BINet: Multivariate Business Process Anomaly Detection Using Deep Learning

Timo Nolle(✉), Alexander Seeliger, and Max Mühlhäuser

Telecooperation Lab, Technische Universität Darmstadt, Darmstadt, Germany
{nolle,seeliger,max}@tk.tu-darmstadt.de

Abstract. In this paper, we propose BINet, a neural network architecture for real-time multivariate anomaly detection in business process event logs. BINet has been designed to handle both the control flow and the data perspective of a business process. Additionally, we propose a heuristic for setting the threshold of an anomaly detection algorithm automatically. We demonstrate that BINet can be used to detect anomalies in event logs not only on a case level, but also on event attribute level. We compare BINet to 6 other state-of-the-art anomaly detection algorithms and evaluate their performance on an elaborate data corpus of 60 synthetic and 21 real life event logs using artificial anomalies. BINet reached an average F_1 score over all detection levels of 0.83, whereas the next best approach, a denoising autoencoder, reached only 0.74. This F_1 score is calculated over two different levels of detection, namely case and attribute level. BINet reached 0.84 on case and 0.82 on attribute level, whereas the next best approach reached 0.78 and 0.71 respectively.

Keywords: Business process management · Anomaly detection
Artificial process intelligence · Deep learning
Recurrent neural networks

1 Introduction

Anomaly detection is an important topic for today's businesses because its application areas are so manifold. Fraud detection, intrusion detection, and outlier detection are only a few examples. However, anomaly detection can also be applied to business process executions, for example to clean up datasets for more robust predictive analytics and robotic process automation (RPA). Especially in RPA, anomaly detection is an integral part because the robotic agents must recognize tasks they are unable to execute to not halt the process. Naturally, businesses are interested in anomalies within their processes, as these can be indicators for inefficiencies, insufficiently trained employees, or even fraudulent activities. Consequently, being able to detect such anomalies is of great value, for they can have an enormous impact on the economic well-being of the business.

M. Weske et al. (Eds.): BPM 2018, LNCS 11080, pp. 271–287, 2018.
https://doi.org/10.1007/978-3-319-98648-7_16

In today's digital world, companies rely more and more on process-aware information systems (PAISs) to accelerate their processes. A byproduct of such PAISs is an enormous data base that often remains unused. The log files these systems are storing can be used to extract valuable information about a process. One key data structure is an event log, which contains information about what activities have been executed in a process, who executed it, at which time, etc. These event logs are a great source of information and are frequently used for different data mining techniques, such as process mining.

In this paper, we propose BINet (Business Intelligence Network), a novel neural network architecture that allows to detect anomalies on attribute level. Often, the actual cause of an anomaly is only captured by the value of a single attribute. For example, a user has executed an activity without permission. This anomaly is only represented by the user attribute of exactly this event. Anomaly detection algorithms must work on the lowest (attribute) level, to provide the greatest benefit. BINet has been designed to process both the control flow (sequence of activities) and the data flow (see [1]).

Due to the nature of the architecture of BINet it can be used for ex-post analysis, but can also be deployed in a real-time setting to detect anomalies at runtime. Being able to detect anomalies at runtime is important because otherwise no counter measures can be undertaken in time. BINet can be trained during the execution of the process and therefore can adapt to concept drift. If unseen attribute values occur during the training, the network can be altered and retrained on the historic data to include the new attribute value in the future. Dealing with concept drift is also important as most business processes are flexible systems. BINet is a recurrent neural network architecture and therefore can detect point anomalies as well as contextual anomalies (see [12]). BINet works under the following assumptions.

- No domain knowledge about the process
- No clean dataset (i.e., dataset contains anomalous examples)
- No reference model
- No labels (i.e., no knowledge about anomalies)

In the context of business processes an anomaly is defined as a deviation from a defined behavior, i.e., the business process. An anomaly is an event that does not typically occur as a consequence of preceding events, specifically their order and combination of attributes. Anomalies that are attributed to the order of activities (e.g., two activities are executed in the wrong order) are called control flow anomalies. Anomalies that are attributed to the attributes (e.g., a user that is not part of a certain security group has illicitly executed an event) of events are called data flow anomalies.

Many anomaly detection algorithms rely on the manual setting of a threshold value to determine anomalies. We propose an unsupervised method for automatically setting the threshold using a heuristic.

We compare BINet to 6 state-of-the art anomaly detection methods and evaluate on a comprehensive dataset of 60 synthetic logs and 20 real-life logs, using artificial anomalies. This work contains four main contributions.

1. BINet neural network architecture[1]
2. Automatic threshold heuristic
3. Comprehensive evaluation of state-of-the-art methods.

2 Related Work

In the field of process mining [1], it is popular to use discovery algorithms to mine a process model from an event log and then use conformance checking to detect anomalous behavior [2,4,27]. However, the proposed methods do not utilize the event attributes, and therefore cannot be used to detect anomalies on attribute level.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior [5]. Specifically, the authors describe a method to extend the likelihood graph to include event attributes. This method works on noisy event logs and includes important characteristics of the process itself by including the event attributes. A drawback of this method is that the attributes are checked in a specific order, thereby introducing a bias towards certain attributes.

A review of classic anomaly detection methodology can be found in [22]. 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 in [7]. 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 estimate the probability distribution of the normal class, and thus can detect anomalies as they come from a different distribution. An important probabilistic technique is the sliding window approach [26]. In window-based anomaly detection, an anomaly score is assigned to each window in a sequence. Then the anomaly score of the sequence can be inferred by aggregating the window anomaly scores. Recently, Wressnegger et al. used this approach for intrusion detection and gave an elaborate evaluation in [28]. While being inexpensive and easy to implement, sliding window approaches show a robust performance in finding anomalies in sequential data, especially within short regions [7].

Distance-based novelty detection does not require a clean dataset, yet it is only partly applicable for process cases, as anomalous cases are usually very similar to normal ones. A popular distance-based approach is the one-class support vector machine (OC-SVM). Schölkopf et al. [23] first used support vector machines [9] for anomaly detection.

Reconstruction-based novelty detection (e.g., neural networks) is based on the idea to train a model that can reconstruct normal behavior but will fail to do so with anomalous behavior. Therefore, the reconstruction error can be

[1] https://github.com/tnolle/binet.

used to detect anomalies [15]. This approach has successfully been used for the detection of control flow anomalies [21] as well as data flow anomalies [20] in event logs of PAISs.

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

The core of BINet is a recurrent neural network, trained to predict the next event and its attributes. The architecture is influenced by the works of Evermann [10,11] and Tax [24], who utilized long short-term memory [13] (LSTM) networks for next event prediction, demonstrating their utility. LSTMs have been used for anomaly detection in different contexts like acoustic novelty detection [18] and predictive maintenance [17]. These applications mainly focus on the detection of anomalies in time series and not, like BINet, on multivariate anomaly detection in discrete sequences of events.

The novelty of BINet lies in the tailored architecture for business processes, including the control and data flow, the scoring system to assign anomaly scores, and the automatic threshold heuristic.

3 Datasets

As a basis for the understanding of the following sections, we first need to define the terms case, event, log, and attribute. A log consists of cases, each of which consists of events executed within a process. Each event is defined by an activity name and its attributes, e.g., a user who executed the event. We use a nomenclature adapted from [1].

Definition 1. *Case, Event, Log, Attribute. Let $\mathcal{C}$ be the set of all cases and $\mathcal{E}$ be the set of all events. The event sequence of a case $c \in \mathcal{C}$, denoted by $\hat{c}$, is defined as $\hat{c} \in \mathcal{E}^*$, where $\mathcal{E}^*$ is the set of all sequences over $\mathcal{E}$. An event log is a set of cases $\mathcal{L} \subseteq \mathcal{C}$. Let $\mathcal{A}$ be a set of attributes and $\mathcal{V}$ be a set of attribute values, where $\mathcal{V}_a$ is the set of possible values for the attribute $a \in \mathcal{A}$. Note that $|\hat{c}|$ is the number of events in case c, $|\mathcal{L}|$ is the number of cases in $\mathcal{L}$, and $|\mathcal{A}|$ is the number of event attributes.*

To evaluate our method, we generated synthetic event logs from random process models of different complexities. We used PLG2 [6] to generate five process models: Small, Medium, Large, Huge, and Wide. The complexity of the models varies in number of activities, breadth, and width; for Small to Huge, activities, breadth, and width increase uniformly, whereas Wide features a much larger breadth than width. Wide was designed as a challenge because it features a high branching factor, thereby making it hard to predict the next activity or attribute. We also use a handmade procurement process model called P2P for demonstrative purposes because it features human readable activity names.

Now, we randomly generate logs from these process models, following the control flow and generating attributes for each event. Each possible sequence of activities was assigned a random probability sampled from a normal distribution with $\mu = 1$ and $\sigma = 0.2$, so that not all sequences appear equally likely.

To generate the attributes, we first create a set of possible values $\mathcal{V}_a$ for each attribute a, with set sizes ranging from 20 to 100. Then we assign random subsets of $\mathcal{V}_a$ to each activity, ranging from 5 to 40 in size. When generating a sequence from the process model, we also sample one possible value for each attribute in each event. While sampling we enforce long term dependencies between attributes. For example, 2 of the 10 attribute values of one activity always occur when 1 of the attribute values for a different event occurred earlier in the sequence; hence, we model causal relationships between attributes within sequences.

Table 1. Overview showing dataset information

Name.	#Logs	#Activities	#Cases	#Events	#Attributes
P2P	10	12	12.5K	102K	0–5
Small	10	20	12.5K	111K	0–5
Medium	10	32	12.5K	73K	0–5
Large	10	42	12.5K	138K	0–5
Huge	10	54	12.5K	100K	0–5
Wide	10	34	12.5K	75K	0–5
BPIC12	1	36	13K	262K	0
BPIC13	3	5–13	0.8K–7.5K	2.4K–66K	2–4
BPIC15	5	355–410	0.8K–1.4K	44K–60K	2–3
BPIC17	2	8–26	31K–43K	194K–1.2M	1
Comp	10	7–18	0.9K–56K	4K–180K	1

In addition to the synthetic logs we also use the event logs from the Business Process Intelligence Challenge (BPIC): BPIC12[2], BPIC13[3], BPIC15[4] and BPIC17[5]. Furthermore, we evaluate our method on 10 real-life event logs of procurement processes, made available to us by a consulting company. Refer to Table 1 for information about the datasets.

Like Bezerra [3] and Böhmer [5], we apply artificial anomalies to the event logs, altering 30 percent of all cases. In addition to the three anomaly types used in [3,5], we introduced a new, attribute-based anomaly. The anomalies are defined as follows: *Skip*, a necessary activity has not been executed; *Switch*, two

[2] http://www.win.tue.nl/bpi/doku.php?id=2012:challenge.
[3] http://www.win.tue.nl/bpi/doku.php?id=2013:challenge.
[4] http://www.win.tue.nl/bpi/doku.php?id=2015:challenge.
[5] http://www.win.tue.nl/bpi/doku.php?id=2017:challenge.

events have been executed in the wrong order; *Rework*, an activity has been executed too many times; *Attribute*, an incorrect attribute value is set (e.g., a user does not have the necessary security level).

Notice that we do apply the artificial anomalies to the real-life event logs as well, which very likely already contain natural anomalies. Thereby, we can measure the performance of the algorithms on the real-life logs to demonstrate feasibility while using the synthetic logs to evaluate accuracy.

When applying the artificial anomalies, we also gather a ground truth dataset. Whenever the anomaly is of type *Skip*, *Rework*, or *Switch*, it is a control flow anomaly, and hence the activity name attribute is marked as anomalous for affected events. If the anomaly is of type *Attribute*, the affected attribute is marked as anomalous. Hence, we obtain ground truth data on attribute level. The ground truth data can easily be adapted to case level by the following rule: A case is anomalous if any of the attributes in its events are anomalous.

We generated 10 flavors of each synthetic process model with different numbers of attributes (0, 1, 2, 3, and 5) and different sizes for $\mathcal{V}_a$, resulting in 60 synthetic logs. Together with BPIC12 (1 log), BPIC13 (3 logs), BPIC15 (5 logs), and BPIC17 (2 logs), and the 10 procurement event logs (Comp), the corpus consists of 81 event logs.

4 Method

In this section we will describe the BINet architecture and all necessary steps for the implementation.

4.1 Preprocessing

Due to the mathematical nature of neural networks, we must transform the logs into a numerical representation. To accomplish this, we encode all string attribute values. Multiple options are available, such as integer encoding or one-hot encoding. We chose to use an integer encoding, which is a mapping $\mathcal{I}_a : \mathcal{V}_a \rightarrow \mathbb{N}$, mapping all possible attribute values for an attribute a to a unique positive integer. The integer encoding is applied to all attributes of the log, including the activity name.

We will represent the event logs as third-order tensors. Each event e is a first-order tensor $\boldsymbol{e} \in \mathbb{R}^A$, with $A = |\mathcal{A}|$, the first attribute always being the activity name, representing the control flow. Hence, an event is defined by its activity name and the event attributes. Each case is then represented as a second-order tensor $\boldsymbol{C} \in \mathbb{R}^{E \times A}$, with $E = \max_{c \in \mathcal{L}} |\hat{c}|$, being the maximum case length of all cases in the log $\mathcal{L}$. To force all cases to have the same size, we pad all shorter cases with event tensors only containing zeros, which we call padding events (these will be ignored by the neural network). The log $\mathcal{L}$ can now be represented as a third-order tensor $\boldsymbol{L} \in \mathbb{R}^{C \times E \times A}$, with $C = |\mathcal{L}|$, the number of cases in log $\mathcal{L}$. Using matrix

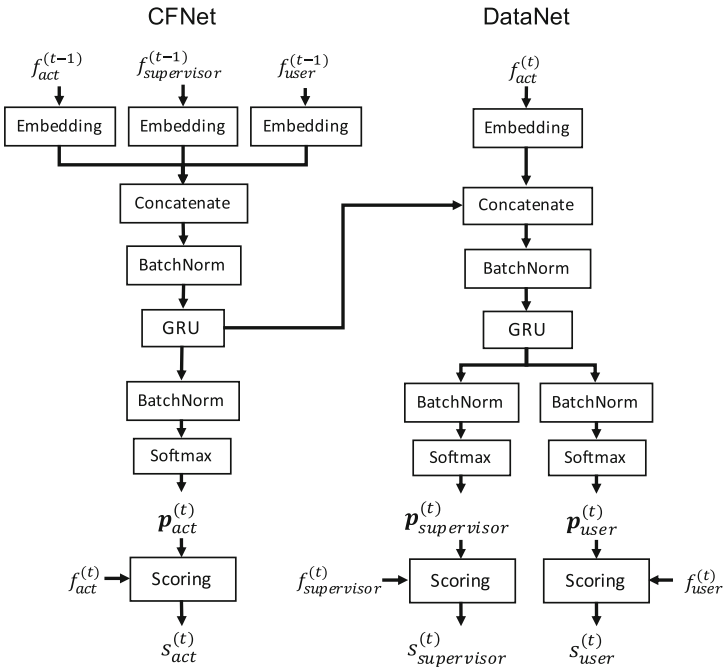

Fig. 1. BINet architecture for a log with two event attributes, *supervisor* and *user*

index notation, we can now obtain the second attribute of the third event in the ninth case with $\boldsymbol{L}_{9,3,2}$. Now we can define a preprocessor as follows:

Definition 2. *Preprocessor Let C, E, and A be defined as above, then a preprocessor is a mapping $\mathcal{P} : \mathcal{L} \to \mathbb{R}^{C \times E \times A}$.*

The BINet preprocessor $\mathcal{P}$ will encode all attribute values and then transform the log $\mathcal{L}$ into its tensor representation. In the following, we will refer to the preprocessed log $\mathcal{L}$ by $\boldsymbol{F}$ (features), with $\boldsymbol{F} = \mathcal{P}(\mathcal{L})$.

4.2 BINet Architecture

BINet is based on a neural network architecture that is trained to predict the next event, including all its attributes. To model the sequential nature of event log data, the core of BINet is a recurrent neural network, using a Gated Recurrent Unit (GRU) [8], an alternative for the popular long short-term memory (LSTM) [13].

We must distinguish between the control flow and the data flow aspect of event log data. Therefore, BINet is composed of two parts: CFNet (control flow) and DataNet (data flow). CFNet is responsible for predicting the next activity name of the next event, while DataNet is responsible for predicting all attributes of the next event.

Figure 1 shows the internal architecture of BINet. CFNet retrieves as input all attributes of an event $e^{(t-1)}$ and is trained to predict the activity name of event $e^{(t)}$, where t is the discrete time step. In the figure we use f_{act} for the activity name feature and $f_{supervisor}$ and f_{user} as examples for attribute features. Note that the architecture will grow automatically if more event attributes are present. The output layer of CFNet is one single softmax layer that outputs a probability distribution $\boldsymbol{p}_{act}^{(t)}$ over all possible activity names.

DataNet retrieves as input the activity name of $e^{(t)}$ and the internal state of the CFNet GRU and is trained to predict all attributes. DataNet will have a separate softmax layer for each event attribute. Note that DataNet is being fed the activity name at time t, which is the same time it is predicting the attributes for. This is crucial because the attributes of an event strongly depend on the activity. Without this information, DataNet will predict the attributes for the most likely next activity (based on the internal state of the CFNet GRU), which can be different from the actual next activity. Because DataNet is only predicting the attributes of an event and not its activity, using $f_{act}^{(t)}$ is legitimate.

BINet is trained on the event log to predict the next event and all its attributes. After the initial training phase, BINet can now be used for anomaly detection. This is based on the assumption that an anomalous attribute will be assigned a lower probability by BINet than a normal attribute.

The last step of the anomaly detection process is the scoring of the events. Therefore, we use a scoring function in the last layer of the architecture. This scoring function receives as input the probability distribution $\boldsymbol{p}_a^{(t)}$ for an attribute a and the actual value of the attribute $f_a^{(t)}$.

A softmax layer outputs a probability distribution over all possible values. When an event log features 5 different activity names, $\boldsymbol{p}_{act}^{(t)}$ will be a first-order tensor of size 5. Each of the dimensions of $\boldsymbol{p}_{act}^{(t)}$ holds the probability that the softmax layer assigns to one of the 5 possible activity values.

We can now define the scoring function σ as the difference between the probability of the most likely attribute (according to BINet) and the probability of the attribute value encountered $f_a^{(t)}$, where $\boldsymbol{p}_a$ is the probability tensor for attribute a and an event e, and i is the corresponding index of $f_a^{(t)}$ in $\boldsymbol{p}_a$.

$$\sigma(\boldsymbol{p}_a, i) = \max_{p \in \boldsymbol{p}_a} p - \boldsymbol{p}_{a_i}$$

The effect of normalization is demonstrated by Fig. 2, which shows three example cases of a P2P dataset with 2 attributes with and without normalization. Without normalization, the scoring function is defined as $\sigma(\boldsymbol{p}_a, i) = 1 - \boldsymbol{p}_{a_i}$, i.e., the inverse probability. We can observe that BINet is able to accurately predict the activities after the first activity, however, the anomaly scores for the user attribute are close to 1 because multiple users are permitted to execute an activity. We can counteract this effect by applying the normalization by confidence as demonstrated in Fig. 2.

The complete BINet architecture is then comprised of CFNet, DataNet and the scoring function σ applied to each softmax layer. We can obtain the anomaly scores tensor S by applying BINet to the feature tensor $\boldsymbol{F}$

$$\boldsymbol{S} = (s_{ijk}) \in \mathbb{R}^{C \times E \times A} = \text{BINet}(\boldsymbol{F}),$$

mapping an anomaly score to each attribute in each event in each trace. The anomaly score for attributes of padding events will always be 0.

4.3 Training

BINet is trained without the scoring function. The GRU units are trained in sequence to sequence fashion. With each event that is fed in, the network is trained to predict the attributes of the next event. We train BINet with a GRU size of $2E$ (two times the maximum case length), on mini batches of size 100 for 50 epochs using the Adam [16] optimizer using the parameters stated in the original paper. We use batch normalization [14] between all layers to counteract overfitting. Every feature is passed through a separate embedding layer (see [19]) to reduce the input dimension.

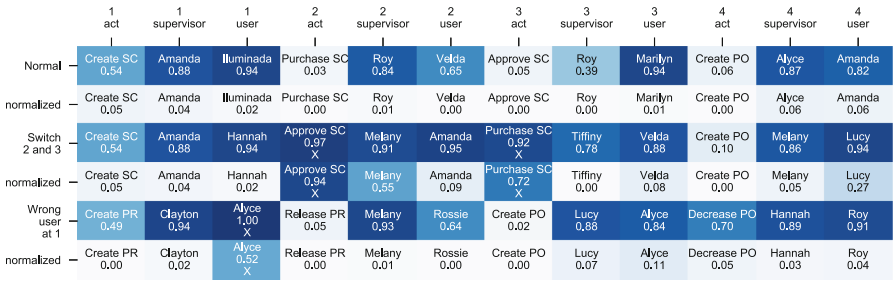

Fig. 2. Effect of confidence normalization on BINet anomaly scores (high scores indicate anomalies); anomalies are marked with X

4.4 Detection

An anomaly detector only outputs anomaly scores. We need to define a function that maps anomaly scores to a label $l \in \{0, 1\}$, 0 indicating normal and 1 indicating anomalous, by applying a threshold t. Whenever an anomaly score for an attribute is greater than t, this attribute is flagged as anomalous. To obtain a separate threshold for each anomaly score in $\boldsymbol{S}$, we define a threshold tensor $\boldsymbol{T} = \alpha \cdot \boldsymbol{\tau}$, where $\boldsymbol{\tau} \in \mathbb{R}^{C \times E \times A}$ is a baseline threshold tensor and $\alpha \in \mathbb{R}$ is a scaling factor. Now we can define the function θ, with inputs $\boldsymbol{S}$, α, and $\boldsymbol{\tau} = (\tau_{ijk})$, as

$$\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = (p_{ijk}) = \begin{cases} 1 & \text{if } (s_{ijk}) > \alpha \cdot (\tau_{ijk}) \\ 0 & \text{otherwise} \end{cases}.$$

To obtain a baseline threshold $\boldsymbol{\tau} \in \mathbb{R}^{C \times E \times A}$, we propose four different strategies τ_0, τ_e, τ_a, and τ_{ea}. In the following we will use $N = \sum_{c \in \mathcal{L}} |\hat{c}|$ to denote the number of non-padding events. The first baseline threshold function, τ_0, is defined by the average anomaly score over all cases, events, and attributes.

$$\tau_0(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{NA} \sum_{a}^{C} \sum_{b}^{E} \sum_{c}^{A} (s_{abc})$$

It is sensible to use a separate threshold for each event position in a case because the branching factor can vary for different points in a process model. Therefore, we define the second baseline threshold function, τ_e, based on the position of an event e in a case c.

$$\tau_e(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{CA} \sum_{a}^{C} \sum_{c}^{A} (s_{ajc})$$

It is also sensible to use a separate threshold for each attribute because $\mathcal{V}_a$ has a different size for each event e and attribute a. Thus, we define the third baseline threshold function, τ_a, to output a separate threshold for each event attribute.

$$\tau_a(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{N} \sum_{a}^{C} \sum_{b}^{E} (s_{abk})$$

The fourth baseline threshold function, τ_{ea} is a combination of τ_e and τ_a, and outputs a threshold for each event position and attribute separately.

$$\tau_{ea}(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{C} \sum_{a}^{C} (s_{ajk})$$

Note that we use matrix index notation to broadcast $\boldsymbol{\tau}$ to the right dimensionality to conform with the definition of θ from before. Utilizing the automatic broadcasting functionality in modern numerical computing libraries, such as TensorFlow[6] or NumPy[7], this can be implemented very efficiently. Remember that anomaly scores for padding events are set to 0, and hence they do not influence the sums. By normalizing with N we calculate the average based only on non-padding events.

4.5 Threshold Heuristic

Most anomaly detection algorithms rely on a manual setting for the threshold. We have proposed four different methods of obtaining a baseline threshold tensor $\boldsymbol{\tau}$ from the anomaly scores tensor $\boldsymbol{S}$. We still need to set the scaling factor α manually. To overcome this, we need to introduce a heuristic to set α automatically.

[6] https://tensorflow.org.
[7] http://numpy.org.

Because anomaly detection is an unsupervised task and no labels are available at runtime, we cannot optimize α based on the detection F_1 score. However, using the F_1 score function we can define the heuristic h_{best} that computes the best possible α for a given anomaly score tensor $\boldsymbol{S}$, a baseline threshold $\boldsymbol{\tau}$, and the ground truth label tensor $\boldsymbol{L}$ as

$$h_{best}(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{\tau}) = \arg\max_{\alpha} F_1(\boldsymbol{L}, \theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau})).$$

As we do not have access to L at runtime, we cannot use h_{best}. We propose a new heuristic that works like the elbow method, commonly used to find an optimal number of clusters for clustering algorithms (see [25]). As we cannot rely on the F_1 score as our metric, we propose the use the anomaly ratio r, which can be defined as, with $\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = (p_{ijk})$.

$$r(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = \frac{1}{CEA} \sum_{i}^{C} \sum_{j}^{E} \sum_{k}^{A} (p_{ijk})$$

The optimal α must lie between α_{low} and α_{high}, where $r(\boldsymbol{S}, \alpha_{low}, \boldsymbol{\tau}) = 1$ and $r(\boldsymbol{S}, \alpha_{high}, \boldsymbol{\tau}) = 0$. We can reduce our search space to this interval. Now we span a grid G of size s between α_{low} and α_{high} to define our candidates for α.

$$G = \left\{ \alpha_{low} + \frac{1}{s}(\alpha_{high} - \alpha_{low}), \ldots, \alpha_{low} + \frac{s}{s}(\alpha_{high} - \alpha_{low}) \right\}$$

In our experiments we found that $s = 20$ is a good choice for s, however, any reasonable choice of $s \in \{5, \ldots, 100\}$ generally works.

Because r is a discrete function, we use the central difference approximation to obtain the second order derivative r'' of r.

$$r''(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) \approx \frac{r(\boldsymbol{S}, \alpha - s, \boldsymbol{\tau}) - 2r(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) + r(\boldsymbol{S}, \alpha + s, \boldsymbol{\tau})}{s^2}$$

Now we can define the elbow heuristic

$$h_{elbow}(\boldsymbol{S}, \boldsymbol{\tau}) = \arg\max_{\alpha \in G} r''(\boldsymbol{S}, \alpha, \boldsymbol{\tau}).$$

h_{elbow} mimics the way a human would set α manually. When given a user interface with a heatmap visualization (like in Fig. 2) for $\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau})$ and control over the value of α, a human would start with a value of α where all attributes are marked as anomalous (i.e., the heatmap shows only blue and no white), and then gradually decrease α until the point where the heatmap switches from mostly showing blue to mostly showing white.

Figure 3 shows the F_1 score and the corresponding anomaly ratio r for a model of BINet, trained on a P2P dataset and using τ_a as the baseline threshold. We can see that the highest possible F_1 score correlates with the maximum of r'', and hence with the "elbow" of r. Interestingly, we found that h_{elbow} works just

as well for other anomaly detection algorithms, such as t-STIDE [26], Naive [3], and DAE [21].

Evaluating h_{elbow} for BINet over all synthetic datasets and all baseline threshold strategies, we can see in Fig. 4 that the best baseline threshold is τ_a. We also find, that the h_{elbow} works remarkably well over all strategies, for the performance of h_{elbow} is very close to h_{best}.

5 Evaluation

We evaluated BINet on all 81 event logs and compared it to two methods from [7]: a sliding window approach (t-STIDE+) [26]; and the one-class SVM (OC-SVM). Additionally, we compared BINet to two approaches from [3]: the Naive algorithm and the Sampling algorithm. Furthermore, we provide the results of the denoising autoencoder (DAE) approach from [20]. Lastly, we compared BINet to the approach from [5], which utilizes 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 scikit-learn[8] using an RBF kernel of degree 3 and $\nu = 0.5$. The Naive, Sampling, Likelihood, and DAE methods were implemented as described in the original papers. t-STIDE+ is an

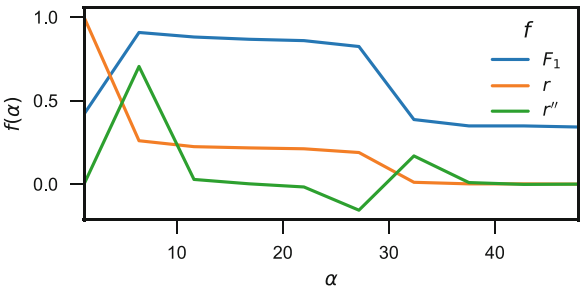

Fig. 3. F_1 score, anomaly ratio r, and second order derivative r'' (scaled for clarity) by α for BINet on a dataset with 5 attributes using τ_a as the baseline threshold

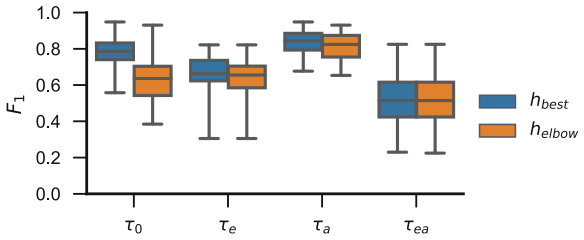

Fig. 4. F_1 score by strategy and heuristic for BINet on the P2P dataset

[8] http://scikit-learn.org.

implementation of the t-STIDE method from [26], which we adapted to work with event attributes (see [20]).

Sampling, Likelihood, Baseline, and the OC-SVM do not rely on a manual setting of the threshold and were unaltered. For the remaining algorithms we used h_{elbow} and chose the following baseline threshold strategies following a grid search: τ_0 for Naive, τ_{ea} for t-STIDE+, τ_a for DAE, and τ_a for BINet.

Figure 5 shows the F_1 score distribution for all methods over all datasets and for the two detection levels. F_1 score is calculated as the macro average F_1 score over the normal and the anomalous class. BINet outperforms all other methods on both detection levels. The more important detection level is the attribute level, as this measure demonstrates how accurately an anomaly detection algorithm can detect the actual attribute that caused an anomaly.

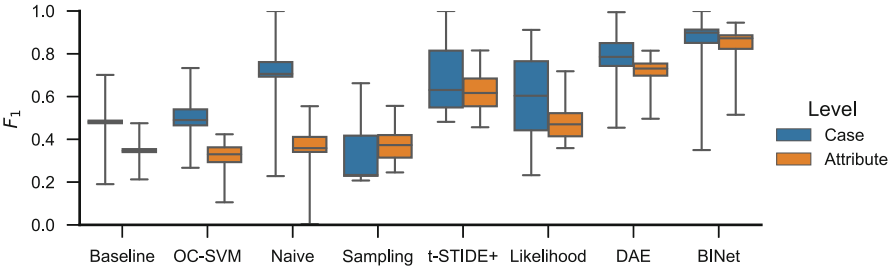

Fig. 5. F_1 score by method and detection level using h_{elbow} where applicable

Table 2. Results showing F_1 score over all datasets by detection level and method; best results are shown in bold typeface

Level	Method	P2P	Small	Medium	Large	Huge	Wide	BPIC12	BPIC13	BPIC15	BPIC17	Comp
Case	Baseline	0.47	0.50	0.47	0.48	0.48	0.50	0.55	0.50	0.49	0.47	0.45
	OC-SVM [23]	0.49	0.50	0.51	0.53	0.52	0.52	0.42	0.52	0.47	0.60	0.51
	Naive [3]	**0.91**	0.80	0.71	0.71	0.76	0.72	0.58	0.47	0.24	0.53	0.63
	Sampling [3]	0.23	0.23	0.44	0.34	0.23	0.23	0.45	0.26	0.22	0.22	0.57
	t-STIDE+ [26]	0.72	0.71	0.73	0.68	0.69	0.67	**0.81**	0.57	**0.51**	0.68	0.68
	Likelihood [5]	0.64	0.65	0.62	0.61	0.65	0.60	0.65	0.29	0.30	0.53	**0.73**
	DAE [20]	0.86	0.81	0.78	0.89	0.80	0.75	0.76	0.52	0.45	**0.75**	0.68
	BINet	**0.91**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	0.58	**0.58**	0.49	0.58	0.66
Attribute	Baseline	0.34	0.36	0.35	0.35	0.35	0.36	0.35	0.35	0.34	0.35	0.36
	OC-SVM [23]	0.33	0.34	0.32	0.35	0.34	0.32	0.11	0.36	0.32	0.37	0.26
	Naive [3]	0.50	0.41	0.35	0.35	0.39	0.37	0.09	0.14	0.00	0.24	0.36
	Sampling [3]	0.28	0.32	0.40	0.46	0.40	0.36	0.37	0.30	0.29	0.32	0.48
	t-STIDE+ [26]	0.66	0.66	0.68	0.64	0.64	0.65	0.67	0.56	0.51	0.62	0.49
	Likelihood [5]	0.50	0.50	0.51	0.48	0.50	0.50	0.47	0.40	0.36	0.47	0.52
	DAE [20]	0.77	0.72	0.72	0.75	0.75	0.71	**0.72**	0.52	0.51	**0.73**	0.61
	BINet	**0.85**	**0.89**	**0.87**	**0.89**	**0.88**	**0.88**	0.59	**0.57**	**0.54**	0.64	**0.68**

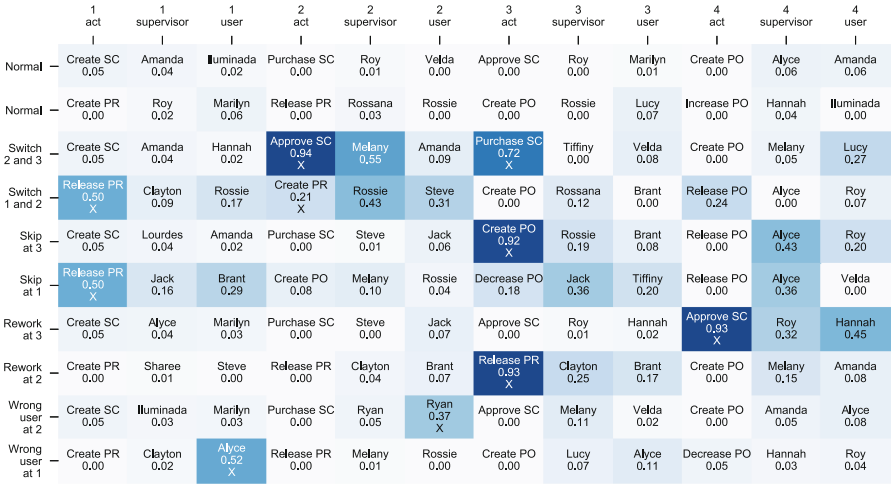

Fig. 6. Anomaly score heatmap for BINet trained on P2P with 2 attributes (supervisor and user); anomalies are marked by X

Expectedly, methods without attribute resolution, like Naive and OC-SVM, perform poorly on attribute level. t-STIDE+, Likelihood, and DAE support detection on attribute level, but show a significantly lower performance than BINet. DAE and BINet are both neural network based. The main advantage of BINet over DAE is that BINet makes use of the time dimension in the sequential data, whereas DAE does not.

Table 2 contains the detailed results for each method by dataset and detection level. BINet performs best across levels on the synthetic logs. On the real event logs, BINet performs best on attribute level, whereas on case level, the field is mixed. An accurate prediction on attribute level is to be favored over an accurate prediction on case level because only the attribute level allows to identify the exact cause of an anomaly.

Figure 6 shows a heatmap of BINet anomaly scores for a P2P dataset with 2 attributes. Two example cases are chosen for each type of anomaly and the normal class to demonstrate how BINet detects anomalies based on attribute level. No threshold has been applied to the anomaly scores. This way, the severity of an anomaly can be illustrated by the colors in the heatmap. Overall, we find that BINet detects control flow anomalies very effectively. For example, a shopping cart (SC) cannot be approved before it has been purchased (first Switch). Similarly, a purchase requisition (PR) cannot be released before it has been created (second Skip). Rework and Attribute anomalies are also detected accurately. In the case of Attribute, only the incorrect attribute is assigned a significantly high anomaly score. In the two examples, Ryan cannot be his own supervisor and Alyce is not permitted to create a PR.

6 Conclusion

In this paper we presented BINet, a neural network architecture for multivariate anomaly detection in business process event logs. Additionally, we proposed a heuristic for setting the threshold of an anomaly detection algorithm automatically, based on the anomaly ratio function.

BINet is a recurrent neural network, and can therefore be used for real-time anomaly detection, since it does not require a completed case for detection. BINet does not rely on any information about the process modeled by an event log, nor does it depend on a clean dataset. Utilizing the elbow heuristic, BINet's internal threshold can be set automatically, reducing manual workload. It can be used to find point anomalies as well as contextual anomalies because it models the time dimension in event sequences and utilizes both the control flow and the data flow information. Furthermore, BINet can cope with concept drift, as it can be setup to continuously train on new cases in real-time.

Based on the empirical evidence obtained in the evaluation, BINet is a promising method for anomaly detection, especially in business process event logs. BINet outperformed the opposition on all detection levels (case, event, and attribute level). Specifically, on the synthetic datasets BINet's performance surpasses those of other methods by an order of magnitude.

For an accurate detection of an anomaly it is essential that an anomaly detection algorithm processes event logs on attribute level; otherwise, a control flow anomaly cannot be distinguished from a data flow anomaly. To allow easy analysis of an event log, the attribute level is the most important detection level. On attribute level, BINet performs significantly better than the other methods.

Overall, the results presented in this paper suggest that BINet is a reliable and versatile method for detecting attribute anomalies in business process logs.

Acknowledgments. This project [522/17-04] is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE, Förderlinie 3: KMU-Verbundvorhaben (State Offensive for the Development of Scientific and Economic Excellence), and by the German Federal Ministry of Education and Research (BMBF) Software Campus project "AI-PM" [01IS17050].

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Bezerra, F., Wainer, J.: Anomaly detection algorithms in logs of process aware systems. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 951–952. ACM (2008)
3. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**(1), 33–44 (2013)
4. Bezerra, F., Wainer, J., van der Aalst, W.M.P.: Anomaly detection using process mining. In: Halpin, T., et al. (eds.) BPMDS/EMMSAD -2009. LNBIP, vol. 29, pp. 149–161. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01862-6_13

5. Böhmer, K., Rinderle-Ma, S.: Multi-perspective anomaly detection in business process execution events. In: Debruyne, C., et al. (eds.) Move to Meaningful Internet Systems. LNCS, pp. 80–98. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_5
6. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings. arXiv:1506.08415 (2015)
7. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection for discrete sequences: a survey. IEEE Trans. Knowl. Data Eng. **24**(5), 823–839 (2012)
8. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078 (2014)
9. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
10. Evermann, J., Rehse, J.-R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: Dumas, M., Fantinato, M. (eds.) BPM 2016. LNBIP, vol. 281, pp. 327–338. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58457-7_24
11. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
12. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier, New York City (2011)
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
14. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
15. Japkowicz, N.: Supervised versus unsupervised binary-learning by feedforward neural networks. Mach. Learn. **42**(1), 97–122 (2001)
16. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv:1412.6980 (2014)
17. Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: LSTM-based encoder-decoder for multi-sensor anomaly detection. arXiv:1607.00148 (2016)
18. Marchi, E., Vesperini, F., Eyben, F., Squartini, S., Schuller, B.: A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks, April 2015
19. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv:1301.3781 (2013)
20. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. arXiv:1803.01092 (2018)
21. Nolle, T., Seeliger, A., Mühlhäuser, M.: Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In: Calders, T., Ceci, M., Malerba, D. (eds.) DS 2016. LNCS (LNAI), vol. 9956, pp. 442–456. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46307-0_28
22. Pimentel, M.A.F., Clifton, D.A., Clifton, L., Tarassenko, L.: A review of novelty detection. Sig. Process. **99**, 215–249 (2014)
23. Schölkopf, B., et al.: Support vector method for novelty detection. In: NIPS. vol. 12, pp. 582–588 (1999)
24. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30

25. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. J. R. Stat. Soc.: Ser. B (Stat. Methodol.) **63**(2), 411–423 (2001)
26. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133–145. IEEE (1999)
27. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Disc. **15**(2), 145–180 (2007)
28. Wressnegger, C., Schwenk, G., Arp, D., Rieck, K.: 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, pp. 67–76. AISec 2013. ACM (2013)