event log quality. Section 3 presents the methods for event log cleaning and reconstruction. Datasets and experimental settings are presented in Section 4. The evaluation is presented in Section 5 and conclusions are eventually drawn in Section 6.

2. Related work and background

This section first discusses related work in the field of process event log quality, then it presents background information on the autoencoders that we use in our framework.

2.1. Event log quality

Several research works have focused on precisely defining and formulating data quality problems (Müller & Freytag, 2005; Oliveira, Rodrigues, & Henriques, 2005; Rahm & Do, 2000). Different approaches lead to different taxonomies, nevertheless, their findings show that the data quality problem manifests in very similar ways. Data quality issues in event logs have been classified by Bose et al. 2013 and, in the specific context of process mining in the health care, byMans et al. (2013). Bose, Mans, and van der Aalst (2013), in particular, have identified missing, incorrect, imprecise, and irrelevant data as type of sources of event log quality degradation. In addition, they analyse the manner in which these issues appear in reality. As a result, the two issues that most frequently occur in real-life logs are missing event, and imprecise activity name.

The issue related to timestamps is also a common cause of inconsistent results in process mining. Mans et al. (2013) point out imprecise resource, i.e., the recorded resource refers to a specific operating room instead of the person who performed the surgery, is more likely to happen than other issues in Hospital Information System (HIS). The process mining manifesto (van der Aalst et al., 2012) defines 5 maturity levels of event logs, which can also be

seen as a way of rating the quality of event logs. In this context, data quality, e.g., completeness and consistency of recorded information, is one of the many dimensions contributing to the definition of event log maturity levels.

Suriadi, Andrews, ter Hofstede, and Wynn (2017) classify a set of event log imperfection patterns that may guide the event log quality improvement phase. These patterns help understanding the sources of imperfection in an event log and, therefore, can guide the improvement of logging activities during process execution. Our paper focuses on a closely related, but different issue, that is, reconstructing anomalous and missing values in an event log that has already been acquired. In our work, we do not assume any knowledge about the process that has generated an event log.

The quality of event logs is strictly related with detecting noise in event logs, i.e., infrequent behaviour, and with repairing event logs, i.e., identifying and reconstructing missing events in an event log. Noise is typically removed in a pre-processing phase, using frequency-based approaches (Cheng & Kumar, 2015). As such, it can be seen in our context as a data *cleaning* activity and the task of detecting noise can be considered as novelty detection. While removing noise helps in many process mining use cases, such as discovering better process models, some decision making tasks may have to be performed on infrequent cases, which often may represent exceptions or critical cases. Therefore, simply removing infrequent behaviour from an event log may not be a viable choice in many practical scenarios.

van der Aalst and de Medeiros (2005) introduce a way to exploit $\alpha-$algorithm to identify anomalous traces. This method requires a complete log containing only normal behaviors to be given as input. In contrast, we assume that the only information available for repairing and reconstructing values in an event log is the event log itself. Ghionna, Greco, Guzzo, and Pontieri (2008) propose a two-step method to filter out the exceptional traces by first extracting the normal patterns and then applying a clustering procedure under the assumption that outliers are characterised by infrequent patterns. Genga, Alizadeh, Potena, Diamantini, and Zannone (2018) propose an approach to extract infrequent process patterns from event logs to help conformance checking. Böhmer and Rinderle-Ma (2017) have proposed methods for multi-instance anomaly detection, by correlating information regarding different instances in the same event log.

Recently, Nolle, Seeliger, and Mühlhäuser (2016) and Nolle, Luettgen, Seeliger, and Mühlhäuser (2018) have proposed to use autoencoders for de-noising event logs, showing remarkable performance, albeit on artificially generated logs. Even in this case, however, noise in event logs is defined at the event level, e.g., missing or duplicated events, rather than at the event attribute level as we consider in this paper. At the attribute level, the approach proposed only deals with categorical values, such as activity labels or resource identifiers, focusing only on anomaly detection.

Several ad-hoc methods have been proposed for repairing event logs by reconstructing missing events. Rogge-Solti, Mans, van der Aalst, and Weske (2013b) propose a method based on stochastic Petri nets and Bayesian networks. A similar approach can then be used to improve process documentation (Rogge-Solti, Mans, van der Aalst, & Weske, 2013a). Bayomie, Helal, Awad, Ezat, and ElBastawissi (2015) have proposed a method to reconstruct the value of case identifiers in an unlabeled event log. These approaches take a different perspective from the one considered in this paper, since they assume that a process model is available. Missing events or attribute values can be then reconstructed by combining process knowledge with knowledge discovery techniques. Dixit et al. (2018) present a set of heuristic methods to identify anomalous timestamps and a method to update anomalous timestamps using a-priori knowledge about the process control flow. While our approach is fully automated and does not require any a-priori knowledge, this approach requires manual interaction of decision makers, which should also have knowledge about the process control flow to repair anomalous timestamps.

Regarding data *imputation*, various methods have been proposed to do this task in large datasets, such as deletion of observations with missing values or reconstructing data using statistical and artificial intelligence techniques. Promising results have been shown particularly in imputation of medical domain datasets (Beaulieu-Jones & Moore, 2017; Shah, Bartlett, Carpenter, Nicholas, & Hemingway, 2014). Beaulieu-Jones and Moore (2017), in the task of dealing data missing completely at random and data missing not at random, find that using bottleneck architecture autoencoder integrated with dropout as a regularizer gives robust results at a variety of information loss. They also show that autoencoders outperform KNN and SVM even when the missing data is increased. One more advantage of this algorithm is that it runs in linear time. Inspired by this paper, in our work, we use the same approach, but only focus on dealing with data missing completely at random in event logs.

For time series, Che, Purushotham, Cho, Sontag, and Liu (2016) propose to deploy a deep neural network based on Gated Recurrent Unit (GRU) (Cho, van Merrienboer, Bahdanau, & Bengio, 2014a). Their model takes two missing patterns, namely masking and time interval. While the model identifies which inputs are observed or missing based on information from masking, it retrieves input observation pattern from time interval. Both representations, masking and time interval, are fed into the model so that GRU is enabled to not only capture the long-term temporal dependencies of time series but take advantage of the missing patterns to enhance the predictions. They also compare the performance of RNN-based and non-RNN-based to see the benefit of using RNN on extracting sequential information. Inspired by this paper, we also consider an RNN-based autoencoder architecture.

Recently, Gondara and Wang (2017) have proposed a multiple-imputation model based on denoising autoencoders, which are able to process a variety of data types (continuous, categorical and mixed), distributions (random and uniform), and missing patterns (missing completely at random and missing not at random). Their model not only performs well on large datasets, but also on small size ones. More remarkably, the model can cope well with the situation in which users do not provide complete observations for training. Nevertheless, their model can only take a fixed length sequence as input, whereas in this paper, we aim to reconstruct information in an event log, where each case may have different length in terms of number of activities.

2.2. Autoencoders

Autoencoders are a class of neural networks used for unsupervised learning and as generative models (Doersch, 2016). Given a dataset $X = \{x^i\} \in \mathbb{R}^n$, an autoencoder tries to learn a vector X' having a distribution similar to X. This is done in two separate processes (see Fig. 1 for a standard representation of an autoencoder). First, a vector Z, i.e., a *code*, is formed (or *encoded*) from X to learn a hidden, or *latent* model $q_\theta(z|x)$ of the data, where θ are the weights and biases of the encoder; second, a decoder $p_\phi(x|z)$ reconstructs a vector X' having similar distribution to X from the code Z (ϕ are the weights and biases of the decoder neural network). Several hidden layers can be stacked in between input and output layers in both processes, allowing a model to create a higher dimensional representation of the data. The autoencoder of Fig. 1, for instance, uses one hidden layer for both encoding and decoding, leading to a 5-layered deep neural network, and adopts a so-called *butterfly* architecture, in which the size of the input is

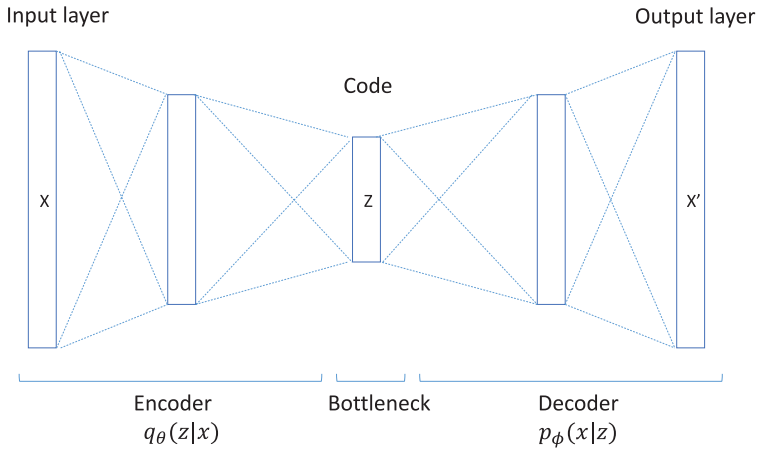


Fig. 1. The *butterfly* architecture of an autoencoder with five layers.

greater than the number of neurons in the hidden layer, which is greater than the code dimension.

The number of hidden units in the bottleneck layer, i.e., the dimension of the code, can be lower or higher than n. If the size of the code is lower than n, then an autoencoder is forced to learn a *compressed* representation of X, by identifying a limited number of interesting features characterising the latent distribution of X. If the size of the code is higher than n, then an autoencoder can still be used to learn interesting structures in data, particularly if specific constraints on hidden units are enforced, e.g., sparcity of hidden units when processing pixels of an image.

Autoencoders initially have been used for dimensionality reduction and feature learning. The basic idea of autoencoders (with code size lower than n) is similar to the one of Principal Components Analysis (PCA) (Abdi & Williams, 2010), that is, to project high dimensional data onto a manifold, which can represent data with a lower dimensional code. Unlike PCA, autoencoders are not restricted to linear transformations and they are able to reconstruct their output from latent variables. Autoencoders have been successfully applied in many real-world problems, such as paraphrase detection (Socher, Huang, Pennin, Manning, & Ng, 2011) or anomaly detection (Sakurada & Yairi, 2014). They also find several applications as generative models, since the learnt code Z can be used to generate new datasets from the same latent distribution of input X.

Similar to other feed-forward neural networks, the neurons in the hidden layers of an autoencoder compute the weighted sums of the input neurons and biases, which are then passed through a non-linear transformation guided by some activation function, such as sigmoid function. The learning process in an autoencoder aims at optimising a loss function $L(X, X')$, where L is a suitable likelihood function. L can be formulated based on the ultimate objective of the training process. When the data resemble a vector of probabilities, i.e., values comprised between 0 and 1, as in the method proposed in the paper, the loss function in Eq. (1) is used to optimise. This loss function considers the average cross-entropy of N observations x_i in X.

$$L(X, X') = \frac{1}{N}\sum_{i=1}^{N} x_i \log x'_i + (1 - x_i)\log(1 - x'_i) \tag{1}$$

Variational autoencoders (VAE) (Kingma & Welling, 2013) are a variant of autoencoders (AE), with additional constraints on the encoded representation to be learnt. A VAE, in fact, is an autoencoder with added constraints on the encoded representation Z. In particular, the features in a code Z learnt by a VAE are forced to roughly follow a given probabilistic distribution $p(z)$, e.g., a unit Gaussian distribution. This helps when VAE are used as a generative model, since new output data roughly similar to the input data can be generated by drawing values from such a distribution and passing them into the decoder part of the neural network.

The loss function to be optimised in a VAE for one data point $x_i \in X$ is shown in Eq. (2). The first term is the reconstruction term, or expected negative log-likelihood of x_i, taken in respect of the encoder's distribution over the code. If the decoder's output does not reconstruct the data well, then this term will be high. The second term is the Kullback–Leibler (KL) divergence (expanded in Eq. (3)). The KL divergence measures how closely the latent code z matches the chosen probability distribution $p(z)$. A common choice is to choose the unit gaussian distribution $N(0, 1)$ as $p(z)$. For more details about the internal functioning of VAE, we refer to Doersch (2016).

$$l_i(\theta, \phi) = -E_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z) + KL(q_\theta(z|x_i)||p(z))\right] \tag{2}$$

$$KL(q_\theta(z|x_i)||p(z)) = E_q[\log q_\theta(z|x)] - E_q[\log p(x|z)] + \log p(x) \tag{3}$$

As previously introduced, we also consider an RNN-based autoencoder architecture. *RNN Encoder-Decoder*, also known as *Sequential* autoencoders, have been first proposed by Cho et al. (2014b) to learn the representation of a sequence of words. Similar to AE and VAE, this model maps the input to a fixed-size vector with the encoder, then decodes this vector back to a target that reconstructs the input. In doing so, it uses one RNN as encoder and another RNN as decoder (see Fig. 2). A RNN Encoder-Decoder is able to estimate the conditional probability $p(y_1, \ldots, y_{T'}) \mid (x_1, \ldots, x_T))$, where $(x_1, \ldots, x_T)$ is the input sequence and $(y_1, \ldots, y_{T'})$ is the corresponding output sequence and, therefore, can be applied in sequence learning tasks.

Different types of recurrent units, such as Long Short-Term Memory unit or Gated Recurrent unit, can be used to build a RNN Encoder-Decoder. Since the work of Tax et al. (2017) has shown the suitability of LSTM networks to the context of predictive monitoring using event logs, in this paper we use LSTM units for the encoder and decoder. We call this model the Long Short-Term Memory Autoencoder (LAE).

3. Methods for event log cleaning and reconstruction

We propose two methods for improving event log quality: Event log cleaning and event log reconstruction. The former deals with identifying anomalous attribute values in event logs, such as

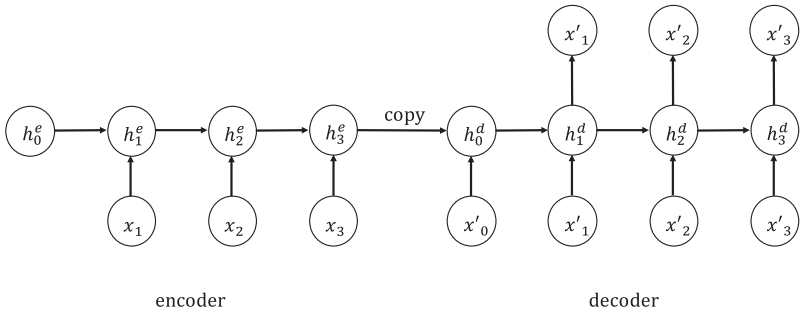


Fig. 2. The architecture of a simple sequence autoencoder to reconstruct output x' from input x.

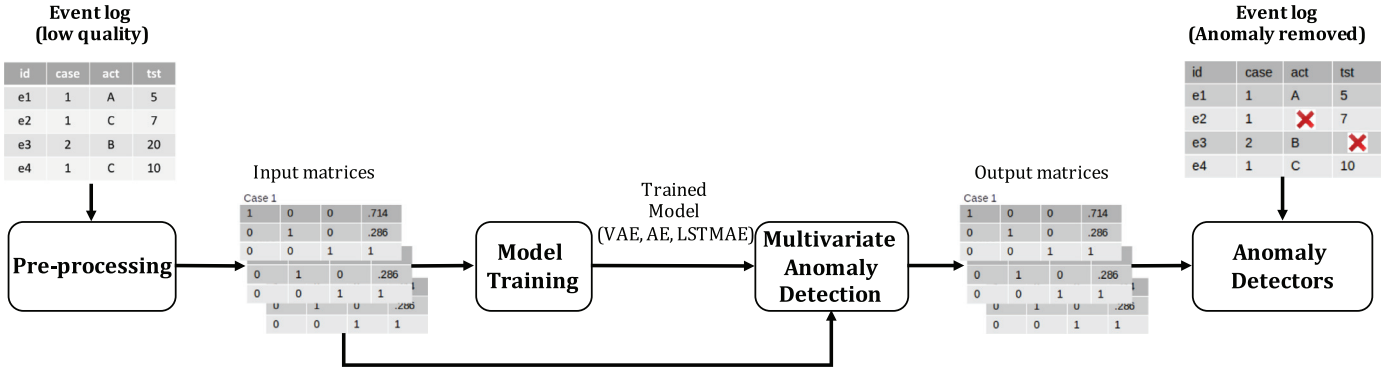


Fig. 3. Event log cleaning procedure.

erroneously logged activity or resource information or abnormal timestamps. The latter deals with reconstructing the value of attribute values missing in an event log by choosing reconstructed values that most truthfully reflect the actual execution of a process. Note that this paper focuses on anomalies at the individual attribute level. This has received little attention in the literature as compared to anomalies at the event level (see Section 2).

The two methods do not rely on any a-priori knowledge about the process that has generated an event log. They are independent methods and, as such, they are presented and extensively evaluated separately. Obviously, one interesting use case is the one in which the two methods are implemented in a pipeline scheme, in which anomalous attribute values, once identified, are removed from an event log, leading to an event log with missing values that need to be reconstructed. However, there may be also other practical scenarios in which event logs are generated with many missing values, i.e., missing values do not necessarily derive from a preceding event log cleaning phase, and for which the event log reconstruction method can be applied.

Autoencoders suit both methods. In both methods, in fact, the input and output are represented by the same type of dataset, that is, an event log. Moreover, in both methods the output event log should resemble the input event log for all values, except for some *erroneous* values, i.e., anomalous attribute values for event log cleaning and missing attribute values for event log reconstruction. Given that, in both methods, erroneous values are likely to represent only a fraction of the overall dataset, autoencoders are used as follows:

- In the event log cleaning phase, anomalous values are identified as those characterised by a high (above threshold) reconstruction error value. That is, after training, autoencoders should not be able to reconstruct anomalous values, because they would lead to high reconstruction errors;
- In the event log reconstruction phase, autoencoders are used in a more traditional way. In particular, autoencoders reconstruct a value for each input attribute value in an event log, including missing values, which are initially conventionally mapped to the numerical value zero.

The next two sections describe in more detail the event log cleaning and reconstruction method, respectively.

3.1. Event log cleaning

Event log cleaning is an instance of anomaly (or outlier) detection, which is concerned with identifying unusual patterns in data. Unlike a standard classification task, anomaly detection is an unsupervised learning task, since labelled observations to train a model are, by definition, not available.

The steps of the proposed method for event log cleaning are shown in Fig. 3. A low quality event log, i.e., with anomalous values, is taken as input. In a pre-processing phase, event logs are transformed into a format that can be fed into autoencoders. As discussed later in this section, for an autoencoder, each case in an event log represents an observation belonging to the input X. Hence, each case in an event log is transformed into a matrix of events and features that can be fed into an autoencoder.

After training an autoencoder, for each case x, we use the trained model to obtain a reconstructed case x'. The reconstruction error is an indication of the distance between the input vector x and the output vector x'. Under the assumption that the model will reproduce the *normal*, i.e., non anomalous, values for all attributes, the reconstruction error of cases containing only normal attribute values is smaller than the error of traces containing anomalous attribute values. Within these cases, the reconstruction error is high for those attributes for which the distance between x and x' is high. If the reconstruction error is less than a given anomaly detection threshold, then a value is considered normal, otherwise it is considered anomalous. As discussed in the previous section, we consider three different autoencoder architecture, namely AE, VAE and LAE.

Event log

id	case	act	tst
e1	1	A	5
e2	1	B	7
e3	2	B	3
e4	1	C	10

Standardisation of attributes "act" (discrete) and "tst" (continuous)

id	case	C_A	C_B	C_c	C_{tst}
e1	1	1	0	0	-0.42
e2	1	0	1	0	0.25
e3	2	0	1	0	-1.09
e4	1	0	0	1	1.26

Input of autoencoder

Case 1	e1	1	0	0	-0.42
	e2	0	1	0	0.25
	e4	0	0	1	1.26
Case 2		0	0	0	0
		0	0	0	0
	e3	0	1	0	-1.09

Fig. 4. Event log pre-processing for anomaly detection: example.

More in detail, reconstruction error and related threshold have to be defined for each feature of interest in a dataset. In this work, we focus on reconstructing anomalous/missing timestamp values and activity labels. The former is an example of numerical attribute in an event log, whereas the latter is an example of categorical value.

For detecting anomalous timestamps, the reconstruction error is defined as the absolute value of the difference between the input timestamp and the reconstructed output timestamp value. The mean reconstruction error is the considered as a threshold to discriminate normal (for which the reconstruction error is below threshold) from anomalous values (with reconstruction error above threshold). In the remainder, we refer to this as a *threshold-based* detector.

In order to detect anomalous activity labels, we consider two approaches. The first one is a threshold-based detector. Given the output vector of probability values assigned by the autoencoder to each possible label for an attribute in x, the error is calculated as the absolute value of the difference between this output probability vector and an input probability vector, in which probability 1 is assigned to the label assumed by the attribute in x and 0 to all other possible labels. As in the case of anomalous time detection, the mean error across attributes in x is used as threshold. The second one uses the activity label as the signal of anomaly. The activity is considered as anomalous when its reconstructed label and input label do not match. Since activity labels are reconstructed using the argmax method, i.e., assigning in x' the activity label with highest probability in the output vector, then we call this *argmax-based* detector.

Autoencoders require as input observations coded as a matrix of numerical values of fixed size. Values should also be scaled to facilitate faster convergence and accuracy. If, in fact, attribute values vary across different ranges, then attributes characterised by larger ranges would be more important than other attributes characterised by smaller ranges during the learning phases. To meet this requirement, an event log is pre-processed as described in the following to transform each case into a matrix of numeric values of fixed size (see Fig. 4 for an example).

Let $\mathcal{E}$ be the event universe, i.e., the set of all possible event identifiers. Events are characterised by attributes, e.g., they belong to a particular case, have a timestamp, correspond to an activity, and are executed by a particular person. Let $AN = \{a_1, \ldots, a_n\}$ be a set of all possible attributes names and $\mathcal{D}_{a_i}$ the domain of attribute named a_i, i.e., the set of all possible values for the attribute a_i. Attributes can be numerical or categorical. Numerical attributes, e.g., timestamps, assume value within a certain numerical interval, that is, $\mathcal{D}_{a_i} = [v_i^{\min}, v_i^{\max}] \subseteq \mathbb{R}$. Categorical attributes assume values within a given set, such as a set of activity identifiers (strings) for the activity label attribute, i.e., $\mathcal{D}_{a_i} = \{v_{i,1}, \ldots, v_{i,K}\}$. For any event $e \in \mathcal{E}$ and attribute name $a \in AN$, $\#_a(e) \in \mathcal{D}_a$ denotes the value of attribute named a for event e. A trace σ_m, i.e., the set of events executed in the m-th case of a process, is a sequence of distinct events $\sigma_m = \{e_{m,n}\}_{n=1,\ldots,N}$ with $e_n \in \mathcal{E}, \forall n$. An event log $\mathcal{L}$ with M cases can then be seen as the union of all traces, i.e., $\mathcal{L} = \bigcup_{m=1}^{M} \sigma_m$. Note that, when the activity label is the only attribute of interest in an event, an event log is often formalised as a multiset of traces. However, since in this paper we consider every event unique (at least because of a different timestamp value), we can define a log as a set of traces.

Categorical attributes (see encoding of the activity attribute *act* in Fig. 4) are encoded using a one-of-K scheme, also known as one-hot encoding. That is, for an attribute a_i a column $c_{i,k}$ in the autoencoder input matrix is created for each value $v_{i,k} \in \mathcal{D}_{a_i}$. A row is then created for each event e in an event log, with values in columns $c_{i,k}$, $k = 1, \ldots, K$ assigned as follows:

$$c_{i,k}(e) = \begin{cases} 1 & \text{if } \#_{a_i}(e) = v_{i,k} \\ 0 & \text{otherwise} \end{cases}$$

Numerical attributes $\#_{a_i}(e)$ (see encoding of timestamps Fig. 4) are encoded by standardising their value with the mean value μ_i and standard deviation value σ_i assumed by attribute a_i in the event log. That is, for a continuous attribute a_i, a column is created such that:

$$c_i(e) = \frac{\#_{a_i}(e) - \mu_i}{\sigma_i}, \tag{4}$$

with:

$$\mu_i = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \#_{a_i}(e) \tag{5}$$

$$\sigma_i = \sqrt{\frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} [\#_{a_i}(e) - \mu_i]^2} \tag{6}$$

Obtained rows are then grouped by case id and ordered by timestamp value. As a result, an individual case is represented by a $p \times q$ matrix, where p is the number of activities in the case and q is the number of columns resulting from the standardisation described above. Since an autoencoder requires a fixed-size matrix as input, zero-padding is applied to all cases for which $p < p_{\max}$, where $p_{\max}$ is the highest number of activities in a case in an event log (see zero-padding of the first two rows for case 2 in Fig. 4, in which it is assumed that case 1 is the one with highest number of activities in the event log). In the experiments that we conducted, the results do not depend on the position of the zero-padding rows in an input matrix.

The objective of the model training step is to train a model that can capture the general behaviour of data in an event log. Once a model has been trained, each case in an event log is reconstructed into an output matrix of elements $c'_{i,j}$ of size $p \times q$. As a result of the generating step, in the output matrix, the anomalous attributes values are replaced by a new value for numerical attributes and

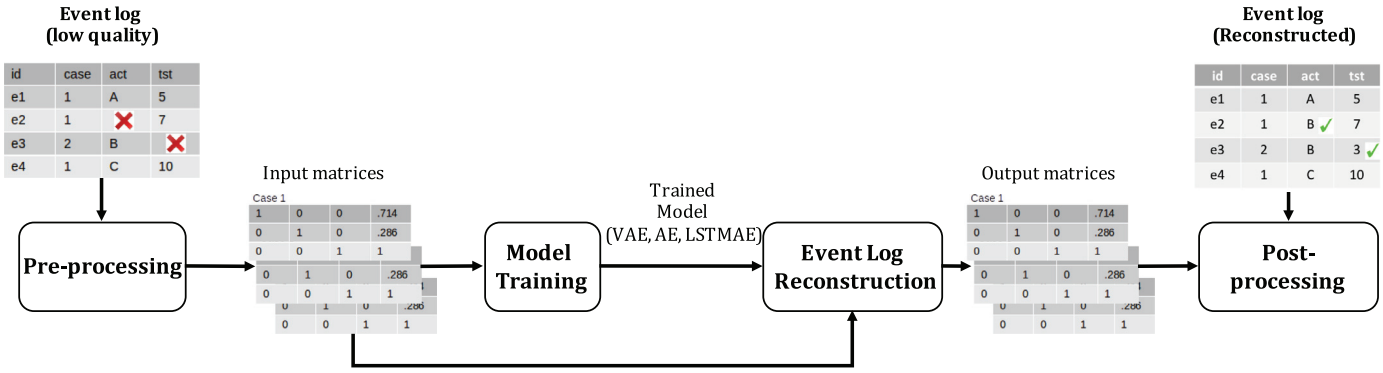


Fig. 5. Event log reconstruction procedure.

probabilities for the categorical attributes which are supposed to be the real values.

We introduce a masking matrix to indicate zero-padding values which should not be taken into consideration when the model computes the loss for updating weights. Elements $m_{i,j}$ of the masking matrix $M \in \mathbb{R}^{p \times q}$ are defined as:

$$m_{i,j} = \begin{cases} 0 & \text{if} c_{i,j} = 0 (0 - \text{padding}) \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

As introduced in the previous section, we use distance-based method to classify normal and abnormal behaviour, hence, we consider a modified mean squared error loss function, which uses the masking matrix of Eq. (8) before averaging across all values in an input matrix:

$$L(c_{i,j}, c'_{i,j}) = \frac{1}{p \times q} \sum_{i=1}^{p} \sum_{i=1}^{q} m_{i,j} \cdot (c_{i,j} - c'_{i,j})^2 \tag{8}$$

3.2. Event log reconstruction

Event log reconstruction is concerned with reconstructing (or *imputing*) a value to be used in place of the missing attribute values in an event log. These values can be missing because they have never been acquired or because they have been removed after having been identified as anomalous, or a combination thereof. In this task, autoencoders are used in a more traditional way. An autoencoder, in fact, reconstructs a value for all attributes in an event log. The value reconstructed for those attributes values originally missing is then considered as the imputed value.

The steps of the proposed method for event log reconstruction are shown in Fig. 5. A low quality event log, i.e., with missing values, is taken as input. After having learnt a model of the input, a model is used to reconstruct missing values in an event log using the latent distribution learnt by an autoencoder. We consider the same types of autoencoders that we consider for event log anomaly detection, i.e., AE, VAE and LAE.

In a post-processing phase, for each case the continuous attribute values reconstructed by the model are then translated back into the traditional format of event logs by applying the inverse of the transformation adopted in the pre-processing phase. Meanwhile, the output corresponding to the categorical attributes is fired by *Softmax* function to get the probability of the likelihood of each label occurring. Before feeding the data into the training network, preprocessing is required on this dataset to make it more appropriate to our model. Similar to the steps described for event log cleaning, the categorical attributes are encoded using one-of-K scheme (see encoding of activity labels in Fig. 6). Note that all values in this case are normalised in the range [0,1] Numerical attributes $\#_{a_i}(e)$ (see encoding of timestamps Fig. 6) are encoded by normalising their value between the minimum value $L_m^{\min}$ the maximum value $L_m^{\max}$ assumed by attribute a_i within the case m to which event e belongs. That is, for a continuous attribute a_i, a column is created such that, for each event e belonging to trace σ_m in an event log:

$$c_i(e) = \frac{\#_{a_i}(e) - L_m^{min}}{L_m^{max} - L_m^{min}}, \tag{9}$$

with $L_m^{\max} = \arg\max_{e \in \sigma_m} \#_{a_i}(e)$ and $L_m^{\min} = \arg\min_{e \in \sigma_m} \#_{a_i}(e)$.

Alternatively, numerical attributes values can be normalised using the minimum and maximum values $l_i^{\min}$ and $l_i^{\max}$ in $\mathcal{D}_{a_i}$. This choice, however, is highly sensitive to abnormally high values in the domain $\mathcal{D}_{a_i}$.

For both categorical and numerical attributes, missing values are encoded to the value 0.

Then, we need to transform the data within a case into matrix through *group-by* and *padding*, in order to get a $p \times q$ matrix as a presentation of a case (see again Fig. 6).

The objective of the model training step is to train a model that can learn the *latent* distribution of data in an event log. Once a model has been trained, each case in an event log is reconstructed into an output matrix of elements $c'_{i,j}$ of size $p \times q$. As a result of the model learning step, in an output matrix missing attributes values (denoted by 0 in the input matrix) are mapped to valid value for numerical attributes and to probabilities for the categorical attributes. In other words, the task of reconstructing a missing value of a numerical attribute is a regression task, while reconstructing values of categorical attributes is a classification task.

To define the loss function to be optimised, during model training we introduce a masking matrix to distinguish missing values and zero-padding values from non-zero values in an input matrix. The loss function, in fact, must be designed in such a way that 0 values in an input matrix should not be learnt, since they correspond to either artificially added zero-padding values or initially missing values. Elements $m_{i,j}$ of the masking matrix $M \in \mathbb{R}^{p \times q}$ are defined as:

$$m_{i,j} = \begin{cases} 0 & \text{if} c_{i,j} = 0 (\text{missingor} 0 - \text{padding}) \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

We consider a cross entropy-based loss function for autoencoders, which uses the masking matrix of Eq. 11 and in which cross-entropy is averaged across all values in an input matrix:

$$L(c_{i,j}, c'_{i,j}) = \frac{1}{p \times q} \sum_{i=1}^{p} \sum_{i=1}^{q} c_{i,j} \cdot m_{i,j} \log c'_{i,j} + (1 - c'_{i,j}) \cdot m_{i,j} \cdot \log(1 - c'_{i,j}) \tag{11}$$

The last step is to post-process the output matrices to reconstruct an event log in its traditional format. In this post-processing

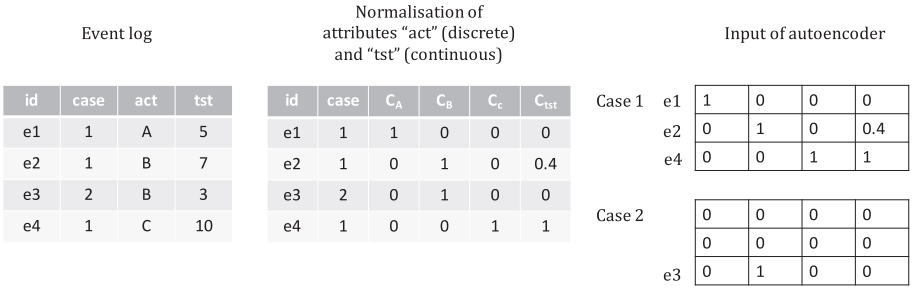


Fig. 6. Event log pre-processing for ELR: Example.

step missing values of numerical attributes are transformed into valid values in an event log by inverting Eq. (9), whereas an output value a_i of a categorical attribute is reconstructed by softmax activation, that is, the attribute value $a_{i,k}$ with highest probability value $c'_{i,k}$ in an output matrix is chosen as reconstructed value.

4. Datasets and experimental settings

This section presents the datasets and experimental settings used in the evaluation of the proposed methods. Specifically, Section 4.1 introduce the event logs considered in the experiments and the procedure followed to introduce anomalous and missing attribute values in them. Section 4.2 presents the settings of experiments, while Section 4.3 introduces the metrics used for evaluating the results of experiments.

4.1. Datasets and simulation of anomalous and missing attribute values

We have considered two publicly available real life event logs from used in the Business Process Intelligence Challenge of 2012 and 2013 (BPI 2012 and BPI 2013) and two artificial event logs event logs (small and large log) generated by simulation using the PLG2 tool[1] using uniformly distributed activity durations. The *BPI 2012* log is an event log of a loan application process at a large dutch financial institution, the *BPI 2013 log* is an event log of an IT incident and problem management at Volvo. Table 1 reports the number of events and cases for each log.[2]

As far as we are aware, there is no publicly available event log labelled with anomalous values and corresponding correct values that we could use in our evaluation. In the evaluation, the two artificial logs are complete, i.e., without missing values, and do not contain anomalies. Therefore, anomalous and missing data can be injected into them by simulation. The two real life logs are also complete. Regarding anomalies, similarly to previous research, we consider also these real life logs as free of anomalies and inject them with anomalous values purposely generated. In practice, it is impossible to guarantee that the original logs are free of anomalies at the moment they have been collected and we can only control for the anomalous values that we inject into the original logs. This means that the results regarding real life logs may be less reliable than the ones on artificial logs, owing to the fact that some anomalous values in the original logs are not labelled as anomalous in our experiment. Previous research in event log anomaly detection, for instance (Böhmer & Rinderle-Ma, 2017; Genga et al., 2018) in the case of anomalous trace detection and (Nolle et al., 2018) in the case of event and attribute level anomaly detection, has implemented consistently a similar approach to generate a viable evaluation dataset for anomaly detection with real life event logs.

Thus, one additional challenge in this paper is to generate random anomalous and missing attribute values in an event log to evaluate the two proposed methods. During the execution of a business process, we assume that abnormal attribute values are generated unexpectedly and the structure of these data cannot be observed or described in closed form, which makes it impossible to learn a model that can fit the pattern of the anomalies.

Given a level L of anomalous values that we want to achieve, $L/2$ anomalous values are assigned to activity labels and $L/2$ to timestamps. In the experiments we consider $L = 30\%$, with both anomalous timestamps and activity labels injected simultaneously in an event log. An activity is anomalous when its label is recorded incorrectly. Therefore, we perturb $L/2$ randomly selected activities in an event log by replacing them with another activity in the log, which is also randomly sampled. Note that replacing an activity with a clearly wrong label, e.g., a misspelled label, is considered trivial, since this type of anomalous values can be identified and reconstructed using standard database cleaning techniques.

Timestamp values are perturbed by considering the duration of the activity corresponding to them. Let μ_{act} and σ_{act} be the mean and standard deviation, respectively, of the duration of an activity $\#_{act}(e)$ of an event e chosen to be anomalous, calculated across all occurrences of this activity in an event log. To introduce anomalous timestamps, the timestamps of event e are modified so that the duration of e is equal to $(\mu_{act} + \sigma_{act}) \cdot (1 + r)$, where r is a random real value sampled uniformly in [0,1]. Note that timestamps are made anomalous by increasing the duration of events in a log. This is consistent with the fact that, in many practical cases, such as with manual logging, events are more likely to be logged with a delay, rather than before they actually occur. Note also that clearly invalid timestamps, e.g., timestamps in a wrong format, can be easily spotted using filtering techniques commonly exposed by process mining tools and, as such, they are not considered in this paper.

For evaluating the event log reconstruction, we need to randomly remove attribute values. Also in this case, our framework focuses on missing activity labels and timestamps.

Since we are interested in investigating the impact of informative missingness on the performance of our algorithm and to better control the experiments, we consider 30%, 35%, 40% and 50% as

[1] http://plg.processmining.it/.

[2] The event logs and the source code to reproduce the experiments presented in this paper are available at https://github.com/IELunist/Autoencoders-for-Improving-Quality-of-Process-Event-Logs/tree/master/multivariate-anomaly-detection-for-event-logs-master.

Table 1
Choices of hidden size for experiments.

Data	Number of events	Number of cases
BPI 2013	65,533	7,554
BPI 2012	262,200	13,087
Small log	28,000	2,000
Large log	120,000	15,000

Table 2
Example of missing attribute value setting, using the BPI 2012 event log.

Case ID	Activity	Complete timestamp
Case 1	A_SUBMITTED-COMPLETE	01/10/2011 07:38:45
Case 1	A_PARTLYSUBMITTED-COMPLETE	01/10/2011 07:38:45
Case 1	A_PREACCEPTED-COMPLETE	NaT
Case 1	NaN	NaT
Case 1	NaN	01/10/2011 18:36:46

Table 3
Choices of hidden size for experiments.

Data	Hidden size [AE, VAE, LAE]
BPI 2013	[100, 5, 10]
BPI 2012	[300, 100, 20]
Small log	[100, 50, 1]
Large log	[50, 20, 5]

ratios of attribute values missingness in event logs and we conduct these experiments in isolation with respect to the experiments about event log anomaly detection. Note that these levels of missingness are likely to be much higher than the ones encountered under reasonable assumptions in practice.

To decide which values to set as missing, we randomly sample two integers x_1 and x_2 from a discrete uniform distribution $U(0, |\mathcal{E}|)$ and $U(1, 2)$, respectively, with $|\mathcal{E}|$ is the number of events in a log $\mathcal{E}$. In this way, x_1 is interpreted as an identifier of the event for which an attribute is set as missing, i.e., the *row* in an event log, and x_2 as the attribute to be set as missing, i.e., 1 for timestamp and 2 for activity label. Then, we set the observation at the location of x_1 and x_2 as missing. Table 2 shows an example of setting missing values using the BPI 2012 dataset.

In this case, x_1 is drawn from a uniform distribution in the range [1,5], that is, 5 events belong to the log, and x_2 from a uniform distribution in range [1,2], i.e., only 2 attributes can be missing (activity or timestamp). In Table 2, 4 missing values have been set, for $(x_1, x_2) \in \{(3, 2), (4, 1), (4, 2), (5, 1)\}$. Note that continuous missing attributes are set to NaN, whereas discrete missing attributes are set to NaT. As a result of this procedure, the likelihood of an attribute value to be missing is completely random.

4.2. Experiment settings

For event log cleaning, to achieve a balance between the number/complexity of experiments and the quality of the results, we consider a fixed neural network structure and we show the results obtained for optimal hyperparameter values of this fixed structure. For VAE and AE, we consider an architecture with two hidden layers (one encoder and one decoder) and one code layer. The number of neurons in hidden layers is smaller than the input size. We also use non-linear activation function and dropout in the hidden layers, which enables faster convergence and avoids overfitting problem. Based on guidance from the literature (Doersch, 2016; Gondara & Wang, 2017; Kingma & Welling, 2013), internal neurons use *Tanh* and *ReLU* activation functions in AE and VAE, respectively. For LAE, we use LSTM cells with hidden size smaller than the number of features to build the RNN encoder-decoder. The hidden layer sizes used in the experiments for event log cleaning for different datasets can be found in Table 3.

We conducted several experiments to find optimal hyperparameters, with early stopping based on the validation loss. Each experiment runs in 100 iterations, in each iteration, the data is loaded in batch size of 16. The models are optimised by Adam algorithm (Kingma & Ba, 2014) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We apply adaptive learning rate with an exponential decay factor of 0.99 to adjust the learning rate after each iteration. The weights of each layer are initialised from a Xavier normal random distribution. In VAE and AE, we used dropout at 0.2 in order to avoid overfitting problem, while in LAE we use clipping to avoid vanishing/exploding gradient problem.

It is critical to choose the right learning rates since they affect the speed of convergence and the quality of models. Therefore, we tune the learning rate manually. We start the training procedure with the learning rate of 0.01 and gradually decrease by 0.001 after each training. By doing this, we find that the learning rate 0.0001 is sufficiently good for the training procedure of the three models. We train the model until the condition of early stopping, in which the error in the validation set does not improve after 10 epochs, is met, or the maximum number of epochs is reached.

For event log reconstruction, the configurations of the autoencoders in terms of activation functions and types of hidden layers are the same as those in event log cleaning, with the exception of using sigmoid as the last activation function since it squashes the output to 0 and 1.

The methods have been implemented using Pytorch.[3] Experiments run on an Intel i7 Linux machine equipped with 16GB memory and a GeForce GTX 1080 GPU. The execution time of one epoch training in the case of event log cleaning is given in Table 4 (runtime is similar for event log reconstruction).

Table 4
Execution time reported in milliseconds.

Data	VAE	AE	LAE
BPI 2013	250	200	350
BPI 2012	2400	1800	9500
Small log	300	200	600
Large log	1900	1300	2500

4.3. Evaluation criteria and baselines models

Regarding event log cleaning, because of imbalanced classes in the dataset, evaluation using the standard accuracy rate is not a good choice. The model, in fact, may act like a general classification model, trying to learn and generalise the majority examples, whereas the minority ones could be misclassified frequently. The performance of the proposed methods should be evaluated based on their ability of identifying the anomalies. Therefore, we assess the performance by considering performance metrics of each class label. There are 4 possible outcomes of the binary classification of a variable labeled either *a* (anomalous) or *n* (non anomalous):

- True positive: A non-anomalous value is assigned a correct label *n*;
- False positive: An anomalous value is incorrectly assigned to an *n* label;
- True negative: An anomalous value is correctly assigned to a label *a*
- False negative: A non-anomalous value is incorrectly is incorrectly assigned to an *n* label.

For an attribute a_i, given the total values TP_i, FP_i, TN_i, and FN_i of true positives false positives, true negatives, and false negatives, respectively, the precision (π_i), recall (ρ_i) and F-score (f_i) metrics are defined as follows:

$$\pi_i = \frac{TP_i}{TP_i + FP_i} \qquad \rho_i = \frac{TP_i}{TP_i + FN_i} \qquad f_i = 2 \times \frac{\pi_i \times \rho_i}{\pi_i + \rho_i} \tag{12}$$

Then, the average values of these metrics weight by the number of elements in each class are considered.

[3] https://github.com/pytorch.

As a baseline for anomaly detection of timestamps (numerical values), we consider Cook's distance in regression analysis. Cook's distance has often been used to investigate a data point with large residual (outliers) for a long time (Cook, 1977). After having fitted a regression model on the data under observation, the Cook's distance $cd(j)$ of an observation (j) is calculated as A/B, where A is the MSE on the entire dataset between the fitted regression model and the regression model obtained by removing j, and B is the MSE between the actual data and the fitted regression model. An observation j is an outlier if $cd(j) > \alpha \cdot E[cd(j)]$, where $E[cd(j)]$ is the average Cook's distance across all observations in the dataset. In order to consider a strong baseline for our experiments, we have conducted several experiments ex-post varying α and we have chosen the values $\alpha = 2.5$ for the artificial event logs and $\alpha = 4.0$ for the BPI logs, which maximise the outlier detection accuracy.

As a baseline for the anomaly detection of activity labels (categorical values), we use the K-Nearest Neighbour (KNN) algorithm (Chomboon, Chujai, Teerarassamee, Kerdprasop, & Kerdprasop, 2015) using $K = 2$ (normal v. anomaly) and the Euclidean distance. For both Cook-s distance and KNN baselines experiments we use the standard implementation in R.

As mentioned previously, the error metrics for the timestamp reconstruction are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE); the error metric for activity label reconstruction is accuracy. The performance of the proposed method for event log reconstruction is evaluated against baseline imputation methods commonly used in the literature. For imputing the values of categorical attributes, i.e., activity name in our experiments, a traditional approach is to consider the most frequent observation (Shah et al., 2014). In our experiments, the baseline BL for missing activity values is the most frequent activity name in an event log. For imputing values of numerical attributes, the median or mean value are often considered (Shah et al., 2014). In our experiments, we consider 4 possible baselines for imputing missing timestamp values, i.e., reconstructing timestamp values using the median (BL_1) and mean (BL_2) duration of all activities in an event log, and using the median (BL_3) and mean (BL_4) duration of the activity to which a missing timestamp belongs. Note that, if both activity name and timestamp are missing for an event, then the activity name is reconstructed first using the proposed method, and this imputed value is used to calculate the baseline values of the missing timestamp.

5. Results and discussion

The results of experiments concerning event log cleaning and reconstruction are discussed in Section 5.1 and 5.2, respectively. Then, Section 5.3 presents a qualitative analysis of the effects of event log cleaning and reconstruction on process discovery.

Note that tables presented in this section report average performance across 10 replications of each experiment. Each replication uses as input a different event log obtained by randomly injecting anomalous or missing values in the original event log according to the procedures described in the previous section. The standard deviation across these repetitions is always very limited (<1.5% of average values) and, as such, it is not shown.

5.1. Event log cleaning

Table 5 shows the results of the anomaly detection of the timestamp attribute. VAE and AE are extremely effective in detecting anomalies in the artificial event logs (small and large), whereas LAE shows worse performance. All 3 proposed architectures fail to achieve acceptable performance on anomalous timestamp detection with the real event logs. This is due to the large range and extreme variability of case and activity duration in these logs. Because timestamps are drawn from normal distributions in the artificial event logs, it appears to be easy to train an autoencoder to discover anomalous values in these cases.

Table 5 also shows the performance obtained by the baseline model (outlier detection based on regression analysis using Cook's distance). The baseline outscores in many cases the proposed approach with real life logs, but its performance is still far from being acceptable for real decision making use cases. This confirms that, mainly due to the irregular distribution of activity durations, detecting timestamp anomalies in the real life event logs considered in this paper is particularly challenging. The performance of the proposed approach is remarkably better than the baseline in the case of artificial logs (small and large), both in terms of precision amd recall. This suggests that the proposed approach fits well event logs in which activity duration distribution is uniform.

Tables 6 and 7 show the results for the threshold-based and argmax-based anomalous activity, respectively. Also in this case, the performance is remarkable with artificial event logs. Regarding real world event logs, the performance is better than in the case of anomalous time reconstruction, with the AE architecture showing better performance than VAE and LAE. Overall, threshold-based anomaly detection appears to perform better than argmax-based detection.

Table 6 also shows the peformance of the KNN baseline for detecting anomalous activities. The AE architecture consistently outscores the baseline in all cases. This performance difference is higher for the case of real life event logs.

Comparing the performance of the three models, the results show that VAE and AE outperforms LAE in all datasets. The performance obtained by AE is slightly higher than those from VAE. In terms of the model complexity, the LAE architecture is more complex than VAE and AE models, resulting in more expensive compu-

Table 5
Performance of anomalous time detector.

Data	Class	AE			VAE			LAE			Cook's D			Support
		Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	
BPI 2013	Normal	0.93	0.65	0.77	0.9	0.63	0.73	0.91	0.63	0.75	0.91	0.12	0.21	963.6
	Anomalous	0.16	0.58	0.25	0.11	0.4	0.17	0.13	0.47	0.2	0.32	0.97	0.48	107.4
	Average	**0.853**	**0.643**	**0.718**	**0.821**	**0.607**	**0.674**	**0.832**	**0.614**	**0.695**	**0.89**	**0.205**	**0.237**	**1071**
BPI 2012	Normal	0.92	0.61	0.73	0.92	0.62	0.74	0.92	0.67	0.78	0.91	0.98	0.94	43268.9
	Anomalous	0.11	0.47	0.18	0.11	0.45	0.18	0.12	0.41	0.18	0.31	0.07	0.11	4554.1
	Average	**0.839**	**0.596**	**0.675**	**0.839**	**0.603**	**0.684**	**0.84**	**0.644**	**0.72**	**0.89**	**0.889**	**0.857**	**47823**
Small log	Normal	0.97	0.99	0.98	0.98	0.99	0.99	0.98	0.68	0.8	0.93	0.94	0.93	5094.10
	Anomalous	0.92	0.69	0.78	0.95	0.8	0.87	0.2	0.83	0.33	0.37	0.34	0.35	505.90
	Average	**0.965**	**0.96**	**0.96**	**0.977**	**0.971**	**0.978**	**0.902**	**0.695**	**0.753**	**0.88**	**0.88**	**0.872**	**5600**
Large log	Normal	0.97	0.99	0.98	0.98	0.99	0.98	0.94	0.63	0.76	0.92	0.84	0.88	21903.1
	Anomalous	0.82	0.71	0.76	0.94	0.8	0.85	0.14	0.59	0.22	0.2	0.35	0.25	2096.9
	Average	**0.955**	**0.962**	**0.958**	**0.976**	**0.971**	**0.967**	**0.86**	**0.626**	**0.706**	**0.79**	**0.79**	**0.82**	**24000**

Table 6
Performance of threshold-based anomalous activity detector.

Data	Class	AE			VAE			LAE			KNN			Support
		Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	
BPI 2013	Normal	0.98	0.96	0.97	0.98	0.96	0.97	0.94	0.68	0.79	0.91	0.94	0.92	969.1
	Anomalous	0.7	0.79	0.74	0.69	0.79	0.74	0.16	0.59	0.25	0.57	0.91	0.7	101.9
	Average	**0.952**	**0.943**	**0.947**	**0.951**	**0.943**	**0.947**	**0.866**	**0.671**	**0.739**	**0.88**	**0.94**	**0.898**	**1071**
BPI 2012	Normal	0.96	0.96	0.97	0.97	0.81	0.89	0.92	0.94	0.93	0.93	0.95	0.94	43050.5
	Anomalous	0.69	0.67	0.68	0.32	0.78	0.45	0.38	0.33	0.36	0.74	0.98	0.84	4772.5
	Average	**0.933**	**0.931**	**0.941**	**0.905**	**0.807**	**0.846**	**0.860**	**0.872**	**0.867**	**0.91**	**0.953**	**0.93**	**47823**
Small log	Normal	0.99	1	0.99	0.99	1	0.99	0.97	0.36	0.52	0.99	0.98	0.99	5037.9
	Anomalous	0.99	0.91	0.95	1	0.9	0.95	0.13	0.9	0.23	0.83	0.93	0.93	562.1
	Average	**0.99**	**0.991**	**0.986**	**0.991**	**0.99**	**0.986**	**0.886**	**0.414**	**0.491**	**0.976**	**0.975**	**0.984**	**5600**
Large log	Normal	0.99	0.99	0.99	0.99	1	0.99	0.95	0.65	0.77	0.99	0.98	0.98	21592.5
	Anomalous	0.92	0.94	0.93	1	0.94	0.97	0.2	0.69	0.29	0.87	0.99	0.93	2407.5
	Average	**0.983**	**0.985**	**0.984**	**0.991**	**0.994**	**0.988**	**0.875**	**0.654**	**0.722**	**0.99**	**0.99**	**0.98**	**24000**

Table 7
Performance of argmax-based anomalous activity detector.

Data	Class	AE			VAE			LAE			Support
		Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**	
BPI 2013	Normal	0.99	0.83	0.9	0.99	0.72	0.84	0.97	0.76	0.85	969.1
	Anomalous	0.36	0.93	0.52	0.27	0.97	0.42	0.25	0.79	0.39	101.9
	Average	**0.927**	**0.84**	**0.862**	**0.918**	**0.745**	**0.798**	**0.898**	**0.763**	**0.804**	**1071**
BPI 2012	Normal	0.99	0.8	0.89	0.99	0.29	0.45	0.93	0.87	0.9	43050.5
	Anomalous	0.35	0.98	0.52	0.13	0.98	0.23	0.28	0.45	0.35	4772.5
	Average	**0.926**	**0.818**	**0.853**	**0.904**	**0.359**	**0.428**	**0.865**	**0.828**	**0.845**	**47823**
Small log	Normal	1	0.99	0.99	1	0.67	0.8	0.95	0.14	0.25	5037.9
	Anomalous	0.88	0.96	0.92	0.25	0.97	0.4	0.11	0.94	0.19	562.1
	Average	**0.988**	**0.987**	**0.983**	**0.925**	**0.7**	**0.76**	**0.866**	**0.22**	**0.244**	**5600**
Large log	Normal	0.99	0.98	0.98	1	0.79	0.88	0.93	0.85	0.89	21592.5
	Anomalous	0.82	0.92	0.87	0.34	0.98	0.51	0.25	0.42	0.31	2407.5
	Average	**0.973**	**0.974**	**0.969**	**0.934**	**0.809**	**0.843**	**0.862**	**0.807**	**0.832**	**24000**

Table 8
Number of anomalous values detected for different values of reconstruction error threshold.

Data	Timestamp (Threshold-based)	Activity (Threshold-based)	Activity (Argmax-based)	Threshold value
BPI 2012	39.61%	58.90%	45.65%	T (mean)
BPI 2013	39.46%	49.14%	17.62%	
Large log	23.63%	43.74%	26.82%	
Small log	39.67%	46.56%	25.22%	
BPI 2012	11.33%	0.00%		$T^{0.5}$ (high)
BPI 2013	16.74%	0.00%		
Large log	7.94%	0.00%		
Small log	3.53%	0.00%		
BPI 2012	80.09%	100.00%		T^2 (low)
BPI 2013	64.02%	100.00%		
Large log	86.02%	100.00%		
Small log	87.61%	100.00%		

tational cost. In addition, tuning LAE requires a lot of effort. From the experiments, we can conclude that AE provides more benefits in the case of event log cleaning.

Finally, we present an analysis of the sensitivity of the results to the choice of threshold to identify anomalous values with threshold-based detectors. As remarked in Section 3, we have considered the mean reconstruction error as threshold for both tasks of activity and timestamp anomaly detection. Choosing the mean reconstruction error as threshold does not rely on any a-priori or domain specific knoweledge about an event log.

Figs. 7 and 8 show the number of attributes identified as anomalous, as a fraction of the total number of attributes in a log, when the threshold varies in the interval $+/-10\%$ of the mean reconstruction error T, for the activity and timestamp reconstruction, respectively. Table 8 shows, in a tabular format, the same for high values (set at $T^{0.5}$) and low values (set at T^2). For the activity anomaly detection, Table 8 also shows the results achieved by the argmax-based detector, which do no depend on the threshold value. Regarding activity label anomaly detection, when the threshold considered is lower or higher than T, a large number of observations quickly are recognised as either anomalous, for lower values of the threshold, or non-anomalous, for higher values of the threshold (see Fig. 7). Fixing the threshold to T appears to overestimate the number of anomalies (set at 30%) when using the threshold-based detector even in the case of artificial logs. So a slightly higher value of the threshold may have been a better choice to identify a correct number of anomalous values. However, Table 8 shows that the argmax-detector identifies a number of anomalies closer to 30%. The anomaly detection of timestamps is much less sensitive to the choice of threshold (see Fig. 8). Finally, we remark that choosing very high or low values of the threshold (see Table 8) leads to unusuable anomaly detection models, in which most attribute values are recognised as either anomalous (when threshold is $T^{0.5}$) or non-anomalous (when threshold is T^2). This analysis shows that choosing the mean reconstruction error as threshold is a viable choice in our case and that choosing a good

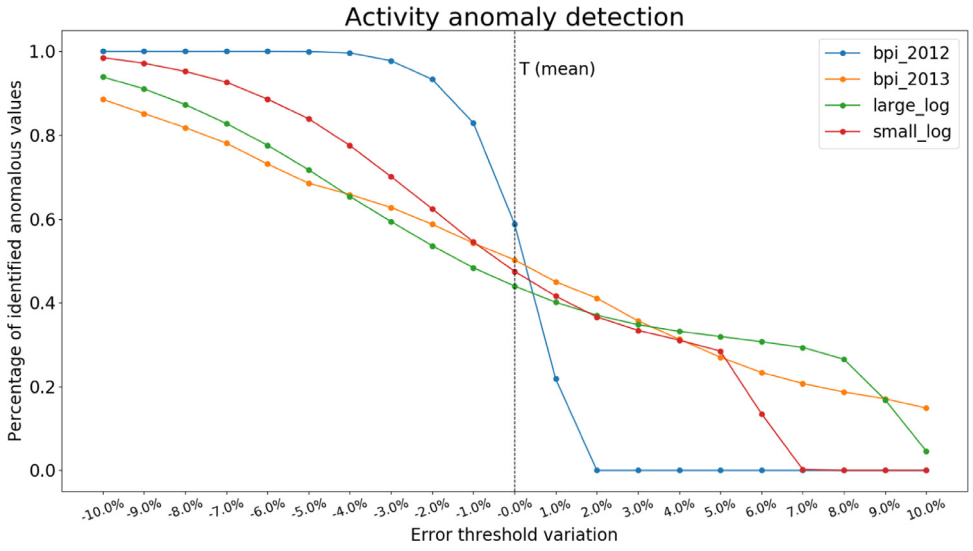


Fig. 7. Sensitivity to threshold variation - activity anomaly detection.

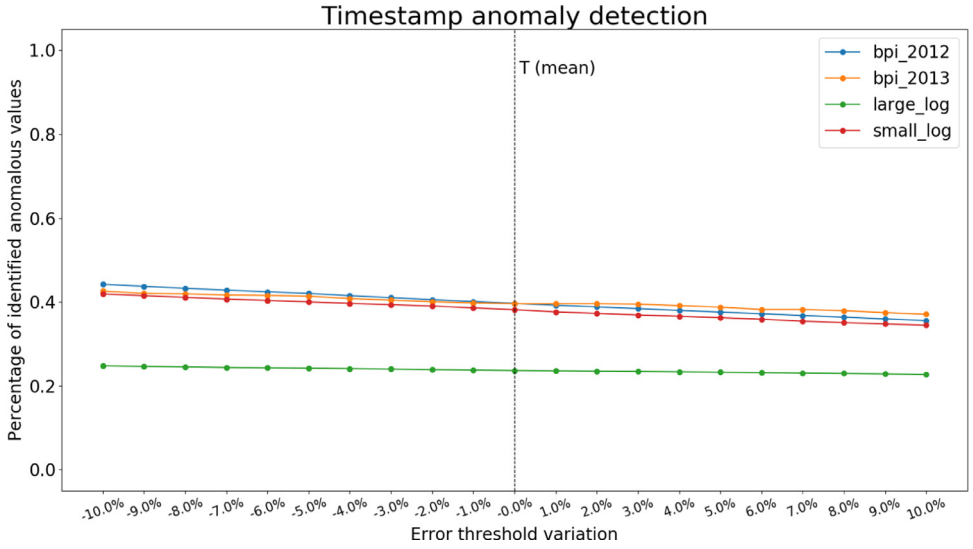


Fig. 8. Sensitivity to threshold variation - timestamp anomaly detection.

value of this threshold is more challenging in the case of the activity label anomaly detection task, which shows to be more sensible to threshold variations.

5.2. Event log reconstruction

Tables 9 and 10 show the time and activity reconstruction results, respectively. The proposed models outperform the baselines in all experiments, Another thing that we have observed during training is that it is difficult to guarantee and accelerate the convergence of training in the small-size dataset compared with the large-size dataset.

Results in Table 9 show that, in relative terms, both VAE, AE and LAE perform better than the baselines BL_{1-4}. The mean and maximum case duration is 8.62 days and 137.22 days, respectively, for the BPI 2012 event log and 178.88 days and 2,254.85 days, respectively, for the BPI 2013 event log. For the small and large event log, the mean case duration are 0.33 and 0.25 days, respectively.

The performance of the proposed method on the BPI 2012 event log is rather stable, with MAE and RMSE not exceeding 13.6% (0.85%) and 27.8% (1.75%) of mean (maximum) case duration, respectively. The performance in respect of the BPI 2013 event log is less stable. The high variability of results for the BPI 2013 event log is due to the distribution of activity durations. Activities in this event log tend to have very short or very long duration, which makes it difficult to learn the time characteristics in the logs. This is also supported by the noticeable gap between mean (BL_1, BL_3) and median (BL_2, BL_4) reconstruction baselines for this event log. Mean values are in fact affected by extreme values or outliers in the dataset, and therefore achieves worse results. On the small log, MAE and RMSE are approximately 9.1% and 12.1% of mean case duration, while these numbers on the large log are 16% and 24%, respectively.

Results in Table 10 show a remarkable efficiency of the proposed method to reconstruct missing activity labels. We can also observe that the performance of reconstructing missing activity labels is more stable compared to missing timestamp reconstruction, even under high levels of information missingness. The model is able to impute missing values efficiently with higher accuracy than baselines from the very first iterations. This may be due to the fact that a sequence of activities in an event log tends to follow a particular pattern determined by the process control flow. This pattern can be learned by our model during the training process, which helps improving the accuracy of activity imputation.

Table 9
Model performance for missing timestamp value reconstruction, measured by Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) in days.

Event log	Missingness	Metric	VAE	AE	LAE	BL_1	BL_2	BL_3	BL_4
BPI 2013	30%	MAE	7.23	6.70	6.69	8.68	14.35	8.97	12.90
		RMSE	13.50	13.32	13.39	16.47	19.40	16.43	18.42
	35%	MAE	7.80	7.34	7.93	15.35	19.88	15.45	18.67
		RMSE	14.38	14.21	14.32	113.84	113.94	113.83	113.85
	40%	MAE	8.52	7.70	7.53	10.14	16.79	10.93	15.34
		RMSE	15.95	13.99	13.95	18.21	23.53	18.42	22.03
	50%	MAE	8.65	8.32	8.42	19.10	26.05	19.68	26.27
		RMSE	14.92	14.73	15.48	139.66	140.02	139.62	140.14
BPI 2012	30%	MAE	0.98	0.77	0.71	1.12	1.46	1.09	1.13
		RMSE	2.12	1.92	1.95	3.80	3.86	3.70	3.68
	35%	MAE	1.07	0.81	1.76	1.15	1.49	1.11	1.13
		RMSE	2.23	1.95	3.01	3.78	3.85	3.66	3.64
	40%	MAE	1.08	0.89	1.12	1.18	1.54	1.13	1.19
		RMSE	2.23	2.10	2.24	3.89	3.97	3.79	3.78
	50%	MAE	1.17	1.06	1.05	1.43	1.85	1.39	1.43
		RMSE	2.40	2.46	2.39	4.29	4.42	4.19	4.18
Small log	30%	MAE	0.02	0.02	0.02	0.06	0.06	0.05	0.05
		RMSE	0.03	0.03	0.03	0.13	0.11	0.11	0.11
	35%	MAE	0.02	0.02	0.02	0.06	0.06	0.05	0.05
		RMSE	0.03	0.03	0.03	0.13	0.12	0.11	0.11
	40%	MAE	0.02	0.02	0.03	0.07	0.06	0.05	0.05
		RMSE	0.04	0.04	0.04	0.08	0.07	0.06	0.07
	50%	MAE	0.03	0.03	0.03	0.13	0.12	0.11	0.11
		RMSE	0.04	0.04	0.05	0.15	0.13	0.14	0.14
Large log	30%	MAE	0.02	0.02	0.02	0.06	0.06	0.06	0.06
		RMSE	0.04	0.04	0.04	0.13	0.12	0.12	0.12
	35%	MAE	0.03	0.03	0.03	0.07	0.07	0.06	0.06
		RMSE	0.04	0.04	0.04	0.13	0.13	0.13	0.13
	40%	MAE	0.03	0.03	0.03	0.07	0.07	0.07	0.07
		RMSE	0.05	0.05	0.05	0.14	0.13	0.11	0.12
	50%	MAE	0.04	0.04	0.04	0.08	0.08	0.08	0.07
		RMSE	0.06	0.06	0.06	0.15	0.14	0.15	0.13

Table 10
Model performance for missing activity label reconstruction, measured by accuracy.

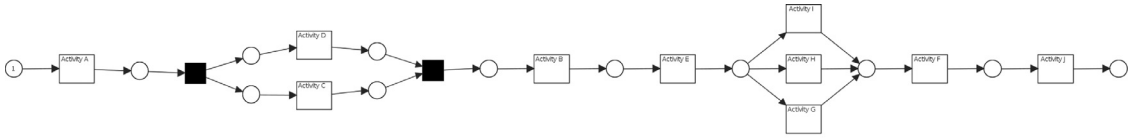
Data	Missingness	VAE	AE	LAE	*BL*
BPI 2013	30%	73.05%	78.57%	74.35%	44.16%
	35%	74.80%	75.33%	73.75%	46.46%
	40%	73.70%	78.76%	76.55%	44.47%
	50%	71.48%	76.33%	72.02%	47.11%
BPI 2012	30%	69.05%	79.19%	48.81%	10.77%
	35%	64.88%	78.69%	27.93%	10.32%
	40%	64.33%	75.92%	31.93%	10.28%
	50%	60.78%	74.90%	36.95%	10.17%
Small log	30%	94.66%	96.36%	61.83%	5.22%
	35%	94.07%	96.90%	67.44%	6.25%
	40%	91.18%	95.41%	66.02%	7.29%
	50%	90.11%	93.74%	65.83%	7.12%
Large log	30%	87.14%	87.69%	83.04%	11.84%
	35%	86.72%	87.02%	82.86%	12.55%
	40%	86.38%	86.84%	82.44%	12.32%
	50%	84.64%	85.20%	82.04%	12.32%

As the number of missing attribute values increases, the performance of the proposed reconstruction models deteriorates only slightly. As remarked before, this may be due to the patterns of sequence of activities. Given enough data in an event log to learn these control flow structures, a trained model can then be easily used to reliably reconstruct missing values. It seems, therefore, that the performance of the model should be evaluated in the future in respect of the complexity of a process model control flow. Note that the results do not vary extensively in the baseline methods using the mean and median values. This is because we only consider completely random missing values, which do not have a significant impact on average values calculated from the dataset.

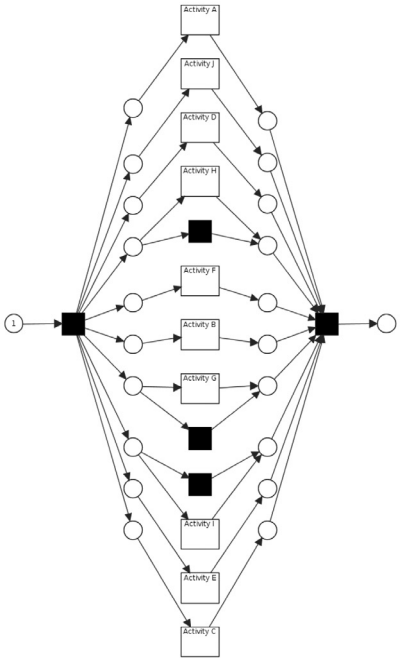
Overall, AE seems to perform better than VAE and LAE in most scenarios. In addition, the latter two models converge slower than AE under the same settings. This should not surprise, since VAE and LAE are by definition better suited to *generative* use cases, i.e., when the objective is to generate new datasets X' with a similar distribution to the input X, whereas AEs suit better the use case of exact reconstruction of the input dataset X, through the steps of encoding and decoding. The problem considered in this paper is more similar to the latter one, since we aim at reconstructing exactly a set of missing values in an event log.

To summarise, the main findings of our experimental evaluation are the following:

- Except in the case of timestamp anomaly detection with real event logs, the proposed framework achieves at least acceptable results detecting anomalies and reconstructing missing values in event logs. Performance is remarkable on artificial event logs, for which attributes values are drawn (and perturbed to achieve anomalies) from well defined distributions;
- The anomaly detection use case is more challenging than the event log reconstruction one. Moreover, the performance of anomaly detection seems to strongly depend on the distribution of values assumed by attributes in an event log. Irregular distribution of attribute values justifies the very poor performance of the proposed framework in timestamp anomaly detection with real event logs;
- Among autoencoder architectures, AE performs better than VAE and LAE. Moreover, being a simpler architecture, the training of AE is quicker than the ones of VAE and LAE. This confirms the better suitability of AE as a *reconstructive* learning model, i.e., a model aiming at reconstructing exactly its own input after having learnt a hidden distribution of input data. VAE and LAE are better *generative* models, i.e., aiming at reconstructing an output *similar* to their input. As such, they are less suitable to fit the use cases considered in this paper, where the objective is to reconstruct exactly the ideal event log cleared of anomalous values.



(a) Process model mined from original (clean) event log



(b) Process model mined from log with injected anomalous values

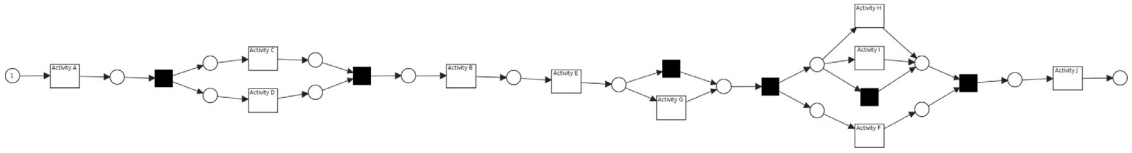


Fig. 9. Process models mined using the large artificial event log.

- Regarding event log reconstruction, the performance deteriorates only slightly with higher levels of missingness of attribute values in an event log. Once, in fact, the model has enough training data to learn control flow relations among activities, missing attributes can be reconstructed effectively. This poses a further challenge to be addressed in the future regarding the study of the relation between the complexity of a process control flow and the level of missingness of data in an event log;
- In both use cases of event log cleaning and reconstruction, the proposed approach performs better than the considered baselines, with the exception of the Cook's distance baseline in timestamp anomaly detection. In this case, however, the performance of the baseline, while being better than the one of the proposed approach, is still far to be acceptable in practical use cases.

5.3. Effects of quality improvement on evidence-based process analysis

While a complete evaluation of event log quality improvement on the outcome of different process analysis use cases is out of the scope of this paper, in this section we present a qualitative evaluation of the effects of attribute-level event log quality improvement on process discovery, that is, one of the main use cases of process mining. Process discovery is particularly significant for the event log quality improvement methods described in this paper, since it relies on activity labels (to identify activities) and timestamps (to detect the order of activities in a case), i.e., the two attributes that we also consider in our approach to anomaly detection and reconstruction.

In the experiments discussed in this section, clean and complete event logs are available and we have run the anomaly detection and reconstruction using the entire event logs as both training and testing set. Logs are perturbed with 10% anomalous values using the same procedure described in the previous section. This enables us to obtain comparable event logs (original, with anomalies and reconstructed) that we can mine to discover process models. Note that considering an entire dataset as both training and testing set is common when using autoencoders, since the objective is for an autoencoder to reconstruct its own input. Each obtained event log is then mined using the inductive miner discovery plugin of ProM [4] with standard settings to obtain a Petri net process model. The obtained model are then visually compared to understand the effects of event log process improvement.

[4] http://www.promtools.org.

From a qualitative standpoint, Fig. 9 shows the process models mined from the original, injected with anomalous values, and reconstructed event logs for the large log. It appears clearly that the effect of anomaly injection is to introduce more variability in an event log, which translates into a process model that allows for more behaviour than the one allowed by an original log. Visually, in fact, the process model mined from the log with anomalies resembles a *flower* WF-net, which allows for all possible execution traces that can be obtained by permuting activity labels in an event log. The process model obtained mining the reconstructed event log is visually similar to the original process model. Most importantly, it allows to execute any trace that can be executed by the original process model. From a quantitative standpoint, we have analysed the distribution of process *variants*, i.e., sets of cases that have been execute following the same trace of activities. In the original large event logs, there are 6 variants, i.e., all traces belong to one of 6 possible different ways to execute the process. Even though the process mined from the reconstructed log has more variants, 95% of the cases in the reconstructed log belong to the 6 most frequent variants, which correspond to the variants of the original process model. In the case of the event log with anomalies, this drops to 40%, that is, the 6 original process variants can replay only 40% of the cases in the log with anomalies. Similar results are obtained for the small artificial event log. As far as real world event logs are concerned, similar results are obtained when only the activity labels are perturbed with anomalous values. The performance of the proposed approach in reconstructing anomalous timestamps, in fact, is too poor for obtaining visually and quantitatively reliable results when anomalous and reconstructed timestamps are used in process discovery.

Based on the preliminary analysis presented in this section, we can conclude that, in many practical scenarios, it would be beneficial for decision makers to base their analyses and decisions on the process models mined from the reconstructed event logs, particularly if compared with the practically useless model obtained by mining the event log in which anomalous values are removed.

6. Conclusions

This paper has presented an innovative approach for improving event log quality using autoencoders, a class of deep neural networks that is suitable to the problem of reconstructing their own input. The proposed method allows to detect abnormal activity labels, e.g., swapped or duplicated, and to reconstruct missing attribute values. It has been evaluated on both artificial and real life event logs. A qualitative evaluation of the impact of event log anomaly detection and reconstruction on process discovery results has also been presented.

This paper has focused on both cleaning and reconstruction of values in event logs at the level of attributes. Previous work has instead focused on identifying anomalous traces in a log and reconstructing missing events in an event log by relying on the existence of a process model. A further main strength of the approach proposed in this paper is that it is fully automated and it does not rely on a-priori knowledge about the process control flow from process stakeholders.

The methods presented in this paper have several limitations. First, the performance obtained using real world event logs in the event log cleaning scenario (anomaly detection) is poor, particularly as far as reconstructing timestamps is concerned. This points towards need for considering and/or integrating other statistical outlier detection techniques into our approach to improve the performance. Also, the distribution of activity durations may play an important role in determining how easy it is to identify anomalies in an event log. As such, event log cleaning methods should be evaluated on many more event logs, characterised by different activity duration distributions. Second, from a theoretical point of view, we argue that not relying on any a-priori knowledge about the process generating data may be a too strong assumption. In many practical scenario, process stakeholders should always have at least a high level understanding of the control flow of processes, and this information should be exploited whenever possible for event log cleaning and reconstruction. In this regard, other approaches in the literature, such as (Bayomie et al., 2015; Dixit et al., 2018) may complement the approach proposed in this paper by providing specific heuristics to detect anomalous values and to incorporate a-priori knowledge about the control flow of a process to improve the event log reconstruction phase. A final limitation of this paper is to consider only randomly introduced anomalous and missing values. While this may capture many practical cases, in many other cases errors in logging are not likely to be random, but they may rather show clear spatial or temporal locality.

Besides what discussed above, this paper can be extended in several ways. First, a more rigorous analysis of the effects of event log reconstruction on evidence-based process analysis should be conducted. In particular, we are currently working on assessing the effect of the methods proposed in this paper on the reliability and quality of process performance indicators calculation.

Second, we are developing a framework to assess and possibly improve the proposed methods to clean and reconstruct event logs in which errors are not introduced randomly, but generated through a set of anomaly and missingness injection patterns based on qualitative studies about data quality in event logs in the literature. For instance, a malfunctioning sensor for a certain period of time may generate bursts of errors in a specific short timeframe, whereas workers not following work practices in logging activities may generate cross-case missing or anomalous values in all activities in which they are involved.

Finally, this work can be extended by considering event log generative models based on autoencoders. Current event log generative techniques are based on detailed process simulation models. These, however, imply the existence of a sound process model, which may be unavailable in case of low quality event logs or *spaghetti* process models. The techniques presented in this paper can be extended to generate new event logs starting from fragments of existing event logs. These can then be used to refine process analysis, e.g., discovering better process models or simulating the values of process performance indicators. Additional generative machine learning techniques, such as GAN (Generative Adversarial Networks) and restricted Boltzmann machines, may be used to solve the same problem considered in this paper or to achieve event log generative models.

Credit author statement

Hoang Thi Cam Nguyen is the first author. She contributed the main idea and initial design of the framework proposed in this paper and a large part of the evaluation section.

Suhwan Lee has contributed extensively to design and implementation of experiments, in particular the comparison with baseline models.

Jongchan Kim has contributed extensively to design and implementation of experiments, in particular the comparison with baseline models.

Jonghyeon Ko has contributed extensively to design and implementation of experiments, in particular the comparison with baseline models.

Marco Comuzzi is the corresponding author. He has overseen the development of the framework and experiments. He also personally developed the part concerning the impact on the process discovery use case.

Acknowledgement

This work has been partially funded by NRF Korea project number 2017076589.

Conflicts of interest

The authors declare that no conflicts of interest exist.

References

van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., et al. (2012). Process mining manifesto. In F. Daniel, K. Barkaoui, & S. Dustdar (Eds.), *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I* (pp. 169–194). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-28108-2 19.

van der Aalst, W., & de Medeiros, A. (2005). Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science, 121*(Supplement C), 3–21. doi:10.1016/j.entcs.2004.10.013. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics, 2*(4), 433–459.

Attaran, M. (2004). Exploring the relationship between information technology and business process reengineering. *Information & Management, 41*(5), 585–596.

Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR), 41*(3), 16.

Bayomie, D., Helal, I. M., Awad, A., Ezat, E., & ElBastawissi, A. (2015). Deducing case ids for unlabeled event logs. In *Business process management workshops* (pp. 242–254). Springer.

Beaulieu-Jones, B. K., & Moore, J. H. (2017). Missing data imputation in the electronic health record using deeply learned autoencoders. In *Pacific symposium on biocomputing 2017* (pp. 207–218). World Scientific.

Bezerra, F., & Wainer, J. (2013). Algorithms for anomaly detection of traces in logs of process aware information systems. *Information Systems, 38*(1), 33–44.

Böhmer, K., & Rinderle-Ma, S. (2017). Multi instance anomaly detection in business process executions. In *International conference on business process management* (pp. 77–93). Springer.

Bose, R. J. C., Mans, R. S., & van der Aalst, W. M. (2013). Wanna improve process mining results? In *2013 IEEE Symposium on computational intelligence and data mining* (pp. 127–134). IEEE.

Che, Z., Purushotham, S., Cho, K., Sontag, D., & Liu, Y. (2016). Recurrent neural networks for multivariate time series with missing values. arXiv:1606.01865.

Cheng, H.-J., & Kumar, A. (2015). Process mining on noisy logs-can log sanitization help to improve performance? *Decision Support Systems, 79*, 138–149.

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014a). On the properties of neural machine translation: encoder-decoder approaches. *CoRR, abs/1409.1259*.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., & Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR, abs/1406.1078*.

Chomboon, K., Chujai, P., Teerarassamee, P., Kerdprasop, K., & Kerdprasop, N. (2015). An empirical study of distance metrics for k-nearest neighbor algorithm. In *Proceedings of the 3rd international conference on industrial application engineering*.

Cook, R. D. (1977). Detection of influential observation in linear regression. *Technometrics, 19*(1), 15–18.

Dixit, P. M., Suriadi, S., Andrews, R., Wynn, M. T., ter Hofstede, A. H., Buijs, J. C., & van der Aalst, W. M. (2018). Detection and interactive repair of event ordering imperfection in process logs. In *International conference on advanced information systems engineering* (pp. 274–290). Springer.

Doersch, C. (2016). Tutorial on variational autoencoders. arXiv:1606.05908.

van Eck, M. L., Lu, X., Leemans, S. J., & van der Aalst, W. M. (2015). *PM²*: A process mining project methodology. In *International conference on advanced information systems engineering* (pp. 297–313). Springer.

Genga, L., Alizadeh, M., Potena, D., Diamantini, C., & Zannone, N. (2018). Discovering anomalous frequent patterns from partially ordered event logs. *Journal of Intelligent Information Systems*, 1–44.

Ghionna, L., Greco, G., Guzzo, A., & Pontieri, L. (2008). Outlier detection techniques for process mining applications. In *Proceedings of the 17th international conference on foundations of intelligent systems* (pp. 150–159). Berlin, Heidelberg: Springer-Verlag. ISMIS'08.

Gondara, L., Wang, K. (2017). Multiple imputation using deep denoising autoencoders. arXiv:1705.02737.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR, abs/1412.6980*.

Kingma, D. P. Welling, M.(2013). Auto-encoding variational bayes. http://arxiv.org/abs/1412.6980.

Mans, R. S., van der Aalst, W. M., Vanwersch, R. J., & Moleman, A. J. (2013). Process mining in healthcare: Data challenges when answering frequently posed questions. In *Process support and knowledge representation in health care* (pp. 140–153). Springer.

Marquez-Chamorro, A. E., Resinas, M., & Ruiz-Cortés, A. (2017). Predictive monitoring of business processes: A survey. *IEEE Transactions on Services Computing*, (1). 1–1.

Meroni, G., Baresi, L., Montali, M., & Plebani, P. (2018). Multi-party business process compliance monitoring through IOT-enabled artifacts. *Information Systems, 73*, 61–78.

Müller, H., & Freytag, J. (2005). Problems, methods, and challenges in comprehensive data cleansing. *Informatik-Berichte // Institut für Informatik, Humboldt Universität zu Berlin*. Humboldt-Univ. zu Berlin.

Nolle, T., Luettgen, S., Seeliger, A., & Mühlhäuser, M. (2018). Analyzing business process anomalies using autoencoders. *Machine Learning, 107*, 1875–1893.

Nolle, T., Seeliger, A., & Mühlhäuser, M. (2016). Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In *International conference on discovery science* (pp. 442–456). Springer.

Oliveira, P., Rodrigues, F., & Henriques, P. R. (2005). A formal definition of data quality problems. . In F. Naumann, M. Gertz, & S. E. Madnick (Eds.), *Iq*. MIT.

Rahm, E., & Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin, 23*(4), 3–13.

Rogge-Solti, A., Mans, R. S., van der Aalst, W. M., & Weske, M. (2013a). Improving documentation by repairing event logs. In *Ifip working conference on the practice of enterprise modeling* (pp. 129–144). Springer.

Rogge-Solti, A., Mans, R. S., van der Aalst, W. M., & Weske, M. (2013b). Repairing event logs using timed process models. In *OTM confederated international conferences" on the move to meaningful internet systems"* (pp. 705–708). Springer.

Sakurada, M., & Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proc. 2nd workshop on machine learning for sensory data analysis* (pp. 4–11). doi:10.1145/2689746.2689747. MLSDA'14.

Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., & Hemingway, H. (2014). Comparison of random forest and parametric imputation models for imputing missing data using mice: A caliber study. *American Journal of Epidemiology, 179*(6), 764–774. doi:10.1093/aje/kwt312.

Socher, R., Huang, E. H., Pennin, J., Manning, C. D., & Ng, A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems* (pp. 801–809).

Suriadi, S., Andrews, R., ter Hofstede, A. H., & Wynn, M. T. (2017). Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. *Information Systems, 64*, 132–150.

Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive business process monitoring with LSTM neural networks. In *International conference on advanced information systems engineering* (pp. 477–492). Springer.

Wang, J., Song, S., Zhu, X., & Lin, X. (2013). Efficient recovery of missing events. *Proceedings of the VLDB Endowment, 6*(10), 841–852.