Information Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information Systems

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

BINet: Multi-perspective business process anomaly classification

Timo Nolle *, Stefan Luettgen, Alexander Seeliger, Max Mühlhäuser

Telecooperation Lab, Technische Universität Darmstadt, Hochschulstr. 10, 64289 Darmstadt, Germany

ARTICLE INFO

Article history:
Received 3 February 2019
Received in revised form 29 June 2019
Accepted 21 October 2019
Available online xxxx
Recommended by Gottfried Vossen

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

ABSTRACT

In this paper, we introduce BINet, a neural network architecture for real-time multi-perspective anomaly detection in business process event logs. BINet is designed to handle both the control flow and the data perspective of a business process. Additionally, we propose a set of heuristics 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 at a case level but also at event attribute level. Finally, we demonstrate that a simple set of rules can be used to utilize the output of BINet for anomaly classification. We compare BINet to eight other state-of-the-art anomaly detection algorithms and evaluate their performance on an elaborate data corpus of 29 synthetic and 15 real-life event logs. BINet outperforms all other methods both on the synthetic as well as on the real-life datasets.

1. Introduction

Corporations are relying on business processes to ensure a smooth operation of their business. Deviations from these processes can have a significant impact on the economic well-being of a company. Naturally, businesses are interested in finding and, subsequently, reducing these deviations. Analyzing the business processes and their performance has traditionally been a human task, and therefore a manual effort. However, in the digital age, every business process leaves a digital footprint, allowing for a completely data-driven analysis of business processes.

Anomaly detection is a popular technique to automatically find these deviations based on the data the businesses are generating. An anomaly, in this context, is a deviation from the behavior defined by the business process. This can be related to the order of activities (control flow perspective), e.g., two activities are executed in the wrong order. Furthermore, it can be related to certain data attributes (data perspective), e.g., an employee has executed an activity without permission or violations against segregation of duty regulations. These two perspectives are inherent to how business process data is saved (see [1]) and must be accounted for by a business process anomaly detection algorithm.

The basis for an accurate anomaly detection is an accurate model of the normal behavior, i.e., the normal business process. Any behavior not captured by this model can be regarded as anomalous. Some methods rely on a human-defined process model as the basis for normal behavior, called a reference model. However, such a reference model is not always available and it might not be economically justifiable to create such a reference model. In those cases, a method that does not require such a reference model is desired.

Autonomous anomaly detection algorithms have to infer this model directly from the patterns in the data and without any domain knowledge about the underlying process. This can be challenging because the data naturally contains anomalous behavior which the algorithm has to ignore. Some algorithms depend on a clean dataset, that is, a dataset that only consists of normal behavior. However, this requires a dedicated preprocessing step based on domain knowledge. Very few business process anomaly detection algorithms exist that infer the normal model from an uncleaned dataset. In Section 3 we give an elaborate summary of existing approaches.

There are further desired properties for a business process anomaly detection algorithm. It is important that an algorithm can be used during the execution of the business process because otherwise no countermeasures can be initiated in time. Furthermore, business processes are dynamic systems, adapting to seasonal effects, market changes, or other external factors. An autonomous anomaly detection algorithm should deal with this dynamic behavior.

Against this background, we propose BINet (Business Intelligence Network), a neural network architecture for real-time multi-perspective business process anomaly detection. BINet has been specifically designed to model both the control flow and the data perspective of business process execution data. Thus, it can capture a variety of different causal dependencies, allowing to detect anomalies related to control flow and data perspective (cf. point and contextual anomalies in [2]). It does not rely on domain knowledge about the business process or a reference model

* Corresponding author.
E-mail address: nolle@tk.tu-darmstadt.de (T. Nolle).

https://doi.org/10.1016/j.is.2019.101458
0306-4379/

and it does not require prior knowledge about the anomalies such as anomaly labels. BINet can be run continuously during the execution of a business process, adapting to concept drift and enabling early detection at runtime. The only input to the BINet algorithm is the business process data, including anomalous behavior. BINet distinguishes between normal and anomalous behavior by automatically defining a threshold solely based on the input data.

This paper is an extension of our previous work on BINet from 2018 [3]. Compared to the original publication, we have simplified the architecture of BINet and present three versions of BINet with different dependency modeling capabilities. Additionally, we elaborate on the threshold heuristics proposed in [3] and introduce an improved set of heuristics mimicking human intuition. We improved the dataset generation algorithm, which now uses an extended likelihood graph to generate causally dependent activities and event attributes. Finally, we propose a simple rule-based classifier to distinguish different anomaly classes, solely based on the outputs of BINet.

This work contains five main contributions.

- BINet neural network architecture
- Automatic threshold heuristics
- Generation algorithm for synthetic event logs
- Comprehensive evaluation of state-of-the-art methods
- Rule-based anomaly classifier

This paper is structured as follows. We start with background information on process mining and neural networks in Section 2, followed by a comprehensive summary of related works and how they relate to BINet in Section 3. Then, we elaborate on the generation algorithm for realistic datasets in Section 4. In Section 5, we describe BINet's preprocessing, neural architecture, detection algorithm, and automatic threshold heuristic. We compare BINet to 8 state-of-the-art anomaly detection methods and evaluate them on a comprehensive dataset of 29 synthetic logs and 15 real-life logs, using artificial anomalies in Section 6. Lastly, we demonstrate that BINet can be used to distinguish anomalies in Section 7. After a short discussion in Section 8, we conclude this paper in Section 9.

2. Background

In this section, we give an introduction to the main concepts behind business process anomaly detection.

2.1. Process mining

For business process data, the most popular techniques originate from the field of process mining. Process mining is centered around the idea of human-readable representations of business processes called process models. Process models are widely used in business process management as a tool for defining, documenting, and controlling business processes inside companies.

During the execution of a digital business process, each process step is stored in a database. This includes information on when the process step was executed (timestamp), what process step was executed (activity), and to which business case it belongs (case identifier). These three fundamental bits of event information are the basis for every process mining algorithm and are usually combined into a single data structure called event log.

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, and 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 $a \in \mathcal{A}$. $|\hat{c}|$ 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.

Based on these event logs, process mining is used to discover the underlying business process and produce a human-readable process model visualization. These algorithms are known as discovery algorithms. An optimal process model should be simple (simplicity), general (generality), precise (precision), and model all behavior found in the event log (fitness). However, these goals are conflicting because, for instance, a process model cannot be both simple and precise, and thus discovery algorithms often emphasize two of the four goals. Simplicity can be emphasized, for example, by ignoring infrequent patterns, since they are unlikely to be part of the real business process.

Some companies document their processes and save them as digital process models. These process models can be used to check whether or not the process is being executed according to it. This is called conformance checking. The result is an annotated event log in which events are marked as conform or non-conform. Non-conform can either indicate that the activity is found in the event log, but is not part of the process, or the activity is found in the model, but not in the event log.

Conformance checking can be used for anomaly detection if a reference model is available. Otherwise, a discovery algorithm can be used to infer such a reference model from the event log. Simplicity and generality are usually preferred over precision and fitness in this setting. However, most discovery and conformance checking algorithms mainly focus on the control flow perspective, and hence they can only partially be applied to the data perspective.

2.2. Artificial neural networks

In fields like computer vision, natural language processing, and speech recognition, artificial neural networks (NNs) have become the state-of-the-art, outperforming other machine learning algorithms by great margins. Key to this success is a concept called representation learning. Aforementioned fields are especially challenging because they require sophisticated feature engineering, namely transforming the raw data into meaningful features to be used as input for the machine learning algorithm. NNs are capable of discovering the necessary representations as part of their training algorithm, reducing, or even removing entirely, the need for feature engineering. This has opened up many areas for the application of machine learning algorithms that were originally considered impractical.

One of these areas is natural language processing, which can be considered particularly relevant to our case since business process executions hold some similarities to natural language, such as the sequential nature. Activities can be considered as words, cases as sentences, and the order of events as well as the event attributes as the grammar of the language. Furthermore, if the same activity is executed by different users, the two are considered distinct. Similarly, if the same activity is preceded by different events, they are also considered distinct.

An NN is a non-linear composite function with multiple degrees of freedom, called weights. These weights can be altered to control the output of the NN, minimizing the error with respect to some desired output. This optimization procedure is referred to as training. During training, the weights are iteratively updated by continuously considering examples of inputs (e.g., a sequence

Please cite this article as: T. Nolle, S. Luettgen, A. Seeliger et al., BINet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458, https://doi.org/10.1016/j.is.2019.101458.

of events) and desired outputs (e.g., the most probable next activity), and changing the weights according to their influence on the output. The goal of the training is to find a set of weights that approximates, as closely as possible, a mapping function from inputs to the respective desired outputs.

Recurrent neural networks (RNNs) have been designed specifically to handle sequential data such as sentences. Instead of processing an event sequence all-at-once, an RNN processes each event individually, recurrently using the same set of weights. To retain information about past events, an RNN uses an internal state (memory), that is updated for each event. This internal state resembles a representation of the event sequence up until that point. The update of the internal state is again controlled by a separate set of weights which are also optimized as part of the training procedure.

A drawback of the design of a classic RNN is that it is forced to update its internal state with each new event, thereby potentially overriding previous information. As a consequence, older events are not as strongly remembered as very recent ones. The RNN quickly forgets about the distant past. Hochreiter and Schmidhuber have addressed this problem with their design of the long short-term memory (LSTM) [4].

Instead of forcing the network to always update its internal state, an LSTM features specific sets of weights that control if the internal state is updated. Thus, through the training procedure, an LSTM can learn when to remember important information, when to retrieve it, and when to forget it. Because an LSTM is not forced to remember everything, it can retain information about past events for arbitrary long sequences, and without loss of memory. Contrary to RNNs, where the internal state at any time is a representation of all events up until that point, the internal state of an LSTM is a representation of important events (i.e., events worth remembering) up until that point.

In the field of natural language processing, LSTMs are used to predict the most probable word to continue an unfinished sentence. This concept can be utilized for anomaly detection in business processes. For an unfinished business case, an LSTM predicts the most probable next event, and if the prediction does not match the actual next event, it can be considered to be anomalous.

3. 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 [5–7]. However, the proposed methods do not utilize the event attributes, and therefore cannot be used to detect anomalies at attribute level.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior [8]. This method includes important characteristics of the process itself by including the event attributes as part of an extended likelihood graph. However, this method relies on a discrete order in which the attributes are connected to the graph, which may introduce a bias towards certain attributes. Furthermore, the same activities are mapped to the same node in the likelihood graph, thereby assigning a single probability distribution to each activity. In other words, control flow dependencies cannot be modeled by the likelihood graph, because the probability distribution of attributes following an activity does not depend on the history of events.

The main drawback of this method is that it uses the initial log to build the likelihood graph, and therefore no case of the original log is classified as anomalous. This is related to the strategy to determine a threshold for the anomaly detection task. We address this caveat in Section 6. However, the notion of the likelihood graph inspired the generation method for synthetic event logs in Section 4.

A review of traditional anomaly detection methodology can be found in [9]. The authors describe and compare many methods that have been proposed over the last decades. Another elaborate summary of anomaly detection in discrete sequences is given by Chandola et al. in [10]. 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 [11]. 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 [12]. While being inexpensive and easy to implement, sliding window approaches show a robust performance in finding anomalies in sequential data, especially within short regions [10].

Distance-based novelty detection does not require a clean dataset, yet it is only partly applicable to 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. [13] first used support vector machines [14] 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 fails to do so with anomalous behavior. Therefore, the reconstruction error can be used to detect anomalies [15]. This approach has successfully been used for the detection of control flow anomalies [16] as well as data flow anomalies [17] in event logs of PAISs.

Domain-based novelty detection requires domain knowledge, which is something we want to avoid. 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 yields 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 [18,19] and Tax [20], who utilized long short-term memory (LSTM) [4] networks for next event prediction, demonstrating their utility. LSTMs have been used for anomaly detection in different contexts like acoustic novelty detection [21] and predictive maintenance [22]. These applications mainly focus on the detection of anomalies in time series and not, like BINet, on multi-perspective anomaly detection in discrete sequences of events.

The novelty of BINet lies in the tailored architecture for business processes, including the control flow and data perspective, the scoring function to assign anomaly scores, and the automatic threshold heuristic. It is a universally applicable method for anomaly detection both in the control flow and the data perspective of business process event logs. Lastly, BINet can handle multiple event attributes and model causal dependencies between control flow and data perspective, as well dependencies between event attributes. This combination is, to the best of our knowledge, novel to the field.

4. Generating realistic event logs

To calculate the performance of a business process anomaly detection algorithm, a labeled dataset is necessary. In this section,

Please cite this article as: T. Nolle, S. Luettgen, A. Seeliger et al., BINet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458, https://doi.org/10.1016/j.is.2019.101458.

we will describe a generation algorithm that can be used to generate realistic event logs from random process models of different complexities. Additionally, we will describe how realistic artificial anomalies can be added to these event logs, resulting in a labeled dataset.

We use a simple paper submission process as the running example to illustrate concepts, methods, and results throughout this paper. The process model in Fig. 1 describes the creation of a scientific paper. Note that the process includes the peer review process, which is executed by a reviewer, whereas the paper is conceptualized and compiled by an author.

4.1. Synthetic event log generation

We used PLG2 [23] to generate six random process models. The models vary in complexity with respect to the number of activities, breadth, and width. Additionally, we use a handmade procurement process model called P2P as in [3], and the paper process shown in Fig. 1.

To generate causally dependent event attributes, we adopt the notion of the extended likelihood graph from [8]. A likelihood graph is a directed graph where each node represents a possible activity and the connection between nodes represents the probability of one activity following another. For example, activity *B* has a 60 percent chance of following *A*. This idea can be extended to include multiple event attributes. For instance, *B* has a 35 percent chance of following *A* if it is Monday or *B* has a 15 percent chance of following *A* if it is Monday and user 1 executed the activity. This pattern can be continued to include an arbitrary numbers of event attributes.

We applied this technique to the process shown in Fig. 1. For each activity, we create a group of possible users allowed to execute the activity and assign probabilities to each user. Fig. 2 demonstrates the final result.

The *Experiment* activity appears twice in this likelihood graph to introduce a long-term control flow dependency. That is, *Conduct Study* always eventually follows *Develop Hypothesis*, and never eventually follows *Develop Method*. Additionally, the user group, as well as the corresponding probabilities, are different.

We can also model causal dependencies between event attributes by including more event attributes as explained before. For example, we could add a weekday attribute to model that *Main Author* only works Mondays and Tuesdays. This is realized by setting the probabilities for the other weekdays to zero.

We have described long-term control flow dependencies as well as data dependencies. By combining the two ideas, data to control flow dependencies are also possible, as *Research Related Work* demonstrates. *Develop Method* always directly follows *Research Related Work* if *Student* is the user.

We can generate event logs by using a random-walk through the likelihood graph, complying with the transition probabilities, and generating activities and attributes along the way. We implemented the generation algorithm so that all these dependencies can be controlled by parameters and the event attributes are automatically generated. Please refer to the code repository,[1] and specifically the notebooks section, for a detailed description of the algorithm as well as examples.

In addition to the synthetic logs, we also use the real-life event logs from the Business Process Intelligence Challenge (BPIC): BPIC12,[2] BPIC13,[3] BPIC15,[4] and BPIC17.[5] Furthermore, we use a set of 4 event logs (referred to by Anonymous) from real-life procurement processes provided by a consulting company.

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

[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.

4.2. Artificial anomalies

Like Bezerra [24] and Böhmer [8], we apply artificial anomalies to the event logs, altering 30 percent of all cases with exactly one anomaly type. Inspired by the anomaly types (*Skip*, *Insert*, and *Switch*) used in [8,24], we identified additional, more elaborate anomalies that frequently occur in real business processes. These anomalies are defined as follows.

- *Skip*: A sequence of up to 3 necessary events has been skipped
- *Insert*: Up to 3 random activities have been inserted
- *Rework*: A sequence of up to 3 events has been executed a second time
- *Early*: A sequence of up to 2 events has been executed too early, and hence is skipped later in the case
- *Late*: A sequence of up to 2 events has been executed too late, and hence is skipped earlier in the case
- *Attribute*: An incorrect attribute value has been set in up to 3 events

We apply the artificial anomalies to the real-life event logs as well, knowing that they likely already contain natural anomalies which are not labeled. However, we can measure the performance of the algorithms on the real-life logs to demonstrate feasibility while using the synthetic logs to evaluate accuracy.

As indicated in Fig. 3 we can gather a ground truth dataset by marking the attributes with their respective anomaly types. Note that we introduce a *Shift* anomaly type, which is used to indicate the place where an *Early* or *Late* event used to be. This is equivalent to a *Skip*. However, we want to differentiate these two cases.

We insert a random event in case of *Insert*, i.e., the activity name does not come from the original process. This is to prevent a random insert from resembling *Rework*, which is possible when randomly choosing an activity from the original process. Attribute values of inserted events are set in the same fashion. When applying an *Attribute* anomaly, we randomly select an attribute value from the likelihood graph that is not a direct successor of the last node.

We created datasets with ground truth data on attribute level. For the anomaly detection task, these labels are mapped to either *Normal* or *Anomaly*, thus creating a binary classification problem. 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 4 likelihood graphs for each synthetic process model with different numbers of attributes, different transition probabilities, and dependencies. Then, we sampled logs from these likelihood graphs, resulting in 28 synthetic logs; 29, including the Paper dataset. Together with BPIC12, BPIC13, BPIC15, BPIC17, and Anonymous, the corpus consists of 44 event logs. We refer to the datasets by their names as defined in Table 1, which gives a detailed overview of the corpus.

5. Method

In this section, we describe the BINet architecture and how it is utilized for anomaly detection.

5.1. Preprocessing

Due to the mathematical nature of neural networks, we must transform the logs into a numerical representation. To accomplish this, we use integer encoding over all nominal attributes. An integer encoding is a mapping $\mathcal{I}_a : \mathcal{V}_a \rightarrow \mathbb{N}$ of all possible attribute values for an attribute a to a unique positive integer. The

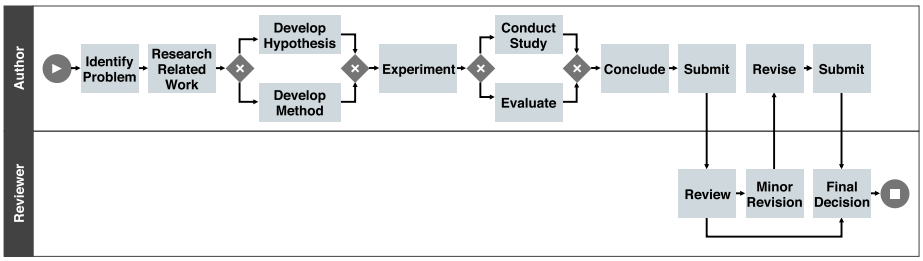

Fig. 1. A simple paper submission process which is used as an example throughout the paper.

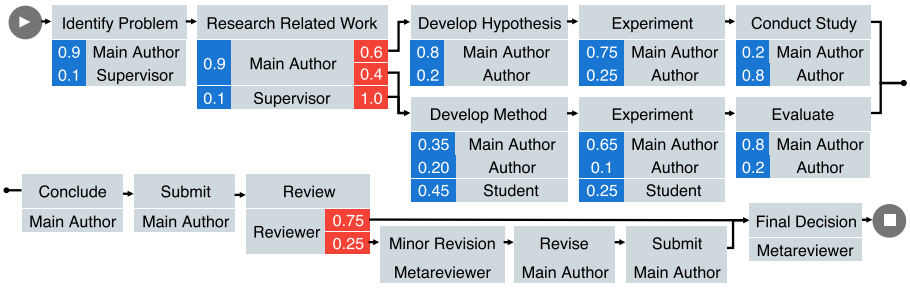

Fig. 2. A likelihood graph with user attribute; 1.0 probabilities omitted for simplicity.

Fig. 3. Anomalies applied to cases of the paper submission process.

Table 1
Overview showing dataset information.

Name.	#Logs	#Activities	#Cases	#Events	#Attr.	#Attr. values
Paper	1	27	5k	66k	1	13
P2P	4	27	5k	48k–53k	1–4	13–386
Small	4	41	5k	53k–57k	1–4	13–360
Medium	4	65	5k	39k–42k	1–4	13–398
Large	4	85	5k	61k–68k	1–4	13–398
Huge	4	109	5k	47k–53k	1–4	13–420
Gigantic	4	154–157	5k	38k–42k	1–4	13–409
Wide	4	68–69	5k	39k–42k	1–4	13–382
BPIC12	1	73	13k	290k	0	0
BPIC13	3	11–27	0.8k–7.5k	4k–81k	2–4	23–1.8k
BPIC15	5	422–486	0.8k–1.4k	46k–62k	2–3	23–481
BPIC17	2	17–53	31k–43k	284k–1.2M	1	289–299
Anonymous	4	19–37	968–17k	6.9k–82k	1	160–362

integer encoding is applied to all attributes of the log, including the activity name.

Event logs can be represented as third-order tensors. Each event e is a first-order tensor $\mathbf{e} \in \mathbb{R}^A$, with $A = |\mathcal{A}|$, the first attribute always being the activity name, representing the control flow perspective. Hence, an event is defined by its activity name and the event attributes. Each case is then represented as a second-order tensor $\mathbf{C} \in \mathbb{R}^{E \times A}$, with $E = \max_{c \in \mathcal{L}} |\hat{c}|$ being the maximum case length of all cases in the log $\mathcal{L}$. For mathematical simplicity and to represent an event log as a tensor, we assume all cases to have the same length. Therefore, we pad all shorter cases with event tensors only containing zeros, which we call padding events. The padding events are ignored by the neural network during training and inference.

The log $\mathcal{L}$ can now be represented as a third-order tensor $\mathbf{L} \in \mathbb{R}^{C \times E \times A}$, where $C = |\mathcal{L}|$ is the number of cases in log $\mathcal{L}$. Using matrix index notation, we can now obtain the second attribute of the third event in the ninth case with $\mathbf{L}_{9,3,2}$. We can also obtain all the second attributes of the third event using $\mathbf{L}_{:,3,2}$, using ":" to denote the cross-section of tensor $\mathbf{L}$ along the case axis. Likewise, we can obtain all the second attributes of case nine with $\mathbf{L}_{9,:,2}$. Thus, 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 $\mathcal{P} : \mathcal{L} \rightarrow \mathbb{R}^{C \times E \times A}$.

The preprocessor $\mathcal{P}$ encodes all attribute values and then transforms the log $\mathcal{L}$ into its tensor representation. In the following, we refer to the preprocessed log $\mathcal{L}$ as $\mathbf{F}$ (features), where $\mathbf{F} = \mathcal{P}(\mathcal{L})$.

5.2. BINet neural architecture

To model the sequential nature of event log data, the core of BINet is a recurrent neural network, based on Gated Recurrent

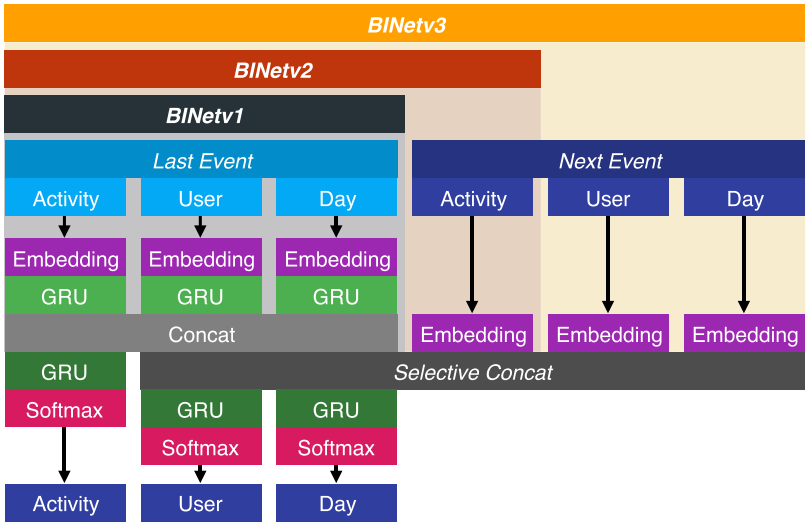

Fig. 4. BINet architectures for a log with two event attributes, *User* and *Day*; the three versions of BINet differ only in the inputs they receive.

Units (GRUs) [25], an alternative to the popular LSTM. BINet processes the distinct sequence of events for each case. For each event, BINet has to predict the next event based on the history of events in the case.

Fig. 4 shows the architecture of BINet. We propose three versions of BINet (BINetv1, BINetv2, and BINetv3). These versions differ in their capability of modeling causal dependencies based on the inputs they receive. BINet consists of dedicated encoder GRUs (light green) for each input attribute (light blue) of the last event. These GRUs are responsible for creating a latent representation of the complete history of a single attribute. Each attribute is fed through an embedding layer to reduce the input dimensionality (see [26,27]).

The counterpart to the encoder GRUs are the decoder GRUs (dark green) which receive as input a concatenation of attribute representations. These GRUs are responsible for combining the control flow and the data perspective, and hence to create a latent representation of the complete history of events that is meaningful to the respective next attribute prediction in the next layer. For prediction, BINet uses a softmax output layer (pink) for each next attribute. A softmax layer outputs a probability distribution over all possible values of the respective attribute, that is, all outputs will sum to one.

In its simplest form, BINet predicts the next activity and all next attributes solely based on past events. This architecture is called BINetv1 (black). However, there likely exist causal dependencies between the activity and the corresponding attribute values for that activity. To model this type of dependency, we propose a second architecture, BINetv2 (red), which in addition to the input of BINetv1, gets access to the activity of the next event. This conditions the attribute decoders onto the actual next activity, as opposed to inferring the next activity from the states of the encoders. Using the BINetv2 architecture, we can model control flow to data dependencies.

There also likely exist dependencies between attributes (i.e., certain users only work certain days of the week). With BINetv1 or BINetv2, these dependencies are not modeled because the attributes are treated as though they were independent. To address this, we propose BINetv3 (orange) which gets access to the complete next event. Hence, BINetv3 can model data to data dependencies.

The selective concatenation layer (dark gray) does not allow information to flow from one of the next event inputs to the respective next attribute decoder GRU. Otherwise, a decoder GRU would gain direct access to the information they are trained to predict, resulting in no learning at all because the decoder can simply copy the input.

To elaborate on the differences in the BINet versions we refer back to the paper submission process from Fig. 2. Suppose the last activity input to BINet is *Research Related Work* and the user was *Main Author*. The activity output should now give a probability of approximately 60 percent to *Develop Hypothesis*. In case of BINetv1, however, the user output does not match the 80 percent for *Main Author* and the 20 percent for *Author*, because the respective decoders also have to take into account the other 40 percent of not going to *Develop Hypothesis*, and hence output a higher probability for *Student*. BINetv2 does not suffer from this problem, because it is certain that *Develop Hypothesis* is the next activity (because of the next activity input), and thus can learn the probabilities appropriately.

To demonstrate the advantage of BINetv3, we have to imagine a third weekday attribute as part of the extended likelihood graph. Suppose for a given activity *Main Author* works only on Fridays, and *Author* works from Monday until Thursday. BINetv3 can correctly predict that if the weekday is Friday, the user must be *Main Author*, whereas BINetv2 cannot. Likewise, BINetv3 can infer that if the user is *Main Author*, the day must be Friday.

For BINetv1, activity, user, and day are entirely independent, for BINetv2, user and day are dependent on the activity, and for BINetv3, user is dependent on activity and day, whereas day is dependent on activity and user. Our implementation of BINet theoretically allows for any number of events and attributes.

BINetv2 is equivalent to the original BINet architecture from [3].

5.3. CaLculating anomaly scores

After the initial training phase, BINet can be used for anomaly detection. This is based on the idea that BINet assigns a lower probability to an anomalous attribute than a normal attribute.

The last step of the anomaly detection process is the scoring of the events. Therefor, we use a scoring function in the last layer of the architecture. This scoring function for an attribute a receives as input the output of the softmax layer for a, that is, a probability distribution $\mathbf{p}_a$, and the actual value of the attribute, v.

Using the example above (the last activity being *Research Related Work* and the user being *Main Author*), the output of the activity softmax layer might look as depicted in Fig. 5. The

Please cite this article as: T. Nolle, S. Luettgen, A. Seeliger et al., BINet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458, https://doi.org/10.1016/j.is.2019.101458.

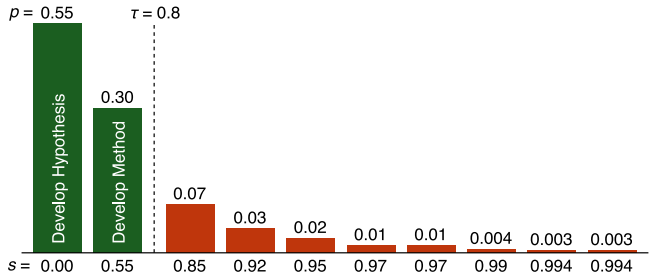

Fig. 5. Output of the activity softmax layer after reading activity *Research Related Work* and user *Main Author*.

probability p for *Develop Hypothesis* is 0.55, and the probability for *Develop Method* is 0.30. Note that BINet gives a high probability for the two correct next activities with respect to the paper process.

We can now define the anomaly score for a possible attribute value v as the sum of all probabilities p of the probability distribution tensor $\mathbf{p}$ greater than the probability assigned to v, p_v. The scoring function σ is therefore defined as follows, with $\mathbf{p}_i$ being ith probability.

$$\sigma(\mathbf{p}, p_v) = \sum_{\mathbf{p}_i > p_v} \mathbf{p}_i$$

Fig. 5 also shows the resulting anomaly scores, s, for each possible activity. Intuitively, an anomaly score of 0.55 indicates that there is a 55 percent chance that the attribute value is anomalous. Thus, we can set a threshold as indicated in Fig. 5 ($\tau = 0.8$), to flag all values as normal where the anomaly score is less than 0.8.

The scoring function σ is applied to each softmax output of BINet, transforming the probability distribution tensor into a scalar anomaly score. We can now obtain the anomaly scores tensor S by applying BINet to the feature tensor $\mathbf{F}$,

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

mapping an anomaly score to each attribute in each event in each case. The anomaly score for attributes of padding events is always 0.

5.4. 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 500 for 20 epochs using the Adam optimizer with parameters as stated in the original paper [28]. Additionally, we use batch normalization [29] after each GRU to counteract overfitting.

5.5. 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 τ. Whenever an anomaly score for an attribute is greater than τ, this attribute is flagged as anomalous. Therefore, we define a threshold function θ, with inputs $\mathbf{S}$ and $\tau \in \mathbb{R}$.

$$\theta(\mathbf{S}, \tau) = \mathbf{Y}_{ijk} = \begin{cases} 1 & \text{if } \mathbf{S}_{ijk} > \tau \\ 0 & \text{otherwise} \end{cases}$$

In the example from Fig. 5, setting $\tau = 0.8$ results in *Develop Hypothesis* and *Develop Method* being flagged as normal, whereas all other activities are flagged as anomalous.

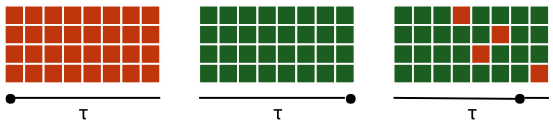

Fig. 6. Example of how an anomaly detection visualization changes with different threshold settings; the rightmost setting corresponds to how a user would likely set the slider manually.

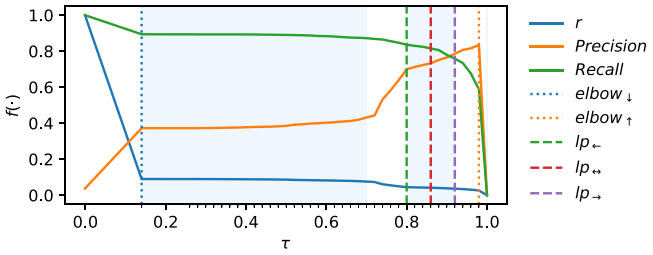

Fig. 7. Thresholds as defined by the heuristics in relation to the anomaly ratio r and its plateaus (blue intervals).

5.6. Threshold heuristic

Most anomaly detection algorithms rely on the user manually setting a threshold or define the threshold as a constant. To determine a threshold automatically, we propose a new heuristic that mimics how a human would set a threshold manually.

Let us consider the following example. A user is presented with a visualization of an anomaly detection result, for instance, a simple case overview showing all events and their attributes as depicted in Fig. 6. Anomalous attributes are shown in red and normal attributes are shown in green. The user is asked to set the threshold manually using a slider. Most people start with the slider either set to the maximum (all attributes are normal, all green) or the minimum (all attributes are anomalous, all red) and then move the slider while observing the change of colors in the visualization. Intuitively, most users fix the slider within a region where the number of shown anomalies is stable, that is, even when moving the slider to the left and right, the visualization stays the same. Furthermore, users likely prefer a threshold setting that shows significantly less anomalous than normal attributes, which corresponds to a slider setting closer to the maximum (the right side). In other words, a setting that produces less false positives.

This behavior of a human setting the threshold can be modeled based on the anomaly ratio r, which can be defined as follows, with $N = \sum_{c \in \mathcal{L}} |\hat{c}|$ denoting the number of non-padding events and $\mathbf{Y} = \theta(\mathbf{S}, \tau)$.

$$r(\theta, \mathbf{S}, \tau) = \frac{1}{NA} \sum_{i}^{C} \sum_{j}^{E} \sum_{k}^{A} \mathbf{Y}_{ijk}$$

By dividing by N we calculate the average based only on non-padding events.

Fig. 7 shows r for a run of BINetv1 on the Paper dataset. In addition to r, the figure shows the values for *Precision* and *Recall* for the anomaly class. Note that r is a discrete function that we sample for each reasonable threshold. Reasonable candidate thresholds are the distinct anomaly scores encountered in $\mathbf{S}$; other values can be disregarded. The candidate thresholds $\mathcal{T}$ are indicated by the minor ticks in Fig. 7.

To define the heuristics in the following, we first have to define the first and second order derivatives of the discrete function

r. They can be retrieved using the central difference approximation. Let $\tau_i \in \mathcal{T}$, then the derivatives are approximated by

$$r'(\theta, \mathbf{S}, \tau_i) \approx \frac{r(\theta, \mathbf{S}, \tau_{i+1}) - r(\theta, \mathbf{S}, \tau_i)}{\tau_{i+1} - \tau_i},$$
$$r''(\theta, \mathbf{S}, \tau_i) \approx \frac{r(\theta, \mathbf{S}, \tau_{i-1}) - 2\, r(\theta, \mathbf{S}, \tau_i) + r(\theta, \mathbf{S}, \tau_{i-1})}{(\tau_{i-1} - \tau_i)(\tau_i - \tau_{i+1})}.$$

To mimic the human intuition, we have to consider regions of r where the slope is close to zero. These are regions where $|r'(\theta, \mathbf{S}, \tau)| < \varepsilon$ (we chose ε to be two times the average slope of r), which we refer to as plateaus (blue regions in Fig. 7). Based on these plateaus we can now define the lowest plateau heuristic lp as follows. Let $LP = \langle \tau_0, \tau_1, \ldots, \tau_n \rangle$ be the sequence of candidate thresholds that lie within the lowest plateau, then

$$lp_{\leftarrow} = \tau_0, \quad lp_{\leftrightarrow} = \frac{1}{n}\sum_{i=0}^{n} \tau_i, \quad lp_{\rightarrow} = \tau_n,$$

corresponding to the left-most ($lp_{\leftarrow}$), the right-most ($lp_{\rightarrow}$), and the mean-centered ($lp_{\leftrightarrow}$) threshold inside the lowest plateau.

In [3], we proposed the elbow heuristic to mimic the same behavior. The definition of the elbow heuristics are given by

$$elbow_{\downarrow} = \arg\max_{\tau \in \mathcal{T}} r''(\theta, \mathbf{S}, \tau), \quad elbow_{\uparrow} = \arg\min_{\tau \in \mathcal{T}} r''(\theta, \mathbf{S}, \tau).$$

So, $elbow_{\downarrow}$ is the threshold where the rate of change of r is maximized, whereas $elbow_{\uparrow}$ is where it is minimized. With respect to r these thresholds are the points where either a steep drop ends in a plateau ($elbow_{\downarrow}$) or a plateau ends in a steep drop ($elbow_{\uparrow}$). Although $elbow_{\downarrow}$ and $elbow_{\uparrow}$ can indicate the beginning and the end of a plateau, these are not necessarily the thresholds a human would naturally pick.

To compare our results to the best possible threshold, we define the *best* heuristic by use of the F_1 score metric. The *best* heuristic is defined as follows, where $\mathbf{L}$ is the set of ground truth labels.

$$best = \arg\max_{\tau \in \mathcal{T}} F_1(\mathbf{L}, \theta(\mathbf{S}, \tau)).$$

It is important to understand that *best* can only be used if the labels are available at runtime. However, in most cases, anomaly detection is an unsupervised problem, and hence no labels are available.

It might be beneficial to apply different thresholds to different dimensions of $\mathbf{S}$. For example, it might be sensible to set a different threshold for the user attribute than the activity because the inherent probability distribution can be different. This is possible by using ":" to apply heuristics on cross-sections of $\mathbf{S}$ by using index notation. Let $h \in \{lp_{\leftarrow}, lp_{\leftrightarrow}, lp_{\rightarrow}, elbow_{\downarrow}, elbow_{\uparrow}, best\}$ then we can define the following threshold strategies

$$h = h(\mathbf{S}),$$
$$h^{(a)} = (\tau_i) = h(\mathbf{S}_{:,:,i}),$$
$$h^{(e)} = (\tau_i) = h(\mathbf{S}_{:,i,:}),$$
$$h^{(ea)} = (\tau_{ij}) = h(\mathbf{S}_{:,i,j}).$$

We only explicitly show the parameter $\mathbf{S}$ for clarity, other parameters are set according to the definition of the chosen heuristic.

Thus, $h^{(a)} \in \mathbb{R}^A$ returns a tensor that holds one threshold for each attribute in an event, whereas $h^{(e)} \in \mathbb{R}^E$ holds a threshold for each event position in a case. Lastly, $h^{(ea)} \in \mathbb{R}^{E \times A}$ combines the two ideas and gives a threshold for each combination of event position and attribute. In other words, instead of applying the threshold heuristic h once for all dimensions of $\mathbf{S}$, we apply it multiple times for different cross-sections of $\mathbf{S}$, obtaining multiple different thresholds.

6. Evaluation

We evaluated BINet on all 44 event logs and compared it to eight state-of-the-art methods. Two methods from [10]: a sliding window approach (t-STIDE+) [11]; and the one-class SVM (OC-SVM). Two methods from [24]: the Naive algorithm and the Sampling algorithm. Furthermore, we provide the results of the denoising autoencoder (DAE) approach from [17]. Lastly, we compared BINet to the approach from [8] (Likelihood). Naive and Likelihood set the threshold statically, so we extended the approaches to support the use of our external threshold heuristics. These extensions are referred to by Naive+ and Likelihood+. For all non-deterministic methods (i.e., DAE, BINet, and Sampling), we executed five independent runs to account for randomness.

For the OC-SVM, we relied on the implementation of scikit-learn[6] 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. Sampling, Likelihood, Baseline, and the OC-SVM do not rely on a manual setting of the threshold and were unaltered. t-STIDE+ is an implementation of the t-STIDE method from [11], which we adapted to support the data perspective (see [17]). Naive+ is an implementation of Naive that removes the fixed threshold of 0.02 and sets the threshold according to the heuristic. Likelihood+ implements the first part of Likelihood (the generation of the extended likelihood graph from the log) and replaces the threshold algorithm with the aforementioned heuristics.

In the last section, we described the intuition of setting separate thresholds using different strategies (e.g., one threshold per attribute). To decide on the best strategy, we evaluate the four strategies (h, $h^{(e)}$, $h^{(a)}$, and $h^{(ea)}$) for all synthetic datasets and all methods that support the heuristics, with $h = best$. The results of the experiments in Fig. 8 indicate that, indeed, it is sensible to set separate thresholds for individual attributes. Interestingly, we also find that setting a single threshold yields similar results. Setting a threshold per event or per event and attribute does perform significantly worse.

Next, we repeated the same experiment for all of the aforementioned heuristics and using $h^{(a)}$ as the strategy. The results can be seen in Fig. 9. Intriguingly, the lowest plateau heuristics perform best for all methods except the DAE. Furthermore, it seems to work best to choose the mean-centered threshold within the lowest plateau ($lp_{\leftrightarrow}$).

Based on the results of the preliminary experiments, we set $h = lp_{\leftrightarrow}^{(a)}$ as the heuristic for the following experiments for all methods apart from the DAE, for which we set $h = elbow_{\uparrow}^{(a)}$. For Likelihood, Sampling, Naive, and OC-SVM we use the internal threshold heuristics.

The overall results are shown in Fig. 10. For the real-life datasets, we do not have complete information, and hence the F_1 score is not a good representation of the quality of the detection algorithms. However, because we compare all methods on the same basis, the results are still meaningful. Furthermore, we only know about the artificial anomalies inside the real-life datasets, and therefore we expect a high recall of the artificial anomalies, whereas we expect a low precision because the dataset likely contains natural anomalies which are not labeled.

This theory is confirmed by the results in Fig. 10. Note also that the recall scores for both the synthetic and the real-life datasets are very similar, indicating comparable performance (for artificial anomalies) on both dataset types.

Finally, we find that BINetv1 works best for the synthetic datasets, whereas the field is mixed for the real-life datasets. However, all three BINet versions perform better than the other

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

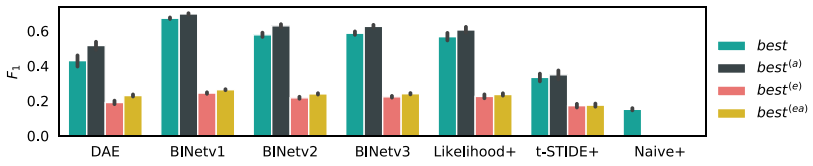

Fig. 8. Average F_1 score by method and strategy over all synthetic datasets, using *best* as the heuristic.

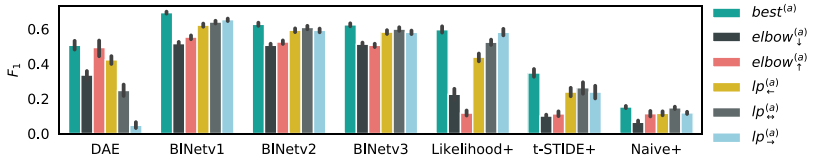

Fig. 9. Average F_1 score by method and heuristic over all synthetic datasets, using $h^{(a)}$ as the strategy.

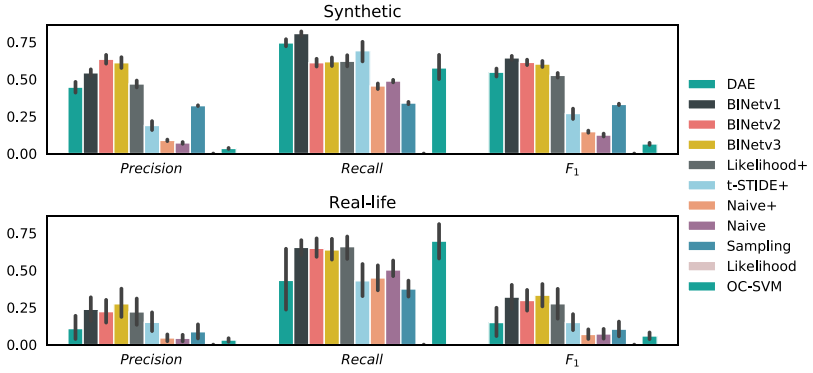

Fig. 10. Average *Precision*, *Recall*, and F_1 by dataset type over all datasets; error bars indicate variance over datasets with different numbers of attributes and multiple runs.

methods. DAE performs significantly worse on real-life data because it ran out of memory on some of the bigger datasets. Therefore, DAE has been penalized defining precision and recall to be zero for these runs.

Detailed results can be found in Table 2, which also gives results for case level (i.e., only anomalous cases have to be detected, not the attributes). An interesting observation is that t-STIDE+ performs best on BPIC12 when evaluating on case level. This might be attributed to BPIC12 being a dataset without event attributes (the only one in the corpus). On attribute level, Likelihood+ is marginally better than BINet on BPIC13. For all other datasets, BINet shows the best performance.

All results are given using the heuristics described above. Labels were not used in the process. Additional material (e.g., evaluation per perspective, per dataset, runtime) can be found in the code repository.

To validate the significance of the results, we apply the non-parametric Friedman test [30] on average ranks of all methods based on F_1 score for all synthetic datasets. Then, we apply the Nemenyi post-hoc test [31], as demonstrated in [32], to calculate pairwise significance.

Fig. 11 shows a critical difference diagram, as proposed in [32], to visualize the results with a confidence interval of 95 percent. Based on the critical difference, we recognize that BINetv1 performs significantly better than all other methods, except BINetv2 and BINetv3. That is, all three BINet versions lie in the same significance group with respect to the critical difference. DAE lies in the same group as BINetv2 and BINetv3, and Likelihood+ in the same as DAE and BINetv3. All other methods lie more than the critical difference away from the three BINets.

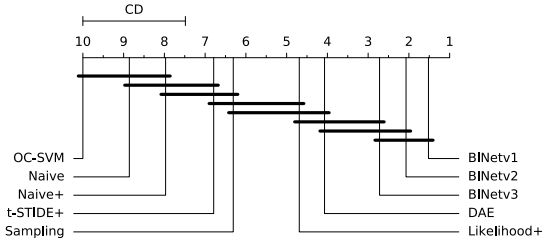

Fig. 11. Critical difference diagram for all methods on all synthetic datasets; groups of methods that are not significantly different (at $p = 0.05$) are connected (cf. [32]).

7. Classifying anomalies

Until now, we have not utilized the predictive capabilities of BINet. Using the probability distribution output of the softmax layers in conjunction with the binarized anomaly scores, we can define simple rules to infer the type of anomaly.

We use the term predictions to denote all possible attribute values with an anomaly score above the threshold. Here, the *Shift* class becomes relevant because it indicates the place where an early or late execution would belong. For each anomalous attribute (according to BINet), we apply the following rules in order.

1. *Skip*: If all predictions do not appear in the case
2. *Insert*: If one of the predictions appears in the case and that occurrence has not been flagged as anomalous
3. *Rework*: If the same activity appears earlier in the case and is not flagged as anomalous

Please cite this article as: T. Nolle, S. Luettgen, A. Seeliger et al., BINet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458, https://doi.org/10.1016/j.is.2019.101458.

Table 2
F_1 score over all datasets by detection level and method; best results (before rounding) are shown in bold typeface.

Level	Method	Paper	P2P	Small	Medium	Large	Huge	Gigantic	Wide	BPIC12	BPIC13	BPIC15	BPIC17	Anonymous
Case	OC-SVM [11]	0.49	0.27	0.25	0.29	0.24	0.23	0.29	0.31	0.55	0.24	0.26	0.35	0.10
	Naive [6]	0.50	0.48	0.49	0.39	0.41	0.40	0.34	0.44	0.55	0.21	0.17	0.31	0.16
	Sampling [6]	0.50	0.49	0.49	0.47	0.49	0.49	0.45	0.49	0.55	0.21	0.17	0.32	0.23
	Likelihood [8].	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Naive+	0.50	0.48	0.49	0.44	0.49	0.45	0.38	0.47	0.55	0.21	0.17	0.28	0.15
	t-STIDE+ [16]	0.40	0.51	0.53	0.43	0.45	0.45	0.41	0.47	**0.68**	0.32	0.29	0.32	0.22
	Likelihood+	**0.85**	0.74	0.76	0.72	0.73	0.73	0.73	0.73	0.62	0.44	0.33	0.45	0.51
	DAE [16]	0.46	0.71	0.72	0.71	0.71	0.70	0.63	0.70	0.60	0.21	0.00	0.30	0.35
	BINetv1	0.74	**0.77**	**0.78**	**0.75**	**0.75**	**0.75**	**0.74**	**0.76**	0.62	0.41	0.37	**0.51**	**0.51**
	BINetv2 [3]	0.76	0.77	0.77	0.72	0.71	0.70	0.68	0.73	0.61	0.40	**0.38**	0.43	0.45
	BINetv3	0.79	0.77	0.76	0.71	0.69	0.69	0.66	0.74	0.66	**0.45**	0.36	0.49	0.50
Attribute	OC-SVM [11]	0.09	0.06	0.05	0.08	0.04	0.05	0.07	0.09	0.05	0.06	0.01	0.09	0.30
	Naive [6]	0.13	0.15	0.14	0.12	0.09	0.11	0.09	0.16	0.05	0.05	0.01	0.10	0.39
	Sampling [6]	0.33	0.33	0.34	0.32	0.34	0.34	0.31	0.32	0.08	0.07	0.01	0.14	0.39
	Likelihood [8]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Naive+	0.13	0.16	0.15	0.16	0.13	0.13	0.13	0.18	0.05	0.05	0.01	0.09	0.33
	t-STIDE+ [16]	0.28	0.33	0.32	0.25	0.25	0.26	0.19	0.28	0.40	0.12	0.05	0.17	0.39
	Likelihood+	**0.74**	0.61	0.63	0.58	0.57	0.58	0.55	0.57	0.35	0.29	0.07	0.28	0.63
	DAE [16]	0.25	0.61	0.61	0.56	0.56	0.55	0.46	0.56	0.06	0.09	0.00	0.24	0.52
	BINetv1	0.64	**0.68**	**0.69**	**0.65**	**0.65**	**0.65**	**0.64**	**0.65**	0.42	0.28	**0.21**	**0.38**	0.60
	BINetv2 [3]	0.67	0.65	0.67	0.60	0.59	0.60	0.56	0.60	0.34	0.25	0.19	0.29	0.55
	BINetv3	0.67	0.65	0.66	0.59	0.57	0.59	0.54	0.61	**0.48**	**0.29**	0.19	0.35	**0.63**

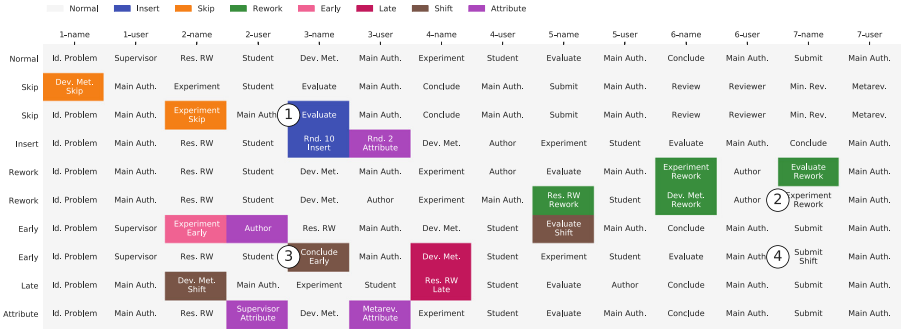

Fig. 12. Classification of anomalies on the Paper dataset based on anomaly scores from BINetv1 using $h = lp^{(a)}_{\rightarrow}$; colors indicate the prediction of the classifier (see legend) and actual classes are shown as text within the cells.

4. *Shift*: If one of the predictions appears earlier or later in the case and is flagged as anomalous
5. *Late*: If the activity appears earlier in the predictions and is flagged as anomalous
6. *Early*: If the activity appears later in the predictions and is flagged as anomalous
7. *Attribute*: Trivially, all anomalous attributes in the data perspective are of type *Attribute*

The result of the classification is visualized in Fig. 12. Interestingly, this set of simple rules performs remarkably well. Anomaly classes inferred by the rules are indicated by the color of the cells, whereas ground truth labels are shown as text in the cells (we omit *Normal* for clarity). Incidentally, this visualization also depicts the binarized anomaly scores according to the threshold since each classified attribute is also an anomalous attribute.

We also included some examples where the classification is incorrect. An interesting case is the second *Skip* example because *Evaluate* has also been marked as anomalous ①. As we have defined in Fig. 2, *Evaluate* is an activity that always eventually follows *Develop Method*. However, *Develop Method* was skipped. Therefore, BINet is never presented with the causing activity, and hence regards *Evaluate* as anomalous.

A different example is that BINet misses the third *Rework* activity in the second example ②. We observed many of these errors, and they are related to the fact that BINet is conditioned on the last input activity and forgets the history of the case (forgetting problem). Under these conditions, *Develop Method* indeed directly follows *Research Related Work*, and hence BINet misses it. This forgetting problem is something we want to address in the future.

The most interesting case is the second *Early* example ③. Here, BINet misclassifies the *Early* activity as *Shift*. Upon closer inspection, we realize that this is indeed a way of explaining the anomaly, albeit not the one the labels indicate. With respect to *Develop Method*, *Conclude* indeed occurs too early in the case. Nevertheless, BINet fails to detect the actual Shift point ④, and thus the rules do not match the pattern correctly.

Using this simple set of rules, we ran the classifier on all synthetic datasets, using BINetv1 as the anomaly detection method and $h = lp^{(a)}_{\rightarrow}$ (the best heuristic for BINetv1). Fig. 13 shows the results in a confusion matrix. Note that the classifier uses as input the anomaly detection result of BINet, and hence can never distinguish normal from anomalous examples. Thus, the errors for the normal class are based on the errors BINet commits in the anomaly detection task. Disregarding these errors, this results in a macro average F_1 score of 0.83 over all datasets for the classification task. Since BINetv1 reaches an average F_1 score of 0.64 on the detection task, this result is truly impressive, considering the simplicity of the rules. For the joint task (detection and classification), BINetv1 reaches an average F_1 score of 0.70.

In Fig. 13, we notice that BINet errs especially often for *Rework*, *Early*, *Late*, and *Shift*. This is connected to the forgetting problem mentioned earlier. Remember that *Rework*, *Early*, and *Late* anomalies are affecting sequences of events, that is, up to 3 events can be part of a rework anomaly, and up to 2 events can

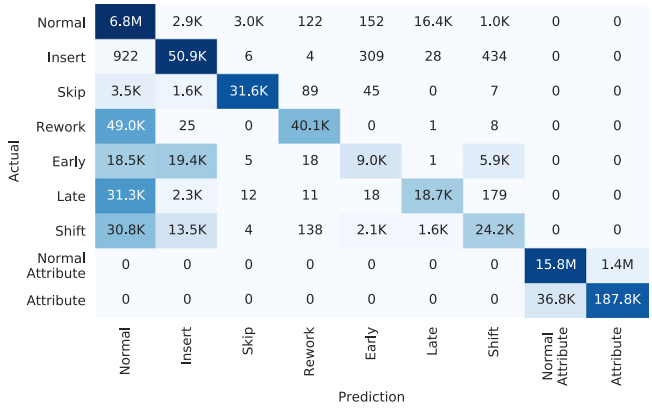

Fig. 13. Confusion matrix for all runs of BINetv1 on synthetic datasets with $h = lp^{(a)}_{\rightarrow}$; color indicates distribution of actual class.

be executed early or late. In the case of Rework, we have already seen an example in Fig. 12, where BINet misclassifies because of forgetting. Fig. 13 confirms that this error occurs quite often since more than 50 percent of all *Rework* anomalies are misclassified as *Normal*. All of these misclassifications happen in cases where a sequence of more than one event has been executed again.

Not every repetition of an event is classified as a *Rework*, only the events identified to be anomalous are. Hence, a repeated event (a loop in the process) is classified as *Normal*, if BINet has learned that it can occur multiple times in a case. In the Paper process, this is demonstrated by the second *Submit* event, which can naturally occur multiple times in a case. In Fig. 13 we can see that BINet very rarely classifies a *Normal* activity as *Rework* (never in the Paper datasets); thus, we can conclude that BINet has learned to model the loop in the Paper process correctly.

As with *Rework*, we can explain the errors for *Early* and *Late* by the same argument. However, these two classes are also often misclassified as *Insert* or *Shift*. The latter goes back to the second *Early* example of Fig. 12 and the ambiguity of labels. The *Insert* errors are of a different kind. They occur because the rule set is not taking into account that multiple events can be executed early or late. We expect to find an early execution somewhere later in the case as the prediction; however, this can only be true for the first event of an early sequence. The same argument can be made for late executions.

The *Shift* errors are related to the fact that the random process models often allow skipping of events. When a *Shift* anomaly is applied to an optional event, BINet, or any other method, has no means of finding the anomaly. This could be accounted for by altering the generation algorithm.

Nevertheless, the results indicate that a simple set of rules can be used to classify the anomalies types we have introduced before. Note that this is a white-box approach and a human user can easily interpret the resulting classification. Even though the different classes are only a subset of all anomaly types, they do cover many of the anomalies encountered in real-life business processes. Importantly, it is quite easy to define new rules for new types of anomalies based on the predictive capabilities of BINet.

8. Discussion

As part of the discussion, we would like to address three matters: The performance on the real-life datasets, the differences between the BINet versions, and the difference in datasets compared to the original publication [3].

Firstly, the difference in performance when comparing synthetic and real-life datasets seems substantial. Yet, the real-life datasets contain natural anomalies that are not labeled and therefore incorrectly represent normal behavior. If BINet correctly detects these anomalies, this is considered to be wrong because the labels indicate otherwise. The results indicate that BINet detects most of the artificial anomalies, which leads to high recall scores. These recall scores are similar to the recall scores reached on the synthetic sets, suggesting that BINet performs equally good on both the synthetic and the real-life datasets. Conversely, we can observe low precision scores on the real-life data. BINet detects natural anomalies that are labeled as normal, thereby affecting the precision. Based on our domain knowledge about the BPIC datasets, we believe most of these misclassified anomalies to be actual anomalies that BINet correctly found, but the lack of labels compromises the precision score.

Secondly, it appears that BINetv1 outperforms BINetv2 and BINetv3. We would have expected a different outcome because BINetv2 and BINetv3 can model causal dependencies. BINetv1 performs particularly well on the synthetic data. We suspect that this is attributed to the generation algorithm and the number of event attributes in the datasets. The design of BINetv2 and BINetv3 is targeted at improving the predictions of event attributes. This design can only be utilized if more than one event attribute exists. In some datasets, this was not the case. However, for the real-life datasets, where more event attributes are available, BINetv3 performs best. Additionally, it appears that for the synthetic datasets the historical information of past events is sufficient to accurately predict the next event. This is not the case for the real-life datasets. In the future, we intend to address this by altering the generation algorithm to generate stronger causal dependencies.

Thirdly, the total number of datasets used in this work is different from the number used in [3]. In [3] we generated multiple redundant logs from the same randomly generated process model. However, we found that a higher number of datasets did not affect the overall evaluation results. Instead, we decided to include more sophisticated anomalies, resulting in a more challenging, diverse, and realistic corpus, albeit a smaller one.

9. Conclusion

In this paper, we presented three versions of BINet, a neural network architecture for multi-perspective anomaly classification in business process event logs. Additionally, we proposed a set of heuristics for setting the threshold of an anomaly detection algorithm automatically, based on the anomaly ratio function. Finally, we demonstrated that a simple set of rules could be used for classification of anomaly types, solely based on the output of BINet.

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, nor does it depend on a clean dataset. Utilizing the lowest plateau heuristic, BINet's internal threshold can be set automatically, reducing manual workload and allowing fully autonomous operation. It utilizes both the control flow and the data perspective. Furthermore, BINet can cope with concept drift, for it can be set up to train on new cases in real-time continuously.

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. Specifically, on the synthetic datasets, BINet's performance surpasses those of other methods by an order of magnitude. We demonstrated that BINet also performs well on the real-life datasets because BINet shows high recall of the artificial anomalies introduced to the original real-life logs.

Please cite this article as: T. Nolle, S. Luettgen, A. Seeliger et al., BINet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458, https://doi.org/10.1016/j.is.2019.101458.

Although the results look very promising, there is still room for improvement. For example, BINet suffers from forgetting when sequences of events are repeated in a case. This issue can be addressed in future work, for example, by using a special attention layer. An interesting option is the use of a bidirectional encoder–decoder structure to read in cases both from left to right and from right to left. Hereby, sequences of repeated events can be identified from two sides, as opposed to just one.

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

Declaration of competing interest

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

Acknowledgments

This project [522/17-04] is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE, Germany, 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 "R2PA" [01IS17050].

References

[1] W.M.P. van der Aalst, Process Mining: Data Science in Action, Springer, 2016.
[2] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, Elsevier, 2011.
[3] T. Nolle, A. Seeliger, M. Mühlhäuser, BINet: Multivariate business process anomaly detection using deep learning, in: Proceedings of the 16th International Conference on Business Process Management – BPM'18, 2018, pp. 271–287.
[4] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[5] L. Wen, W.M.P. van der Aalst, J. Wang, J. Sun, Mining process models with non-free-choice constructs, Data Min. Knowl. Discov. 15 (2) (2007) 145–180.
[6] F. Bezerra, J. Wainer, W.M.P. van der Aalst, Anomaly detection using process mining, in: Proceedings of the 10th International Workshop on Enterprise, Business-Process and Information Systems Modeling – BPMDS'09, Springer, 2009, pp. 149–161.
[7] F. Bezerra, J. Wainer, Anomaly detection algorithms in logs of process aware systems, in: Proceedings of the 23rd Annual ACM Symposium on Applied Computing – SAC '08, 2008, pp. 951–952.
[8] K. Böhmer, S. Rinderle-Ma, Multi-perspective anomaly detection in business process execution events, in: Proceedings of on the Move to Meaningful Internet Systems, OTM'16, Springer, 2016, pp. 80–98.
[9] M.A.F. Pimentel, D.A. Clifton, L. Clifton, L. Tarassenko, A review of novelty detection, Signal Process. 99 (2014) 215–249.
[10] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection for discrete sequences: A survey, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 823–839.
[11] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: Alternative data models, in: Proceedings of the 1999 IEEE Symposium on Security and Privacy – SP'99, 2008, pp. 133–145.
[12] C. Wressnegger, G. Schwenk, D. Arp, K. Rieck, 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, 2013, pp. 67–76.
[13] B. Schölkopf, R.C. Williamson, A.J. Smola, J. Shawe-Taylor, J.C. Platt, et al., Support vector method for novelty detection, in: Proceedings of the 12th International Conference on Neural Information Processing Systems – NIPS'99, Vol. 12, 1999, pp. 582–588.
[14] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[15] N. Japkowicz, Supervised versus unsupervised binary-learning by feedforward neural networks, Mach. Learn. 42 (1) (2001) 97–122.
[16] T. Nolle, A. Seeliger, M. Mühlhäuser, Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders, in: Proceedings of the 19th International Conference on Discovery Science – DS'16, Springer, 2016, pp. 442–456.
[17] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Analyzing business process anomalies using autoencoders, Mach. Learn. 107 (11) (2018) 1875–1893.
[18] J. Evermann, J.-R. Rehse, P. Fettke, A deep learning approach for predicting process behaviour at runtime, in: Proceedings of the 14th International Conference on Business Process Management – BPM'16, Springer, 2016, pp. 327–338.
[19] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140.
[20] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Proceedings of the 29th International Conference on Advanced Information Systems Engineering – CAiSE'17, Springer, 2017, pp. 477–492.
[21] E. Marchi, F. Vesperini, F. Eyben, S. Squartini, B. Schuller, A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks, in: Proceedings of the 40th International Conference on Acoustics, Speech and Signal Processing – ICASSP'15, 2015.
[22] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, G. Shroff, LSTM-Based encoder-decoder for multi-sensor anomaly detection, 2016, arXiv preprint arXiv:1607.00148.
[23] A. Burattin, PLG2: Multiperspective processes randomization and simulation for online and offline settings, 2015, arXiv preprint arXiv:1506.08415.
[24] F. Bezerra, J. Wainer, Algorithms for anomaly detection of traces in logs of process aware information systems, Inf. Syst. 38 (1) (2013) 33–44.
[25] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014, arXiv preprint arXiv:1406.1078.
[26] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, arXiv preprint arXiv:1301.3781.
[27] P. De Koninck, S. vanden Broucke, J. De Weerdt, Act2vec, trace2vec, log2vec, andmodel2vec: Representation learning for business processes, in: Proceedings of the 16th International Conference on Business Process Management – BPM'18, Springer, 2018, pp. 305–321.
[28] D. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[29] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning – ICML'15, 2015, pp. 448–456.
[30] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Amer. Statist. Assoc. 32 (200) (1937) 675–701.
[31] P. Nemenyi, Distribution-Free Multiple Comparisons Ph.D. thesis, Princeton University, 1963.
[32] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.