Mach Learn (2018) 107:1875–1893
https://doi.org/10.1007/s10994-018-5702-8

Analyzing business process anomalies using autoencoders

Timo Nolle[1] · **Stefan Luettgen**[1] ·
Alexander Seeliger[1] · **Max Mühlhäuser**[1]

Received: 31 March 2017 / Accepted: 15 March 2018 / Published online: 27 April 2018

Abstract Businesses are naturally interested in detecting anomalies in their internal processes, because these can be indicators for fraud and inefficiencies. Within the domain of business intelligence, classic anomaly detection is not very frequently researched. In this paper, we propose a method, using autoencoders, for detecting and analyzing anomalies occurring in the execution of a business process. Our method does not rely on any prior knowledge about the process and can be trained on a noisy dataset already containing the anomalies. We demonstrate its effectiveness by evaluating it on 700 different datasets and testing its performance against three state-of-the-art anomaly detection methods. This paper is an extension of our previous work from 2016 (Nolle et al. in Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In: International conference on discovery science, Springer, pp 442–456, 2016). Compared to the original publication we have further refined the approach in terms of performance and conducted an elaborate evaluation on more sophisticated datasets including real-life event logs from the Business Process Intelligence Challenges of 2012 and 2017. In our experiments our approach reached an F_1 score of 0.87, whereas the best unaltered state-of-the-art approach reached an F_1 score of 0.72. Furthermore, our approach can be used to analyze the detected anomalies in terms of which event within one execution of the process causes the anomaly.

Editors: Toon Calders and Michelangelo Ceci.

✉ Timo Nolle
nolle@tk.tu-darmstadt.de

Stefan Luettgen
stefan.luettgen@stud.tu-darmstadt.de

Alexander Seeliger
seeliger@tk.tu-darmstadt.de

Max Mühlhäuser
max@tk.tu-darmstadt.de

[1] Telecooperation Lab, Technische Universität Darmstadt, Darmstadt, Germany

Keywords Deep learning · Autoencoder · Anomaly detection · Process mining · Business intelligence

1 Introduction

Anomaly detection is becoming an integral part of business intelligence. Businesses are naturally interested in detecting anomalies in their processes, because these can be indicators for inefficiencies in their process, badly trained employees, or even fraudulent behavior. Consequently, being able to detect such anomalies is of great value, for they can have enormous impact on the economic well-being of the businesses.

More and more companies rely on process-aware information systems (PAISs) (Dumas et al. 2005) to improve their processes. This increasing number of PAISs has generated a lot of interest in the data these systems are gathering. The log files these systems are storing can be used to extract the events executed in the process, and thereby create so called event log files. Event logs are comprised of activities (and other miscellaneous information) that occurred during the execution of the process. These event logs enable process analysts to explore the underlying process. In other words, the event log consists of footprints of the process. Consequently, it is possible to recreate the process model by evaluating its event log. This is known as *process model discovery* and is one of the main ideas in the domain of process mining (Van der Aalst et al. 2011).

Process mining provides methodologies to detect anomalies in the execution of a process; e.g., by discovering the as-is process model from the event log (Van der Aalst et al. 2004) using discovery algorithms and then comparing the discovered model to a reference model. This is known as *conformance checking* (Rozinat and Van der Aalst 2008). Another way of detecting anomalies is to compare the event log to the reference model. However, this approach requires the existence of such a reference model.

If no reference model is available, process mining relies on discovering a reference model from the event log itself (Bezerra et al. 2009; Bezerra and Wainer 2013). These methods make use of a threshold to deal with infrequent behavior in the log, so that the discovered model is a good representation of the normal behavior of the process. Hence, this model can be used as a reference model for the conformance check.

A key assumption in anomaly detection is that the anomalous executions occur less frequent than normal executions. This skewed distribution can be taken advantage of when applying anomaly detection techniques.

In this paper, we propose a method for detecting anomalies in business process data. Our method works under the following assumptions.

- No prior knowledge about the process
- Training data already contains anomalies
- No reference model needed
- No labels needed (i.e., no knowledge about anomalies)
- The algorithm must detect the exact activity at which the anomaly occurred

The system must deduce the difference between normal and anomalous executions purely based on the patterns in the raw data. Our approach is based on a special type of neural network, called an autoencoder, that is trained in an unsupervised fashion.

The main contribution of this work is the application of an autoencoder to analyze the detected anomalies in terms of which event within a sequence is anomalous as opposed to the whole sequence at once. This can be refined further by analyzing which characteristic of the event (e.g., the executing user) is anomalous, not just the event itself. We demonstrate that, using this approach, we can accurately identify activities that have been executed in the wrong order, skipped, or unnecessarily reworked. Furthermore, we can detect when unauthorized users have illegally executed an activity.

To demonstrate the feasibility of our approach we compare its performance to seven state-of-the-art methods for anomaly detection. In addition to these six methods, we also present an adaptation of one of the methods. All methods were applied to a comprehensive set of 600 different artificial event logs featuring authentic business process anomalies as well as 100 real-life event logs coming from the Business Process Intelligence Challenge (BPIC).

In summary, the contributions of this paper are as follows.

1 Novel application of autoencoders to automatically analyze anomalies in the domain of business process intelligence.
2 Adaptation of the t-STIDE anomaly detection method from Warrender et al. (1999) to work with event attributes.
3 Comprehensive evaluation of state-of-the-art anomaly detection methods in the domain of business process intelligence.
4 Provision of a representative, labelled, set of artificial process event logs containing authentic anomalies.

2 Related work

In the field of process mining (Van der Aalst et al. 2011) anomaly detection is not very frequently researched. Most proposed methods work by using discovery algorithms to mine a reference model from the event log (Bezerra et al. 2009) and then using it for conformance checking to detect anomalous behavior. The bigger part of these methods relies on a clean dataset to work correctly. Unfortunately, this violates our assumptions, as the data coming from the PAISs will naturally contain anomalies.

Recently there has been some research on approaches that can deal with noisy event logs. Through the use of special discovery algorithms, that can deal with noise and infrequent behavior in the process, the approach from Bezerra et al. (2009) can be refined to work with noisy logs (Bezerra and Wainer 2013). The authors in Bezerra and Wainer (2013) give three different algorithms in their paper. Within this work we will compare our approach to two of the proposed approaches.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior (Böhmer and Rinderle-Ma 2016). Specifically, the authors describe a method to extend the likelihood graph to include event attributes. This method works both on noisy event logs and includes important characteristics of the process itself by including the event attributes. We will also compare our method to the method from Böhmer and Rinderle-Ma (2016) in the evaluation section.

A review of classic anomaly detection methodology can be found in Pimentel et al. (2014). Here, the authors describe and compare many methods that have been proposed over the last decades. Another elaborate summary on anomaly detection in discrete sequences is given by Chandola et al. (2012). The authors differentiate between five different basic methods for novelty detection: probabilistic, distance-based, reconstruction-based, domain-based, and information-theoretic novelty detection.

Probabilistic approaches try to estimate the probability distribution of the normal class, and thus can detect anomalies as they were sampled from a different distribution. In speech recognition (Juang and Rabiner 1991), hidden Markov models (HMMs) (Rabiner and Juang 1986; Rabiner 1989) are a popular choice for modeling sequential data. HMMs can also be used for anomaly detection as shown in Warrender et al. (1999) and Jain and Abouzakhar (2012), where they are used successfully for system intrusion detection. However, as Chandola et al. pointed out in 2008, the performance of such HMMs strongly depends on the fact that the raw data can be sufficiently modeled by a Markov process.

Another important probabilistic technique is the sliding window approach as proposed in Forrest et al. (1996), where it is used for intrusion detection. In window based anomaly detection, every window of a sequence is assigned an anomaly score. Then the anomaly score of the sequence can be inferred by aggregating the window anomaly scores. Recently, Wressnegger et al. (2013) used this approach for intrusion detection and give an elaborate evaluation. While being inexpensive and easy to implement, sliding window approaches show a robust performance in finding anomalies in sequential data, especially within short regions of the data (Chandola et al. 2012).

Distance-based novelty detection does not require a cleaned dataset, yet it is only partly applicable for process traces, as anomalous traces are usually very similar to normal ones. A popular distance-based approach is the one-class support vector machine (OC-SVM). Schölkopf et al. (1999) first used support vector machines (Cortes and Vapnik 1995) for anomaly detection. Tax, in his PhD thesis (Tax 2001), gives a sophisticated overview over one-class classification methods, also mentioning the OC-SVM. OC-SVMs have shown to be successful in the field of intrusion detection as demonstrated by Heller et al. (2003).

Reconstruction-based novelty detection (e.g., neural networks) is similar to the aforementioned approaches in Hawkins et al. (2002) and Japkowicz (2001). However, training a neural network usually also requires a cleaned dataset. Nevertheless, we will show that our approach works on the noisy dataset by taking advantage of the skewed distribution of normal data and anomalies, as demonstrated in Eskin (2000).

Domain-based novelty detection requires domain knowledge, which violates our assumption of no prior knowledge about the process. Information-theoretic novelty detection defines anomalies as the examples that most influence an information measure (e.g., entropy) on the whole dataset. Iteratively removing the data with the highest impact will yield a cleaned dataset, and thus a set of anomalies.

The approach within this paper is highly influenced by the works in Dong and Japkowicz (2016), Hawkins et al. (2002) and Japkowicz (2001), in which they propose the use of replicator neural networks (Hecht-Nielsen 1995) for anomaly detection, i.e., networks that reproduce their input, which are based on the idea of autoencoders from Hinton (1989). Autoassociative neural network encoders use a similar concept and have been used to model the nominal behavior of complex systems (Thompson et al. 2002). They have also been used for residual generation in Diaz and Hollmén (2002), demonstrating that these models can also model behavior not directly observed in the training data, which increases generalization. A comprehensive study of replicator neural networks for outlier detection can be found in Williams et al. (2002). The approaches from Diaz and Hollmén (2002), Dong and Japkowicz (2016), Hawkins et al. (2002), Japkowicz (2001) and Thompson et al. (2002), however, do not work well with variable length input. In our approach, we address this problem by using a padding technique. We opted to use a neural network based approach, for recent achievements in machine translation and natural language processing indicate that neural networks are an excellent choice when modeling sequential data (Dai and Le 2015; LeCun et al. 2015).

The main distinction between all other methods and the proposed approach is that it can be used to identify which exact event and furthermore which attribute characteristic is the cause of the anomaly. The only other approach that can deal with event attributes is

Table 1 Example event log of a procurement process

Trace ID	Timestamp	Activity	User
1	2015-03-21 12:38:39	PR created	Roy
1	2015-03-28 07:09:26	PR released	Earl
1	2015-04-07 22:36:15	PO created	James
1	2015-04-08 22:12:08	PO released	Roy
1	2015-04-21 16:59:49	Goods receipt	Ryan
2	2015-05-14 11:31:53	SC created	Marilyn
2	2015-05-21 09:21:26	SC purchased	Emily
2	2015-05-28 18:48:27	SC approved	Roy
2	2015-06-01 04:43:08	PO created	Johnny

the method from Böhmer and Rinderle-Ma (2016). However, it can not deal with long-term dependencies, because it works on a general likelihood graph, which disregards the past events when calculating the probability of an event occurring at a specific point in the process. Our approach can deal both with the attributes and with non-local dependencies in the logs.

3 Dataset

PAISs keep a record of almost everything that happened during the execution of a business process. This information can be extracted from the systems in form of an event log. Event logs are the most common data structure when working with process data from PAISs, especially in the field of process mining.

3.1 Event logs

An event log consists of traces, each consisting of the activities that have been executed. Table 1 shows an excerpt of such an event log. In this case, it is representative for the execution of a procurement process. Notice that an event log must consist of at least three columns: a trace ID, to uniquely assign an executed activity to a trace; a timestamp, to order the activities within a trace; and an activity label, to distinguish the different activities. Optionally, the event log can contain so called event attributes. In the example event log from Table 1, the user column is such an event attribute, indicating which user has executed the respective activity.

3.2 Process model generation

To create a test setting for our approach we randomly generated process models and then sampled event logs from them. The process models were generated using PLG2 (Burattin 2015), a process simulation and randomization tool. Each process model has a different complexity, with regard to the number of possible activities and the branching factors (i.e., out-degrees). The complexity of a process model can also be measured by the number of possible variants. A variant is a valid path through the complete process model from a valid start activity to a valid end activity. Table 2 shows the process models with their corresponding complexities. Note that the *Wide* process model was specifically generated to evaluate the

Table 2 Overview over the 5 different randomly generated process models and the P2P process

Model	#Nodes	#Edges	#Variants	Max length	∅ Out-degree
P2P	14	16	6	9	1.14
Small	22	26	6	10	1.18
Medium	34	48	25	8	1.41
Large	44	56	28	12	1.27
Huge	56	75	39	11	1.34
Wide	36	53	19	7	1.47

approach on a dataset that has low complexity in terms of the number of variants, but a high branching factor.

Now, we generated authentic event logs from these process models by randomly sampling variants of the process with replacement. In real process models these variants are not equally distributed. Therefore, we randomly generated a distribution for the variants each time we were sampling an event log. These probabilities were sampled from a normal distribution with $\mu = 1$ and $\sigma = 0.2$, and then normalized so they sum up to 1. Furthermore, we randomly generated a set of users in the process (between 10 and 30 different users per process). Then we sampled subsets of the user set for each activity, denoting which users are permitted to execute the activity. The number of possible users per activity lies between 1 and 5. After computing all variants, we also introduced a long-term dependency for the user variable in each variant at random. Therefore, we randomly chose two activities in each variant that must be executed by the same user.

3.3 Example process

In addition to the five randomly generated models, we also used a simplified version of a purchase to pay (P2P) process model as is depicted by the BPMN model in Fig. 1. This model was mainly created for purposes of evaluation, as it features interpretable activity names unlike the randomly generated models. The resulting event log for the P2P model was generated in the same fashion as those of the randomly generated models using the same parameters as mentioned above. Notice the possible users for each activity as indicated by the italic names in Fig. 1.

3.4 Anomalies

To introduce noise into the event logs we randomly applied mutations to a fixed percentage of the traces in the event log. These mutations represent the anomalies in the data. Each trace can be affected by one of the following five anomalies (we will use their respective names from now on):

1. *Skipping* A necessary activity has not been executed,
2. *Switching* Two consecutive events have been executed in the wrong order,
3. *Reworking* An activity has been executed twice in a row,
4. *Incorrect user* A user has executed an activity to which he was not permitted,
5. *Incorrect LTD* The wrong user has executed the long-term dependent activity.

Compared to our work in Nolle et al. (2016), we have added two more anomalies that we found occur very frequently in real-life scenarios. A classic problem in real-life business

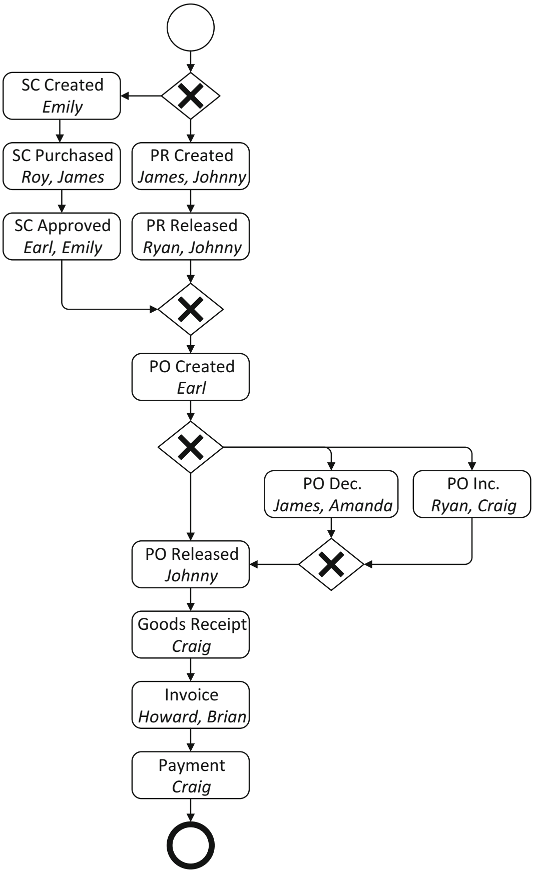


Fig. 1 BPMN model of a simplified purchase to pay process; the italic names represent the users allowed to execute that activity

processes is the segregation of duty. For example, a user that approves a purchase order must not be the same user that has initially created it. Many anomalies in real-life processes are related to the users executing the events, which is why we included this event attribute here.

Our way of generating the artificial event logs is very similar to the methods of Bezerra and Wainer (2013) and Böhmer and Rinderle-Ma (2016). One difference is, that we also introduce anomalies affecting event attributes. We will make these datasets, the generation algorithm, and our implementation of the algorithm publicly available. For more information on this, please consider contacting the corresponding author.

For each process model, we randomly generated a set of permitted users for each activity. We did this ten times, resulting in 60 different process models. For each of these 60 process models, we then generated 10 event logs, each featuring a different percentage of anomalies and a random variant distribution. The percentage of anomalous traces in the training log ranged from 10, 20, up to 100%. That is, we generated training logs containing 10% anomalies and 90% normal traces, as well as logs with 80% anomalies and 20% normal traces, and so on up to a log which entirely consists of anomalies, i.e., 100%. In total, we generated 600 different artificial event logs. Each event log consisted of 12,500 traces. For each event log we created a separate test event log containing 2500 traces featuring the same variant distribution and users.

3.5 Real-life event logs

In addition to the artificial event logs we also generated training and test event logs from the public datasets of the Business Process Intelligence Challenge 2012[1] and 2017,[2] which we will refer to as BPIC12 and BPIC17 respectively. BPIC17 is an updated version of BPIC12, representing the same loan application process. However, BPIC17 contains data from the last 5 years, after the company has introduced a new workflow system.

Similarly to the artificial logs, we used the event logs as a basis and randomly applied anomalies to a fixed percentage of traces in the logs. As these logs did not feature a user attribute we did not include the *Incorrect user* and *Incorrect LTD* anomalies. For BPIC12 and BPIC17 we generated training sets featuring between 10 and 100% anomalies, as was done for the artificial logs. We also generated separate test sets for both logs, resulting in 100 real-life training event logs with artificial anomalies.

4 Method

Recently, artificial neural networks have gotten a lot of attention by outclassing the state-of-the-art methods in many domains such as object recognition in images (Krizhevsky et al. 2012) or machine translation (Bahdanau et al. 2014). Before we introduce our method, we first want to give a brief overview over the neural network architecture we employed.

A feed-forward neural network consists of multiple layers, each containing many neurons. Every neuron in one layer is connected to all neurons in the preceding and succeeding layers. These connections have weights attached to them, which can be used to control the impact a neuron in one layer has on the activation of a neuron in the next layer. To calculate the output of a neuron we apply a non-linear activation function [a popular choice is the rectifier function $f(x) = \max(0, x)$ (Nair and Hinton 2010) to the sum over all outputs of the neurons in the previous layer times their respective connection weights. The initialization of these weights is important, as pointed out in Glorot and Bengio (2010), for no two weights within one layer must be initialized to the same value. Then, the back-propagation algorithm (Rumelhart et al. 1988) is used to iteratively tune the weights, so that the neural network produces the desired output, or a close enough approximation of it.

In a classification setting, the desired output of the neural network is the class label. However, a neural network can also be trained without the use of class labels. One such type of neural network is called an autoencoder. Instead of using class labels, we are using the original input as the target output when training the autoencoder. Obviously, a neural network, if given enough capacity and time, can simply learn the identity function of all examples in the training set. To overcome this issue, some kind of capacity limitation is needed. This can be done by forcing one of the autoencoder's hidden layers to be narrow (i.e., narrower than the input dimension), thereby not allowing the autoencoder to learn the identity function. Another common way of limiting the capacity is to distribute additive Gaussian noise over the input vector of the autoencoder. Thus, the autoencoder—even if repeatedly trained on the same trace—will always receive a different input. We use a combination of both these strategies for our method.

[1] http://www.win.tue.nl/bpi/doku.php?id=2012:challenge.

[2] http://www.win.tue.nl/bpi/doku.php?id=2017:challenge.

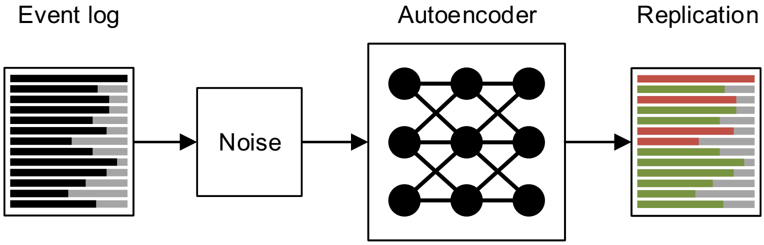


Fig. 2 Autoencoder is trained to replicate the traces in the event log after the addition of Gaussian noise

4.1 Setup

To train an autoencoder on the generated event logs, we first must transform them. The first step is to encode each activity and user using a one-hot encoding. Each activity is encoded by an n-dimensional vector, where n is the number of different activities encountered in the event log. To encode one activity, we simply set the corresponding dimension of the one-hot vector to a fixed value of one, while setting all the other dimensions to zero. We use the same method to encode the user event attribute. This results in a one-hot vector for the activity and another for the user for each event in a trace. Now we combine these vectors by concatenating them into one vector. If the activity vectors are a_1, a_2, ..., a_n and the respective user vectors are u_1, u_2, ..., u_n, the resulting vector will be $a_1 \| u_1 \| a_2 \| u_2 \| ... \| a_n \| u_n$, where $\|$ denotes concatenation.

Note that another option of dealing with variable size traces is dividing the traces into subsequences of equal size (n-grams). However, using n-grams of events loses the connection between distant events, if the n-gram size is too narrow. Consequently, the system is unable accurately model long-term dependencies between events. Therefore, we chose to use the one-hot encoding method.

Because feed-forward neural networks have a fixed size input, we must apply one more step of pre-processing. To force all encoded trace vectors to have the same size we pad all vectors with zeros, so each vector has the same size as the longest vector (i.e., the longest trace) in the event log.

Suppose an event log consists of 10 different activities, 20 different users, and the maximum length of all traces in the event log is 12. The longest trace within the event log will have a size of $(10 + 20) \cdot 12 = 360$. Therefore, we must pad all shorter vectors with zeros so they reach size 360.

Using the one-hot encoded event log we can train the autoencoder with the backpropagation algorithm (Rumelhart et al. 1988), using the event log both as the input and the label. Figure 2 shows a simplified version of the architecture. The special noise layer adds Gaussian noise before feeding the input into the autoencoder. This layer is only active during training. Now the autoencoder is trained to reproduce its input, that is, to minimize the mean squared error between the input and its output.

We trained on mini batches of size 50 for 200 epochs, allowing early stopping when the loss on the validation set did not decrease within the last 10 epochs. We used the Adam optimizer (Kingma and Ba 2014), which utilizes the momentum technique (Sutskever et al. 2013). We set the optimizer parameters to $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$. The learning rate was set to 0.001 initially, and was scaled by a factor of 0.1 when the validation loss did not improve within the last 5 epochs. Additionally, we used a dropout of 0.5 between all layers, as suggested in Srivastava et al. (2014); the additive noise applied to the input was sampled from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.1$. Each autoencoder consists of an input and an output layer with linear units, and 2 hidden layers with rectified linear units. These training parameters were used for each of the different event logs, but the size of the hidden layer was adapted depending on the event log, i.e., the number of neurons in the hidden layer was set to be half the size of the input layer. For the real-life BPIC event logs we only used 1 hidden layer.

4.2 Classifying traces

After training the autoencoder, it can be used to reproduce the traces in the test event logs, but without applying the noise. Now, we can measure the mean squared error between the input vector and the output vector to detect anomalies in the event log. Because the distribution of normal traces and anomalous traces in the event log is one sided, we can assume that the autoencoder will reproduce the normal traces with less reproduction error than the anomalies. Therefore, we can define a threshold τ, where if the reproduction error of a trace succeeds this threshold τ, we consider it an anomaly. To set the threshold we use the mean reproduction error over the training dataset and apply a scaling factor α. We define the threshold as in Eq. 1, where e_i is the reproduction error for trace i, and n the number of traces in the dataset.

$$\tau = \frac{\alpha}{n} \sum_{i=1}^{n} e_i \tag{1}$$

4.3 Classifying events and attributes

We have described how to detect anomalous traces in the event log; now we want to refine this method. Not only can we detect that a trace is anomalous, but also which event in the trace influences the reproduction error the most. Hence, we must change our calculation of the reproduction error from trace based to event based. Up until now, we calculated the reproduction error as the mean squared error between the entire one-hot encoded input and output sequence of the autoencoder. However, we can also consider the mean squared error for every event in the sequence separately. Furthermore, we can also compute the error for each activity and user separately.

Let us consider the example input vector i from Eq. 2. We can divide the vector into the corresponding subvectors, as indicated by the curly braces. This gives us $a_1, u_1, a_2, u_2, ..., a_n, u_n$. Now we can split the reproduced version of i (i.e., the output vector) identically, obtaining $\hat{a}_1, \hat{u}_1, \hat{a}_2, \hat{u}_2, ..., \hat{a}_3, \hat{u}_3$.

$$i = [\ \underbrace{00001}_{a_1}\ \underbrace{0100}_{u_1}\ \underbrace{10000}_{a_2}\ \underbrace{0010}_{u_2}\ ...\ \underbrace{01000}_{a_n}\ \underbrace{0100}_{u_n}\ ] \tag{2}$$

The error E for an activity vector a_i is then given by the mean squared error between a_i and $\hat{a}_i$. For a user vector u_i the method works analogously. Thus, we can compute the error for all activity vectors and all user vectors over the whole dataset. Notice that this works for any number of event attributes.

The benefit is that we can distinguish between activity related anomalies and user related anomalies. We will elaborate on this in the evaluation section below.

5 Evaluation

We evaluated the autoencoder approach (DAE) on all 700 event logs and compared it to state-of-the-art anomaly detection methods mentioned (Chandola et al. 2012). Namely: a sliding window approach named t-STIDE (Warrender et al. 1999); the one-class SVM approach (OC-SVM); and the Markovian approach using a hidden Markov model (HMM) (Warrender et al. 1999). In addition to that, we also compared our approach to two approaches proposed in Bezerra and Wainer (2013), the Naive algorithm and the Sampling algorithm. Lastly, we compared our approach to the most recent approach proposed in Böhmer and Rinderle-Ma (2016), using an extended likelihood graph (Likelihood). As a baseline we provide the results of a random classifier.

For the OC-SVM we relied on the implementation of the scikit-learn package for Python (Pedregosa et al. 2011) using an RBF kernel of degree 3 and a $\nu = 0.6$. The HMM approach was implemented using the hmmlearn package for Python. We implemented the t-STIDE algorithm ourselves using a window size $k = 4$. The hyperparameters for both approaches were optimized using grid search. The Naive, Sampling, and Likelihood methods were implemented as described in the original papers.

At last, we used our own implementation of the t-STIDE method which we will refer to as t-STIDE+. The classic t-STIDE approach only takes into account the activities of an event log, but not the attributes. To make use of the attributes we must adapt the original method.

A window of size k is a tuple of k events, where each event consists of a tuple of the activity name a and the corresponding user u. Let us consider an example window size of three. A window w is defined as $w = \{(a_1, u_1), (a_2, u_2), (a_3, u_3)\}$. The approach works by employing a frequency analysis over all windows of size k in the training set, and then comparing the relative frequencies of all windows in the test set to the corresponding ones from the training set. Whenever a window's relative frequency is significantly lower than its frequency in the training set, the trace containing this window is considered an anomaly. We evaluated the t-STIDE and the t-STIDE+ approach on all datasets and for all feasible choices of k, i.e., k was chosen to lie between 2 and the maximum trace length in the dataset. The evaluation showed that $k = 4$ performed the best for both approaches.

We used the threshold technique from Eq. 1 for all approaches except the OC-SVM, for the scikit-learn implementation automatically optimizes the threshold. For the other approaches, we optimized the scaling factor α by an exhaustive grid search. One requirement when setting α was that α must be the same for all event logs, i.e., we strive for a general setting of α.

We also considered isolation forests (Liu et al. 2008, 2012) for our experiments; however, this approach relies on setting a contamination parameter indicating the noise level of the data, rendering the approach unusable as we assume no prior knowledge about the noise level.

We evaluated the 9 methods on all 600 artificial, as well as the 100 real-life event logs. In total, we evaluated 6300 models.

5.1 Experiment results

Figure 3 shows the F_1 score of all methods for each process model. The F_1 score per model was calculated using the macro average for each model. Then all F_1 scores were averaged over all models for the corresponding process model. A more detailed evaluation is given in Tables 3 and 4, which show the F_1 scores and their standard deviation for each process model, best results being shown in bold typeset. Notice that the DAE approach performs best in all settings, closely followed by t-STIDE+, whereas the other approaches perform significantly

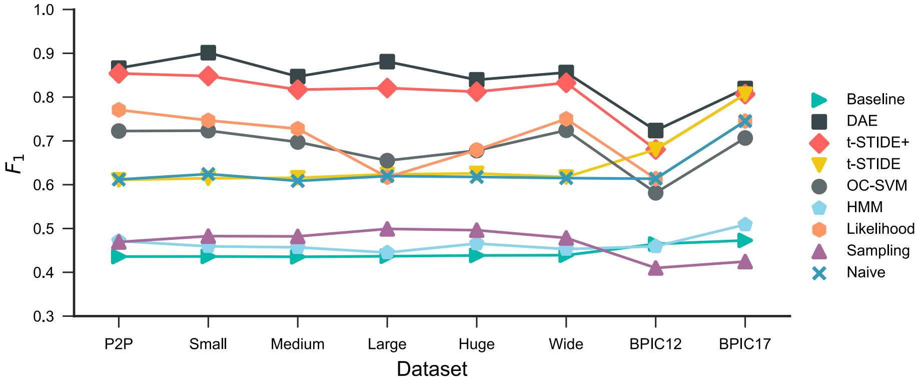


Fig. 3 F_1 score by process model and method

Table 3 Results of the experiments for all evaluated methods for each process model

	P2P	Small	Medium	Large	Huge	Wide
Baseline	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01
HMM (Warrender et al. 1999)	0.47 ± 0.02	0.46 ± 0.01	0.46 ± 0.01	0.45 ± 0.02	0.47 ± 0.01	0.45 ± 0.02
OC-SVM (Schölkopf et al. 1999)	0.72 ± 0.06	0.72 ± 0.05	0.70 ± 0.05	0.65 ± 0.04	0.68 ± 0.05	0.72 ± 0.05
Naive (Bezerra and Wainer 2013)	0.61 ± 0.01	0.62 ± 0.01	0.61 ± 0.03	0.62 ± 0.01	0.62 ± 0.02	0.62 ± 0.02
Sampling (Bezerra and Wainer 2013)	0.47 ± 0.03	0.48 ± 0.05	0.48 ± 0.06	0.50 ± 0.06	0.50 ± 0.06	0.48 ± 0.05
t-STIDE (Warrender et al. 1999)	0.61 ± 0.01	0.61 ± 0.03	0.62 ± 0.02	0.62 ± 0.01	0.63 ± 0.01	0.62 ± 0.02
Likelihood (Böhmer and Rinderle-Ma 2016)	0.77 ± 0.17	0.75 ± 0.14	0.73 ± 0.15	0.62 ± 0.10	0.68 ± 0.11	0.75 ± 0.17
t-STIDE+	0.85 ± 0.09	0.85 ± 0.08	0.82 ± 0.09	0.82 ± 0.07	0.81 ± 0.05	0.83 ± 0.11
DAE	**0.87 ± 0.09**	**0.90 ± 0.07**	**0.85 ± 0.08**	**0.88 ± 0.07**	**0.84 ± 0.08**	**0.86 ± 0.08**

Best results are shown in bold typeface

worse. Another interesting point is that the HMM approach performs no better than the random baseline, which supports Chandola's claim that HMMs are not a good method for anomaly detection in sequential data (Chandola et al. 2008). Also the sampling approach performs only slightly better than chance. However, this is due to the fact that we average all results over all training sets including training event logs with higher share of anomalies. In Fig. 4 we can see that the Sampling method works only for low noise levels.

Overall, we can conclude that the DAE performs better than the state-of-the-art methods in all of our test settings.

5.2 The impact of the noise level

As described before, we used different noise levels when generating the datasets, by generating training sets which included between 10 and 100% anomalous traces. We use the word

Table 4 Results on the BPIC event logs

	BPIC12	BPIC17
Baseline	0.46 ± 0.01	0.47 ± 0.01
HMM (Warrender et al. 1999)	0.46 ± 0.00	0.51 ± 0.00
OC-SVM (Schölkopf et al. 1999)	0.58 ± 0.07	0.71 ± 0.04
Naive (Bezerra and Wainer 2013)	0.61 ± 0.12	0.75 ± 0.12
Sampling (Bezerra and Wainer 2013)	0.41 ± 0.00	0.42 ± 0.00
t-STIDE (Warrender et al. 1999)	0.68 ± 0.14	0.81 ± 0.02
Likelihood (Böhmer and Rinderle-Ma 2016)	0.61 ± 0.12	0.75 ± 0.12
t-STIDE+	0.68 ± 0.14	0.81 ± 0.02
DAE	**0.72** ± **0.08**	**0.82** ± **0.05**

Best results are shown in bold typeface

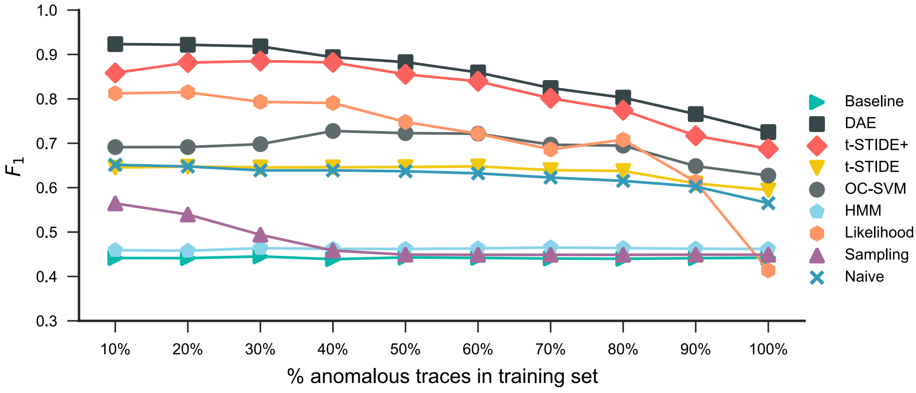
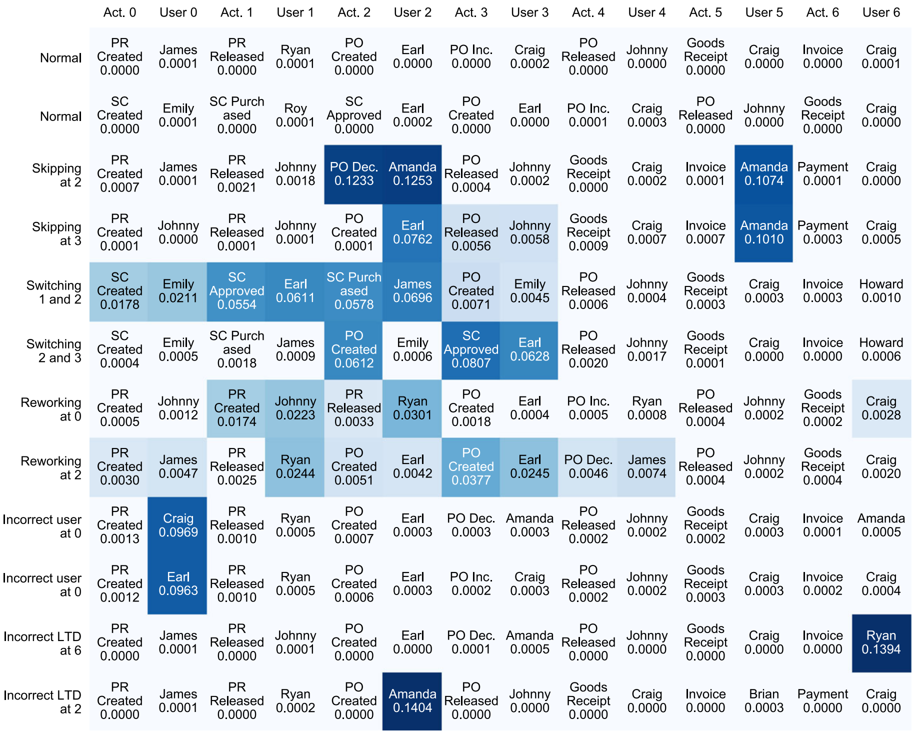


Fig. 4 F_1 score by percentage of anomalous traces in the training set

noise to refer to the share of traces in the training set which are anomalous. Notice that an anomalous trace still contains normal subsequences of events. Only a small part of the trace is affected by the anomaly in our test settings. Hence, there is still normal behavior present in parts of each trace, even when each trace has been affected by an anomaly, as in the 100% case. We specifically included these harsh noise levels to test the different approaches on their ability to generalize. We want to point out, however, that noise levels greater than 50% are extremely unlikely in real-world settings.

One can also argue that a noise level greater than 50% is illogical, because the classification task just gets inverted; hence, the anomaly class becomes the normal class. This is not true for the same reason as before. As we are dealing with sequential data and many different events in sequence (i.e., a trace) are assigned one label, there are still events that carry information about the normal behavior of the process. And in most cases the normal events in an anomalous trace, still overpower the anomalous ones. Hence, a noise level of 60% is not the same as a noise level of 40% with classes inverted.

Figure 4 shows the F_1 score for all methods for the different noise levels. Again, we find that the DAE outperforms the other approach at all noise levels, again closely followed by t-STIDE+.



	Act. 0	User 0	Act. 1	User 1	Act. 2	User 2	Act. 3	User 3	Act. 4	User 4	Act. 5	User 5	Act. 6	User 6
Normal	PR Created 0.0000	James 0.0001	PR Released 0.0000	Ryan 0.0001	PO Created 0.0000	Earl 0.0000	PO Inc. 0.0000	Craig 0.0002	PO Released 0.0000	Johnny 0.0000	Goods Receipt 0.0000	Craig 0.0000	Invoice 0.0000	Craig 0.0001
Normal	SC Created 0.0000	Emily 0.0001	SC Purch ased 0.0000	Roy 0.0001	SC Approved 0.0000	Earl 0.0002	PO Created 0.0000	Earl 0.0000	PO Inc. 0.0001	Craig 0.0003	PO Released 0.0000	Johnny 0.0000	Goods Receipt 0.0000	Craig 0.0000
Skipping at 2	PR Created 0.0007	James 0.0001	PR Released 0.0021	Johnny 0.0018	PO Dec. 0.1233	Amanda 0.1253	PO Released 0.0004	Johnny 0.0002	Goods Receipt 0.0000	Craig 0.0002	Invoice 0.0001	Amanda 0.1074	Payment 0.0001	Craig 0.0000
Skipping at 3	PR Created 0.0001	Johnny 0.0000	PR Released 0.0001	Johnny 0.0001	PO Created 0.0001	Earl 0.0762	PO Released 0.0056	Johnny 0.0058	Goods Receipt 0.0009	Craig 0.0007	Invoice 0.0007	Amanda 0.1010	Payment 0.0003	Craig 0.0005
Switching 1 and 2	SC Created 0.0178	Emily 0.0211	SC Approved 0.0554	Earl 0.0611	SC Purch ased 0.0578	James 0.0696	PO Created 0.0071	Emily 0.0045	PO Released 0.0006	Johnny 0.0004	Goods Receipt 0.0003	Craig 0.0003	Invoice 0.0003	Howard 0.0010
Switching 2 and 3	SC Created 0.0004	Emily 0.0005	SC Purch ased 0.0018	James 0.0009	PO Created 0.0612	Emily 0.0006	SC Approved 0.0807	Earl 0.0628	PO Released 0.0020	Johnny 0.0017	Goods Receipt 0.0001	Craig 0.0000	Invoice 0.0000	Howard 0.0006
Reworking at 0	PR Created 0.0005	Johnny 0.0012	PR Created 0.0174	Johnny 0.0223	PR Released 0.0033	Ryan 0.0301	PO Created 0.0018	Earl 0.0004	PO Inc. 0.0005	Ryan 0.0008	PO Released 0.0004	Johnny 0.0002	Goods Receipt 0.0002	Craig 0.0028
Reworking at 2	PR Created 0.0030	James 0.0047	PR Released 0.0025	Ryan 0.0244	PO Created 0.0051	Earl 0.0042	PO Created 0.0377	Earl 0.0245	PO Dec. 0.0046	James 0.0074	PO Released 0.0004	Johnny 0.0002	Goods Receipt 0.0004	Craig 0.0020
Incorrect user at 0	PR Created 0.0013	Craig 0.0969	PR Released 0.0010	Ryan 0.0005	PO Created 0.0007	Earl 0.0003	PO Dec. 0.0003	Amanda 0.0003	PO Released 0.0002	Johnny 0.0002	Goods Receipt 0.0002	Craig 0.0003	Invoice 0.0001	Amanda 0.0005
Incorrect user at 0	PR Created 0.0012	Earl 0.0963	PR Released 0.0010	Ryan 0.0005	PO Created 0.0006	Earl 0.0003	PO Inc. 0.0002	Craig 0.0003	PO Released 0.0002	Johnny 0.0002	Goods Receipt 0.0003	Craig 0.0003	Invoice 0.0002	Craig 0.0004
Incorrect LTD at 6	PR Created 0.0000	James 0.0001	PR Released 0.0000	Johnny 0.0001	PO Created 0.0000	Earl 0.0000	PO Dec. 0.0001	Amanda 0.0005	PO Released 0.0000	Johnny 0.0000	Goods Receipt 0.0000	Craig 0.0000	Invoice 0.0000	Ryan 0.1394
Incorrect LTD at 2	PR Created 0.0000	James 0.0001	PR Released 0.0000	Ryan 0.0002	PO Created 0.0000	Amanda 0.1404	PO Released 0.0000	Johnny 0.0000	Goods Receipt 0.0000	Craig 0.0000	Invoice 0.0000	Brian 0.0003	Payment 0.0000	Craig 0.0000

Fig. 5 DAE error heatmap, trained on a P2P event log with 10% anomalous traces

Notice that the DAE still performs remarkably well, even when trained on the 100% training set. This is due to its ability to generalize over multiple traces. The t-STIDE approaches can also generalize over multiple traces, because they classify based on windows; and the windows itself can contain a completely valid sequence of events. These approaches can learn what a normal trace ought to look like, by combining the knowledge they gathered of normal subsequences over multiple traces. For the t-STIDE approaches this is obvious, as the window size is usually smaller than the trace is long; hence, it is only trained on subsequences in the first place. The DAE, on the other hand, is trained on the whole trace at once, which makes this level of generalization much more remarkable and unique among all the approaches.

5.3 Interpreting the anomalies

An interesting feature of the DAE approach is that it can be used to detect not only anomalous traces, but also which event or which event attribute has influenced the reproduction error the most. This can be done by computing the reproduction error for each event attribute separately, as described earlier. Figure 5 shows 12 example traces of the P2P test dataset for a DAE trained on a training set with 10% anomalous traces. For clarity, we only show the first 6 events omitting the remaining events. The cells are colored according to the reproduction error; the higher the error the darker the color.

As we can clearly see, it is never the whole trace that leads to a high reproduction error. The DAE succeeds to reproduce the normal parts of the traces quite well, whereas it fails to reproduce the anomalous parts. For example, the first two *Normal* traces are reproduced with almost no error at all, which is exactly what we expected. Let us now look at the four examples at the bottom (*Incorrect user* and *Incorrect LTD*). The DAE is remarkably good at detecting incorrect users. Neither Craig, nor Earl, are permitted to execute the activity *PR Created* (cf. Fig. 1). Detecting *Incorrect LTD* works just as fine.

Moving to the three anomalies from the original paper, we want to recall one problem that we have observed during the evaluation in Nolle et al. (2016). Whenever an activity is skipped or reworked, the remaining subsequence is shifted by one to the left, or the right respectively. In Nolle et al. (2016) this led to the effect that all activities after the initial skipped (or reworked) activity had high reproduction error. This phenomenon does not occur as severely in the extended approach, but it is still noticeable. We assume that the additional hidden layers provide enough abstraction so the DAE can adapt to this problem.

Overall, we can see that the approach is very precise in narrowing down the exact cause of the anomaly. In fact, this approach can be used to perform an automatic root cause analysis on the detected anomalies, without the need of an extra processing step. Most other anomaly detection algorithms can only be used to divide the normal examples from the anomalies, but then an additional algorithm has to be used to, for instance, cluster the anomalies. Another important point about this is that it allows to follow what the DAE has learned as well as to interpret it. Usually, not being interpretable is a notorious problem for neural network based approaches. Not in this case.

5.4 Discussion

At last we want to point out some interesting observations. You might notice that in the two *Skipping* examples the user at index 5, Amanda, produces a high reproduction error. This is due to the fact that this event has a long-term dependency to an earlier event. In the first case the event is connected to the *PO Decreased* event, in the second one the connected event is the event that has been skipped. Now that the trace has, in part, been shifted due to the skipping, the original event from index 6 is now at index 5. Essentially, we have detected a fluke anomaly, that was not supposed to be there, yet the DAE approach has found it, demonstrating the feasibility of the approach.

This indicates, as already mentioned in Nolle et al. (2016), that the DAE is sensitive towards the actual position of an event within a trace, which also becomes apparent in the second *Skipping* example. The event *PO Created* is wrong, yet the DAE reproduces it correctly. This is due to the fact the *PO Created* can be correct here when the trace starts with the *SC Created* event.

Furthermore, we still observe some cross-talk between adjacent events. If we inspect the first *Switching* example, we notice that *SC Approved* and *SC Purchased* have been switched, as correctly identified by the DAE. However, the first event also produces a high reproduction error, albeit being correct at that location. This error, compared to the error at indices 2 and 3, is significantly lower.

Table 5 provides the average F_1 score of the approaches when classifying traces, events, and attributes respectively. When classifying attributes we classify the activity and the user separately, whereas when classifying events we do not separate the attributes. Therefore, we also produced labels indicating anomalous and normal event attributes, when generating the datasets. Any event attribute that had not been affected by any of the anomalies, has been labeled normal, whereas all other attributes have been labeled as anomalous. An event is an anomalous when any of its attributes is anomalous and similarly a trace is anomalous when

Table 5 Results of the experiments for the anomalous event classifier per label and process model

Resolution	Method	Average	Normal	Anomaly
Traces	Baseline	0.44 ± 0.01	0.25 ± 0.01	0.62 ± 0.01
	HMM (Warrender et al. 1999)	0.46 ± 0.02	0.17 ± 0.06	0.75 ± 0.05
	OC-SVM (Schölkopf et al. 1999)	0.70 ± 0.06	0.51 ± 0.09	0.89 ± 0.05
	Naive (Bezerra and Wainer 2013)	0.62 ± 0.02	0.49 ± 0.02	0.74 ± 0.02
	Sampling (Bezerra and Wainer 2013)	0.48 ± 0.05	0.11 ± 0.18	0.86 ± 0.09
	t-STIDE (Warrender et al. 1999)	0.62 ± 0.02	0.49 ± 0.02	0.74 ± 0.02
	Likelihood (Böhmer and Rinderle-Ma 2016)	0.72 ± 0.15	0.52 ± 0.26	0.91 ± 0.07
	t-STIDE+	0.83 ± 0.08	0.73 ± 0.12	0.93 ± 0.05
	DAE	**0.87 ± 0.08**	**0.78 ± 0.13**	**0.95 ± 0.03**
Events	Sampling (Bezerra and Wainer 2013)	0.44 ± 0.17	0.64 ± 0.20	0.25 ± 0.14
	t-STIDE (Warrender et al. 1999)	0.66 ± 0.03	0.90 ± 0.02	0.42 ± 0.04
	t-STIDE+	0.61 ± 0.03	0.86 ± 0.03	0.36 ± 0.05
	DAE	**0.72 ± 0.02**	**0.92 ± 0.02**	**0.53 ± 0.04**
Attributes	t-STIDE+	0.59 ± 0.03	0.87 ± 0.03	0.31 ± 0.06
	DAE	**0.70 ± 0.03**	**0.94 ± 0.01**	**0.47 ± 0.05**

Best results are shown in bold typeface

any of its events is anomalous. Consequently, we can now calculate the performance of the DAE based on single events.

Sampling, t-STIDE, and t-STIDE+ can all also naturally be used to classify events. Apart from DAE, only t-STIDE+, due to our adaption of the algorithm, can also be used to classify single attributes. This can be done, for instance, by assigning the specific window anomaly scores to the respective last event or attribute in the window. Sampling relies on a conformance check, which per definition gives a per event resolution. Table 5 shows that the DAE outperforms the other approaches in all three categories.

We can conclude that this approach can discover the special characteristics of anomalies in an otherwise unknown process, while still being able to correctly identify normal behavior. All together, we can say that the DAE approach is the most versatile out of all the approaches, as it works well in all of our test settings.

6 Conclusion

We have presented a novel application of denoising autoencoders to detect anomalies in business process data. Our approach does not rely on any prior knowledge about the process itself. Also, we do not rely on a clean dataset for the training; our approach is trained on a noisy dataset already containing the anomalies. Furthermore, we have demonstrated that the autoencoder can also be used to easily identify the anomalous event(s) or event attribute(s), making results interpretable with regards to why an anomaly has been classified as such. Even though we showed that this approach works for business process data, it can be applied just as easily to other domains with discrete sequential data.

We conducted a comprehensive evaluation using representative artificial and real-life event logs. These event logs featured a range of different anomalies, different complexities in terms of the process model, variable variant probabilities, random user sets for each activity, and different shares of anomalous traces, ranging from 10 to 100%. We compared the autoencoder approach to 7 other state-of-the-art anomaly detection methods, as described in Chandola et al. (2012), Bezerra and Wainer (2013), Böhmer and Rinderle-Ma (2016) and Warrender et al. (1999), showing that our approach outperforms all other methods in all of the test settings, reaching an F_1 score of 0.87 on average, whereas the second-best approach, our own adaption of the t-STIDE algorithm reached 0.83. The next best unaltered anomaly detection algorithm, using an extended likelihood graph, reached an F_1 score of 0.72. To our knowledge, this is the most sophisticated evaluation and comparison of anomaly detection methodology within the domain of process intelligence to date.

The biggest advantage of the autoencoder approach over the other methods is that it allows to analyze the detected anomalies even further. Computing the anomaly score for each event attribute individually, the approach indicates the anomalous attribute very convincingly. To our knowledge, this method of analyzing the anomalies is novel to the field of discovery science, as well as business intelligence and process mining.

The presented approach is an extended version of the approach from Nolle et al. (2016). In the original paper, we postulated that the approach is susceptible to anomalous behavior in the event log that is very frequent. However, by showing that the approach works well for all noise levels, especially the higher noise levels where the exact same anomaly can occur many times, we have shown this not to be the case. We also showed, by using skewed variant distributions, that the autoencoder is robust towards process models with unequally distributed variants, that is, some variants (i.e., one valid path through the process model) are more likely than others. By including the user as an event attribute, we demonstrated that more dimensions can be added easily to the approach, without a significant loss of accuracy.

As an inspiration for future work on the matter we want to give a few remarks. Note that for the DAE approach to work in a real-time setting the trace length of all future traces must be conform with the input size of the neural network. If traces surpass the input size, they cannot be fed into the autoencoder. There are some strategies to compensate for this problem. For example, the autoencoder can be set up with spare padding input units. Instead of padding all traces to match the maximum length encountered in the training set, we pad all traces to an arbitrary length greater than the maximum length. If we do want to reuse the already trained autoencoder, we can use another strategy. Every trace that is too long to feed into the autoencoder is divided into subsequences of exactly the size of the input. For example, if an autoencoder has input size 10 and a trace has size 12, we would first feed the sequence starting from the first event until the tenth event, then the sequence from the second until the eleventh, and so on. Then we can average the anomaly scores over all subsequences. Another solution to the problem is the use of recurrent neural networks, which can be used to consume sequences of arbitrary length.

Another problem arises if one of the attributes is set to a value not encountered during training. Consequently, there will be no dimension allocated in the one-hot encoding for it. A simple solution to this problem is to add one extra dimension to the encoding vector which is used to encode all unknown attribute characteristics. Nevertheless, the autoencoder should be retrained regularly to counteract concept drift.

With t-STIDE+ and DAE we have presented two approaches to detect anomalies in business process data. It is quite costly to train a neural network on big datasets, because the dataset needs to be iterated many times. The t-STIDE+ approach has the advantage that it can be trained with just iteration.

However, due to the nature of the algorithm, it has some drawbacks; it cannot capture long-term dependencies if the window size is too small and if the windows size is too big the accuracy decreases. Furthermore, it is not trivial to assign an anomaly score to a single attribute of an event, because anomaly scores are based on windows. Lastly, it cannot deal with numerical event attributes (e.g., prices), without resorting to binning or grouping, which is not obvious.

The DAE approach does not have these drawbacks. Numerical data can easily be modelled by using a single linear input and output neuron for real-valued numbers. Certainly, it does require more training time, but with the introduction of evermore powerful GPUs and lately TPUs the trade-off between accuracy and efficiency is not as severe.

Overall, the results presented in this paper suggest that a denoising autoencoder is a reliable and versatile method for detecting—and interpreting—anomalies in unknown business processes.

Acknowledgements This project (HA Project No. 522/17-04) is funded in the framework of Hessen Modell-Projekte, financed with funds of LOEWE Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbund-vorhaben (State Offensive for the Development of Scientific and Economic Excellence).

References

Bahdanau, D., Cho, K.,& Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

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

Bezerra, F., Wainer, J., & Van der Aalst, W. M. P. (2009). Anomaly detection using process mining. In T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, & R. Ukor (Eds.), *Enterprise, business-process and information systems modeling* (pp. 149–161). Berlin, Heidelberg: Springer.

Böhmer, K., & Rinderle-Ma, S. (2016). Multi-perspective anomaly detection in business process execution events. In: OTM confederated international conferences On the move to meaningful internet systems (pp. 80–98). Springer.

Burattin, A. (2015). PLG2.: Multiperspective processes randomization and simulation for online and offline settings. CoRR abs/1506.0.

Chandola, V., Mithal, V., & Kumar, V. (2008). Comparative evaluation of anomaly detection techniques for sequence data. In: 2008 Eighth IEEE international conference on data mining (pp. 743–748).

Chandola, V., Banerjee, A., & Kumar, V. (2012). Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *24*(5), 823–839.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Dai, A. M.,& Le, Q. V. (2015). Semi-supervised sequence learning. In: Advances in neural information processing systems (pp. 3079–3087).

Diaz, I., & Hollmén, J. (2002). Residual generation and visualization for understanding novel process conditions. In: Neural networks, 2002. international joint conference on IJCNN'02. Proceedings of the 2002 (vol. 3, pp. 2070–2075). IEEE.

Dong, Y.,& Japkowicz, N. (2016). Threaded ensembles of supervised and unsupervised neural networks for stream learning. In: Canadian conference on artificial intelligence (pp. 304–315). Springer.

Dumas, M., Van der Aalst, W. M., & Ter Hofstede, A. H. (2005). *Process-aware information systems: Bridging people and software through process technology*. Hoboken: Wiley.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In: In Proceedings of the international conference on machine learning. Citeseer.

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for unix processes. In: EEE symposium on security and privacy. Proceedings, 1996 (pp. 120–128). IEEE.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Aistats*, *9*, 249–256.

Hawkins, S., He, H., Williams, G., & Baxter, R. (2002). Outlier detection using replicator neural networks. In: Data warehousing and knowledge discovery (pp. 170–180). Springer.

Hecht-Nielsen, R. (1995). Replicator neural networks for universal optimal source coding. *Science*, *269*(5232), 1861.

Heller, K. A., Svore, K. M., Keromytis, A. D., & Stolfo, S.J.(2003). One class support vector machines for detecting anomalous windows registry accesses. In: In Proceedings of the workshop on data mining for computer security.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, *40*(1), 185–234.

Jain, R.,& Abouzakhar, N. S. (2012). Hidden markov model based anomaly intrusion detection. In: 2012 International conference for internet technology and secured transactions (pp. 528–533).

Japkowicz, N. (2001). Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, *42*(1), 97–122.

Juang, B. H., & Rabiner, L. R. (1991). Hidden markov models for speech recognition. *Technometrics*, *33*(3), 251–272.

Kingma, D.,& Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems (pp. 1097–1105).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Liu, F. T., Ting, K. M.,& Zhou, Z. H.(2008). Isolation forest. In: Eighth IEEE international conference on data mining, 2008. ICDM'08 (pp. 413–422). IEEE.

Liu, F. T., Ting, K. M., & Zhou, Z. H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *6*(1), 3.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10) (pp. 807–814).

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.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pimentel, M. A., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, *99*, 215–249.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Rabiner, L., & Juang, B. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, *3*(1), 4–16.

Rozinat, A., & Van der Aalst, W. M. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems*, *33*(1), 64–95.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., Platt, J. C., et al. (1999). Support vector method for novelty detection. *NIPS*, *12*, 582–588.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G.(2013). On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th international conference on machine learning (ICML-13) (pp. 1139–1147).

Tax, D. M. J. (2001). One-class classification: Concept learning in the absence of counter-examples. Ph.D. thesis, Technische Universiteit Delft.

Thompson, B. B., Marks, R. J., Choi, J. J., El-Sharkawi, M. A., Huang, & M. Y., Bunje, C. (2002). Implicit learning in autoencoder novelty assessment. In: Neural networks, 2002. Proceedings of the 2002 international joint conference on IJCNN'02 (vol. 3, pp. 2878–2883). IEEE.

Van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, *16*(9), 1128–1142.

Van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., et al. (2011). Process mining manifesto. In: Business process management workshops (pp. 169–194). Springer.

Warrender, C., Forrest, S., & Pearlmutter, B.(1999). Detecting intrusions using system calls: Alternative data models. In: IEEE symposium on proceedings of the security and privacy, 1999 (pp. 133–145). IEEE.

Williams, G., Baxter, R., He, H., Hawkins, S., & Gu, L. (2002). A comparative study of rnn for outlier detection in data mining. In: IEEE international conference on data mining, 2002. ICDM 2003. Proceedings. 2002 (pp. 709–712). IEEE.

Wressnegger, C., Schwenk, G., Arp, D., & Rieck, K.(2013). A close look on n-grams in intrusion detection: Anomaly detection vs. classification. In: Proceedings of the 2013 ACM workshop on artificial intelligence and security, AISec '13 (pp. 67–76). ACM.