Predicting Remaining Cycle Time from Ongoing Cases: A Survival Analysis-based Approach

Fadi Baskharon[1], Ahmed Awad[1,2][0000−0003−1879−1026] and Chiara Di Francescomarino[3]

[1] Information Technology and Computer Science School, Nile University, Giza, Egypt
{f.zaki, aawad}@nu.edu.eg
[2] University of Tartu, Tartu, Estonia
[3] Fondazione Brunno Kessler, Trento, Italy dfmchiara@fbk.eu

Abstract. Predicting the remaining cycle time of running cases is one important use case of predictive process monitoring. Different approaches that learn from event logs, e.g., relying on an existing representation of the process or leveraging machine learning approaches, have been proposed in literature to tackle this problem. Machine learning-based techniques have shown superiority over other techniques with respect to the accuracy of the prediction as well as freedom from knowledge about the underlying process models generating the logs. However, all proposed approaches learn from *complete* traces. This might cause delays in starting new training cycles as usually process instances might last over long time periods of hours, days, weeks or even months.
In this paper, we propose a machine learning approach that can learn from *incomplete* ongoing traces. Using a time-aware survival analysis technique, we can train a neural network to predict the remaining cycle time of a running case. Our approach accepts as input both complete and incomplete traces. We have evaluated our approach on different real-life datasets and compared it with a state of the art baseline. Results show that our approach, in many cases, is able to outperform the baseline approach both in accuracy and training time.

Keywords: Predictive process monitoring · Remaining time prediction · Survival analysis · Incomplete traces

1 Introduction

Predictive process monitoring [7] is a sub-field of process mining that is concerned with predicting an outcome of interest while an execution is still running, for instance with the purpose of proactively taking corrective actions before things go wrong. Different types of outcomes can be predicted, as for instance, the remaining time for a case to finish [15,11,14], which activity to be executed next [3,9], or the fulfillment of a certain goal [5,4]. Different techniques have been used to tackle the prediction challenge, such as machine learning, statistical methods, annotated transition systems and hybrid approaches [7]. However, all these techniques require as a major input a history of *complete* cases, in order

to train a model that will be used at run-time to predict the respective outcome for running cases.

Among the techniques for predictive process monitoring, machine learning and deep learning-based approaches have shown superiority with respect to the accuracy of the prediction [14,17]. Yet, as known, training deep learning models requires more resources and a large dataset to obtain better results. Moreover, process instances generally run for long time, days, weeks or even months. A direct threat in this case is the possibility of concept drifts [6] that render the currently-used model for prediction useless and triggers the need to train a new model on a larger set of traces containing newly complete traces. In such case, the retraining has to be delayed until a sufficiently large set of newly completed cases has been collected.

A main limitation of contemporary predictive monitoring techniques is the need for *complete* traces to train their models. This causes delays of retraining cycles until new completed cases are collected. In this paper, we alleviate this limitation by allowing learning from ongoing cases, i.e. *incomplete* traces, by employing survival analysis techniques [2]. Treating incomplete traces as censored data, we are able to train a neural network to predict the remaining time for a running case. Compared to the state-of-the-art, our results show at least comparable accuracy to methods that require complete traces with better results on several data sets; additionally, our training takes much less time to complete.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 provides the necessary details about survival analysis and the specific techniques we employ. Our contributions are detailed in Section 4 for the encoding of (incomplete) traces and the architecture of the neural network and Section 5 for the experimental evaluation and comparison to the baseline method. Finally, Section 6 concludes the paper.

2 Related Work

In general, literature on predictive process monitoring can be classified based on the type of targeted prediction: next activity, outcome, delays or the remaining cycle time. Due to space limitations, we will discuss the literature related to the prediction of the remaining cycle time, as this is the focus of our work. We refer the user to [7] for a survey on the wider domain of predictive process monitoring.

A recent survey by Verenich et al. [17] has benchmarked the different approaches for predicting the remaining cycle time of a running instance. In general, prediction approaches have been categorized as generative or discriminative. Generative approaches are process-aware, that is, they require a pre-existing representation of the process whose execution generates the traces. Discriminative approaches, instead, are process-agnostic and can learn directly from traces.

Concerning the generative approaches, in [15] the authors discover a transition system from the log and augment it with information about the remaining time of cases; in [11] stochastic Petri nets are leveraged for making predictions;

in [18], flow analysis is used to aggregate the remaining cycle time on the case level over its individual activities.

Discriminative methods can rely on different approaches. Some of them leverage non-parametric regression models [16,12], others propose clustering-based techniques, as the work in [1], while others rely on neural networks [14] for the prediction of the remaining cycle time. We will use this latter approach and the work in [14] as baseline for our work.

2.1 Baseline Approach

The baseline approach in [14] predicts the next activity in a process as well as its timestamp. Each event occurring in the trace is transformed into a feature vector $x_1, ..., x_k$ to be fed as input to an LSTM network as follows: (i) **activity type (A)**, i.e., the type of the activity in a one-hot-encoding representation; (ii) **delta t** (fv_{t1}), i.e., the time between the previous and the current event in the trace that allows the network to learn the time dependencies between the process' events; (iii) **two time-based features,** (fv_{t2}) **and** (fv_{t3}), that correspond to the hour of the event within the day in 24-hour format and the hour of the event since the start of the week, respectively, so as to learn when the event has happened with respect to a working day or a working week. The LSTM has two outputs: (i) O_a^k that corresponds to a one-hot-encoding representation of the type of the next event with an extra bit representing whether or not this event occurs at the end of the case; and (ii) O_t^k representing the relative time difference between the current event and the next event. The remaining cycle time can be computed by summing O_t^k for all the events from the current event until the last event of the trace. This model needs to be trained with complete traces, as the model needs to learn the process sequence, as well as when the sequence ends. The model also suffers when dealing with sequences containing loops, as loops cause the model to predict overly long sequences.

3 Background

In this section we will briefly introduce the notations we use throughout the rest of the paper and a quick overview on survival analysis as the inspiring method to the contribution of this paper.

3.1 Events, Trace, Logs

The occurrence of an event is the manifestation of the evolution of a running process. Each event contains at least three pieces of information: a reference to the activity, a reference to the process instance, and a timestamp. Events can have more information, e.g., cost, the human resource who executed it, and the lifecycle transition, e.g., *started, completed, etc.* In this paper, we require just the three basic pieces of information.

Let $\mathcal{A}$ be the set of all activities that an event can reference. The set $\mathcal{A}_{end} \in \mathcal{A}$ contains end activities. Additionally, the set $\mathcal{T}$ is the time domain and $\mathcal{C}$ is the universe of case identifiers. Finally, the set $\mathcal{E}$ is the universe of events. Thus, an event $e \in \mathcal{E}$ represents the execution of some activity $a \in \mathcal{A}$ within a case $c \in \mathcal{C}$ that occurred at time $t \in \mathcal{T}$. A shorthand for these notations are $e.a, e.c, e.t$ respectively.

Definition 1 (Trace). *A trace is a finite non-empty sequence of events $\sigma \in \mathcal{E}*$. $|\sigma|$ defines the length of the trace. $\sigma_i \in \mathcal{E}$ is the event at position $i, 1 \leq i \leq |\sigma|$. A trace is called* complete *if and only if $\sigma_{|\sigma|}.a \in \mathcal{A}_{end}$, otherwise it is* incomplete.

A prefix of a trace is defined as a function $pre : \mathcal{E} * \times \mathcal{N} \rightarrow \mathcal{E}*$ that returns a sub-sequence of a trace σ up to and including the event at position i in the trace.

A log $L \subset \mathcal{E}*$ is a set of traces where each trace appears at most once.

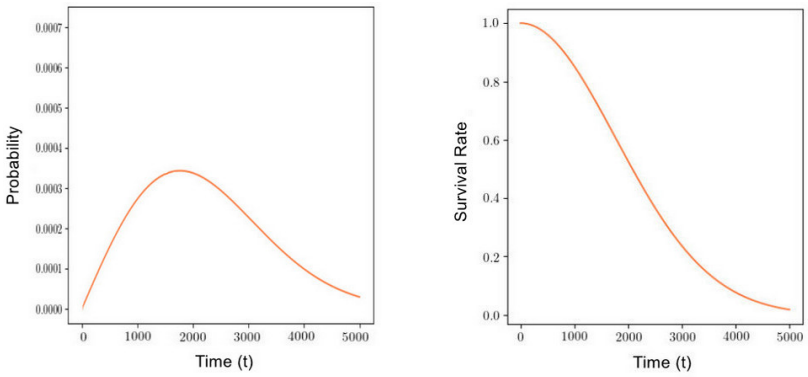
3.2 Survival Analysis and Censored-learning

Survival analysis models [2] are key players in statistical studies that focus on analyzing the waiting duration or the remaining time until an event happens, such as a death, failure, churn, or any other event of interest. These kinds of models are capable of answering even more complex questions like "What is the probability that an event does/doesn't happen within an amount of time T?", "What is the probability distribution of the event occurrence over time?" or "What is the proportion of a population which will survive passed a certain time?".

Let T be a random variable denoting the waiting time until the occurrence of an event, survival models provide information about the probability density function $f(t) = Pr(T = t)$ and the survival function $S(t) = Pr(T > t)$, among other functions. The probability density function gives information about the likelihood of the occurrence of the event at time t. The survival function represents the probability of surviving until a certain time t without experiencing the event of interest. (See Figure 1 for the difference between the two functions).

There are three types of survival models: Non-parametric, semi-parametric, and parametric approaches [13]. Only the latter has the ability to extrapolate or predict beyond the data time limit, since it fits the survival curve to a time distribution. There are many suitable distributions to represent a time-based random variable T as stated in [10], such as exponential, log-logistic, log-normal, gamma, Poisson, Geometric, and the Weibull distribution.

A capability of survival models is the ability to learn from unobserved events which are called "censored data". That is, the collected data represent a snapshot in time, where some samples have experienced the event of interest, but most of the samples have not. Yet, these samples contain useful information telling that "at least we have not observed an event until the time t".



(a) f(t) (b) S(t)

Fig. 1. Probability density function Vs. Survival function

Fig. 2 illustrates the random variable T that represents the time to an event of interest. We say that T is observed when we observe the actual waiting time within the period of our study, hence we call it **un-censored (observed) event** $T \in [0, t]$ ($e1$ in Figure 2), where the event of interest is exactly observed. We say that T is censored when we partially know about the event occurrence time. We have three types of censored data [13]:

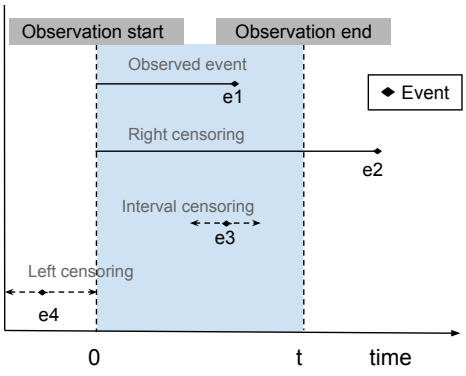


Fig. 2. Observed Vs. Censored events

Right-censored event $T \in [t, \infty)$ refers to cases that have started with the study but the event of interest was not observed during the study time ($e2$ in Figure 2). Right-censored events are useful under the assumption of *non-informative censoring*. That is, censoring is independent of the likelihood of the occurrence of the event of interest. In other words, we assume that the cases whose data are censored would have the same distribution of time to event if they were actually observed. **Interval-censored** event $T \in [a, b],\ where\ 0 < a, b < t$, rather than knowing the exact time of the event, all we know is that the event occurred between two known time points ($e3$ in Figure 2). **Left-censored** Mathematically, left censored is no different from interval censoring. It indicates that the event occurred at some point prior to the period of study. ($e4$ in Figure 2).

In the context of predictive process monitoring, only observed events and right-censored events are relevant. That is because, for the set of traces used for learning, we either know the exact end time of the trace, or we don't capture the trace end in our study (in case of incomplete traces).

4 Learning from Incomplete Traces

Traditional survival models are designed to handle records with static features that affect the waiting time. It is not meant to handle time varying features. In the context of predictive process monitoring, cases are time varying as the same activity may have different durations across different cases.

Martisson [8] proposed a model that benefits from the survival analysis interpretation and is able to deal with the time varying features by training a gated recurrent neural network (GRU) that captures the temporal relations between the time steps. The network is trained to predict the parameters of the Weibull distribution by optimizing the log of a special likelihood function to consider both observed and censored events, as explained in Section 4.2. The Weibull distribution turns out to be a suitable choice since it is controlled by two parameters α and β, that makes it flexible to interpret complex outputs, and because of its ability to fit both discrete and continuous problems.

In our work, we adapt the network architecture of the baseline method [14] by changing the encoding of the input traces to account for incomplete traces (Definition 1) and adapt the likelihood function from [8] to train the network to predict the parameters of the Weibull distribution that fits the time to the end event of a case. We discuss these two steps in detail in the following subsections.

4.1 Neural Network Setup

The problem of measuring the remaining time till a process ends can be tackled using a similar approach like [8]. However, the original work was designed to predict the waiting time to recurrent events, e.g. the time to the next failure of a machine. In our case, we are interested in the time until a process instance ends.

To adapt the approach to the prediction of the remaining time to an end event, we kept the same loss function and Weibull parameters as in [8]. Then, we adapted the network design from many-to-many to many-to-one to account for non-recurrent end events.

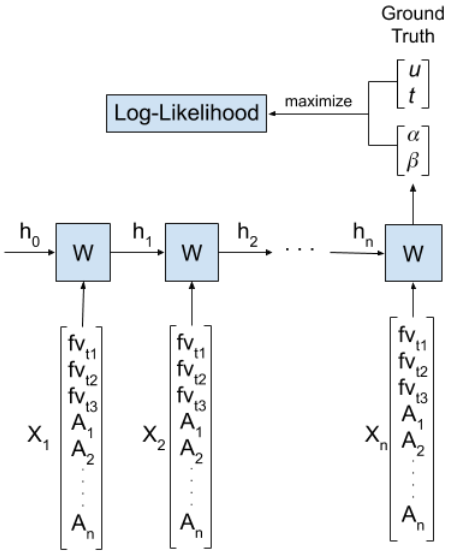


Fig. 3. Network architecture

Considering a log $\mathcal{L}$, we use N_{max} to denote the length of the longest trace $\sigma_l \in \mathcal{L}$. We train a model for each possible prefix $pre(\sigma, p)$ (Definition 1), where $\sigma \in \mathcal{L}$ and $p \in \{2, 3, 4, \ldots, N_{max} - 1\}$.

As shown in Figure 3, the model consists of an input layer $[X_1, X_2, ...X_n]$, which is the vector representation of the prefix, as explained in Section 2.1,

connected to two GRU layers; and a dense layer of 2 neurons for the output representing the α and the β of the distribution of the random variable T for a given trace. Having the Weibull distribution, we need to find the most likely value in the curve, which turns out to be the mode of the distribution.

4.2 Optimization Function

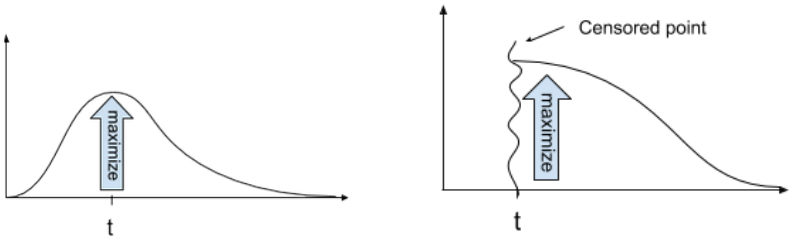
Let T be a random variable for the waiting time having some parameters θ, and t the observed cycle time, we are interested in the negative likelihood as a loss function. In other words, we aim at maximizing the likelihood of T being around the true observation t for complete traces or at pushing it to the right beyond the censored point t in case of incomplete traces:

$$\mathcal{L}(t,\theta) \propto \begin{cases} P(T=t|\theta) & \text{Observed events (complete trace)} \\ P(T>t|\theta) & \text{Censored events (incomplete trace)} \end{cases} \tag{1}$$

This can be expressed mathematically as follows (detailed proof in [8]):

$$\begin{aligned} \mathcal{L}(t,\theta) &\propto log\Big[f_T(t)^u S_T(t)^{1-u}\Big] \\ &= u.\Big[\beta.log(\frac{t}{\alpha}) + log(\beta)\Big] - \Big(\frac{t}{\alpha}\Big)^\beta \end{aligned} \tag{2}$$

where u is the event indicator, meaning that $u = 1$ in case of observed events and $u = 0$ in case of censored events. This is equivalent to optimizing θ to maximize the PDF $f_T(t)$ around t for the observed cases, and maximize the survival function $S_T(t)$ beyond t for the censored cases. Figure 4 illustrates what the objective function aims to do.



(a) Maximize $f(t)$ (b) Maximize $S(t)$

Fig. 4. Illustration of the optimization function

In order to train the neural network, we need a (t, u) pair for each observation. For complete traces, t is the actual time till the end of the trace and u = 1 means we observed the end of the trace. For incomplete traces, t is the time till the last event observed (not the end event) and u = 0 means that we have not observed the trace end till this time.

5 Evaluation

In this section we report on the implementation, used datasets, procedure and results of the experiments performed for the evaluation of the proposed approach and its comparison with the baseline.

Implementation

We have used Tensorflow 2.0.1 to build and train our network. All experiments were run on an Intel Core i7-8650U CPU @ 1.90GHz 2.11 GHz. The code for the network and the experiments can be found on our Git hub repo.

Datasets

We have evaluated our approach on four datasets (real life logs) that are described next.

a. **Helpdesk dataset:** This log contains events from a help desk ticketing system of an Italian software company[4]. The process consists of 9 activities, and all cases start with the ticket creation into the system. Each case ends when the issue is resolved and the ticket is closed.
b. **BPI'12 subprocess W dataset:** [5] The log contains data from the application procedure for financial products at a large financial institution. This process consists of three sub-processes. Two of them have events corresponding to automatic activities, whereas the third sub-process (items sub-process) contains events for manual (human executed) tasks. We only used the items sub-process.
c. **BPI'12 subprocess W dataset with no repetition:** This is the same dataset as "b" but without loops.
d. **Environmental permit dataset:** This is a log of an environmental permitting process at a Dutch municipality [6]. Each case refers to one permit application.

The four datasets have very different characteristics in terms of trace length and the number of unique activities, as shown in Figure 5 and Figure 6 respectively. Datasets a and c are the simplest with few unique activities and short traces, while dataset b contains loops in the process which affects the performance of the baseline method. Finally, dataset d is considered to be the most complex with very long traces and different activities. Moreover, it also presents a large variation of the distribution of the log duration.

Experimental Procedure

Each dataset is pre-processed in order to (i) remove the traces with length below $p+1$, for each prefix $p \in \{2, ..., N_{max} - 1\}$, with N_{max} the longest trace of the log; and (ii) build the needed features using the first p events and the time till the last event. The dataset is then split into three equal parts: training set (TS), validation set and test set.

[4] doi:10.17632/39bp3vv62t.1
[5] doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[6] doi:10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

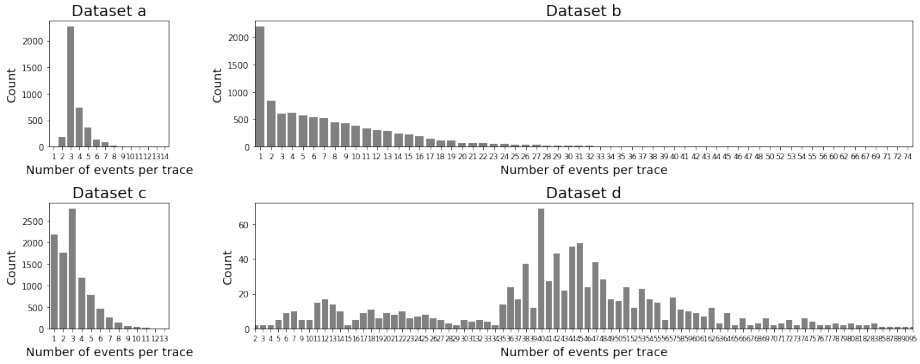


Fig. 5. Length of traces in each dataset

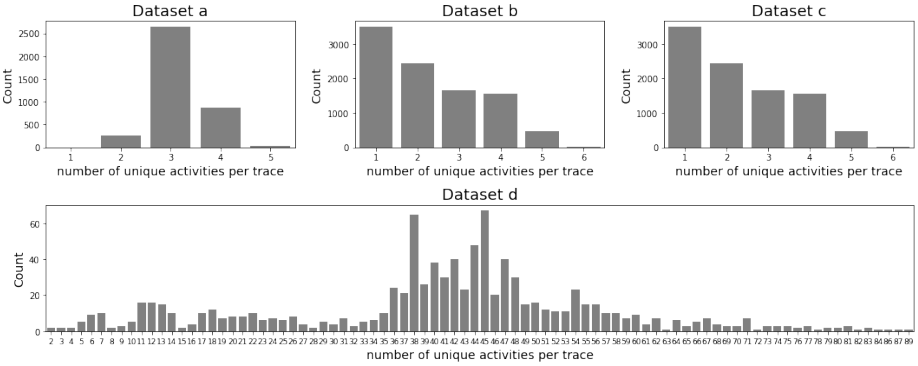


Fig. 6. Unique activities per trace in each dataset

In order to evaluate the effect of censored data, we built a special training set (TS_C) composed of 50% of complete traces and 50% of incomplete traces[7]. To this aim, we split the training set TS further into two sets of traces: the first representing observed cases and the second representing censored cases, which are simulated by randomly cutting the traces. If n is the trace length and p the length of the prefix used for the features, the time till the last observed event is computed looking at the event at position n, if the trace belongs to the first set, or looking at the event in position j, where j is randomly chosen between $p+1$ to n, if the trace belongs to the second set. For censored data, traces need to have at least $p+2$ events.

The following three experiments have been conducted:

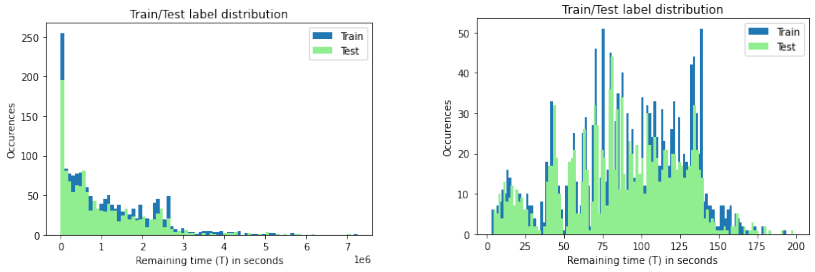
- **Experiment 1**: training on TS using the time to event approach (Section 4).
- **Experiment 2**: training on TS using the time to event approach and transformation of the output to a new less biased space. For the output transformation we used a root cubic transformation where $\phi(x) = \sqrt[3]{x}$, and inverse

[7] The choice of 50% as a fixed ratio for complete/incomplete traces is to reduce the variables in the experiments for better comparison

transformation where $f(x) = x^3$, which turns out to give the best results given the output distribution. (See Figure 7)

- **Experiment 3**: training on TS_C using the time to event approach and transformation of the output to a new less biased space.

We used the same hyperparameters for simplicity, and preserve the same validation and test sets per dataset and prefix throughout the 3 models. The validation set is used to avoid over-fitting, and the test set to compute the performance.



(a) original label (b) transformed label

Fig. 7. Label distribution before and after transformation

Results

We have executed the three experiments described above and compared the results with the ones of the baseline [14] in terms of the mean absolute error (MAE) expressed in days. Fig 8 summarizes the results.

In general, our approach better captures long term dependencies for longer traces (datasets b and d). The accuracy is further enhanced when transforming the label (Experiment 2). With 50% censored data, we find that the accuracy of the model is not harmfully impacted. Instead, it sometimes outperforms the baseline due to its ability to further remove any bias. The number on top of each prefix represents the number of traces used for testing. Obviously, it decreases as the prefix increases since we have less traces matching the length criteria. Yet, our model is able to learn from the smallest set and outperforms the baseline.

There is a little variation in performance with datasets a and c due to their short traces and limited number of possible activities. However, we can see that training with 50% of right-censored traces improves the model performance in almost all the prefixes. This is due to the balance achieved from having both observed and censored traces. In other words, when the majority of the traces are very short, the network tends to predict zero remaining time producing a large MAE for the large traces, affecting the overall performance. The existence of right-censored traces reduces this tendency and forces the network to compromise between short and long traces. Datasets b and d are very challenging due to their very long traces, loops, and random behavior especially in dataset d (Fig. 6). This is obviously affecting the performance of the baseline method since it tries to predict all the remaining events.

Our three experiments have close performance, and surprisingly having 50% of right-censored traces did not harm the performance. This empirically proves that incomplete traces are quite insightful and the network did learn from them.

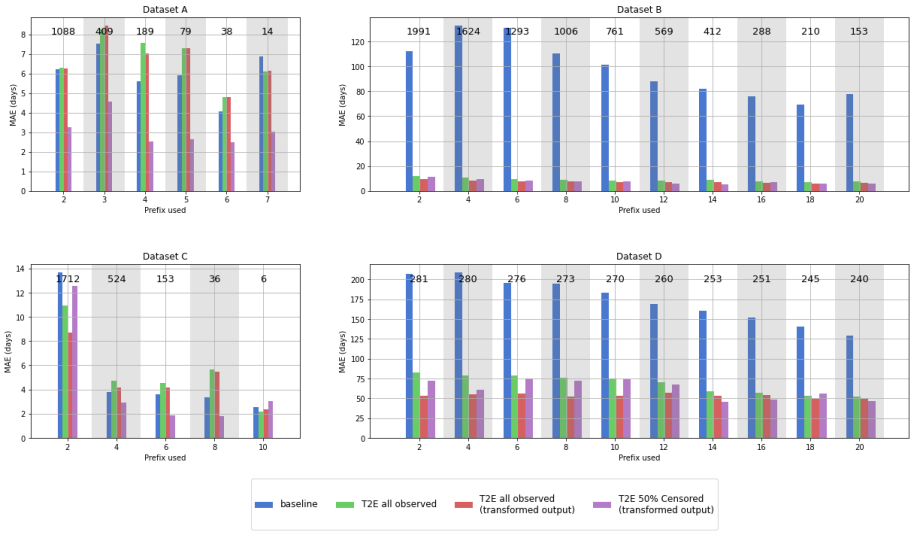


Fig. 8. MAE for experiments 1-3 and the baseline method for the four datasets

Fig. 9 reports the training time of our approach compared to the baseline. We run the training using the setup mentioned in Section 4.1. This huge difference in training time is expected because the network focuses only on predicting the remaining time instead of learning the actual events sequence till the end of the trace.

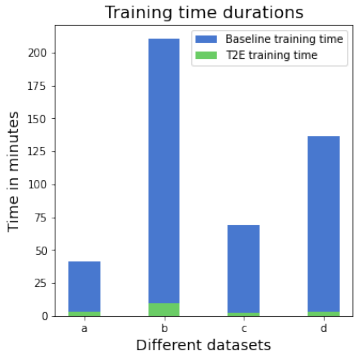


Fig. 9. Training duration

6 Conclusion

In this paper, we present an approach to predict the remaining cycle time of ongoing cases based on learning from incomplete traces. The approach employs survival analysis techniques for this purpose. Our results show, in general, besides a reduced training time, a lower MAE compared to the baseline approach. Using incomplete traces can be useful in several cases, especially when concept drifts occur. Waiting until collecting complete traces, indeed, might compound the impact of degrading model performance as process instances usually take long time to complete.

As future work, we will investigate the applicability of survival analysis techniques and learning from incomplete traces to predict the outcome of a running case. Additionally, we intend to experiment with more data sets and with different percentages of censored traces.

References

1. Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering context-aware models for predicting business process performances. In *OTM*, volume 7565 of *LNCS*, pages 287–304. Springer, 2012.
2. John P Klein and Melvin L Moeschberger. *Survival analysis: techniques for censored and truncated data*. Springer Science & Business Media, 2006.
3. Geetika T Lakshmanan, Davood Shamsi, Yurdaer N Doganata, Merve Unuvar, and Rania Khalaf. A markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems*, 42(1):97–126, 2015.
4. Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *BPM*, volume 9253 of *LNCS*, pages 297–313. Springer, 2015.
5. Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *CAiSE*, volume 8484 of *LNCS*, pages 457–472. Springer, 2014.
6. Marco Maisenbacher and Matthias Weidlich. Handling concept drift in predictive process monitoring. In *SCC*, pages 1–8. IEEE Computer Society, 2017.
7. Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. Predictive monitoring of business processes: A survey. *IEEE Trans. Serv. Comput.*, 11(6):962–977, 2018.
8. Egil Martinsson. *Wtte-rnn: Weibull time to event recurrent neural network*. PhD thesis, Chalmers University of Technology & University of Gothenburg, 2016.
9. Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances, 2016.
10. Germán Rodrıguez. Parametric survival models. *Rapport technique, Princeton: Princeton University*, 2010.
11. Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In *ICSOC*, volume 8274 of *LNCS*, pages 389–403. Springer, 2013.
12. Arik Senderovich, Chiara Di Francescomarino, and Fabrizio Maria Maggi. From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. *Inf. Syst.*, 84:255–264, 2019.
13. LLC StataCorp. Stata survival analysis reference manual, 2017.
14. Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. In *CAiSE*, volume 10253 of *LNCS*, pages 477–492. Springer, 2017.
15. Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475, 2011.
16. Boudewijn F van Dongen, Ronald A Crooy, and Wil MP van der Aalst. Cycle time prediction: When will this case finally be finished? In *OTM*, pages 319–336. Springer, 2008.
17. Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Irene Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *ACM Trans. TIST*, 10(4):1–34, 2019.
18. Ilya Verenich, Marlon Dumas, Marcello La Rosa, and Hoang Nguyen. Predicting process performance: A white-box approach based on process models. *J. Softw. Evol. Process.*, 31(6), 2019.

Time Matters: Time-Aware LSTMs for Predictive Business Process Monitoring

An Nguyen[1⋆], Srijeet Chatterjee[1 ⋆⋆], Sven Weinzierl[2], Leo Schwinn[1], Martin Matzner[2], and Bjoern Eskofier[1]

[1] Department of Computer Science, Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Erlangen, Germany {an.nguyen, srijeet.chatterjee, leo.schwinn, bjoern.eskofier} @fau.de

[2] Insitute of Information Systems, Friedrich-Alexander-University Erlangen-Nürnberg (FAU), Nürnberg, Germany {sven.weinzierl, martin.matzner} @fau.de

Abstract. Predictive business process monitoring (PBPM) aims to predict future process behavior during ongoing process executions based on event log data. Especially, techniques for the next activity and timestamp prediction can help to improve the performance of operational business processes. Recently, many PBPM solutions based on deep learning were proposed by researchers. Due to the sequential nature of event log data, a common choice is to apply recurrent neural networks with long short-term memory (LSTM) cells. We argue, that the elapsed time between events is informative. However, current PBPM techniques mainly use "vanilla" LSTM cells and hand-crafted time-related control flow features. To better model the time dependencies between events, we propose a new PBPM technique based on time-aware LSTM (T-LSTM) cells. T-LSTM cells incorporate the elapsed time between consecutive events inherently to adjust the cell memory. Furthermore, we introduce cost-sensitive learning to account for the common class imbalance in event logs. Our experiments on publicly available benchmark event logs indicate the effectiveness of the introduced techniques.

Keywords: Predictive Business Process Monitoring, Deep Learning, Recurrent Neural Network, LSTM, Time-Awareness.

1 Introduction

In the last years, a variety of predictive business process monitoring (PBPM) techniques that base on machine learning (ML) were proposed by researchers [6] to improve the performance of operational business processes [4]. PBPM is a class of techniques aiming at predicting future process characteristics in running process instances [12], like next activities, next timestamps or process-related performance indicators. Such PBPM techniques produce predictions through

⋆ Corresponding author

⋆⋆ Equal contribution with An Nguyen

predictive models. These models are in turn constructed based on historical event log data.

A current trend in PBPM is to apply deep neural networks (DNNs) to learn more accurate predictive models from event log data than with "traditional" ML algorithms like probabilistic automata [7]. DNNs belong to the ML-sub-field deep learning (DL) and achieve that by identifying the intricate structures in high-dimensional data through multi-representation learning [11].

Existing DL-based PBPM techniques often rely on DNN architectures consisting of out-of-the-box constructs like layers with a "vanilla" long short-term memory (LSTM) cell [9] or state-of-the-art loss functions for parameter learning.

Event logs can be seen as sequences of events in continuous time with irregular intervals (i.e., elapsed time between consecutive events). We argue that these time intervals are informative in the case of event logs in PBPM. Intuitively, these time intervals describe human behavior of executing business processes. Thus, a time-aware PBPM technique considering information on time intervals could potentially achieve a higher predictive quality. Time information is currently only exploited via hand-crafted control-flow features as inputs to "vanilla" LSTM cells [14]. To better account for the time information in event log data, we propose a new PBPM techniques using time-aware LSTM (T-LSTM). T-LSTM extends the "vanilla" LSTM cells by incorporating the elapsed time between consecutive events in order to adjust the memory state and is inspired by work from Baytas et al. [2].

Furthermore, the problem of next activity prediction is commonly modeled as a supervised multi-class classification problem. The distribution of activities in event logs are commonly skewed. Therefore, we additionally introduce cost-sensitive learning to address the inherent class-imbalances. The main contributions of this work are summarized below:

- We introduce a time-aware LSTM model for the tasks of predicting next activities and timestamps in PBPM
- We tackle the problem of skewed class distributions via cost-sensitive learning

We evaluate the effectiveness of our proposed techniques by conducting experiments for the next activity and timestamp prediction on publicly available benchmark event logs commonly used for PBPM.

The remainder of the paper is structured as follows: Section 2 presents related work on DL-based next activity and timestamp prediction. Section 3 introduces preliminaries and the concept of a LSTM. Section 4 and 5 describes the architecture of T-LSTM and our experimental setup respectively. Then, in Sections 6 and 7, we present and discuss our results. Section 8 concludes our paper and discusses future research directions.

2 Related Work

Inspired by the field of natural language processing (NLP), Evermann et al. [7] applied recurrent neural network-based and LSTM-basd DNN architectures for

the next activity and next sequence of activity prediction in PBPM. They made use of word embeddings to encode activities of event log's process instances.

Navarin et al. [13] used a "vanilla" LSTM-based DNN architecture for predicting the completion time of running process instances. They one-hot encoded the activity attributes, computed temporal control-flow attributes, and considered additional real-valued or categorical context attributes.

Tax et al.[14] proposed a multitask learning approach using "vanilla" LSTM cells for next activity and timestamp prediction respectively. Like in [13], they one-hot encoded the activity and computed temporal control-flow features. However, the authors did not consider additional data attributes in their approach. This work acts as a baseline for a variety of other techniques such as [17].

Khan et al. [10] introduced memory augmented neural networks (MANNs) in PBPM. MANNs reduce the number of trainable parameters. In general, the network's architecture consists of an externalized state memory and two "vanilla" LSTM cells manipulating the memory. One LSTM cell works as encoder and the other one as decoder. Concerning the predictive quality, their approach is comparable to the one presented in [14].

Camagro et al. [5] extended the implementation of [14] and fed the resource attribute into the DNN model. Additionally, instead of one-hot encoding, they applied embeddings, as proposed by Evermann et al. [7].

Taymouri et al. [15] introduced generative adversarial networks (GANs) for the next activity and timestamp prediction. The network's architecture comprises two "vanilla" LSTM cells. One for the generator and the other one for the discriminator.

To date, several studies have investigated DNN-based PBPM techniques. However, none of the related works models the elapsed time between two successive events. We address this gap through adapting time-aware LSTM cells [2].

Further, to the best of the our knowledge, there exists no work to date which addresses the event class imbalance problem for the next activity and/or next timestamp prediction. We address this gap through adapting cost-sensitive learning.

3 Background

3.1 Preliminaries

Definition 1 (Event, Trace, Event Log). *An event is a tuple* (c, a, ts) *where* c *is the case id,* a *is the activity (label) and* ts *is the timestamp. A trace is a non-empty sequence* $\sigma = \langle e_1, \ldots, e_{|\sigma|} \rangle$ *of events such that* $\forall i, j \in \{1, \ldots, |\sigma|\}$ $e_i.c = e_j.c$ *and* $e_i.ts \leq e_j.ts$, $1 \leq i < j \leq |\sigma|$. *An event log* L *is a set* $\{\sigma_1, \ldots, \sigma_{|L|}\}$ *of traces. A trace can also be considered as a sequence of vectors which contain derived control flow information or features. Formally,* $\sigma = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(|\sigma|)} \rangle$, *where* $\mathbf{x}^{(t)} \in \mathbb{R}^{n \times 1}$ *is a vector, and the superscript indicates the time-order upon which the events happened.* n *is the number of features derived for each event.*

Definition 2 (Prefix and Label). *Given a trace* $\sigma = \langle e_1, \ldots, e_k, \ldots, e_{|\sigma|} \rangle$, *a prefix of length* k, *that is a non-empty sequence, is defined as* $f_p^{(k)}(\sigma) = \langle e_1, \ldots, e_k \rangle$, *with* $0 < k < |\sigma_c|$. *A next activity label for a prefix of length* k *is defined as* $f_{l,a}^{(k)}(\sigma) = e_{k+1}.a$, *whereas a next timestamp label for a prefix of length* k *is defined as* $f_{l,ts}^{(k)}(\sigma) = e_{k+1}.ts$. *The above definition also holds for an input trace representing a sequence of vectors. For example, the tuple of all possible prefixes, the tuple of all possible next activity labels and the tuple of all possible next timestamp labels for* $\sigma = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)} \rangle$ *are* $\langle \langle \mathbf{x}^{(1)} \rangle, \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle \rangle$, $\langle e_2.a, e_3.a \rangle$, *and* $\langle e_2.ts, e_3.ts \rangle$.

3.2 Long Short-term Memory Cells

Most of the DNN architectures proposed for the next activity and timestamp prediction in PBPM [16] use "vanilla" LSTM cells [9]. LSTMs belong to the class of recurrent neural networks [11] and are designed to handle temporal dependencies in sequential prediction problems [3].

Given a sequence of inputs $\sigma = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, ..., \mathbf{x}^{(k)} \rangle$, a LSTM computes sequences of outputs $\langle \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}, ..., \mathbf{h}^{(k)} \rangle$ via the following recurrent equations:

$$
\begin{aligned}
\mathbf{f}_g^{(t)} &= sigmoid(\mathbf{U}_f \mathbf{h}^{(t-1)} + \mathbf{W}_f \mathbf{x}^{(t)} + \mathbf{b}_f) && \text{(forget gate)}, \\
\mathbf{i}_g^{(t)} &= sigmoid(\mathbf{U}_i \mathbf{h}^{(t-1)} + \mathbf{W}_i \mathbf{x}^{(t)} + \mathbf{b}_i) && \text{(input gate)}, \\
\tilde{\mathbf{c}}^{(t)} &= tanh(\mathbf{U}_g \mathbf{h}^{(t-1)} + \mathbf{W}_g \mathbf{x}^{(t)} + \mathbf{b}_g) && \text{(candidate memory)}, \\
\mathbf{c}^{(t)} &= \mathbf{f}_g^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}_g^{(t)} \circ \tilde{\mathbf{c}}^{(t)} && \text{(current memory)}, \\
\mathbf{o}_g^{(t)} &= sigmoid(\mathbf{U}_o \mathbf{h}^{(t-1)} + \mathbf{W}_o \mathbf{x}^{(t)} + \mathbf{b}_o) && \text{(output gate)}, \\
\mathbf{h}^{(t)} &= \mathbf{o}_g^{(t)} \circ tanh(\mathbf{c}^{(t)}) && \text{(current hidden state)}, \\
& \forall t \in \{1, 2, \ldots, k\}.
\end{aligned}
\tag{1}
$$

$\{\mathbf{U}_{f,i,g,o}, \mathbf{W}_{f,i,g,o}, \mathbf{b}_{f,i,g,o}\}$ are trainable parameters, $\circ$ denotes the Hadamard product (element-wise product), $\mathbf{h}^{(t)}$ and $\mathbf{c}^{(t)}$ are the hidden state and cell memory of a LSTM cell. Additionally, a LSTM cell uses four gates to manage its states over time to avoid the problem of exploding/vanishing gradients in the case of longer sequences [3]. $\mathbf{f}_g^{(t)}$ (forget gate) determines how much of the previous memory is kept, $\mathbf{i}_g^{(t)}$ (input gate) controls the amount new information is stored into memory, $\tilde{\mathbf{c}}^{(\mathbf{t})}$ (candidate memory) defines how much information is stored into memory and $\mathbf{o}_g^{(t)}$ (output gate) determines how much information is read out of the memory. The hidden state $\mathbf{h}^{(t)}$ is commonly forwarded to a successive layer.

4 Methodology

4.1 Time-Aware Long Short-term Memory Cells

"Vanilla" LSTM cells, as described in Section 3.2, assume a uniform distribution of the elapsed time between events ($\boldsymbol{\Delta}^{(t)} := {x_{ts}}^{(t)} - {x_{ts}}^{(t-1)}$). This assumption does not hold for most event logs analyzed in PBPM though (see Fig. 4). The elapsed time between consecutive events might have an impact on the next activity and timestamp prediction. Hence, a LSTM cell should be able to take irregular elapsed times into account when processing event logs.

Time-aware long short-term memory (T-LSTM) cells are an extension of the LSTM. Fig. 1 depicts the T-LSTM cell and highlights its differences with regard to the LSTM cell.

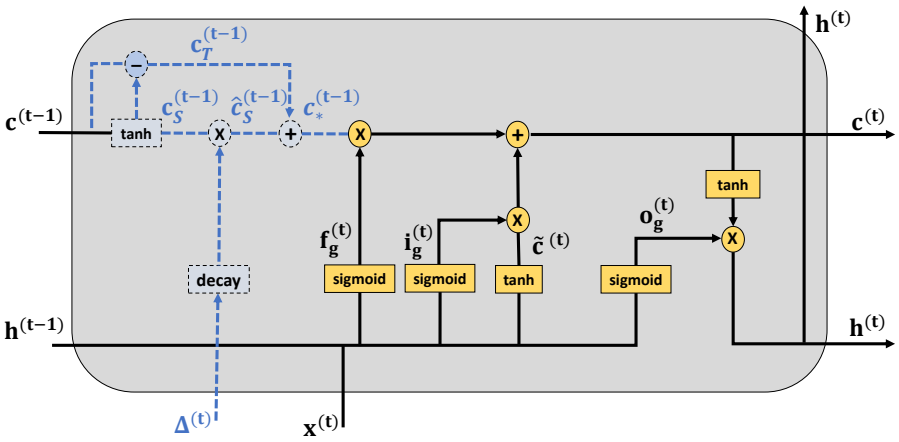


Fig. 1. Illustration of a T-LSTM cell with its computational components at time step t. The dashed and blue components indicate the extensions to the "vanilla" LSTM cell. The previous cell memory $\mathbf{c}_S^{(t-1)}$ is adjusted to $\mathbf{c}_*^{(t-1)}$ (see Eq. (2)) and is then processed together with $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ via the LSTM computations, as formalized in Eq. (1).

The main idea behind T-LSTM is to perform a subspace decomposition of the previous cell memory $\mathbf{c}^{(t-1)}$. First, a short term memory component $\mathbf{c}_S^{(t-1)}$ is extracted via a network. Next, the short term memory is discounted via a decay function of the elapsed time and yields $\hat{\mathbf{c}}_s^{(t-1)}$. Then, the long term memory ($\mathbf{c}_T^{(t-1)} = \mathbf{c}^{(t-1)} - \mathbf{c}_S^{(t-1)}$) is calculated. Finally, the previous cell memory is adjusted $\mathbf{c}_*^{(t-1)} = \mathbf{c}_T^{(t-1)} + \hat{\mathbf{c}}_s^{(t-1)}$. The adjusted previous memory $\mathbf{c}_*^{(t-1)}$ is then, together with $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$, further processed as in LSTM cells by substituting $\mathbf{c}^{(t-1)}$ with $\mathbf{c}_*^{(t-1)}$ in Eq. (1). The following equations summarize the T-LSTM

specific computations for the subspace decomposition and adjustment of the previous memory.

$$
\begin{aligned}
\mathbf{c}_S^{(t-1)} &= tanh(\mathbf{W}_d \mathbf{c}^{(t-1)} + \mathbf{b}_d) && \text{(short term memory)},\\
\hat{\mathbf{c}}_s^{(t-1)} &= \mathbf{c}_S^{(t-1)} * decay(\mathbf{\Delta}^{(t)}) && \text{(discounted short term memory)},\\
\mathbf{c}_T^{(t-1)} &= \mathbf{c}^{(t-1)} - \mathbf{c}_S^{(t-1)} && \text{(long term memory)},\\
\mathbf{c}_*^{(t-1)} &= \mathbf{c}_T^{(t-1)} + \hat{\mathbf{c}}_s^{(t-1)} && \text{(adjusted previous memory)},\\
&... && \text{(LSTM computations as in Eq. (1))},\\
&\forall t \in \{1, 2, \ldots, k\}.
\end{aligned}
\tag{2}
$$

Note, that we only add $\{\mathbf{W}_d, \mathbf{b}_d\}$ as trainable parameters compared to the LSTM cell. As recommended in Baytas et al. [2], we chose $decay(\mathbf{\Delta}^{(t)}) = 1/log(e + \mathbf{\Delta}^{(t)})$ since we input the elapsed times in seconds and therefore have large values for $\mathbf{\Delta}^t$. Any other monotonic decreasing function and scale for $\mathbf{\Delta}^t$ would be valid as well, but our initial choice proved to be effective. The intuition behind the subspace decomposition is that the short term memory should be discounted if the elapsed time is very large, while the long term memory should be maintained in the adjusted previous cell memory $\mathbf{c}_*^{(t-1)}$. Similar as for LSTMs, the hidden state $\mathbf{h}^{(t)}$ is forwarded to successive layer for further processing. Hence, it is straightforward to substitute LSTM with T-LSTM cells in a given DNN architecture.

4.2 Network Architecture

We adapted the multitask architecture proposed by Tax et al. [14] as a baseline (see Fig. 2). The predicted next activity $\hat{e}_{k+1}.a$ is the output of a softmax activation after the last dense layer, where the output dimension is equal to the number of unique activity labels. $\hat{e}_{k+1}.a$ is evaluated against the one-hot encoded ground truth label $e_{k+1}.a$ by using the Cross-Entropy (CE) loss. The predicted next timestamp $\hat{e}_{k+1}.ts$ is a scalar output of a dense layer. We do not apply any additional activation after the time specific dense layer to be consistent with the implementation[3] of Tax et al. [14]. $\hat{e}_{k+1}.ts$ is compared with the ground truth timestamp $e_{k+1}.ts$ using the Mean Absolute Error (MAE). The total loss is the sum of both losses, as implemented in Tax et al. [14]. Further, they applied one-hot encoding for the activities and compute time-related control-flow features, which we also used in our experiments. We refer to the baseline architecture as "**Tax**". We performed an ablation study and made three modifications to the baseline DNN architecture:

– We weighted the CE loss function based on the distribution of activity labels in the training set. Hence, the classification of under-represented event classes had larger influence during training. We refer to this model as "**Tax+CS**".

[3] `https://github.com/verenich/ProcessSequencePrediction`

– We replaced all LSTM layers with T-LSTM layers and refer to this model as "**Tax+T-LSTM**".
– We added cost-sensitive learning and replaced all LSTM layers with T-LSTM layers. We call this model "**Tax+CS+T-LSTM**"

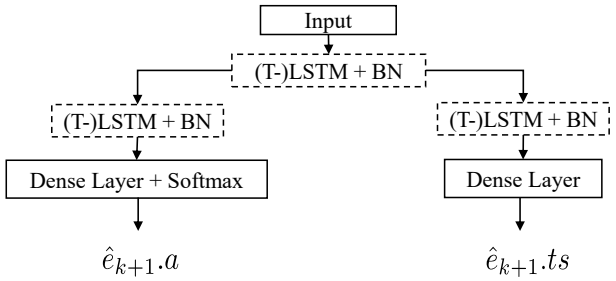


Fig. 2. Network architecture for this work based on the multitask learning approach proposed by Tax et al. [14]. The dashed components are either LSTM or T-LSTM layers. The input is of the network is a sequence of vectors representing a prefix $\langle e_1, \ldots, e_k \rangle$ as in Tax et al. [14]. For the baseline architecture we applied one-hot encoding and LSTM layers as in [14]. The outputs of the model are the predicted next activity ($\hat{e}_{k+1}.a$) and timestamp ($\hat{e}_{k+1}.ts$). Each of the LSTM layers is followed by a batch normalization layer (BN) to speed up training, as used in Tax et al. [14].

5 Experiments

5.1 Datasets

We performed our experiments on the same publicly available datasets as Tax et al. [14] to validate the effectiveness of our proposed techniques. Fig. 3 shows the distribution of the activities (labels) for the different datasets. It is evident that the distributions of activities are skewed for both event logs. Table 1 presents descriptive statistics of the datasets used in this work.

Helpdesk[4]**:** This event log originates from a ticket management process of an Italian software company.

BPI'12 W Subprocess[5] **(BPI12W):** The Business Process Intelligence (BPI) 2012 challenge provided this event log from a German financial institution. The data come from a loan application process. The 'W' indicates state of the work item for the application.

[4] https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb
[5] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

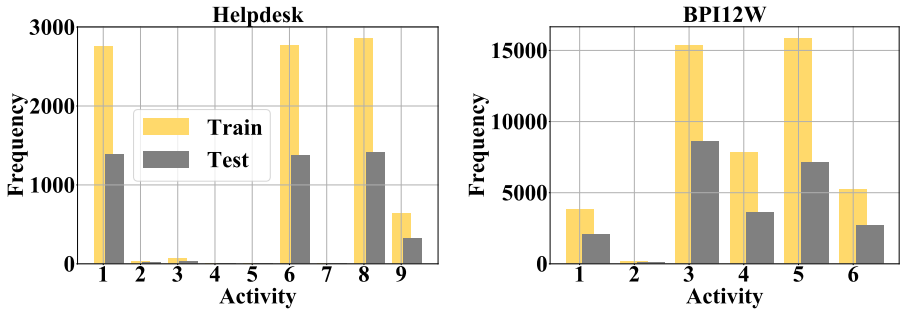


Fig. 3. Activity distribution in training and test set for Helpdesk and BPI12W datasets. It is evident that the distributions of the activity labels are skewed.

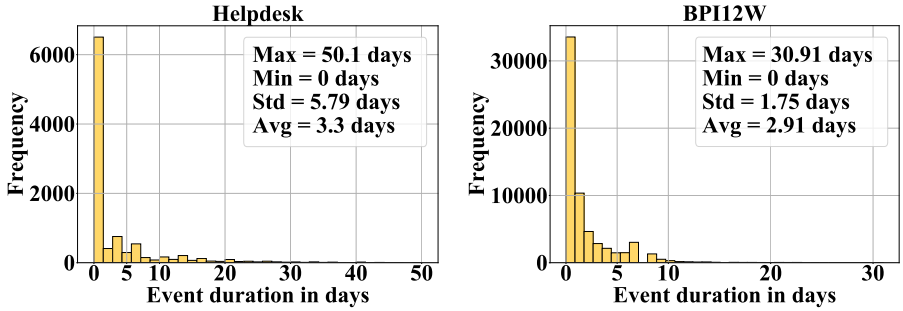


Fig. 4. Event duration distribution for the complete Helpdesk and BPI12W datasets. It can be observed that the majority of the events are completed within one day. However, there are many events with longer duration. Note that we input the elapsed time between events ($\mathbf{\Delta}^t$) in seconds for T-LSTM.

5.2 Preprocessing

We used the cleaned and prepared datasets as in Tax et al. [14]. The datasets can be found on the corresponding GitHub repository[6]. The preprocessing steps include splitting the data into training and test set, calculating time divisors, and ASCII encoding activities and sequence generation. Datasets were split into 2/3rd and 1/3rd for training and testing respectively and preserve the temporal order of cases. We additionally used the last 20% of the training data as a validation set in order to tune the hyperparameters. We adapted the sequence and feature generation methods by Tax et al. [14]. The features include the activity of the event, position of the event in the case, time since the last event, time from the starting event of the case, time from midnight, and day of the week. We create one-hot encoded versions of the ground truth labels $e_{k+1}.a$ for the next activity prediction in order to compare them with the predicted next activity labels $\hat{e}_{k+1}.a$.

[6] `https://github.com/verenich/ProcessSequencePrediction/tree/master/data`

Characteristic	Helpdesk	BPI12W
Number of instances	3,804	9,658
Case variants	154	2,263
Unique activities	9	6
Events	13,710	72,413
Max case length	14	74
Min case length	1	1
Avg case length	3.604	7.497

Table 1. Descriptive statistics of the datasets used in this study.

5.3 Training Setup

For hyperparameter tuning, we performed a grid search on the training set and chose the model with the lowest validation loss. The validation loss is the sum of activity-related validation loss and time-related validation loss. The number of LSTM or T-LSTM units were set to 64 or 100. For the dropout rate (for both dense layers), we tried the values 0.0 and 0.2. We choose Nadam as an optimization algorithm, as used in [14]. Nesterov accelerated gradient (NAG) calculates the step using the 'lookahead' algorithm, which approximates the next parameters. Adam optimizer estimates learning rates based on initial moments of the gradients. Nadam is a combination of both and is robust in noisy datasets. Furthermore, we tested a range of different learning rates $\{0.0001, 0.0002, 0.001, 0.002, 0.01\}$ since this is known to have a large impact on LSTMs [8]. We trained each model for 150 epochs, with a batch size of 64 and apply early stopping with patience 25 for regularization.

5.4 Evaluation

We applied the same evaluation metrics as in [14]. We used the *Accuracy* metric to evaluate the next activity prediction. For the next timestamp prediction, we used the *Mean Absolute Error (MAE)* to evaluate our models.

5.5 Implementation

We conducted all experiments on a workstation with 24 CPU cores, 748 GB RAM and a singe GPU NVIDEA QUADRO RTX6000. We implemented the experiments in Python 3.7. We used the DL framework TensorFlow 2.1[7]. The source code is available on GitHub[8].

[7] `https://www.tensorflow.org`
[8] `https://github.com/annguy/time-aware-pbpm`

6 Results

6.1 Next Activity Prediction

Table 2 shows the results for the next activity prediction in terms of Accuracy. For Helpdesk and BPI12W, the approach Tax+CS+T-LSTM achieved the highest Accuracy (0.724 and 0.778) among all approaches. The approach's improvement compared to the baseline is 0.012 and 0.018. While the two approaches, Tax+CS and Tax+T-LSTM, outperformed the baseline for Helpdesk, these approaches achieved a lower Accuracy for BPI12W than the baseline.

Approach	Helpdesk	BPI12W
Tax (baseline)	0.712	0.760
Tax+CS	0.713	0.757
Tax+T-LSTM	0.718	0.693
Tax+CS+T-LSTM	**0.724**	**0.778**

Table 2. Results for the next activity prediction in terms of Accuracy. The best result for each dataset is highlighted (larger is better).

6.2 Next Timestamp Prediction

Table 3 shows the results for the next timestamp prediction task in terms of MAE in days. All approaches with a T-LSTM cell, clearly outperformed the baseline for both event logs. Thereby, the approach Tax+CS achieved the lowest MAE of 2.87 days and 0.88 days for Helpdesk and BPI12W respectively. Compared to the baseline, this approach reduced the MAE by 0.88 days (Helpdesk) and 0.68 days (BPI12W). The other two approaches, Tax+T-LSTM and Tax+CS+T-LSTM, achieved a slightly worse MAE values compared to Tax+CS for both event logs. It is worth noticing that for Helpdesk Tax+CS+T-LSTM and for BPI12W Tax+T-LSTM yielded the second best results with MAE close to Tax+CS.

Approach	Helpdesk	BPI12W
Tax (baseline)	3.75	1.56
Tax+CS	**2.87**	**0.88**
Tax+T-LSTM	3.01	0.88
Tax+CS+T-LSTM	2.94	0.90

Table 3. Results for next step time prediction in terms of MAE in days. The best result for each dataset is highlighted (lower is better).

7 Discussion

In this paper, we argued that the elapsed time between consecutive events carries valuable information on human behavior in running business processes. Therefore, we introduced T-LSTM cells for PBPM which inherently model the elapsed time between consecutive events. Further, we introduced of cost-sensitive learning to better cope with the problem of imbalanced data.

The obtained results indicate that the elapsed time between consecutive events is informative and that a DNN architecture relying on T-LSTM cells cab yield more accurate models for PBPM. Especially, with the approach Tax+CS+T-LSTM, we could outperform the baseline (Tax) for both datasets (i.e., Helpdesk and BPI12W) and both prediction tasks (i.e., next activity prediction and next timestamp prediction). Thereby, we could observe that cost-sensitive learning plays a crucial role for the predictive quality of a DNN architecture using T-LSTM cells instead of "vanilla" LSTM cells. Interestingly, the effectiveness of the introduced techniques is more evident for the next timestamp prediction compared to the next activity prediction

Even though our presented results on DNN architectures using T-LSTMs seem promising, there are a few limitations to our work. First, we need to verify our findings by performing experiments on more datasets. Second, a better hyperparameter tuning approach like Bayesian optimization [1] could be applied for all configurations to get a better estimate of their effectiveness. Further, several runs with random initialization should be performed to estimate the stability of the models.

8 Conclusion and Future Work

We propose T-LSTM as an alternative to the commonly used "vanilla" LSTM cell to better exploit information on the elapsed time between consecutive events. Furthermore, we introduced the concept of cost-sensitive learning to account for the common class-imbalance in event log data. Our results indicate the effectiveness of the introduced techniques for the next activity and timestamp prediction. This suggests that integrating specific mechanisms into neural network layers to incorporate event log specific characteristics might be an interesting direction for future research. Here, we mainly demonstrated the benefit of replacing a normal LSTM with a time-aware LSTM cell for a given baseline approach [14].

An avenue for future research is to investigate if T-LSTM cells might also improve other LSTM-based PBPM approaches such as Camargo et al. [5] involving resource attributes or Taymouri et al. [15] generating fake event logs. Another direction for future research is to further customize an LSTM cell in terms specifically for PBPM. For example, a process-aware LSTM cell could not only deal with time information but also with resource information.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25rd International Conference on Knowledge Discovery and Data Mining (KDD) (2019)
2. Baytas, I.M., Xiao, C., Zhang, X., Wang, F., Jain, A.K., Zhou, J.: Patient subtyping via time-aware LSTM networks. In: Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining (KDD). pp. 65–74 (2017)
3. Bengio, Y., Simard, P., Frasconi, P., et al.: Learning long-term dependencies with gradient descent is difficult. Transactions on Neural Networks **5**(2), 157–166 (1994)
4. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Quarterly **40**(4), 1009–1034 (2016)
5. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: Proceedings of the 17th International Conference on Business Process Management (BPM). pp. 286–302. Springer (2019)
6. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: Which one suits me best? In: Proceedings of the 16th International Conference on Business Process Management (BPM). pp. 462–479. Springer (2018)
7. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017), evermann2017predicting
8. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey. IEEE Transactions on Neural Networks and Learning Systems **28**(10), 2222–2232 (2017)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
10. Khan, A., Le, H., Do, K., Tran, T., Ghose, A., Dam, H., Sindhgatta, R.: Memory-augmented neural networks for predictive process analytics. arXiv preprint arXiv:1802.00938 (2018)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436 (2015)
12. Maggi, F., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE). pp. 457–472. Springer (2014)
13. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–7. IEEE (2017)
14. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE). pp. 477–492. Springer (2017)
15. Taymouri, F., La Rosa, M., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In: Proceedings of the 18th International Conference on Business Process Management (BPM) (2020)
16. Weinzierl, S., Zilker, S., Brunk, J., Revoredo, K., Nguyen, A., Matzner, M., Becker, J., Eskofier, B.: An empirical comparison of deep-neural-network architectures for next activity prediction using context-enriched process event logs. arXiv:2005.01194 (2020)
17. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: Proceedings of the 18th Conference on Business Process Management (BPM). Springer (2020)

A Preliminary Study on the Application of Reinforcement Learning for Predictive Process Monitoring

Andrea Chiorrini, Claudia Diamantini, Alex Mircoli, and Domenico Potena

Department of Information Engineering, Polytechnic University of Marche, Ancona
{a.chiorrini,c.diamantini,a.mircoli,d.potena}@univpm.it

Abstract. The present paper explores the opportunity of applying reinforcement learning to various typical tasks in the field of predictive process monitoring. The tasks considered are the prediction of both next event activity and time completion as well as the prediction of the whole progression of running cases. Experiments have been conducted on the popular benchmark dataset, BPI' 2012, on which we compare the proposed learning system with state of the art methods adopting LSTM networks trained through supervised learning. Results enlighten promising features of the approach and interesting research issues and challenges, as well as proving the applicability of reinforcement learning to predictive process monitoring.

Keywords: Predictive Process Monitoring · Reinforcement Learning · Outcome and time prediction · Process Mining

1 Introduction

Recently the field of predictive process monitoring is receiving increasing attention [7,8]. It aims at improving process monitoring through the introduction of predictive capabilities, allowing both process improvement and proactive problem handling. Predictive process monitoring relies on models, obtained from historical process logs, able to forecast the evolution of a process instance based only on the first part of it. In detail, predictive process monitoring tries to handle several different problems, such as the one step ahead prediction of the next activity that will be performed and the estimation of its execution time, or the prediction of all the activities to be performed until the end of the trace, i.e. the trace suffix, and the total execution time of the trace, i.e. the trace time. Recently, different machine learning techniques have been adopted to deal with predictive process monitoring tasks, with a particular focus on Long Short-Term Memory (LSTM) networks. In this paper we investigate the use of reinforcement learning to predict, both suffix and one step ahead, activities and execution times. Reinforcement Learning (RL) is a particular kind of machine learning paradigm that trains models to directly maximize a reward signal, without assigning any label or necessarily trying to find some hidden structure in the data. Reinforcement

learning has been gaining increasing attention since 2015, when [9] trained an agent that bested many human professional players over various Atari games. This has led the scientific community to further investigate the techniques, leading to various interesting results (e.g., [14,13]), up until the latest astonishing artificial agent [17] that managed to beat professional human player in Starcraft II, an extremely complex real time strategy game. An interesting feature of this family of algorithms is that learning is guided by an objective function that takes into account all the chain of future decisions and its effects, instead of focusing only on the decision at hand. This could be an interesting feature in the predictive process mining field, where events to be predicted are conditioned by the process workflow.

Motivated by past success of RL and this observation, we set as our goal to study if it is possible to apply reinforcement learning to the field of predictive process monitoring. At the best of our knowledge this is the first study of this kind. The only other study that applied reinforcement learning in the process mining field is [4], where the problem of efficient resource allocation is considered. Our results enlighten promising features of the approach and interesting research issues.

The rest of the paper is organized as follows: Section 2 is devoted to introduce related work. In Section 3 background knowledge about reinforcement learning is provided, while Section 4 explains our proposed approach. Section 5 presents the performed experiments and discusses achieved results. Finally, Section 6 concludes the paper and outlines some directions for future works.

2 Related work

Recent efforts in predictive process monitoring exploits Deep Neural Networks, specifically LSTM and CNN [1,3,6,10,11,16].

In particular, [16] trained an LSTM for the one step ahead event prediction, in particular the activity associated to the next event and its completion time, as well as the suffix of the trace, iteratively using the one step ahead event prediction. They encoded each event into a feature vector that is a combination of the one-hot encode of the associated activity and three temporal features related to the event's timestamp such the time of the day, the time since the previous event, and the accumulated duration since the start of a process case. LSTM has also been used in [3] to predict the activity associated with the next event of a case, but this approach, differently from [16], uses the embedded dimension of LSTM to both reduce the input's size and include extra attributes like the resource associated to each event. Their experiments show that the proposed approach sometimes outperforms [16] for the prediction of the next event. However, [3] only focuses on predicting event activity types and cannot predict the next event's timestamp as it cannot handle numerical variables.

In [1] a combination of approaches in [3] and [16] is proposed for the one step ahead event prediction. The approach considers and predicts the next activity, the timestamp and type of resource of next event. To do so they introduce a

notion of abstract class of resource, i.e. group of resources that usually performs similar activities, this way they manage to avoid the main limit of [3] which is the inability of handling numerical variables and therefore predicting next event's timestamp.

Another LSTM model has been proposed by Lin et al. [6] for the prediction of the next activity and all the other categorical attributes of the next event (e.g the associated resource) of the past events, using an approach similar to an attention mechanism for weighting the event attributes on the basis of their relevance in the prediction of future events. Again this approach suffers from the inability of handling numerical values and therefore predicting timestamps.

Even more recently, the usage of CNN has been investigated. The basic idea is to convert the temporal data enclosed in an event log into spatial data so as to treat them as images [10] This idea has been been further extended in [11], where an RGB encoding of process instances is used to train a 2-D CNN based on two inception blocks. Both papers only tackle the one step ahead activity prediction task.

3 Background

In this section we provide general background knowledge on RL.

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. A reinforcement learning problem is formalized using ideas from dynamical systems theory, specifically, as the optimal control of incompletely-known Markov decision process [15].

The learner (agent) interacts with an environment during a sequence of timesteps composing the learning episode. In the domain of process mining we can think the learning episode as the evolution of the trace, and the occurrence of an event in it as a timestep. At each timestep, an interaction between agent and environment occurs through observations (x) of the environment, actions (a) and rewards (r). The observation is the trace event, the action is the prediction for the next event, and the reward is derived from the comparison between the next event information and the predicted one. The agent's goal is to maximize its cumulative future reward performing its actions, with respect to the state of the environment. The state s_i of the environment is defined as the full sequence of observations and actions performed until timestep i, formally: $s_i = x_1, a_1, x_2, ..., a_{i-1}, x_i$. However, it is complex to use a state composed of a variable number of observations and actions as input. Hence, it is usually preferred to use a constant fixed number of observations and actions. In this paper we refer to the number of past timesteps considered to define the state as *window size*. Having fixed the window size to a generic k, the state is written as $s_i = x_{i-k}, a_{i-k}, x_{i-k+1}, ..., a_{i-1}, x_i$.

The objective function of the agent at timestep i can be expressed as:

$$R_i = \sum_{t=i}^{T} \gamma^{t-i} r_t, \tag{1}$$

where $\gamma < 1$ is the discount factor of future rewards, used to prioritize more recent rewards, and T is the number of timesteps in the whole learning episode. In particular, the methodology adopted in this paper considers Q learning agents. This type of agents tries to approximate the optimal action-value function $Q^*(s, a)$, learning it from the transitions from a state s_i to a next state s_{i+1} on the basis of the performed action a_i and the received reward r_i. The optimal action-value function may be expressed as:

$$Q^*(s, a) = \max_{\pi} \mathbf{E}\left[R_i | s_\mathrm{t} = s, a_t = a, \pi\right] \tag{2}$$

which is the maximum expected reward achievable after seeing sequence s and taking action a, by following any behaviour policy ϕ for mapping sequences to actions.

This reinforcement learning algorithm is based on the fact that knowing $Q^*(s, a)$ an agent can choose the best sequence of actions at any state, maximizing its reward.

Obviously, perfectly knowing $Q^*(s, a)$ it is not always possible, especially in complex environment. Still it is possible to discover $Q(s, a, \theta)$, through a machine learning model, where θ are the parameters of the trained model so that $Q(s, a, \theta) \approx Q^*(s, a)$. In the case of deep Q network (DQN) agents, the model adopted is a deep neural network, and θ are its weights, used to approximate the optimal action-value function. In our study we used as underlying model to approximate the Q-function an LSTM based neural network. The network weights may be adjusted through training using as loss function, that varies at each timestep, the mean squared error defined as follows:

$$L_i(\theta_i) = \mathbf{E}_{(s,a,r,s')}\left[(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)\right]^2 \tag{3}$$

in which γ is a discount factor determining a penalty for more future reward, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i used in place of the optimal and unknown $max_{a'} Q^*(s', a')$.

Therefore, in contrast to supervised learning where targets are fixed before learning begins, the targets depends on the network weights. Though, since θ_i^- is kept fixed at the ith optimization of $L_i(\theta_i)$, all the optimization problems at each iteration are well defined. In our case we used a so called soft update of θ which updates the parameters at each iteration on the basis of a coefficient β accordingly to the formula: $\theta_{i+1} = (1 - \beta)\theta_i + \beta\theta_i^-$, where $\beta \in (0, 1)$.

It is worth noting that this algorithm is model-free as it solves the reinforcement learning task directly, without estimating the system transition dynamics. Also it is an off-policy algorithm, since it learns a greedy policy where

$a = argmax_{a'}Q(s, a', \theta)$ but it still ensure, through its behaviour policy, an adequate exploration of the state space through a random action. This allows to discover if there are better actions to perform with respect to the recommended one. In our particular case, for training, we used a Boltzmann Q Policy, which builds a probability law on q values and returns an action selected randomly according to this law while, for prediction purposes, a GreedyQPolicy is adopted which selects the action with the highest reward.

For further details, the full description of the algorithm can be found in [9], which originally proposed it.

4 Methodology

This section is devoted to describe the proposed methodology, which uses two agents trained through reinforcement learning to predict activity and execution time of both the one step ahead event as well as the activities suffix and trace time. First, we give some preliminary definitions of event, trace, and event log.

Let ε be the event universe, i.e., the set of all possible event identifiers. An event $e_i \in \varepsilon$ is characterized by a set of properties. In the context of the present paper, we assume the availability of the following properties: the activity associated to the event, denoted by a_i and the complete timestamp t_i. A trace is a finite non-empty sequence of events $\sigma =< e_1, e_2, \ldots, e_k >, e_i \in \varepsilon, e_i \neq e_j$ for $i \neq j$. We assume the events are ordered with respect to their timestamp, i.e. $t_i < t_j$ for $i < j$. An event log L is a set of traces such that each event appears at most in one trace.

In the following, we describe the pre-processing performed to make event log data suitable for being fed to our system, and the details on the architecture adopted.

4.1 Pre-processing

As it will be clear in the following we use an LSTM as agent's model. Here we describe how log data are processed to generate the input sequences to the model. An event e_i in the sequence is logically represented by 4 components, namely the activity a_i, and three temporal features. Each activity is expressed by a binary vector built using the one-hot encoding of the activity type. One-hot encoding has been chosen as it is an effective and popular way of representing categorical data. Its main advantage is that one-hot encoding transformation does not introduce any order or similarity among the representation of categorical data.

Regarding the three added temporal features, the first is the time passed between Sunday midnight and the event e_i ($\mathrm{t_{w_i}}$ in eq. 4) which is useful to express the seasonality of the process. The second is the time passed between the completion of an event e_i and the completion of the previous event e_{i-1} ($\mathrm{t_{e_i}}$ in eq. 5), thus substantially corresponding to the event duration (plus possible idle time between the two events). The last temporal feature is the time passed between the start of the trace and the event e_i ($\mathrm{t_{t_i}}$ in eq. 6), which gives information

about the progression of the trace. This last one is particularly relevant since there may be a strong correlation between the performed actions and the "age" of the process case.
Formally:

$$t_{\mathrm{w_i}} = \frac{t_{\mathrm{i}} - t_{\mathrm{w_0}}}{\Delta t_{\mathrm{w}}} \tag{4}$$

$$\begin{cases} t_{\mathrm{e_i}} = 0 & \text{if } i = 1, \\ t_{\mathrm{e_i}} = \frac{t_{\mathrm{i}} - t_{\mathrm{i\text{-}1}}}{\Delta_{\max_e}} & \text{otherwise} \end{cases} \tag{5}$$

$$t_{\mathrm{t_i}} = \frac{t_{\mathrm{i}} - t_{\mathrm{0}}}{\Delta_{\max_t}} \tag{6}$$

where t_i is the timestamp of the event at index i, t_{w_0} is the timestamp of the last passed Sunday midnight, and $\mathrm{t_0}$ is the start timestamp of the process. Δt_w $\Delta_{\max_e}$ and $\Delta_{\max_t}$ are normalization factors to make features varying in the range $[0, 1]$, as it improves the performance of the network. Δt_w is the amount of time in a week, while $\Delta_{\max_e}$ and $\Delta_{\max_t}$ are, respectively, the maximum event duration and the maximum trace duration. It is also worth noting that given t_{e_i} and both $t_{w_{i\text{-}1}}$ and $t_{t_{i\text{-}1}}$, it is possible to derive the value of both t_{w_i} and t_{t_i}.

4.2 Learning Architecture

The overall architecture is shown in figure 1. In the Figure, dashed lines enlighten the learning phase, while solid lines refer to the prediction phase. In the system, we have two different agents. Both take as input a sequence of events, in which every event is defined by the three temporal features and the one-hot encoding of the activity as explained before. One agent predicts the one step ahead activity, the next one that will be performed, while the other is devoted to predict its completion time. As said, every DQN agent has an underlying neural network that models the reward function. For each of our two agents, we used an LSTM based neural network to learn and approximate the optimal $Q^*(s, a)$, instead of training them using ground-truth labels, typical of supervised learning. This is done, through the agents' interaction with their respective environment, thanks to which they receive their reward. The LSTM architecture have been chosen because of its widespread adoption in predictive process mining. We hasten to note that DQN agents only work with a discrete action space and therefore they are unable to produce continuous outputs. To address this issue we divided the output time in bins, each representing the range in which the estimated time falls, and we designed the time agent so as to produce bin indexes as outputs.

As explained in section 3, the learning process of our RL agents is based on the notion of transition from a state of the environment to another on the basis of the performed action and its associated reward at each timestep.

We set the reward functions in each environment as binary reward functions: in the time environment, the reward gives a plus one when the predicted bin included the true time, and zero otherwise; similarly in the activity environment the reward gives a plus one when the prediction is correct and a zero otherwise.

For the one step ahead prediction of the next activity and time the two agents work in isolation exploiting their underlying LSTM network model to perform their prediction. For suffix prediction the situation is more complex, as each agent has access only to the information of the first part of the trace. In particular, it reads only the first k events where k is the window size. Hence, each agent needs to rely both on its own prediction and on the other agent's prediction to have all the required inputs for predicting more than one step ahead, as the true information is not available. In a way, the two agents cooperates exchanging messages to inform the other of their prediction, at each timestep. This way the whole sequence may be predicted using the predicted information when the true one is not available. All this is iterated until the end of the trace is predicted.

Formally, at the first iteration we consider the sequence $\sigma_k =< e_1, e_2, \ldots, e_k >$ of events of length k (window size), where e_j be the j-th event of a trace, which is characterized by the tuple $< a_j, t_{w_j}, t_{e_j}, t_{t_j} >$. The time predictor agent α_t and the action predictor agent α_a are defined as follows:

$$\alpha_t : \sigma_k \mapsto t'_{e_{k+1}},$$

$$\alpha_a : \sigma_k \mapsto a'_{k+1},$$

where apex denotes the predicted value. Each agent will inform the other of its prediction and therefore the predicted next event e'_{k+1} will be characterized by the tuple $< a'_{k+1}, t'_{w_{k+1}}, t'_{e_{k+1}}, t'_{t_{k+1}} >$, where $t'_{w_{k+1}}$ and $t'_{t_{k+1}}$ are derived from $t'_{e_{k+1}}$, t_{t_k} and t_{w_k}. Then a new prediction will be performed by each agent using as input $\sigma_{k+1} =< e_2, \ldots, e_k, e'_{k+1} >$. Iterating at the i-th step, the sequence σ_i will be formed by $k-i$ real events and i predicted ones. The algorithm is iterated until the end event of the process is predicted.

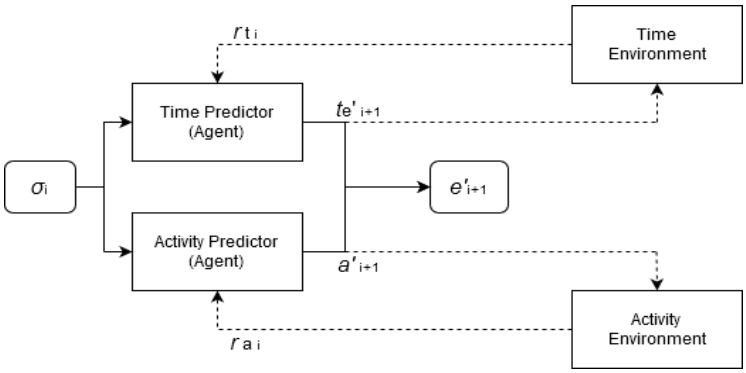


Fig. 1. Overall architecture.

5 Evaluation

In this section we empirically evaluate the performance of the proposed approach. Results are compared with those of other approaches using LSTM networks for uniformity reasons, so to remark the contribution of RL paradigm.

The following subsections describe the experimental setup, reference metrics and results.

5.1 Experimental setup

Dataset The experimental dataset is a subset of an event log from the Business Process Intelligence Challenge (BPI'12) [2] which contains data from the application procedure for financial products at a large financial institution. This process consists of three subprocesses: one that tracks the state of the application, one that tracks the states of work items associated with the application, and a third one that tracks the state of the offer. Since our goal is to predict the coming events and their timestamps, events that are performed automatically aren't considered relevant. Therefore, we limit our evaluation to the work items subprocess (BPI'2012 W): the one containing events that are manually executed. As done in [16], to perform our experiments we used chronologically ordered first 2/3 of the traces as training data, and evaluate the activity and time predictions on the remaining 1/3 of the traces.

The dataset has been pre-processed as explained in section 4. For what concerns the setting of bins defining the output values of the time agent, we analyzed the whole distribution of events duration in the dataset. This allowed to set the various ranges so as to both balance the number of elements in a bin and to maintain a reasonable similarity between elements in the same bin.

The resulting bin endpoints are [0, 1, 10, 60, 120, 240, 480, 1440, 2880, 4320, 7200, 10080, 14400, 20160, 30240, 40320, 50400] expressed in minutes. Also note that the chosen endpoints correspond to meaningful time frames such as hours, days or weeks. Figure 2 shows the distribution of events duration in the dataset. The x-axis is in logarithmic scale for visualization purposes.

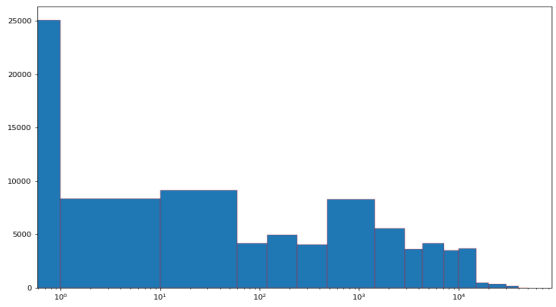


Fig. 2. t_e distribution in bins

Agents We performed the experiments using Keras-rl [12], running on a machine with two NVIDIA GeForce GTX 1080, a i7 8700K CPU @3.70 GHz and 32GB RAM. Each agent was trained for 600000 steps and is characterized by the use of a sequential memory of dimension 500000, a BoltzmannQPolicy clipped in range (-15,15) as behaviour train policy, and a GreedyQPolicy as test policy; the target function was updated through soft update using $\beta = 10^{-2}$ as coefficient. The underlying neural network has two hidden LSTM layers with 200 neurons each and ReLU activation; during training we used an Adam optimizer with a learning rate of 10^{-3}. This configuration was kept for all the tested window size, as it had the best performance for approximating the Q function, between those tested.

5.2 Metrics

In order to properly compare our results with previous work, we adopted the same evaluation metrics.

One-step ahead prediction We evaluate our results in terms of accuracy, for the next activity prediction, and in terms of mean absolute error (MAE) in days, for the predicted time. For the purpose of comparison with the baselines we use the MAE but it is important to remember that our time agent predicts ranges of time. Therefore, since we need a continuous value for the time in order to compute the MAE, we choose for this the inferior endpoint of the bin predicted as value. For example, if the predicted bin is the third one, which corresponds to range $[10, 60)$, the time used for computing the MAE will be 10 minutes.

Suffix prediction For the suffix completion time prediction we consider the absolute trace duration error (TDE)

$$TDE = |t'_{t_f} - t_{t_f}| \tag{7}$$

where, with some abuse of notation, f refers to the final event in the true and estimate trace, hence t_{t_f} (t'_{t_f}) represents the total duration of the true (estimated) trace. The TDE is then averaged over all traces.

For evaluating the accuracy of the activity suffix prediction the most well known and used distance is the Damerau-Levenshtein distance, which is defined as the minimum number of deletion, insertion, substitution and transposition operations needed to transform the first string to the second. In particular, this distance can be normalized dividing its value for the length of the longer string. What we adopted for comparison purposes is the Damerau-Levenshtein similarity expressed as one minus the normalized Damerau-Levenshtein distance.

5.3 Results

In Table 1 we present the performances achieved for the one step ahead prediction tasks. For next completion time prediction (Table 1.(a)) we compare

our results with the best reported by Tax et al. [16] for different window sizes. Table 1.(b) reports the accuracy of the next activity prediction of our method, and the ones reported by Tax et al. [16] and Camargo et al. [1]. In [1] the next completion time task is not addressed. In Table 1.(a) the row "All" reports the average performance over all the tested window sizes. In [16] these correspond to all the values in the range [2,20], whereas in our case we considered the set {2,3,4,5,6,7,10,20}.

It can be seen that our performance in the next completion time prediction are clearly better than the baseline, whilst our accuracy is worse. In particular, the relative improvement in the case of completion time prediction is about 27%, and the relative accuracy degradation is only about 8% with respect to best result provided by [1]. These results may be justified as follows. DQN agents optimize a cumulative reward function that takes into account rewards on future actions, in a sense trying to simulate the future. Completion times show a form of dependency on the total trace duration. For instance, overestimating the duration of early events will lead to an excessively long overall trace duration estimate. This may guide the learner through states with a better generalization ability. On the contrary, a similar relation does not exist for activities in the considered setting, where only the structural perspective of the process (i.e the workflow) is taken into account. Thus enriching the log with other perspectives and in particular with data regarding case-specific and event-specific properties may likely highlight dependencies among activities and thus lead to improved results. We plan to verify such hypothesis in future work.

Table 1. Comparison of performances for the one step ahead prediction tasks. (a) Next completion time. (b) Next activity.

	MAE (days)	
Window size	Ours	Tax et al.
2	1.34	1.69
10	1.05	1.45
20	0.62	0.98
All	1.17	1.59

(a)

Accuracy		
Ours	Tax et al.	Camargo et al.
71.3%	76%	77.8%

(b)

We also show in Table 2 the performance achieved in suffix prediction tasks. The results confirm a better behavior of the proposed RL architecture on the completion time prediction than on the activity prediction task. For the former, the relative improvement is about 21%, which is in line with the one step ahead performance. For the latter, we observe a much worse performance degradation of about the 66% with respect to [1], and about 50% with respect to [16]. This is in part due to an expected error propagation effect, since errors committed at the early suffix prediction stages progressively compromise all the subsequent ones. As another issue reducing our systems performance, we observed that the

event agent struggle to predict the end of the trace, leading to excessively long traces. To verify this, we calculated the DL-similarity truncating the predicted traces to the length of the true traces, discovering that performances improves up to a DL-similarity of 0.2974, which is comparable with the accuracy obtained by [16]. For what concerns computational complexity, clearly the time required to train an RL agent is much higher than the LSTM alone. We experimented an increase factor of about 20x of the required training time. This is a well known characteristics of RL training although alternative techniques with better computational performance have been proposed [5]. We plan to investigate them in future work.

Table 2. Comparison of performances for the suffix prediction tasks. (a) Completion time. (b) Next activities.

	mean TDE (days)		
Window size	Ours	Tax et al.	Camargo et al.
2	12.66	≈ 14	≈ 11
10	6.17	≈ 9	≈ 9
20	4.63	≈ 6	≈ 9

(a)

DL-similarity		
Ours	Tax et al.	Camargo et al.
0.174	0.3533	0.525

(b)

6 Conclusions and Future Works

The main contribution of this paper is to provide a preliminary study of the applicability of reinforcement learning techniques to predictive process monitoring tasks. In particular we used Deep Q Networks agents to address both the one step ahead activity and completion time prediction, and the trace suffix outcome prediction. Through our experiments on the BPI'2012 popular benchmark dataset, we showed that DQN agents can fully exploit time information, achieving results that significantly outperforms state of the art approaches based on LSTM architectures, while the plain workflow information seems to be insufficient to train an RL agent for the activity prediction task. The present paper also highlights several interesting research directions. First of all, as already noticed the proposed approach may be further refined through the use of case-specific data, and event-specific data. Second, more complex reward functions may be used in order to weight the activity and/or case importance, for instance something like the amount of money involved in a loan procedure or the cost to perform a specific activity. Third, alternative RL techniques can be considered, to investigate both their efficiency and accuracy performances.

References

1. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate lstm models of business processes. In: Proceedings of the 17th International Conference on Busi-

ness Process Management (BPM'19), Lecture Notes in Computer Science, 11675. pp. 286–302. Springer International Publishing (2019)
2. van Dongen, B.: BPI Challenge 2012 event log (2012). https://doi.org/doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
3. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129 – 140 (2017), smart Business Process Management
4. Huang, Z., van der Aalst, W., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. Data & Knowledge Engineering **70**(1), 127 – 145 (2011)
5. Lange, S., Gabel, T., Riedmiller, M.: Batch reinforcement learning. In: Reinforcement Learning. Adaptation, Learning, and Optimization, vol. 12. Springer, Berlin, Heidelberg (2012)
6. Lin, L., Wen, L., Wang, J.: MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence, pp. 118–126. Proceedings, Society for Industrial and Applied Mathematics (2019)
7. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Advanced Information Systems Engineering. pp. 457–472. Springer International Publishing, Cham (2014)
8. Marquez-Chamorro, A., Resinas, M., Ruiz-Cortes, A.: Predictive monitoring of business processes: A survey. IEEE Transactions on Services Computing **11**(6), 962–977 (2018)
9. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
10. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: 2019 International Conference on Process Mining (ICPM'19). pp. 129–136 (2019)
11. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Predictive process mining meets computer vision. In: "Business Process Management Forum (BPM'20), Lecture Notes in Business Information Processing. vol. 392, pp. 176–192. Springer International Publishing (2020)
12. Plappert, M.: keras-rl. https://github.com/keras-rl/keras-rl (2016)
13. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science **362**(6419), 1140–1144 (2018)
14. Silver, D., Huang, A., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018)
16. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Advanced Information Systems Engineering. CAiSE 2017. Lecture Notes in Computer Science, vol 10253. pp. 477–492 (2017)
17. Vinyals, O., Babuschkin, I., et al.: Grandmaster level in starcraft "ii" using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)

An Alignment Cost-Based Classification of Log Traces Using Machine-Learning

Mathilde Boltenhagen[1], Benjamin Chetioui[2], and Laurine Huber[3]

[1] Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Gif-sur-Yvette, (France)
mathilde.boltenhagen@lsv.fr
[2] University of Bergen, Department of Informatics, Bergen, Hordaland, (Norway)
benjamin.chetioui@uib.no
[3] Université de Lorraine, LORIA (UMR 7503), Nancy, (France)
laurine.huber@loria.fr

Abstract. Conformance checking is an important aspect of process mining that identifies the differences between the behaviors recorded in a log and those exhibited by an associated process model. Machine learning and deep learning methods perform extremely well in sequence analysis. We successfully apply both a Recurrent Neural Network and a Random Forest classifiers to the problem of evaluating whether the alignment cost of a log trace to a process model is below an arbitrary threshold, and provide a lower bound for the fitness of the process model based on the classification.

1 Introduction

With the cost of computer memory becoming negligible, organizations have become able to store extremely complex event logs from their systems. Process Mining (PM) is a field of study that attempts to make sense of these logs by producing corresponding *process models*. As decision makers increasingly rely on such models, it is crucial to ensure that they model the targeted systems reliably. *Conformance checking* is an entire subfield of PM that aims at defining the key criteria of a good process model [1]. As of today, the four main criteria that are considered are *fitness*, *precision*, *generalization*, and *simplicity*. Because of the complexity of the involved data and of the resulting process models, the fitness criterion is the only one unanimously accepted in the community. Computing the fitness requires alignments of the event logs with the process model, which often is costly [2,3] and for which a trade-off is possible between higher result quality and lower computational complexity. The need for such a compromise begs the question: is it possible to extract high-quality conformance checking information through a less complex process?

To motivate such research, the 2016 *Process Discovery Contest* invited scientists to study model compliance from a classification-oriented perspective [4]. The event logs were classified in two classes — *compliant* and *deviant* — using pure data mining techniques. By encoding event logs into sequences of activities called *log traces*, it is possible to perform such a classification using Recurrent

Neural Networks (RNNs). RNNs are at the core of significant progress in other fields of Computer Science such as Natural Language Processing, or Bioinformatics [5]. The PM community has recently shown significant interest in RNNs, but principally on the topic of *Predictive Business Process Monitoring* [6,7,8,9,10,11].

In this paper, we focus on the efficiency of Deep Learning (DL) and classical Machine Learning (ML) methods in conformance checking scenarios. Our core contribution is an application of a RNN and a Random Forest (RF) classifier to the problem of classifying traces based on their alignment costs to a reference process model. We provide some theoretical properties of the fitness along with reproducible experiments.

2 Related Work

The classification of log traces has been studied in the context of system deviation analysis. Such works generally consider two classes of processes (*normal* and *deviant*) and aim at explaining why discrepancies occur and deviant processes arise. Nguyen et al. defined trace classes from data attributes and investigated the problem of classification using decision trees, the k-Nearest Neighbors algorithm and neural networks [12]; Sun et al. and Bose et al. investigated labeled traces and association rules mining methods that can be used to extract human readable results from them [13,14]. Similarly, Bellodi et al. provided a method to classify log traces using Markov Logic formulas [15]. One glaring difference between these works and ours is that we have an oracle at our disposal to classify our traces, i.e. a process model.

The application of Long Short-Term Memory (LSTM) networks to the problem of predicting the next event in a business process was previously investigated in several works [6,7,8,9]. In lieu of RNNs, Pasquadibisceglie et al. investigated Convolutional Neural Networks for the same purpose [10]. Building on top of all these approaches, Taymouri et al. tackled the problem by implementing a Generative Adversarial Network, with promising results [11].

The present paper is probably most similar to the work of Nolle et al. [16], whose results, which are based on RNN-based alignments, are extremely promising, though they perform anomaly detection instead of log trace classification.

3 Preliminaries

In this section, we provide some background and notation for both PM and ML.

3.1 Log Traces, Process Model, Fitness and Alignments

We represent event data as log traces.

Definition 1 (Log traces). *Let $\mathcal{A}$ be a set of activities. We define a* log *L as a finite multiset of sequences $\sigma \in \mathcal{A}^*$, which we refer to as* log traces.

$\langle open, read, wait, wait, close \rangle$
$\langle open, read, close \rangle$
$\langle write, wait, close \rangle$
$\langle open, wait, write, close \rangle$

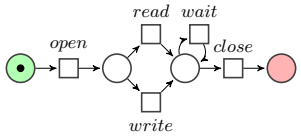


Fig. 1: A log L and an associated process model M

Process models can be generated from an event log; these models extrapolate a set of possible *runs* from the recorded log traces exhibited in the aforementioned event log. An example of a log and associated process model is provided in Figure 1.

Definition 2 (Runs of a process model). *Let M be a process model defined over a set of activities $\mathcal{A}$. We write $Runs(M) \subseteq \mathcal{A}^*$ the set of sequences generated by M.*

This paper does not discuss the structure of process models; for a given model M, we consider the set $Runs(M)$ to be a sufficient description of M. How well M models a log is measured by the *fitness* criterion and can be computed based on $Runs(M)$ as the minimal cost of aligning each log trace to a run of M.

Definition 3 (Alignment Cost, Optimal Alignment). *Given a log trace $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle \in L$, and a process model M, we define the alignments of σ with M as sequences of pairs (*moves*) $\langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$ with $p \leq m + n$ such that, for a given index i and a given run $u = \langle u_1, \ldots, u_n \rangle \in Runs(M)$:*

- *each move (σ'_i, u'_i) is either: a synchronous move (σ_j, u_k) with $\sigma_j = u_k$, a log move $(\sigma_j, \gg)$, which represents the deletion of σ_j in σ, or a model move $(\gg, u_k)$, which represents the insertion of u_k in σ, where $j \in \{1, \ldots, m\}$ and $k \in \{1, \ldots, n\}$;*
- *the left projection $\langle \sigma'_1, \ldots, \sigma'_p \rangle$ of the alignment to $\mathcal{A}^*$ (which drops the occurrences of $\gg$), yields σ;*
- *the right projection $\langle u'_1, \ldots, u'_p \rangle$ of the alignment to $\mathcal{A}^*$ (which drops the occurrences of $\gg$), yields u.*

We call alignment cost *the count of non-synchronous moves in the alignment. An* optimal alignment *is an alignment in which the alignment cost is the minimum possible given σ and M.*

The table below describes an optimal alignment of the log trace $\langle open, wait, write, close \rangle$ with the process model drawn in Figure 1. Since the alignment contains one non-synchronous move, its cost is 1.

trace	*open*	*wait*	*write*	*close*
run	*open*	$\gg$	*write*	*close*

We compute the fitness of a process model with regards to a trace as follows:

$$\text{fitness}(\sigma, M) = 1 - \frac{\text{minCost}(\sigma, \text{select}(\sigma, M))}{|\sigma| + \min\limits_{u' \in Runs(M)} |u'|} \quad (1)$$

where $\text{select}(\sigma, M)$ returns a run $u \in Runs(M)$ such that the set of alignments of σ with M using u contains an optimal alignment, and $\text{minCost}(\sigma, u)$ returns the minimum cost of aligning σ with M using a run u.

A trace is said to be *fitting* when its fitness is 1, i.e. when its optimal alignment has a cost of 0. We define the fitness of a process model M with regards to a log L to be the average of the fitness of M with regards to each log trace of L.

3.2 Supervised Learning from Sequences

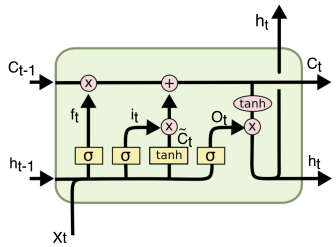
There are several approaches towards training classification models from sequential data in a supervised way. They have in common that they must encode sequences of variable lengths as fixed-size vectors; these vectors are subsequently used as training examples for the classifier, which learns a classification model from them. The quality of the model is then assessed using several metrics and methods, based on its ability to accurately classify new inputs.

Building a Model One can construct the vectors referenced above in different ways, e.g. by ignoring the order of the sequences (Bag-of-words) in the hope that knowledge about the frequency of each word in the sequences is sufficient to train a classifier (e.g. a RF classifier), or by training Deep Neural Networks able to encode the ordering of the sequences in the vectors (e.g. a LSTM network).

Long Short-Term Memory networks are RNNs able to learn and remember over long sequences of inputs [5]. They achieve that by integrating neurons specifically designed to determine whether a piece of information should be remembered or forgotten, depending on whether it is relevant for classification. Figure 2 gives the structure and relevant equations of an LSTM cell.

When one uses LSTM networks for sequence classification, the sequences (represented as sequences of integers) are usually first passed through an embedding layer before being passed through the LSTM layer; the prediction is then the output of a dense layer. One may add dropout layers to the network, in order to randomly ignore a percentage of units during training to avoid overfitting. The specificity of this architecture is that the whole sequence is fed as input to the network and that the embedding is learned through the training process; this permits learning a representation of the sequence that somehow embeds its sequential properties.

Definition 4 (Bag-of-words (BoW) encoding). *For an alphabet $\mathcal{A}$ and a sequence $\sigma \in \mathcal{A}^*$, a Bag-of-words encoding canonically maps σ to a multiset of words of $\mathcal{A}$.*



$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

where b_g is the bias added at gate g, W_g is the weight vector for gate g, x_t is the current input, C_{t-1} the memory of last hidden unit, and h_{t-1} the output of last hidden unit.

Fig. 2: LSTM cell (adapted from [17])

In its simplest version, the multiset is encoded as a vector of integers $\mathcal{X}_\sigma$, and the element at index i in $\mathcal{X}_\sigma$ gives the count of the word at index i in $O_\mathcal{A}$ in the sequence σ, where $O_\mathcal{A}$ is a vector containing exactly all the elements of $\mathcal{A}$ in some arbitrary order. For instance, the respective BoW encodings of the log traces in Figure 1 are $\langle 1,1,2,1,0\rangle$, $\langle 1,1,0,1,0\rangle$, $\langle 0,0,1,1,1\rangle$, and $\langle 1,0,1,1,1\rangle$ for $O_\mathcal{A} = \langle open, read, wait, close, write\rangle$.

Random Forests (RFs) are an ensemble learning method for classification. A RF constructs a bootstrapped collection of decision trees, i.e. a collection of decision trees that are sampled with replacement. Each decision tree consists of inner dichotomous nodes representing tests on random subsets of features, and of leaf nodes representing the possible output classes. The class of a given input can be predicted by taking the majority vote of the classification trees [18]. These decision trees can help to understand which features are important for classification, since every output can be represented as a list of decisions taken at the dichotomous nodes.

Validating a Model We recall some metrics used to evaluate classification models, as well as one famous validation technique, namely the *K-fold cross-validation*.

Definition 5. *In the following, given a classification model C and a given input i, we write $y_{C,i}$ the actual class of the input and $\hat{y}_{C,i}$ its predicted class by C.*

Definition 6 (Accuracy). *For a given classification model C and an input i, we say that the classification is* accurate *when $y_{C,i} = \hat{y}_{C,i}$. For a set of inputs S, we define $E_{S,C} = \{i : i \in S,\ \hat{y}_{C,i} = y_{C,i}\}$. The accuracy $acc_C(S)$ of the classification of S by C is given as $acc_C(S) = \frac{|E_{S,C}|}{|S|}$.*

Definition 7 (Cross-Entropy Loss). *For a given binary classification model C and a given set of inputs S, there exists an error function called* cross-entropy loss *$loss_C(S)$ defined by $loss_C(S) = \frac{1}{|S|}\sum_{i\in S} -\log(P(\hat{y}_{C_i} = y_{C_i}))$.*

K-fold cross-validation K-fold cross-validation is a model validation technique used to lower the biases that may emerge when one only selects one training set and one testing set. Given $K \in \mathbb{N}^*$, the dataset D is split into K i-indexed subsets D_i. For each subset, one trains a model using $D \setminus D_i$ as the training set, and subsequently evaluates it using D_i as the testing set. The performance of the model is then summarized using the mean and variance of the evaluation scores.

4 Classifying Traces and Bounding the Fitness of a Model

The fitness of a log trace to a process model represents important information in conformance checking. Computing the fitness requires computing alignments of the trace with the model, which is a costly process. In this section, we present a binary classification of log traces based on their closeness to a process model: the *Alignment Cost Threshold-based Classification* (ACTC). This classification provides means of extracting relevant information at a much lower cost than alignments, while still guaranteeing a lower bound for the fitness of a process model to a log.

Definition 8 (Alignment Cost Threshold-based Classification). *Let M be a process model and L be a log. For a given alignment cost threshold $t_{\mathcal{AC}} \in \mathbb{N}$, the ACTC maps each log trace $\sigma \in L$ to one of two classes depending on its minimal alignment cost $c_{\sigma,M}$:*

- *the* positive *class L_{pos} if $c_{\sigma,M} \leq t_{\mathcal{AC}}$;*
- *the* negative *class L_{neg} otherwise.*

The $t_{\mathcal{AC}}$ parameter allows us to have more flexibility — in that we can now work with arbitrarily close traces instead of only fitting ones — and to control the balance of our two classes.

Theorem 1. *Given the ACTC of a log L for a model M and a cost threshold $t_{\mathcal{AC}}$, the following holds:*

$$\mathit{fitness}(L, M) \geq \frac{\sum_{\sigma \in L_{pos}} 1 - \frac{t_{\mathcal{AC}}}{|\sigma| + \min_{u \in Runs(M)} |u|}}{|L|} \tag{2}$$

i.e. $\mathit{fitness}(L, M)$ is bounded from below.

Proof. The fitness of a process model M with regards to a log L is defined as the average of the fitness of M with regards to each log trace of L, i.e.

$$\text{fitness}(L, M) = 1 - \frac{\sum_{\sigma \in L} \frac{\text{minCost}(\sigma, \text{select}(\sigma, M))}{|\sigma| + \min_{u' \in Runs(M)} |u'|}}{|L|}. \tag{3}$$

Let there be an ACTC of cost threshold $t_{\mathcal{AC}}$. For every $\sigma \in L$, we have

$$\text{fitness}(\sigma, M) \geq \begin{cases} 0 & \text{if } \sigma \in L_{\text{neg}} \\ 1 - \frac{t_{\mathcal{AC}}}{|\sigma| + \min\limits_{u' \in Runs(M)} |u'|} & \text{if } \sigma \in L_{\text{pos}} \end{cases}, \tag{4}$$

since $t_{\mathcal{AC}}$ is the highest allowed alignment cost for a trace to be classified into L_{pos}. It follows trivially that:

$$\text{fitness}(L, M) \geq \frac{\sum\limits_{\sigma \in L_{\text{pos}}} 1 - \frac{t_{\mathcal{AC}}}{|\sigma| + \min\limits_{u \in Runs(M)} |u|}}{|L|}. \tag{5}$$

In the following, we write $B = \text{fitness}(\sigma, M)$ for any $\sigma \in L_{\text{pos}}$. □

Taking a small value for $t_{\mathcal{AC}}$ results in a large B, but a potentially smaller cardinality for L_{pos}; on the other hand, a large $t_{\mathcal{AC}}$ will induce a larger cardinality for L_{pos} but a smaller B. The aim of the following is to compute B from predictions, i.e. in a case where L_{pos} is built using a predictive approach. In this case, there is a risk that traces will be classified erroneously. We show in the next sections that classification models are good enough in practice to guarantee a lower bound of their fitness that is very close to the one outlined above.

5 Experiments

In this section, we present our datasets; we follow by describing how we parameterize our classification models; finally, we present our experimental results.

5.1 Alignment Datasets

The ACTC requires a training set of alignments; for that purpose, we have created alignments datasets that contain the trace variants of each dataset (i.e. the unique sequences in the log) and their minimal alignment costs for several process models[4]; that way, we rid our results of the noise induced by duplicate traces.

We ran our experiments on the three largest logs from the Business Process Intelligence Challenges available at the time of writing, using models from the work of Augusto et al. [19]. The models were discovered using the preprocessing method of Conforti et al. [20], and then either the Inductive Miner (IM) [21], the Split Miner (SM) [22], or the Heuristic Miner (SHM) [23]. Table 1 summarizes the relevant pieces of information pertaining to the datasets.

[4] https://github.com/BoltMaud/An-Alignment-Cost-Based-Classification-of-Log-Traces-Using-ML

Log	Number of Trace Variants	Method of Model discovery	Average Alignment Cost	Median Alignment Cost	Dataset Name
BPIC_2012	4 366	Noise Filter + IM	2.14	2.00	A^{im}_{2012}
		Noise Filter + SM	3.02	3.00	A^{sm}_{2012}
		Noise Filter + SHM	7.60	6.00	A^{shm}_{2012}
BPIC_2017	15 930	Noise Filter + IM	14.90	13.00	A^{im}_{2017}
		Noise Filter + SM	15.03	13.00	A^{sm}_{2017}
		Noise Filter + SHM	16.31	14.00	A^{shm}_{2017}
BPIC_2019	11 973	Noise Filter + IM	24.38	6.00	A^{im}_{2019}

Table 1: Event log description and alignment costs

For each log, we also generate a set of 1000 random mock traces of lengths varying between 1 and the length of the longest trace in the log. These traces have, in most cases, a very high alignment cost with regards to the process models.

5.2 Learning Methods

We train two classifiers, namely a RF on BoW-encoded sequences, and a LSTM network on sequences whose encoding embeds the sequential properties of the activities. The general overview of the training process is shown in Fig. 3.

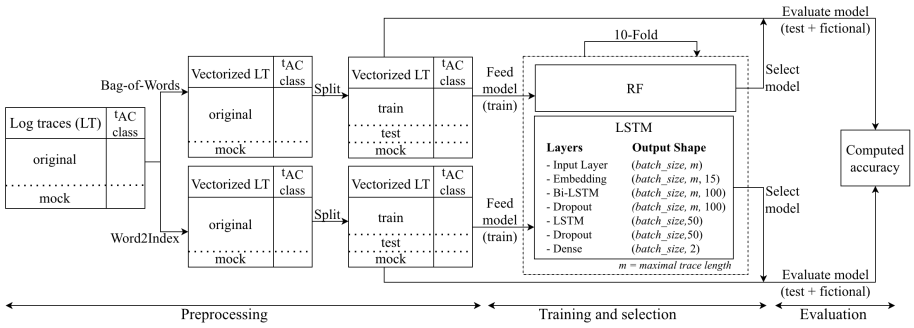


Fig. 3: Overview of the experimental setup

LSTM Network This model takes constant-length vectors of integers as inputs, in which a given integer corresponds to exactly one activity. Traces that are shorter than the expected length of the vectors are padded as needed.

The architecture of the model we train is given in Figure 3. The *input* layer takes a vector of size m (corresponding to the length of the longest trace in the log) containing elements belonging to the set of all the actions taken in the log traces. The vector is encoded into a vector of 15 elements using an *embedding* layer. The resulting vector is then fed to a *bi-LSTM* layer — ensuring that the left and right contexts of the actions in the input traces are remembered — and then to another simpler LSTM layer. *Dropout* layers with a frequency rate of

0.5 are added to prevent overfitting. The *dense* layer uses the softmax activation function to output the predicted classes, thus ensuring that they are mutually exclusive. We train the model for 10 epochs and with a batch size of 50 instances [5].

RF Classifier The RF classifier does not take into account the order of the events, as it takes as input vectors that represent an ensemble of features, in our case activities. The classifier is thus trained with vectors resulting from a BoW encoding of the traces.

The target values, i.e. the prediction classes, are 0 (negative) or 1 (positive) depending on the minimal alignment cost of the sequence.

We set up 3 verification steps: first, we split the dataset into a training set (67%) and a testing set (33%) using a 10-fold cross-validation on the training sets to find the best predictive model in terms of accuracy. Second, we predict the classes of the sequences in the testing sets, and compare the accuracy during training to the accuracy during testing; they should be similar. Finally, we feed randomly generated traces with a high alignment cost to the predictive model; they should always be classified negatively.

5.3 Results and Interpretation

We built two distinct classifiers — one RNN and one RF — for each pair (d, m), with d one of the 7 datasets in Table 1, and m one of the possible alignment costs for the model; each pair represents an ACTC problem.

Table 2 summarizes the results of the experiments, where $t_{\mathcal{AC}}$ is the median of the alignment costs given in Table 1. The table contains the accuracies and losses for our testing data, and we compare our running times with the ones of ProM[6] for computing the alignments.

Both learning models exhibit good accuracy and low losses, thus confirming the potential of predictive approaches for the problem of alignment. The predicted lower bound of the fitness is computed from the traces classified as positive and is very close to the exact fitness lower bound. However, we note a significant difference between the actual fitness and these lower bounds. This is because the fitness function we use is coarse-grained, in that it gives a purely binary score denoting whether a log trace is classified as negative or positive. Despite this weakness, it remains somewhat useful as a heuristic to decide which of two models better fits a trace. It is also worth noting that our binary classification is straightforward to understand, whereas understanding alignments tends to require more expertise; such a classification is therefore likely to be very valuable to decision makers.

[5] The size of the embedding layer, the number of epochs, the batch size, and the stack of LSTM layers were chosen after several initial experiments, as they were the parameters that yielded the best results.

[6] https://www.promtools.org

Alignments	Fitness	$t_{\mathcal{AC}}$	% of positive	Fitness Lower Bound	RNN				Random Forest				ProM
					Acc	Loss	Predicted Fitness Lower Bound	Avg. Runtime (ms)	Acc	Loss	Predicted Fitness Lower Bound	Avg. Runtime (ms)	Avg. Runtime (ms)
A^{im}_{2012}	0.950	2	73	0.695	0.999	0.011	0.695	12.00	0.988	0.057	0.700	0.06	42.28
A^{sm}_{2012}	0.932	3	73	0.670	0.829	0.377	0.745	19.72	0.820	0.472	0.713	0.08	52.85
A^{shm}_{2012}	0.837	6	56	0.476	0.969	0.104	0.491	23.75	0.972	0.136	0.479	0.06	99.89
A^{im}_{2017}	0.874	13	53	0.463	0.984	0.047	0.473	10.01	0.979	0.056	0.467	0.03	5.12
A^{sm}_{2017}	0.819	13	52	0.415	0.985	0.049	0.420	2.70	0.985	0.053	0.421	0.03	7.72
A^{shm}_{2017}	0.794	14	52	0.400	0.981	0.055	0.410	4.05	0.984	0.055	0.405	0.03	33.23
A^{im}_{2019}	0.561	6	53	0.328	0.973	0.078	0.338	15.11	0.958	0.103	0.344	0.03	1.09

Table 2: Alignment Cost Threshold-based Classification by using a RNN and a Random Forest Classifier, with $t_{\mathcal{AC}}$ the median of the alignment costs.

Once the model has been trained, predicting the class of a trace is, in most cases, significantly faster than computing its exact alignment, as summarized in Table 2. One glaring exception is in the case of A^{im}_{2019}, in which computing exact alignments remains roughly 14 times more efficient than performing a prediction using the RNN. This is because the model is very simple (made of only 13 transitions, without loops); this is not surprising and should not matter in practice, as predictive approaches are tools designed to outperform exact approaches in complex cases with big or even intractable search spaces. One noteworthy caveat of using predictive approaches, however, is the fact that the models must be trained before they become able to output predictions. In our experiments, training a model took from 3.18s to 8.97s for our RF classifier, and from 2675.87s to 34837.31s for our LSTM network —both of which involved a 10-fold cross validation.

To better assess the impact of $t_{\mathcal{AC}}$ on our results, we perform a comparison of the predictions with varying $t_{\mathcal{AC}}$ values in Table 3. We summarize the accuracy, loss, and distribution into the two output classes for the testing data, as well as for randomly generated mock data. We notice that the accuracy drops very fast as $t_{\mathcal{AC}}$ grows larger for the mock data; this is induced by an equally quick drop in the percentage of log traces classified as negative. Given actual log traces however, both classifiers are reasonably accurate in each one of the considered cases. As was the case in Table 2, we note that the predicted lower bound of the fitness is close to the one given by our exact formula. This is also a nice result, although the actual fitness of the process model with regards to the log is pretty far off at 0.837.

6 Conclusion and Opening

We presented a compelling use of ML for conformance checking by constructing binary oracles — using a RF classifier and a LSTM network — that are able to predict with high accuracy whether the minimal alignment cost of a log trace with regards to a process model is below an arbitrary threshold. The method

$t_{\mathcal{AC}}$	Class	%	Fitness Lower Bound	RNN			Random Forest		
				Acc	Loss	Predicted Fitness Lower Bound	Acc	Loss	Predicted Fitness Lower Bound
2	all	100	0.071	0.992	0.029	0.076	0.998	0.009	0.073
	pos	8		0.982	0.214		1.000	0.043	
	neg	92		0.992	0.013		0.998	0.006	
	mock	100	/	0.961	0.108	/	0.904	0.207	/
4	all	100	0.169	0.991	0.042	0.166	0.999	0.016	0.170
	pos	20		0.968	0.151		1.000	0.021	
	neg	80		0.997	0.016		0.998	0.015	
	mock	100	/	0.937	0.303	/	0.876	0.317	/
6	all	100	0.476	0.971	0.104	0.491	0.972	0.150	0.479
	pos	56		0.990	0.066		0.978	0.063	
	neg	44		0.944	0.156		0.962	0.268	
	mock	100	/	0.871	0.543	/	0.837	0.548	/
8	all	100	0.500	0.976	0.092	0.498	0.984	0.077	0.501
	pos	65		0.980	0.079		0.989	0.031	
	neg	35		0.970	0.116		0.974	0.161	
	mock	100	/	0.818	1.189	/	0.782	0.911	/
10	all	100	0.524	0.937	0.165	0.508	0.971	0.103	0.522
	pos	73		0.943	0.100		0.979	0.055	
	neg	27		0.921	0.336		0.949	0.233	
	mock	100	/	0.364	3.759	/	0.620	1.650	/

Table 3: Comparison of the prediction results for different $t_{\mathcal{AC}}$ values for the testing set of A^{shm}_{2012}. The exact fitness for the used sublog is 0.837.

we proposed is more flexible, cheaper, and easier to understand for humans than the one usually used for exact alignments. We furthermore proved the existence of a lower bound for the fitness of a process model. Our work shows that there is a lot of value to be gained in exploring the use of ML methods in conformance checking. Future investigations may include whether the exact minimal alignment cost of a trace with a process model can be predicted from a regression model; another interesting project could build on the work of Nolle et al. [16] to predict optimal alignments of a log trace to a process model.

References

1. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking. Springer (2018)
2. Adriansyah, A.: Aligning observed and modeled behavior. (2014)
3. Van Dongen, B., Carmona, J., Chatain, T., Taymouri, F.: Aligning modeled and observed behavior: a compromise between computation complexity and quality. In: CAISE, Springer (2017)
4. Carmona, J., de Leoni, M., Depaire, B., Jouck, T.: Summary of the process discovery contest 2016. In: Proceedings of the Business Process Management Workshops, Springer. (2016)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation (1997)
6. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems (2017)

7. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: CAISE, Springer (2017)
8. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: BPM, Springer (2019)
9. Lin, L., Wen, L., Wang, J.: Mm-pred: A deep predictive model for multi-attribute event sequence. In: International Conference on Data Mining, SIAM (2019)
10. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: International Conference on Process Mining, ICPM, IEEE (2019)
11. Taymouri, F., La Rosa, M., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In Proc. of BPM, Springer (2020)
12. Nguyen, H., Dumas, M., La Rosa, M., Maggi, F.M., Suriadi, S.: Mining business process deviance: a quest for accuracy. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer (2014)
13. Sun, C., Du, J., Chen, N., Khoo, S.C., Yang, Y.: Mining explicit rules for software process evaluation. In: Proceedings of the 2013 International Conference on Software and System Process
14. Bose, R.J.C., van der Aalst, W.M.: Discovering signature patterns from event logs. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), IEEE
15. Bellodi, E., Riguzzi, F., Lamma, E.: Probabilistic declarative process mining. In: International Conference on Knowledge Science, Engineering and Management, Springer (2010)
16. Nolle, T., Seeliger, A., Thoma, N., Mühlhäuser, M.: Deepalign: Alignment-based process anomaly correction using recurrent neural networks. In: CAISE, Springer (2020)
17. Olah, C.: Understanding LSTM networks (08 2015) [Online; acceded on 02-September-2020].
18. Breiman, L.: Random forests. Machine learning (2001)
19. Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., La Rosa, M., Reißner, D.: Abstract-and-compare: A family of scalable precision measures for automated process discovery. In: BPM, Springer (2018)
20. Conforti, R., La Rosa, M., ter Hofstede, A.H.: Filtering out infrequent behavior from business process event logs. IEEE Transactions on Knowledge and Data Engineering (2016)
21. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM, Springer (2013)
22. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems (2019)
23. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated discovery of structured process models from event logs: the discover-and-structure approach. Data & Knowledge Engineering (2018)

Improving the Extraction of Process Annotations from Text with Inter-Sentence Analysis

Luis Quishpi, Josep Carmona, and Lluís Padró

Computer Science Department
Universitat Politècnica de Catalunya
Barcelona, Spain.
{quishpi,jcarmona,padro}@cs.upc.edu

Abstract. The automatic extraction of formal process information from textual descriptions of processes is a challenging problem, but worth exploring, since it enables organizations to align complementary information that talks about processes. In this paper we continue our previous work on this area, based on defining hierarchical/tree patterns on the dependency trees that arise from the linguistic analysis. We now incorporate a new abstraction layer on these patterns, that consider relationships between nearby sentences. The aim of this extension is to capture inter-sentence relationships that typically arise in textual descriptions of processes. The experiments done on publicly available benchmarks corroborate this intuition, showing a significant rise in the ability to capture all the important control-flow relationships defined in the text.

1 Introduction

As it has been recently acknowledged, there are quite important challenges on applying Natural Language Processing (NLP) techniques in the field of Business Process Management (BPM) [12]. Among the important ones, the extraction of process models from textual process descriptions is a very attractive use case, since the creation of process models consumes up to 60% of the time spent on process management projects. This paper focuses on this challenging task.

Although different approaches have been considered in the last years (see section 2), a number of open challenges remain for reaching a maturity level enabling its widespread adoption. For instance, techniques must be able to identify sentences that provide contextual information, rather than describe process steps. Furthermore, the inherent ambiguity of natural language can lead to different interpretations regarding the process that is described [14].

In this paper we significantly expand the techniques and results recently presented in [9], where we described robust tree-based patterns to be queried over the dependency trees arising from the NLP analysis of the textual descriptions. Patterns in [9] where only applicable in the context of a single sentence, which made our approach unable to extract inter-sentence relationships. The contribution of this paper is therefore the extension of our previous contribution with a

more general set of patterns, resulting in a significant boost in the recall of the original framework (see experiment results in section 5).

The paper is organized as follows: next section shortly describes the work related to this contribution. Section 3 overviews the main components of our proposal, presented in Section 4. Experiments and tool support are reported in Section 5, whilst Section 6 concludes the paper and outlines future work.

2 Related work

For the sake of space, we only report here the related work that focuses on the extraction of process knowledge from textual descriptions [1,4,13], or the work that considers textual annotations in the scope of BPM [7,10].

For the former, the work by Gonçalves et al. [1] adopts important steps to extract the different BPMN elements and the work by Friedrich et al. [4] is acknowledged as the state-of-the-art for extracting process representations from textual descriptions, so we focus our comparison on this approach. As we will see in the evaluation section, our approach is significantly more accurate with respect to the state-of-the-art in the extraction of the main process elements. Likewise, we have incorporated as well the patterns from [13], and a similar outcome is reported in the experiments. The main reason is that approach relies on a deep NLP analysis and patterns on the syntactic structure of the sentence, instead of a shallow analysis and flat patterns.

For the later type of techniques ([7,10]), we see these frameworks as the principal application for our techniques. In particular, we have already demonstrated in the platform `https://modeljudge.cs.upc.edu` an application of the use of annotations in the scope of teaching and learning process modeling[1].

3 Preliminaries

The core of our proposal is the use of deep NLP analyzers to convert a textual description of a process into a syntax-semantic structure. Then, this structure is mined using tree-shaped patterns to obtain a conceptual representation of the process. Although other tools could be used, we resort to FreeLing as a NLP analyzer, TRregex as a tree-oriented pattern matching tool, and ATDP as a conceptual representation support. We describe each of them below.

3.1 Natural Language Processing

Linguistic analysis tools can be used as a means to structure information contained in texts for its later processing in applications less related to language itself. This is our case: we use NLP analyzers to convert a textual description of a process model into a structured representation.

[1] The reader can see a tutorial for annotating process modeling exercises in the `ModelJudge` platform at `https://modeljudge.cs.upc.edu/modeljudge_tutorial/`.

The NLP processing software used in this work is FreeLing[2] [8], an open–source library of language analyzers providing a variety of analysis modules for a wide range of languages. More specifically, the natural language processing layers used in this work are: tokenization & sentence splitting, morphological analysis, PoS-Tagging, Named Entity Recognition, Word sense disambiguation, Dependency parsing, Semantic role labeling and Coreference resolution. The three last steps are of special relevance since they allow the top-level predicate construction, and the identification of actors throughout the whole text: dependency parsing identifies syntactic subjects and objects (which may vary depending, e.g., on whether the sentence is active or passive), while semantic role labeling identifies semantic relations (the *agent* of an action is the same regardless of whether the sentence is active or passive). Coreference resolution links several mentions of an actor as referring to the same entity.

3.2 Annotated Textual Descriptions of Processes (ATDP)

ATDP is a formalism proposed in [10], aiming to represent process models on top of textual descriptions. This formalism naturally enables the representation of a wide range of behaviors, ranging from procedural to completely declarative, but also hybrid ones. Different from classical conceptual modeling principles, this highlight ambiguities that can arise from a textual description of a process, so that a specification can have more than one possible interpretation[3].

ATDP specifications can be translated into linear temporal logic over finite traces [5,2], opening the door to formal reasoning, automatic construction of formal models (e.g. in BPMN) from text, and other interesting applications such as simulation: to generate end-to-end executions (i.e., an *event log* [15]) that correspdond to the process described in the text, which would allow the application of *process mining* algorithms.

ATDP models are defined over an input text, which is marked with *typed text fragments*, which may correspond to *entities*, or *activities*. Marked fragments can be related among them via a set of *fragment relations*.

Entity fragments. The types of entity fragments defined in ATDP are:

- *Role.* The role fragment type is used to represent types of autonomous actors involved in the process, and consequently responsible for the execution of activities contained therein.
- *Business Object.* This type is used to mark all the relevant elements of the process that do not take an active part in it, but that are used/manipulated by process activities.

Activity fragments. ATDP distinguishes two types of activity fragments:

- *Condition.* It is considered discourse markers that mark conditional statements, like: *if*, *whether* and *either*. Each discourse marker needs to be tailored to a specific grammatical structure.

[2] http://nlp.cs.upc.edu/freeling

[3] In this work we consider a flattened version of the ATDP language, i.e., without the notion of *scopes*.

– *Task and Event.* Those fragment types are used to represent the atomic units of work within the business process described by the text. Usually, these fragments are associated with verbs. Event fragments are used to annotate other occurrences in the process that are relevant from the point of view of the control flow, but are exogenous to the organization responsible for the execution of the process.

Fragment Relations. Text fragments can be related to each other by means of different relations, used to express properties of the process emerging from the text:

– *Agent.* Indicates the role responsible for the execution of an activity.
– *Patient.* Indicates the role or business object on which an activity is performed.
– *Coreference.* Induces a coreference graph where each connected component denotes a distinct process entity.
– *Sequential.* Indicates the sequential execution of two activity fragments `A1` and `A2` in a sentence. We consider two important relations from [10]: Precedence and Response. Moreover, to cover situations where ambiguities in the text prevent selecting any of the two aforementioned relations, we also incorporate a less restrictive constraint WeakOrder, that only applies in case both activities occur in a trace.
– *Conflicting.* A conflict relation between two condition activity fragments $\langle C1, C2 \rangle$ in a sentence indicates that one and only one of them can be executed, thus capturing a choice. This corresponds to the relation NonCoOccurrence from [10].

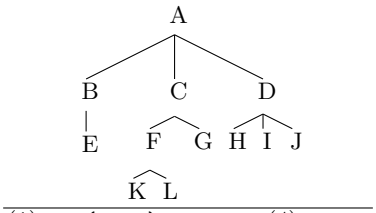
3.3 TRegex

In this paper, we use Tregex[4] [6], a query language that allows the definition of regular-expression-like patterns over tree structures. Tregex is designed to match patterns involving the content of tree nodes and the hierarchical relations among them. In our case we will be using Tregex to find substructures within syntactic dependency trees. Applying Tregex patterns on a dependency tree allows us to search for complex labeled tree dominance relations involving different types of information in the nodes. The nodes can contain symbols or a string of characters (e.g. lemmas, word forms, PoS tags) and Tregex patterns may combine those tags with the available dominance operators to specify conditions on the tree. Additionally, as in any regular expression library, subpatterns of interest may be specified and the matching subtree can be retrieved for later use. This is achieved in Tregex using unification variables as shown in pattern (2) in Figure 1.

Figure 1 describes the main Tregex operators used in this research to specify pattern queries.

[4] `https://nlp.stanford.edu/software/tregex.html`

Operator	Meaning
`X << Y`	X dominates Y
`X >> Y`	X is dominated by Y
`X !>> Y`	X is not dominated by Y
`X < Y`	X immediately dominates Y
`X > Y`	X is immediately dominated by Y
`X >, Y`	X is the first child of Y
`X >- Y`	X is the last child of Y
`X >: Y`	X is the only child of Y
`X $-- Y`	X is a right sibling of Y
`X $. Y`	X is the immediate left sibling of Y



(1) `E>>(A<<G)` (4) `F!>>A`
(2) `E>>(A=x)>:(B=y)` (5) `H>:D`
(3) `K!>>B>>(A<D)` (6) `A<J`

Fig. 1. Some operators provided by Tregex (left). The tree on the right would match patterns (1), (2), (3), and would not match patterns (4), (5), (6). Note that unless parenthesized, all operators refer to the first element in the pattern. Pattern (2) uses operator `=` to capture nodes `A` and `B` into variables `x` and `y` respectively.

4 Generalized Approach

4.1 Basic Approach: Intra-Sentence Analysis

In this paper we describe an extension to the approach presented in [9]. This subsection summarizes the basic original approach, and following subsections provide details on the added extensions, which mainly consist of the extraction of relations between actions or conditions in different sentences, as well as an extended evaluation covering not only entities and actions, but also relations.

In [9] we presented a proposal to extract Business Process elements (entities, actions, conditions, events, and relations) from a process textual description.

The approach consists of: (a) Use a full-fledged NLP analysis pipeline [8] to analyze the text and extract verbal predicates, involved actors and objects, syntactic trees of all sentences, and coreferences between different mentions of the same actor/object, and (b) apply a cascade of TRegEx patterns on the output of the NLP preprocess to extract and elaborate the relevant process information. These patterns perform the following tasks:

1. Select the appropriate description for an entity or object. For instance, in the sentence "*The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination*" the results of the NLP semantic role labeling step for *Agent* would return the whole subtree headed by *physician* (i.e. *an outpatient physician, who decides... examination*).
   The used Tregex patterns will strip down such a long description removing the determiner and the relative clause, while keeping the core actor/object and its main modifiers, thus extracting respectively `outpatient physician` as a role, and `female patient` as a business object.
2. Next step is identifying relevant activities. The NLP preprocess detects all predicates in the text (mainly, all verbs are considered a predicate, plus

some deverbal nouns such as "reception", "meeting", etc). However, although many verbs in a process description may be predicates from a linguistic perspective, they do not correspond to actual process activities. Thus, we use a set of patterns that discard predicates unlikely to be describing a relevant process task, or relabel them as *condition* or *event* fragments:

(a) More specifically, we use a set of predicates that check for syntactic structures involving conditional clauses (*if*, *whether*, *either*, ...) and the appropriate nodes in the tree are marked as *condition* fragments. In this step, we determine, for instance, that `she is healthy` and `needs an additional examination` are conditions in the sentence "*... who decides whether she is healthy or needs an additional examination.*".

(b) Another set of patterns deal with syntactic structures involving keywords like *when*, *once*, *as soon*, *whenever*, etc, and mark the related predicates as *event* fragments. These patterns allow us to identify the fragment `confirm(payment)` as an event fragment in the sentence "*Once the payment is confirmed, the ZooClub department can print the card...*"

(c) A third batch of patterns takes care of discarding activities that are not relevant to the process. To this end, we use two different strategies: one is removing all activities related to auxiliary, control, or subjective verbs (*be*, *have*, *start*, *want*, *think*, *believe*, etc.) which are unlikely to describe an actual process task. The second strategy relies on removing actions described in a subordinate clause. For instance, in the sentence "*..., the examination is prepared based on the information provided by the outpatient section*", the verbs *base* and *provide* would be removed as activities, since the main action described by this sentence is just *prepare (examination)*, and the subordinate clause just gives additional details on the object or procedure, but not on the actual process activity.

3. The last set of patterns deal with relations between activities. In our original work we tackled only relations between two activities in the same sentence. We considered different types of relations:

(a) Precedence: We use patterns to detect sentences relating one *event* and one *activity* in a precedence relation. E.g. in the sentence "*An intaker keeps this registration with him at times when visiting the patient*", it would extract the sequential relation from `visit(patient)` to `keep(registration)`.

(b) Response: This relation is identified between *condition* and *activity* fragments, which typically occur in conditional sentences such as "*If the patient signs an informed consent, a delegate of the physician arranges an appointment with one of the wards and updates the HIS selecting the first available slot*". From this sentence, we would extract the relation that `arrange (appointment)` *responds to* `sign(consent)`.

(c) Weak Order: There are many pairs of activities appearing in the same sentence where some kind of sequential order can be deduced, but it is not possible for an automatic system to determine their exact kind of relation. In these cases, we take a conservative approach and extract

the least restrictive constraint, WeakOrder. For instance, in the sentence "*The Payment Office of SSP generates a payment report and then pays the vendor*", we could extract that `generate` and `pay` are in WeakOrder.

(d) Conflict: Conflict relations can be determined between condition fragments, provided they are in the right syntactic structure. In this way, we can extract the constraint that the sample can not be safely used and contaminated at the same time from the sentence "*... decides whether the sample can be used for analysis or whether it is contaminated*", or that conditional fragments `approve` and `deny` from the sentence "*The next step is for the IT department to analyse the request and either approve or deny it.*" are considered in conflict.

4.2 Inter-Sentence Analysis

Patterns used in [9] for relation extraction summarized in Section 4.1 aimed to capture relations between two activities/events mentioned in the same sentence. The main contribution of this paper is the extension of these patterns to capture also relations between activities or events located in different sentences.

To achieve this goal, since TRegEx is able to handle a single tree at a time, we first need to join together the syntactic trees for all sentences in the text in a single tree. For this, we add two kinds of artificial parent nodes: A `<PARAGRAPH>` node that has as children the root nodes for each of the sentences in the same paragraph, and a `<DOCUMENT>` node that has as children all the `<PARAGRAPH>` nodes. With that, we obtain a unique tree for all the document, and we can apply TRegEx patterns that span over more than one sentence. Figure 2 shows an example of a tree representing a short document. We apply patterns on the document tree to extract conflict and sequence relations between activities, events, or conditions detected in previous steps (see sec. 4.1.)

Conflicts. Conflicts between activities in the same sentence are detected using patterns described in [9]. The following patterns deal with conflicts between activities in different sentences. Their goal is to instantiate in variables `originRef` and `destinationRef` verbs that head sentences which may contain nodes marked as `<ACTIVITY>` or `<CONDITION>` on which the relation will be extracted.

```
PC1  /verb/=originRef > /<PARAGRAPH>/
                      << /<CONDITION>/
                      $. (/verb/=destinationRef << /<CONDITION>/)
PC2  /verb/=originRef > /<PARAGRAPH>/
                      << /<CONDITION>/
                      $. (/verb/ !<< /<CONDITION>/
                              $. (/verb/=destinationRef << /<CONDITION>/))
```

Pattern PC1 checks for a verb directly under a `<PARAGRAPH>` (i.e. main sentence verb) that has a condition as a child, and that its right sibling (i.e, main verb in the following sentence) also has a condition. This would extract a conflict

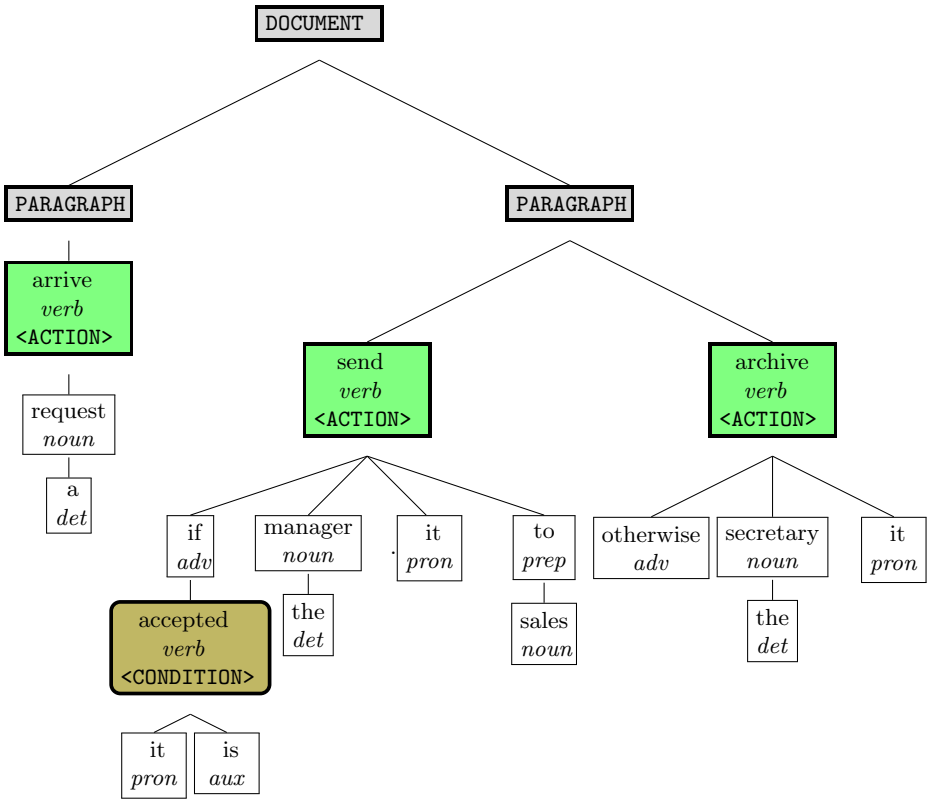


Fig. 2. Document tree for a text with two paragraphs: The first one with the sentence "*A request arrives*", the second with two sentences: "*If it is accepted, the manager sends it to sales. Otherwise, the secretary archives it*". Nodes in the syntactic dependency trees have been marked as `<ACTION>` or `<CONDITION>` in previous steps.

between *proceed* and *repeat* in the pair of sentences "*If sample is ok, proceed with examination. If contamination is detected, repeat sampling.*" Pattern PC2 captures the same kind of structure, when there is an additional sentence without a condition in between (e.g. "*If sample is ok, proceed with examination. Fill out treatment request form. If contamination is detected, repeat sampling.*")

Sequences. A second batch of patterns takes care of extracting sequence relations between activities in contiguous sentences. As in the case of conflicts, the patterns instantiate the variables `originRef` and `destinationRef` to candidate subtrees that are then searched for `<ACTIVITY>` or `<CONDITION>` nodes. Some patterns directly instantiate the variable `destination`, the actual target of the extracted relation.

PS1
```
/verb/=originRef > /<PARAGRAPH>/
                 $. (/verb/=destinationRef
```

```
                                    < /afterwards|then|immediately/)
PS2  /verb/=originRef > /<PARAGRAPH>/
                      $. (/verb/ << (/<CONDITION>/=destination << /or/))
PS3  /verb/=originRef > /<PARAGRAPH>/
                      $. (/verb/ << (/or/ << /<CONDITION>/=destination))
PS4  /verb/=originRef > /<PARAGRAPH>/
                      $. (/verb/ << /<CONDITION>/=destination
                                 < /otherwise|else/)
PS5  /verb/=originRef > /<PARAGRAPH>/
                      $. (/verb/ << /<CONDITION>/
                                 < (/otherwise|else/=destination)
```

Pattern PS1 extracts a sequence relation between the main verb of a sequence and the main verb of the next one provided the latter has a modifier such as *afterwards*, *then*, *immediately*, etc. Patterns PS2 and PS3 establish sequence relations between an activity and or-ed conditions in the following sentence (e.g extract sequences *send→fill* and *send→reject* in sentences "*Send form to customer. The customer can fill the form or reject to do it.*") Patterns PS4 and PS5 check a similar case, but where the second sentence has an "if-else" structure. They would extract the sequence relations *send→accept* and *send→cancel* in the sentence "*A budget is sent to the customer. If he accepts it, the bill is issued, otherwise the operation is cancelled.*"

5 Tool Support and Experiments

This section presents experiments evaluating the performance gain obtained when including patterns to capture relations between activities or events located in different sentences. We report two different results: First, we report relations extraction performance using a baseline based on [9] where we extract relations only for pairs in the same sentence. Second, we report results applying patterns to extract both intra- and inter-sentence relations.

The evaluation is performed comparing the relations extracted against gold standard manual annotations. For both table 1 and table 2, the test data set used in our experiments are the same as that used in the original proposal [9], which consists of 18 text-model pairs, each example includes a textual process description paired with the corresponding BPMN models created by a human.

The first 13 models stem from material in the appendix of [3], and the last 5 from our academic dataset[5] used in [11].

As a gold reference for evaluation, we manually created one ATDP for each example following the activities and relations in those BPMN models, i.e. marking as activity fragments only the text pieces that had a corresponding element in the BPMN model and connecting only the activities fragments that had a corresponding relation in the BPMN model.

[5] https://github.com/setzer22/alignment_model_text/tree/master/datasets/NewDataset

Source	Conflict						Sequence					
	#gold	#pred	#ok	P	R	F_1	#gold	#pred	#ok	P	R	F_1
1-1_bicycle_manufacturing	2	1	1	100	50	67	59	8	6	75	10	18
1-2_computer_repair	1	0	0	0	0	0	59	9	7	78	12	21
2-1_sla_violation	5	2	0	0	0	0	372	46	14	30	4	7
3-1_2009-1_mc_finalice_sct	0	0	0	0	0	0	52	4	3	75	6	11
3-2_2009-2_conduct	1	0	0	0	0	0	20	8	4	50	20	29
3-6_2010-1_claims_notification	2	0	0	0	0	0	63	9	7	78	11	19
4-1_intaker_workflow	0	0	0	0	0	0	596	17	5	29	1	2
5-1_active_vos_tutorial	3	0	0	0	0	0	15	4	3	75	20	32
6-1_acme-	1	0	0	0	0	0	340	22	11	50	3	6
7-1_calling_leads	1	0	0	0	0	0	13	2	1	50	8	13
8-1_hr_process_simple	0	0	0	0	0	0	15	7	6	86	40	55
9-2_exercise_2	3	0	0	0	0	0	11	6	6	100	55	71
10-2_process_b3	3	0	0	0	0	0	114	7	3	43	3	5
1081511532_rev3	1	0	0	0	0	0	47	9	5	56	11	18
1120589054_rev4-	0	0	0	0	0	0	66	9	6	67	9	16
1364308140_rev4	1	0	0	0	0	0	21	8	4	50	19	28
20818304_rev1	3	2	2	100	67	80	36	11	6	55	17	26
784358570_rev2	2	0	0	0	0	0	126	11	6	55	5	9
TOTAL	29	5	3	**60**	**10**	**18**	2025	197	103	**52**	**5**	**9**

Table 1. Evaluation of relation extraction using only intra-sentence patterns. Sequence relations are evaluated on the transitive clausure of both the sets of gold annotations and annotations produced by the system.

Intra-Sentence. Results for the first scenario (only intra-sentence patterns) are shown in Table 1, and correspond to results obtained using the patterns described in [9], which rely on extracting relations just within sentences. Precision is the percentage of right relations over predicted relations ($P = \#ok/\#pred$). Recall is the percentage of expected relations extracted ($R = \#ok/\#gold$). F_1 score is the harmonic mean of precision and recall ($F_1 = 2PR/(P + R)$). We only count extracted relations as right if they match the gold annotations in type (`<SEQUENCE>`, `<CONFLICT>`). In both experiments, sequence relations are evaluated over the transitive closure of the sequence annotations.

Inter-Sentence. In the second evaluation scenario, in addition to patterns created in [9], we use inter-sentence patterns described in Section 4.2.

Obtained results presented in Table 2 show that our new contribution extracts more relations, thus obtaining a large boost in recall (from 0.05 to 0.70 overall) with a very mild loss of precision (from 0.52 to 0.50 overall). Recall is boosted both for conflict and sequence relations, while precision is increased for conflicts, but slightly decreased for sequences.

Source	Conflict						Sequence					
	#gold	#pred	#ok	P	R	F_1	#gold	#pred	#ok	P	R	F_1
1-1_bicycle_manufacturing	2	2	2	100	100	100	59	90	54	60	92	72
1-2_computer_repair	1	0	0	0	0	0	59	65	33	51	56	53
2-1_sla_violation	5	4	2	50	40	44	372	572	152	27	41	32
3-1_2009-1_mc_finalice	0	0	0	0	0	0	52	57	42	74	81	77
3-2_2009-2_conduct	1	0	0	0	0	0	20	33	19	58	95	72
3-6_2010-1_claims	2	1	1	100	50	67	63	76	53	70	84	76
4-1_intaker_workflow	0	0	0	0	0	0	596	906	455	50	76	61
5-1_active_vos_tutorial	3	3	3	100	100	100	15	16	12	75	80	77
6-1_acme-	1	0	0	0	0	0	340	561	287	48	84	61
7-1_calling_leads	1	1	1	100	100	100	13	41	13	32	100	48
8-1_hr_process_simple	0	0	0	0	0	0	15	21	15	71	100	83
9-2_exercise_2	3	6	3	50	100	67	11	10	9	90	82	86
10-2_process_b3	3	1	1	100	33	50	114	138	83	60	73	66
1081511532_rev3	1	1	1	100	100	100	47	41	30	73	64	68
1120589054_rev4	0	0	0	0	0	0	66	78	66	85	100	92
1364308140_rev4	1	0	0	0	0	0	21	26	10	38	48	43
20818304_rev1	3	3	3	100	100	100	36	29	19	66	53	58
784358570_rev2	2	3	2	67	100	80	126	118	85	72	67	70
TOTAL	29	25	19	**76**	**66**	**70**	2025	2878	1437	**49**	**71**	**59**

Table 2. Evaluation of relation extraction using both intra- and inter- sentence patterns. Sequence relations are evaluated on the transitive clausure of both the sets of gold annotations and annotations produced by the system.

6 Conclusions and Future Work

We have presented an extension of our work in [9], consisting in adding syntax-tree based patterns to capture relations between activities or events located in different sentences. Results show that crossing the sentence boundaries is a highly productive strategy, since many more relations can be extracted. Also, the fact of using syntax-aware patterns, and not just flat regular expressions allows this extension to be done with almost no loss of precision.

Acknowledgments This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R, and by the Ecuadorian National Secretary of Higher Education, Science and Technology (SENESCYT).

References

1. João Carlos de A. R. Gonçalves, Flávia Maria Santoro, and Fernanda Araujo Baião. Business process mining from group stories. In *Proceedings of the 13th International Conference on Computers Supported Cooperative Work in Design, CSCWD 2009, April 22-24, 2009, Santiago, Chile*, pages 161–166. IEEE, 2009.
2. Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1027–1033. AAAI Press, 2014.

3. Fabian Friedrich. Automated generation of business process models from natural language input. *School of Business and Economics. Humboldt-Universität*, 2010.
4. Fabian Friedrich, Jan Mendling, and Frank Puhlmann. Process model generation from natural language text. In Haralambos Mouratidis and Colette Rolland, editors, *Advanced Information Systems Engineering*, pages 482–496, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
5. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 2013.
6. Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234. Citeseer, 2006.
7. Hugo A. López, Søren Debois, Thomas T. Hildebrandt, and Morten Marquard. The process highlighter: From texts to declarative processes and back. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018*, pages 66–70, 2018.
8. Lluís Padró and Evgeny Stanilovsky. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, pages 2473–2479, 2012.
9. Luis Quishpi, Josep Carmona, and Lluís Padró. Extracting annotations from textual descriptions of processes. In *Proceedings of 18th International Conference on Business Process Management (BPM 2020), Sevilla, Spain, September 13-18, 2020*, pages 66–70, 2020.
10. Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, and Lluís Padró. Formal reasoning on natural language descriptions of processes. In *Int. Conference on Business Process Management*, pages 86–101. Springer, 2019.
11. Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona, and Lluís Padró. Aligning textual and model-based process descriptions. *Data & Knowledge Engineering*, 118:25–40, 2018.
12. Han van der Aa, Josep Carmona, Henrik Leopold, Jan Mendling, and Lluís Padró. Challenges and opportunities of applying natural language processing in business process management. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 2791–2801, 2018.
13. Han van der Aa, Claudio Di Ciccio, Henrik Leopold, and Hajo A Reijers. Extracting declarative process models from natural language. In *International Conference on Advanced Information Systems Engineering*, pages 365–382. Springer, 2019.
14. Han van der Aa, Henrik Leopold, and Hajo A. Reijers. Dealing with behavioral ambiguity in textual process descriptions. In Marcello La Rosa, Peter Loos, and Oscar Pastor, editors, *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science*, pages 271–288. Springer, 2016.
15. Wil M.P. van der Aalst. *Process Mining*. Springer, second edition, 2016.

Case2vec: Advances in Representation Learning for Business Processes

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

Technical University of Darmstadt, Germany
Telecooperation Lab
{luettgen,seeliger,nolle,max}@tk.tu-darmstadt.de

Abstract. The execution of a business process is often determined by the surrounding context, e.g., department, product, or other attributes an event provides. Process discovery mainly focuses on the executed activities, although the context of a case may be needed to accurately represent a process instance, e.g., for clustering, prediction, or anomaly detection. Hence, in this paper, we present a representation learning technique (Case2vec) using word embeddings for business process data to better encode process instances. Our work extends Trace2vec and incorporates an additional semantic level by using not only the activity name but also the attributes and thereby incorporating the context. We evaluate our approach in the context of trace clustering. Additionally, we show that Case2vec can be used to abstract events which are semantically similar but syntactically different. We also show that word embeddings allow for interpretability when employing vector space arithmetic.

Keywords: Representation learning · Word embeddings · Process context.

1 Introduction

In recent years, process mining has become an important technology for organizations analyzing their business processes. Event logs recorded by process-aware information systems can be analyzed with process mining to obtain valuable insights about how a business process is executed in reality. However, process mining techniques primarily focus on the control-flow of a process without considering the context a case is executed in, e.g., department, product, customer, or other attributes an event provides. This additional process context may help to further reveal patterns within the event log, which are not visible in the control-flow perspective, to enhance process mining techniques. Our goal is to learn vector representations of process cases that include this context information that can be used in various process mining techniques.

Vector representations of cases are required by many techniques in process mining such as trace clustering [12, 11, 4], prediction [3], and anomaly detection [10, 13]. Trace clustering aims to improve the discovery of process models by grouping similar cases. Clusters of cases that are executed in similar contexts

can be generated, allowing the user to compare process models of different contexts. Improved prediction models can be learned that also consider the process contexts. Furthermore, anomaly detection methods based on extended vector representations can provide more reliable results. These are just a few examples for potential use cases of context-including vector representations.

Our work is based on a technique proposed in the area of natural language processing (NLP) for learning vector representations of words and sentences. Similar to a sentence with words, a case of a business process consists of a sequence of activities. Activities are also not executed in random order, but according to a predefined grammar, the underlying process model. The core idea is to model similarities and intentionally avoid comparing by words only, because we know that different words or sentences can bear the same meaning.

A previous work, Trace2vec [4], showed that the representation learning approach Word2vec [8], which constructs a vector space of the words of a corpus to capture similarities, can also be used on process data. To model such similarities, a large event log is crawled to order activities which occur together within this vector space. However, Trace2vec also showed some difficulties in the experiments with the BPIC15 event log: First, the vocabulary of event logs is much smaller compared to the vocabulary of natural language. Second, the context of a case is not taken into account, which can provide further details about the dependencies between activities and attributes. For instance, if the BPIC15 event log is clustered into the municipalities without considering the process context, it is assumed that the control-flow alone clearly determines the municipality. In highly standardized processes like governmental processes, the control-flow is the very part that does not separate one trace from another, but rather its context, e.g., an officer working exclusively in one or a few municipalities.

In this paper, we present an extended approach based on Trace2vec that can indeed lead to sensible results when evaluating these representations for a trace clustering task. We name this extension Case2vec, because it uses event and case attributes to capture the process context. Our extension increases the vocabulary that allows to better exploit case relationships. Besides our extension, we examine a proper hyperparameter strategy that can better deal with the sparse vocabulary in business process data. We revisit the original approach using the BPIC15 event log and show how parameter tuning and especially incorporating attributes improves results. We also show a wider range of results on the BPIC19 event log, which holds not only more traces, but also more attributes.

As additional tasks we investigate two useful applications of the neural network architecture presented: (1) *Event abstraction* allows to show that syntactically different activities are semantically similar, given enough traces in a similar context. (2) Arithmetic operations within the vector space keep semantic meaning which we show in an *interpretability task*. This is done on an artificial paper writing process to show the task more clearly because we know how the activities in this process depend on each other.

2 Related Work

Process case representations are used by various process mining techniques such as trace clustering, anomaly detection, and prediction. Different representations have been proposed in the related work. A simple representation technique is the bag-of-words model which is used to compute the similarity of sentences based on the co-occurrences. Song et al. [12] encode sequences of activities as one hot vectors, in which each component corresponds to an activity. Transitions between activities are used instead by Bose et al. [1] to compare cases.

Besides manually defined case representations, automatically generated representation vectors can be learned. For instance, a word embedding is a feature learning technique in which words are mapped to a vector space. Words appearing together frequently within a text corpus will be mapped close together within a vector space to capture their semantic relationship. Word embeddings do not rely on syntactical features and, therefore, can compute a similarity value of two sentences, even if none of the words of each sentence is the same. De Koninck et al. [4] transferred the idea of Word2vec [8] and Doc2vec [7] to process data. An LSTM and CBOW-based approach was introduced by Bui et al. [2]. A supervised representation learning approach based on conditional random fields for event abstraction was introduced by Tax et al. [14].

Representation learning has been used for different analysis methods. Trace2vec representations were used to cluster traces into similar groups in [4]. Tavares et al. [13] use the same representations to identify anomalous cases.

A drawback of most related work in this field is the limitation to the pure control-flow, namely the sequence of activities to learn case representations. Thus, the process context of the cases is not considered.

3 Case2vec

In NLP, word embeddings use the context of the words in a document to exploit semantic similarities of words by mapping them to a vector space. The closer these words appear together in the document the closer they are mapped together in the vector space. Thereby, semantic similarity of different statements can be confirmed as long as they are mapped close together.

As already mentioned, a popular technique for modeling word embeddings is Word2vec [8]. The task is to model what is in the neighborhood of a word. This can be done using two different approaches. We can predict a word given its surrounding words (continuous bag of words, short CBOW), or the other way round, predict the surrounding words given one word (skip-gram). For example, in the sentence *I like ... process mining*, continuous bag of words would insert words of similar representation to fill the gap, e.g., *business*. Vice versa, skip-gram would take the word *business* and amend it with preceding and succeeding words given by example sentences in the training data.

Word2vec learns a CBOW or skip-gram model using a neural network and implicitly constructs an abstract representation of the vocabulary and its rela-

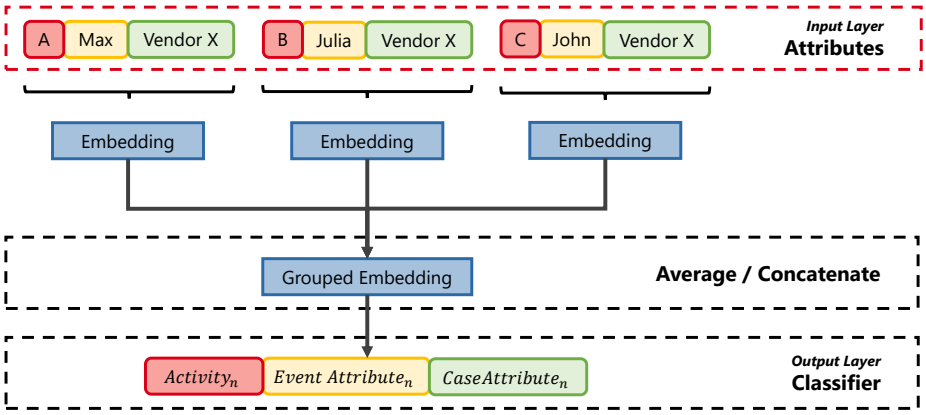


Fig. 1. Architectural overview of Case2vec where each trace's activity names, event, and case attributes are concatenated as single words.

tionships between each other. Similarly, activities within a sequence of a business process are also dependent on the preceding and succeeding activities which form the context. We can employ the idea of word embeddings and map activities to a vector space such that activities in similar regions are related to each other according to their function in the underlying process. Doc2vec serves as a representation of a collection of words, namely a document. Analogous to the Word2vec model, in Doc2vec a word is a document and we want to predict the surrounding documents. The structure of a trace from an event log is of similar form when considering activity names as words in a trace sequence. The resulting embedding space is a representation where activities and traces, given enough sample traces, are projected according to their role in the overall process model.

The embedding on the control-flow level is constructed by using the activity name as a single word. The set of different activity names forms the vocabulary of the embedding, and a Doc2vec representation is constructed by treating a trace as a document. The control-flow level (Fig. 1 without event and case attributes) has been introduced as Trace2vec [4]. One drawback of this approach is the focus on the control-flow. Therefore, we introduce Case2vec, which incorporates the different kinds of attributes by concatenating them with the corresponding activity name. The key idea is to incorporate attributes in addition to activity names to enlarge the vocabulary and induce a better separation of cases. If attributes are taken into account, the concatenation of the activity and its respective attributes becomes an additional word and, therefore, includes the process context.

Fig. 1 shows the architecture with the attribute extension, where the words of the vocabulary are constructed by concatenating `Activity`, `Resource`, and `Vendor`. We also evaluate the approach either using event or case attributes.

4 Experimental Evaluation

We implemented[1] the described representation learning techniques using *gensim*, *scikit-learn* and *fastcluster* in Python to evaluate their performance. We use two Business Process Intelligence Challenge (BPIC) event logs, an amended version of them, and a fully synthetic paper writing process to evaluate on the following objectives: Trace clustering, event abstraction, and interpretability through vector arithmetic operations in the vector space.

In the following, we describe the event logs, the experimental setup and report the results.

4.1 Datasets

We use real-life and artificial event logs to evaluate the different objectives.

Real-life Event Logs We use the BPIC15 [5] and BPIC19 [6] event logs to compare the applicability of the different approaches. We select a case attribute for both event logs that can be considered as the ground truth label for clustering. Although we do not know in advance if this process provides features that will lead to good clustering results with this label, we are not necessarily interested in the best clustering result, but rather how incorporating different attributes can influence the clustering performance.

For BPIC15, event logs are already split into five different municipalities. In BPIC19, the case attribute `Item Type` is used as the cluster label without the `Standard` cases to obtain evenly distributed clusters.

During the experiments for event abstraction we amended the real-life event logs with noise or additional attributes. For the event abstraction task, we amended activity names with random numbers in a certain amount of traces to show that the method is robust to small changes in activity names.

Artificial Event Log: Paper Writing Process The artificial example event log is based on a synthetically generated process depicted in Fig. 2. It describes the main steps in a scientific paper writing process from identifying a problem to the submission of the paper. The activities are dependent on each other according to their sequential order. This event log is more comprehensible for interpreting the results of the vector arithmetic experiment. For the experiments we sampled 5 000 traces of this process according to [9].

4.2 Real-life Event Logs: Trace Clustering

In the first experiment, we use the case representations for clustering cases into their classes to show applicability for process context separation.

[1] Source code publicly available at: https://github.com/alexsee/case2vec

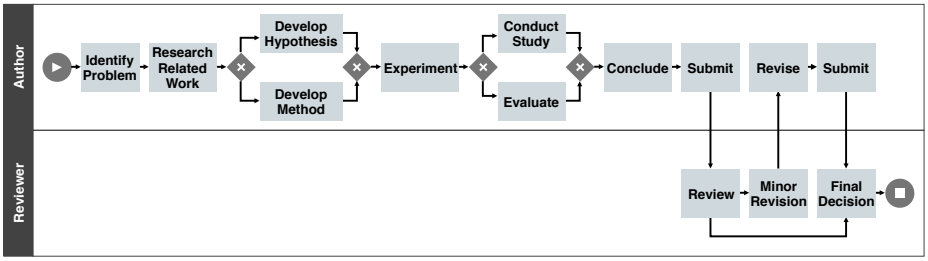


Fig. 2. Overview of the paper writing process [9].

Experimental Setup Each event log is used individually to train the network according to the description in Sect. 3. For training, activities and attributes are used and none of the sequences are trimmed. Afterwards, we obtain the internal representation of each case and use the feature vectors as input for clustering.

Taking into account that process data in comparison to natural language has shorter sentence length and substantially smaller vocabulary, we employ a hyperparameter strategy to overfit the dataset for the clustering task. This is done using the ground truth label as an attribute with the goal to reach a Normalized Mutual Information (NMI) measure of 1.0 to ensure that the trained vector space has the capacity to model the underlying processes. This step is important before running the actual experiments to exclude weak results because of an impaired modeling capability of the underlying neural network. After a set of parameters is found that can overfit the dataset, the same optimization strategy can be employed during the actual experiments to maximize the NMI without the ground truth label.

In our parameter optimization strategy, we first optimize the vector size. We vary the vector size of the hidden and the embedding layer (2, 3, 4, 8, 16, 32, 64, 128, 256), and the number of epochs $(10, 25, 50)$. Next, we optimize the window size of the embedding which determines how many activities before and after the current activity are considered. A value 5 or 7 seems optimal, and similar to the vector size, larger values do not improve the result and only run the risk of overfitting. Training epochs are varied between 10 and 50. The other parameters were standard parameters according to [4]. We trained the embedding with $sg = 0$ for the CBOW model, a learning rate set constant with $lr = 0.025$ for both Trace2vec and Case2vec and the decay factor alpha to 0.002. The number of inference epochs is set to 50. For clustering we opted to use hierarchical clustering. As a distance metric we use cosine distance to avoid a bias when dealing with traces of different length.

As an evaluation metric, we measure the NMI. We analyze the results using the non-parametric Friedman test. The Bonferroni corrected pairwise Wilcoxon signed-rank test is used for post-hoc analysis. We further report Kendall's W effect size.

Table 1. Best clustering performance grouped by approach, configuration, and event log.

Log	Approach	Vector Size	Epochs	NMI
BPIC15	Trace2vec (original)	64	40	0.080
	Trace2vec (optimized)	4	50	0.132
	Case2vec (org:resource)	8	25	**0.980**
	Case2vec (Case Type)	3	25	0.010
	Case2vec (org:resource + Case Type)	8	50	**0.983**
	Case2vec (Responsible Actor)	128	25	0.398
	Case2vec (org:resource + Responsible Actor)	4	50	0.424
BPIC19	Trace2vec	32	10	0.560
	Case2vec (org:resource)	128	50	0.657
	Case2vec (org:resource + Document Type)	16	50	0.566
	Case2vec (Document Type)	128	25	0.591
	Case2vec (org:resource + Item Category)	16	25	0.626
	Case2vec (Item Category)	256	50	**0.805**
	Case2vec (org:resource + Vendor)	128	25	0.330
	Case2vec (Vendor)	2	50	0.296

Results As a first step, we recreated the results by De Koninck et al. using the BPIC15 event log. As depicted in Table 1, Trace2vec reaches an NMI of 0.080 and increases to 0.132 after hyperparameter optimization. Using Case2vec with the case attribute `Responsible Actor` leads to a significant performance increase up to 0.398. The event attribute `org:resource`, which refers to the executing user, shows a performance of 0.980. Combining `org:resource` with one of the case attributes `Case Type` or `Responsible Actor` decreased the performance.

For the BPIC19 event log, Trace2vec reaches a performance up to 0.560.The case attribute `Item Category` reached the highest results with 0.805. However, using the `Vendor` results in a lower NMI than the control-flow only. Also, combining attributes also does not guarantee better results. Used separately, `org:resource` results in an NMI of 0.657 and `Document Type` in an NMI of 0.591. Combining the two leads to an NMI of 0.566, which results in a lower NMI than used separately.

Detailed results regarding the vector size are depicted in Fig. 3. The analysis of the results confirmed significant differences ($\chi^2(2) = 108$, $p < .001$, $W = 1$) between the approaches with a large effect. Post-hoc tests confirmed differences ($p < .001$) between all approaches with Case2vec performing better than Trace2vec. Incorporating `org:resource` lead to a significant better performance ($p < .001$) for BPIC15. For BPIC19, we discovered consistent results across the different parameter configurations, still there are significant differences ($\chi^2(2) = 24.111$, $p < .001$, $W = .223$) between the approaches. Similar to BPIC15, post-hoc tests confirm significant ($p < .001$) differences between all approaches.

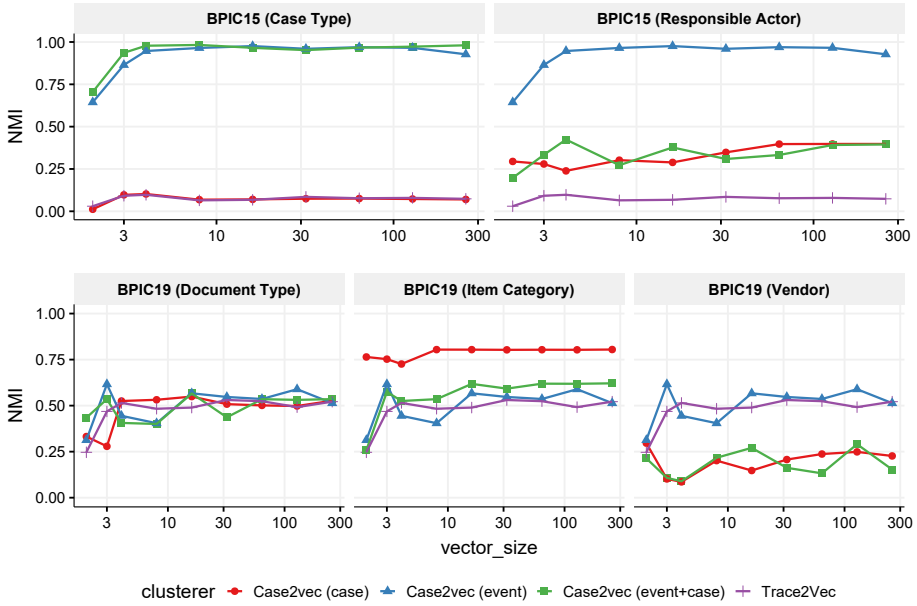


Fig. 3. Clustering results grouped by vector size, approach, configuration, and event log with 50 epochs.

4.3 Amended Real-life Event Logs: Event Abstraction

The goal of event abstraction is to identify similar traces although activity names are slightly different. Eventually, these activity names can be adjusted to clean the event log. An example would be the activity name `PR_created` and `Create_PR`, which describes the same action with just a different name. The idea is to identify activities with similar function within the process so that a vector representation will allocate both activities close together in the vector space despite their different names.

The level of distorted activity names ranges from 0%, which is the normal case, up to 50%. The number of variations indicates the number of noise which is added to the activity name, e.g. letters or numerals. For example, if there are 2 variations and a 20% distortion level, the same random number is added to 20% of the traces, and the remaining 80% describes the unmodified variation. In case of 6 variations, besides the undistorted traces, there are traces distorted with 5 different random numbers to further increase uncertainty. Fig. 4 shows results for different levels for different variations of the activity names. Case2vec with event attribute shows consistent results with deviations of ≤ 0.1 in NMI for distortion levels from 0% to 40%. A larger deviation of ~ 0.1 can only be seen with Trace2vec for 20% distortion.

Fig. 4. Overview of the results of the event abstraction task.

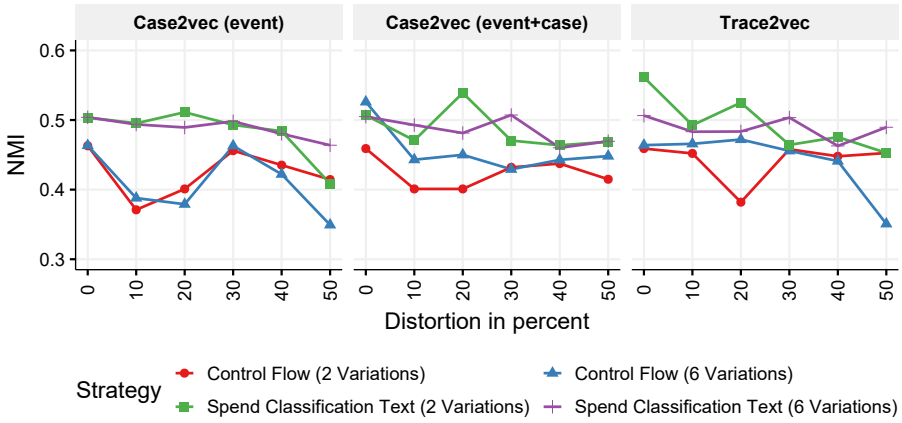


Table 2. More paper process interpretability tasks

Add	Subtract	Top Result
Experiment, Develop Method, Submit	Final Decision	Conduct Study
Conclude, Review, Submit	Final Decision	Develop Method
Develop Method, Submit	Final Decision	Conclude
Final Decision	Submit	Review
Experiment, Conclude	Submit	Develop Method

4.4 Synthetic Paper Process: Vector Arithmetic Interpretability

Since representation vectors are spanned within a vector space, arithmetic operations can be performed between vectors. The famous `king - man + woman = queen` example from Word2vec showed that representations can contain important semantic relationships. In this experiment, we investigate if vector arithmetic operations can also be used with process data. For testing the interpretability task, we have to come up with a certain scenario which allows a semantic interpretation. We use the paper writing process (see Fig. 2) because it is not pseudonymized and the activities can be read and understood by an analyst.

The first scenario is that the experiment was done and the paper was submitted, but the final decision has not taken place, because something is still missing that fulfills the criteria for an accepted paper. A possible composition would be to add the experiment and submission but subtract the final decision. When performing `Experiment` + `Submit` − `Final Decision` we would expect that something between `Experiment` and `Submit` is missing so that the `Final Decision` is still pending. The result of this computation returns `Evaluate` as the top result. The second top result is `Conduct Study` and the third top result

is `Review`. Table 2 shows more example interpretability tasks. The result column shows the top result.

5 Discussion

In this section, we elaborate on the results from the experiment section and follow the order presented there.

5.1 Trace Clustering

BPIC15 Our experiments showed that hyperparameter optimization increases the performance of the original approach, but did not lead to useful results. However, it is not known if the separation by municipality solely based on the control-flow is possible. The process of a building permit application may be presumably highly standardized and, therefore, not a useful criterion for separating by municipality.

The results of our experiments show that the user of an activity is an attribute that is able to separate the cases into the five municipalities. This may be an obvious observation because persons may only work for a specific municipality. However, the event log also contains several persons that work across multiple municipalities. Case2vec, which includes the control-flow and the attributes, is able to find case representations for clustering that discriminate between the municipalities.

BPIC19 For the BPIC19 experiments, we found that Trace2vec performed significantly better compared to the BPIC15 event log. The best result was achieved by incorporating the `Item Category` case attribute, which seems to be strongly related to the `Item Type`. Interestingly, not all attributes improve the control-flow performance. For instance, the case attribute `Vendor` decreased the performance down to 0.290. This could be explained by the fact that a vendor is not a good separating attribute when categorizing according to an item type a company purchases. This would be the case if the company acquires most of its items from the same vendor regardless of the category of the item. Hence, even if the control-flow is able to separate by item type to some extent, an attribute, which is identical for most items, like a vendor, can obfuscate the results. This means that we cannot arbitrarily add more attributes for better results.

Case2vec seems to be sensitive to the selection of the attributes. Even though we showed that those methods provide good results after hyperparameter tuning, applying them to real-life event logs can be difficult because the quality of the result can usually not be determined since the ground truth is unknown. Attributes that contain random values or do not contribute to the desired clustering result lead to a significant drop in performance. However, when selecting appropriate attributes, Case2vec can outperform Trace2vec significantly. Still, finding good attributes can be difficult without prior knowledge.

5.2 Event Abstraction

For event abstraction, we ran several experiments with different amounts of traces including random numbers. We also changed the amount of different random numbers. Every attribute including a different random number will increase the vocabulary size. Still, Case2vec was able to identify and group traces according to their function despite them being amended. An even more interesting application than finding functionally similar traces with different names would be finding functionally similar traces with different performance metrics like cost or time. An analyst could study why these traces are similar in their role but differ in cost or time.

5.3 Interpretability Task

The example computation `Experiment` + `Submit` − `Final Decision` returns `Evaluate` as the top result and `Conduct Study` as the second top result. Both are sensible choices when we assume that `Evaluate` has already taken place and both are performed before `Submit`. The third top result is `Review`, which takes place directly after `Submit` and also shows a sensible reason assuming `Evaluate` and `Conduct Study` have been taken place and therefore cannot be the reason the submission is still blocked. Further results shown in Table 2 can be interpreted with similar reasoning.

However, in real-life event logs the interpretation of activities is not as clear because often they do not have interpretable names and even if, these names do not necessarily relate to a role in the process its name might suggest. Additionally, the developers of the Word2vec framework remark that vector arithmetic is not guaranteed to always produce sensible results. It is still interesting to see that on a small and well-defined event log the vector representation can deliver these results.

6 Conclusion

In this paper, we presented Case2vec, a representation learning technique based on a vector space model. It is trained using a neural network in an unsupervised fashion by using the sequence of activities including event attributes. It does not rely on any prior knowledge about the process and is able to learn robust and compact representations automatically.

The results of the evaluation in a trace clustering task showed that Case2vec is able to learn a good representation given useful control-flow or case attributes. When selecting appropriate attributes Case2vec can outperform Trace2vec significantly as shown in our real-life evaluation. The experiments on the additional tasks like event abstraction or arithmetic operations in the constructed vector space support that the learned representation is able to capture semantic characteristics of the process. However, Trace2vec and Case2vec seem to be sensitive to the selection of the attributes and finding good attributes can be difficult without prior knowledge. Feature selection methods from machine learning may help

to identify attributes with a high information value, helping analysts to select useful attributes. Another limitation is that Case2vec only supports categorical attributes. Numerical values could be incorporated by grouping them into bins beforehand.

In conclusion, the internal representation of Case2vec is highly useful for trace clustering, finding functionally similar traces or executing vector space arithmetic operations for interpretability tasks.

Acknowledgement. This work is funded by the German Federal Ministry of Education and Research (BMBF) research project KI.RPA [01IS18022D].

References

1. Bose, R.P.J.C., van der Aalst, W.M.P.: Context Aware Trace Clustering: Towards Improving Process Mining Results. In: International Conference on Data Mining (SIAM) (2009)
2. Bui, H.n., Vu, T.s., Nguyen, H.h., Nguyen, T.t.: Exploiting CBOW and LSTM Models to Generate Trace Representation for Process. In: Intelligent Information and Database Systems (2020)
3. Camargo, M., Dumas, M., González-Rojas, O.: Learning Accurate LSTM Models of Business Processes. In: Business Process Management (2019)
4. De Koninck, P., vanden Broucke, S., De Weerdt, J.: act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes. In: Business Process Management (2018)
5. van Dongen, B.: BPI Challenge 2015. https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
6. van Dongen, B.: BPI Challenge 2019. https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1
7. Le, Q., Mikolov, T.: Distributed Representations of Sentences and Documents. In: International Conference on Machine Learning (2014)
8. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. In: Advances in Neural Information Processing Systems (2013)
9. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: BINet: Multi-perspective Business Process Anomaly Classification. Information Systems (2019)
10. Nolle, T., Seeliger, A., Mühlhäuser, M.: BINet: Multivariate Business Process Anomaly Detection Using Deep Learning. In: Business Process Management (2018)
11. Song, M., Yang, H., Siadat, S.H., Pechenizkiy, M.: A comparative study of dimensionality reduction techniques to enhance trace clustering performances. Expert Systems with Applications (2013)
12. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace Clustering in Process Mining. In: Business Process Management Workshops (2009)
13. Tavares, G.M., Barbon, S.: Analysis of Language Inspired Trace Representation for Anomaly Detection. In: ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium (2020)
14. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Event Abstraction for Process Mining Using Supervised Learning Techniques. In: SAI Intelligent Systems Conference (IntelliSys) (2016)

Supervised Conformance Checking using Recurrent Neural Network Classifiers

Jari Peeperkorn[1], Seppe vanden Broucke[2,1], and Jochen De Weerdt[1]

[1] Research Center for Information Systems Engineering (LIRIS), KU Leuven, Leuven, Belgium
[2] Department of Business Informatics and Operations Management, Ghent University, Ghent, Belgium
{jari.peeperkorn, seppe.vandenbroucke, jochen.deweerdt}@kuleuven.be

Abstract. Conformance checking is concerned with the task of assessing the quality of process models describing actual behavior captured in an event log across different dimensions. In this paper, a novel approach for obtaining the degree of recall and precision between a process model and event log is introduced. The approach relies on the generation of a so-called "antilog", randomly constructed from the activity vocabulary, on one hand, and a simulated "model log", which is played-out from the given model. In the case of recall the antilog and model log are used to train a recurrent neural network classifier. This network allows for calculating the probability of a trace being part of the model log or the antilog. If thereupon the event log is fed to the neural network, a value for recall can be obtained. In the case of precision the neural network is trained using a given event log and the antilog, and the model log is fed to it afterwards. We show that this new method can be used to measure global recall and precision correctly in some common examples.

Keywords: Process Mining · Conformance Checking · Machine Learning · Neural Networks· RNN.

1 Introduction

Conformance checking covers different process mining techniques to compare event logs with process models. The latter can either be normative or an automatically discovered model. Conformance checking techniques can include both global conformance analysis, typically represented in the form of metrics, as well as local diagnostics, i.e. pinpointing conformance problems at a more fine-granular level, either in the log or in the model. Global conformance metrics typically measure conformance across one of the following quality dimensions: recall (or fitness), precision, generalisation and simplicity.

In this work, we propose a novel, fully data driven technique that can be used to measure recall and precision between process models and event logs and thus works at a global level, although it also has potential to provide insights into local conformance diagnostics in future work. For now, the technique relies on

the generation of a model log and comparing this with the given event log. This comparison is carried out by first generating a so-called "antilog" of either one of these two logs (the given event log or the model log). The antilog represents behavior not present in the log and could be generated in multiple ways. In this work, we investigate the potential of using the simplest of strategies to produce such an antilog, i.e. traces consisting of events representing activity executions randomly selected from the activity vocabulary. Once we have such an antilog, a recurrent neural network is trained to discriminate whether process instances belong to the (model or event) log or the antilog. Using this network we can obtain scores for the instances of the other (model or event) log. These scores represent how well an instance is described by the behavior in the log used for training.

Our technique has several compelling advantages. First of all, it proposes an new alternative to the common alignment or replay based algorithms, by utilizing a recurrent neural network classifier (RNN) model. Second, the RNN models are intrinsically probabilistic, thus giving a fine-grained analysis of model-log conformance. Furthermore, they are able to automatically detect temporal relations in sequences, making them a fitting tool to assess process instances. Moreover, by increasing the sample size of the antilog, we can investigate convergence and stability of metrics. Third, while being a black-box model, the RNN model could be complemented with visualizations that can pinpoint conformance problems at a local level, indicating at which timestep (activity) the prediction changes to antilog. Fourth, our technique allows for incremental updating of the model. Once the RNN is trained, conformance analysis can happen very fast, e.g. when checking the precision of different models. Fifth, despite the fact that we rely on model simulation to obtain a representation of the model behaviour, our technique is intrinsically more model-agnostic than other techniques which assume a certain model representation, and can be applied on any model that defines execution semantics. Finally, our technique links very well with predictive process monitoring techniques and has a lot of potential to extend conformance analysis towards other dimensions than control-flow, i.e. also including the resource and data dimensions. The use of RNNs in process mining is not new and has been applied in several areas such as predictive process monitoring [11,22]. However, the application of RNNs to conformance checking has not been investigated before.

The remainder of our paper is structured as follows: first the technique and its different components are introduced in Section 2. In Section 3 a set of initial experiments are detailed, which show our approach's potential for global conformance analysis. The paper is concluded by a section discussing previous related work (Section 4) and conclusions and directions for future work in Section 5.

2 RNN-based Conformance Checking

2.1 Overview

Figure 1 presents the key idea behind our proposed technique. RNN based conformance checking relies on first "playing-out" the process model to obtain a

model log. From this point onward the approaches to obtain either precision or recall differ. If we want to calculate recall, we take the model log and generate its model antilog. Again, we need to generate the antilog to capture non-conforming behaviour (not present in the model), in order to train the RNN. A recurrent neural network classifier is then trained to discriminate whether a certain process instance comes from the model log or its antilog. In practice this discriminator uses a sequence of activities as input, and outputs the probability of the instance belonging to either antilog or log (a value between 0 and 1). If we now use a new process instance as input in this network, we obtain a score on how well the model log (and therefore the model) describes it. Using the event log as input grants us an opportunity to obtain a global recall score. We can either use the average predicted value over all instances or we can use the fraction of instances with a score higher than a certain threshold (e.g. 0.5).

When precision of the process model is of interest, the technique starts with the event log and trains an RNN-classifier by combining it with its event antilog. By then training the RNN-classifier in a similar fashion and letting it classify the traces in the model log, we can obtain precision scores for every trace and thus for the entire model. Because the method as a whole provides labels for the RNN to train with automatically, it can be regarded as "self-supervising".

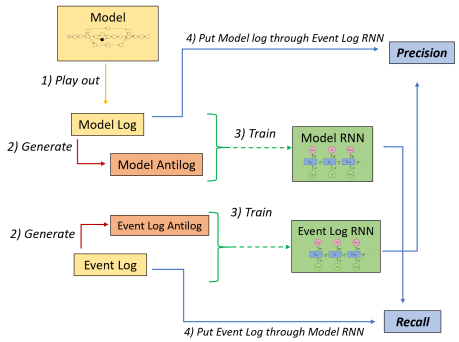


Fig. 1: Overview of the technique.

2.2 Model log generation

For the time being, the method relies on capturing the model behavior by means of a generated model log. In this work, we opted to "play-out" the model stochastically. This means that not necessarily all possible execution sequences allowed by the model are captured by the model log. However by making the model log sizeable enough, this problem can be partially solved. Only in extreme cases (like e.g. the flower model [9]), where the model entails a very large (infinite) set of potential process variants, this may lead to lower recall values than imposed by ground truth. However, we argue that this should not be seen as a major downside of our technique because even for models with a very large behavioural size (thus allowing a large number of variants), we can continually increase the size of

the model log by simulating more cases. Moreover, because of the probabilistic nature of the approach, it will be possible to assess stability and convergence of recall and precision scores when increasing the size of the model log. In addition, the technique itself is fairly robust to infinite behaviour due to loops, given that the RNN discriminator in itself has sufficient generalization power to deal with such constructs. One advantage of using a model log is that the method is model agnostic and can be applied on any model that defines play-out execution semantics. For the time being, we have chosen to use the play-out functionality of the Python Process Mining library PM4PY [4].

2.3 Antilog generation

Multiple methods to obtain the antilog were considered before settling on the intuitive approach proposed here. That is, experiments demonstrated that a strategy as straightforward as random generation produced satisfactory results, which indeed is an appealing proposition towards end-users. More specifically, we generate process instances with a sequence length uniformly selected between the minimal and maximal sequence length found in the log it is based on. Each activity is randomly selected from the activity vocabulary present in both event log and model log. This choice of vocabulary is not set in stone, as one could also opt to use only the vocabulary of the log in question. Therefore the model antilog and the event log antilog are likely to be very similar, only the array of different potential sizes of the instances might be different. Finally, a check is performed whether the newly generated antilog instance does not correspond to a certain instance variant from the log it is based on. In this way another possible difference between the model antilog and the event log antilog is created. However if the event log or simulated model log is large enough (i.e. has enough instances of each possible variant), this check is not strictly necessary and can be ignored. After all, the classifier will usually have significantly more examples of this particular instance with the correct (real) label than with the incorrect (antilog) label in its training set.

Observe that, despite the fact that in the current configuration the two antilogs will typically not differ much, we decided to still make a distinction between model antilog and event log antilog, given that we investigated some alternative antilog generation methods, which might lead to more significant differences between the two types of antilog. For instance, we considered a strategy that involved the addition of noise to the log. More sophisticated antilog generation strategies, e.g. relying on artificial negative event generation or generating the model antilog directly from the model also seem worthwhile. Nonetheless, given both the satisfactory results detailed below, as well as the fact that the random model log generation and random antilog generation provide the technique with a clear probabilistic nature, we argue that this is a proper choice. The antilog size was taken to be equal to the log it is based on. This was chosen in order to not introduce class imbalance in the training set of the classifier. If the training set would be wildly imbalanced, the likelihood of the classifier getting stuck in a

local minimum whilst training would increase, with the end result of simply predicting the majority label with a near-certain probability, ultimately resulting in recall/precision scores of either 0 or 1.

2.4 Recurrent neural network classifier

Recurrent neural networks (RNNs) are a type of artificial neural network specifically designed to handle sequential data. RNNs can be seen as a combination of multiple feedforward neural networks (one for each time step in the sequence). The hidden layers (recurrent layers) passing on messages, either one directional forward or bi-directional back and forth. Due to the vanishing/exploding gradient problem, simple recurrent neural networks do not handle long term dependencies well. Multiple solutions for this have been proposed, the most popular being Long Short-term Memory (LSTM) [15] and Gated Recurrent Units (GRU) [5]. Using such networks provides several advantages when there is a reason to incorporate these long term dependencies. However in this particular setting, we noticed that (non-fitting) switching of two subsequent activities was not being discriminated as nonconforming by the network when using an LSTM, which might be a problem in a conformance checking context. For example, when "sign contract" occurs before "check contract", this did not lead to a large change in predictions by the network, but could still indicate a problematic conformance error. Therefore we opted to use a bi-directional simple recurrent layer. The full architecture can be found in Figure 2.

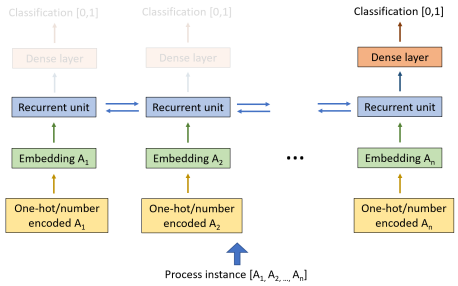


Fig. 2: Overview of the Recurrent Neural Network discriminator architecture.

The input is a process instance, i.e. a sequence of activities. These activities are presented one by one to the network, in an integer encoded fashion. In a first hidden layer this integer vector is converted into an embedding vector. An embedding is a vector representation, trained to be meaningful in the sense of constructing a lower-dimensional vector which retains as much of the original topology of the input space as possible. Embeddings can be trained in self-supervised fashion on a big corpus (e.g. word2vec [18]) or in a supervised fashion, added as an extra layer, as done here. The one-hot encoded vectors are multiplied by a weight matrix with dimensions *dimension embedding* $\times$ *activity vocabulary size*. This weight matrix thus contains the embeddings of the different

activities in each column and is trained while training the entire network. You could also extract these embeddings for other purposes afterwards. Or use pre-trained embeddings, trained in a word2vec like fashion, as done by [7], albeit in different contexts. The activity embeddings are fed to the bi-directional simple RNN layer. The output of this layer is then send through a dense layer which predicts the label of the sequence. This output is a number between 0 and 1 and can be interpreted as a probability. This label is taken from the output of the last time step in the sequence (the final activity), but in theory the label could be predicted at each time step. This could provide extra insights into at which specific activity the nonconforming behavior takes place and could provide interesting visualizations. For now, however, the focus lies on predicting one overall label, i.e. whether a trace stems from a log or its antilog. In this research setting, the neural network implementation was executed by means of the the Python library Keras [3] and sequences were padded to the maximal length, such that all eventual input sequences had the same size. The dimension of the embedding layer was set to 4 and the dimension of the recurrent layer was set to 8. The RNN uses a sigmoid activation function and was trained to optimize the binary cross entropy loss using RmsProp [24] for 40 epochs with a minibatch size of 64. In order to counteract overfitting a dropout of 0.5 was added between the recurrent layer and the dense output layer.

3 Experimental Evaluation

In this section the newly introduced technique will be tested[4]. In this experiment, we focus on global recall and precision calculation. Hereto, we use the event log in Table 1 together with the models 1-10 in Figure 3, obtained from van Dongen et al. [9]. This set of models was supplemented with model 11, a model with low recall and high precision. We also use a model discovered from the event log by the Alpha miner [25] and 3 models discovered by the inductive miner [17], with the noise parameter set to 0, 0.5 and 1 respectively. These models can also be found in Figure 3. The discovery algorithms were used as implemented in the ProM framework [10].

Table 1: The test log used for the experimental setup [9].

Instance	#
$\langle A, B, D, E, I \rangle$	1207
$\langle A, C, D, G, H, F, I \rangle$	145
$\langle A, C, G, D, H, F, I \rangle$	56
$\langle A, C, H, D, F, I \rangle$	23
$\langle A, C, D, H, F, I \rangle$	28

[3] `https://keras.io`

[4] The implementation of the technique, tests and the synthetic data used can be found on `https://github.com/jaripeeperkorn/Supervised-Conformance-Checking-using-Recurrent-Neural-Network-Classifiers`

For each of the models we calculate the global fitness and precision, both by averaging the probability values as well as by counting the fraction of instances with a probability above the 0.5 threshold. This was done using the setting described above. We each time generated a model log with 1500 process instances, which is not sufficient to capture all possible behavior in some of the extreme models (e.g. the flower model). In most cases however, this was more than sufficient. The technique was performed 10 times for each model, and the median is taken to be the eventual value as well as the standard error $\sigma/\sqrt{10}$. These values are compered with different methods from literature: alignment based recall and precision [1], behavioral recall [13] and precision [27] and Markovian recall and precision with $k = 3$ [3]. We used the implementations in CoBeFra [26] for the alignment and behavioral based metrics. The results can be found in Table 2.

Table 2: Resulting recall and precision values.

Model	Recall					Precision				
	Prob.	Count	[1]	[13]	[3]	Prob.	Count	[1]	[27]	[3]
1	1.00 ± 0.00	1.00 ± 0.00	1.00	1.00	1.00	1.00 ± 0.04	1.00 ± 0.03	0.98	1.00	0.88
2	0.83 ± 0.00	0.83 ± 0.00	0.92	0.81	0.23	1.00 ± 0.00	1.00 ± 0.00	1.00	0.89	1.00
3	0.79 ± 0.01	1.00 ± 0.00	1.00	1.00	1.00	0.00 ± 0.00	0.00 ± 0.00	0.14	0.12	0.00
4	1.00 ± 0.00	1.00 ± 0.00	1.00	0.99	1.00	1.00 ± 0.01	1.00 ± 0.00	1.00	0.94	1.00
5	1.00 ± 0.00	1.00 ± 0.05	1.00	1.00	1.00	0.91 ± 0.02	0.89 ± 0.02	0.95	0.95	0.56
6	1.00 ± 0.00	1.00 ± 0.00	1.00	1.00	1.00	0.77 ± 0.01	0.77 ± 0.02	0.95	0.87	0.18
7	1.00 ± 0.00	1.00 ± 0.00	1.00	1.00	1.00	0.58 ± 0.04	0.60 ± 0.06	0.80	0.72	0.35
8	0.52 ± 0.02	0.52 ± 0.14	0.74	1.00	1.00	0.00 ± 0.00	0.00 ± 0.00	0.34	0.16	0.01
9	1.00 ± 0.00	1.00 ± 0.00	1.00	1.00	—	0.50 ± 0.01	0.50 ± 0.01	0.84	0.60	—
10	0.43 ± 0.11	0.45 ± 0.15	0.62	0.59	0.09	0.00 ± 0.00	0.00 ± 0.00	0.89	0.19	0.06
11	0.15 ± 0.01	0.17 ± 0.02	0.62	0.35	0.64	1.00 ± 0.01	1.00 ± 0.00	1.00	0.36	1.00
Alpha	0.96 ± 0.00	0.97 ± 0.00	1.00	0.99	0.77	0.73 ± 0.01	0.72 ± 0.01	0.96	0.92	0.38
Ind. 0	1.00 ± 0.00	1.00 ± 0.00	1.00	1.00	1.00	0.86 ± 0.04	0.89 ± 0.05	0.72	0.59	0.42
Ind. 0.5	1.00 ± 0.00	1.00 ± 0.00	1.00	0.99	0.77	0.84 ± 0.03	0.86 ± 0.04	0.79	0.69	0.44
Ind. 1	0.18 ± 0.11	0.17 ± 0.12	0.86	0.84	0.68	0.82 ± 0.06	0.81 ± 0.07	0.87	0.64	0.48

Our new technique approximately agrees with existing metrics from the literature for most of the models. The recall values obtained for model 2 can be explained by looking at the fraction of instances in the event log corresponding to the most frequent trace $1207/1459 \approx 0.83$. Similarly, for model 11, the fraction of the least frequent traces in the event log $(56 + 23 + 28)/1459 \approx 0.07$ can be obtained. The recall score provided by the RNN is slightly higher, but not as high as the scores provided by alignment or Markovian based recall. For both model 2 and model 11 the precision is 1, as it should be. Noticeable differences in recall and/or precision appear in the special cases, i.e. model 3 (flower), model 8 (all parallel) and model 10 (round-robin). Whereas the extremely low precision values can be seen as correct, the incorrect low recall values can be attributed to an (incomplete) random model log generation. Due to the infinite (for the flower model) or high number of possible traces that can be generated by these models, the chance that not all or even few traces present in the event log are not simulated is high. This leads to incorrectly low recall values. This might be solvable by generating the model log in a different way (e.g. complete log generation with a maximum on the number of times a particular marking

can be seen). However, for these overly general models, simply generating a bigger model log will not work, as the random antilog grows simultaneously (we chose to keep them the same size). The chance for a flower model to generate something close to the instances in the event log, is similar to the chance of the random antilog to generate it. In a way, an extra punishment on the recall value is given to overly general and imprecise models. It however also clear that each of the other method from literature have different examples they cannot handle properly. Another important difference in recall values can be seen in the model discovered by the inductive miner with noise parameter set to 1. Our newly proposed technique outputs a recall value significantly lower than the ones obtained by the methods from the literature. When looking at the model, you can actually see that it is not able to replay the most frequent instance from the event log $\langle A, B, D, E, I \rangle$, but is able to replay the other ones. The value obtained by our technique corresponds approximately with the fraction of the replayable traces in the event log $(145 + 56 + 23 + 28)/1459 \approx 0.17$, while the other techniques return values significantly higher. Apart from the extreme case in model 8, the method using the average of the probabilities and the method using the counts do not differ much.

We further investigated the convergence of the recall and precision values with increasing model log size (and therefore with increasing model antilog size as well). It was observed that convergence happens at a significantly higher number of instances than the amount of different variants the model can produce. This indicates that not necessarily the model log generation, but rather the random antilog generation requires enough examples in order to obtain a stable result. Because the RNN outputs the probability of an instance belonging to either the event log or the antilog, we can also show a probability distribution over all instances. This might provide end-users with a valuable visualisation. We also manually confirmed the method was able to pick up on small (unwanted) change. There is off course a trade-off. If it is necessary that small changes (e.g. switch "sign contract" before "check contract") not yet seen, result in a nonconforming label, having a RNN which is trained to generalize its predictions maximally, might provide an issue in combination with the current antilog generation. By reducing possible overfitting measures (e.g. dropout between layers) the shape of the distribution becomes more bimodal with two peaks (at 0 and 1) and no or little mass in between. On the other hand, if small changes are not a problem and we are more interested in the distribution of the fitness values, it is necessary to make sure the neural network is still generalizing enough. If you already know which specific behavior is certainly not desired, you can add this behavior manually to the antilog. This means adding multiple (different) traces containing the unwanted behavior to the antilog. The time it takes to run the method (precision or recall) with two event logs (and one antilog) of 1500 instances (size 20 activities, vocabulary size of 10) one time (including) training is around $9s$, as performed on an Intel(R) Core(TM) i7-9850h CPU @ 2.60ghz. If the RNN is already trained and you would only need to put through one log this reduces to $0.2s$.

4 Related work

One of the earliest research on Conformance Checking can be found in Cook et al. (1999) [6]. Multiple techniques for obtaining recall or precision have been proposed ever since. One example of an early precision metric was introduced by Greco et al. (2006) [14] which calculates a "set difference", between a set of traces representing the log's behavior and one representing the model's behavior. Other noteworthy earlier contributions are proper completion, token based sequence replay and the advanced behavioral appropriateness [20]. Behavioral recall (using a percentage of correctly classified positive events) and behavioral specificity (replaying the sequences and taking the percentage of correctly classified negative events) were introduced by Goedertier et al. (2009) [13] and was later supplemented with a similar method using the amount of "false positives" (behavior allowed by model, but labeled a negative event based on the log) [8] and by the behavioral precision [27]. Another method using behavioral profile based metrics, based on different constraints a process model can impose on a log, was introduced by Weidlich et al. (2011) [28]. Another metric is ETC precision which uses log prefix automatons and the number of "escaping" edges [19]. This was later altered in projected conformance checking (PCC), better scaled to real-life logs [16]. A lot of focus has been on the (average) alignment based trace recall and precision approach introduced in [1], supplemented by the one align precision and best align precision [2]. Van Dongen et al. (2016) proposed the use of "anti-alignments" to obtain a model's precision [9]. Recently promising work comparing Markovian abstractions of both event log and model has been shown as a potential efficient alternative to alignments based methods [3]. In recent years different axioms were proposed as well, describing (un)wanted behavior that conformance checking metrics should (not) adhere to [21].

The method introduced in this work draws some resemblance to Classifier-Adjusted Density Estimation for Anomaly Detection and One-Class Classification (or CADE) [12], which is an anomaly detection method that uses a classifier trained on discriminating between real data and synthetic data generated uniformly over all features. RNN's have earlier been used in predictive process monitoring [11, 22]. Recent work in predictive monitoring has also introduced the use of Generative Adverserial Nets [23]: a self-supervising machine learning technique based on training a discriminator and a generator simultaneously, which draws some similarities to the technique introduced here.

5 Conclusion and Future Work

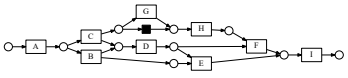
We have presented a new technique to obtain recall and precision of event logs and process models. The technique was tested on a small example log and models to show its potential for global conformance analysis. In future work, it should be interesting to hold our new technique against the recently proposed conformance checking axioms [21]. However, due to the intrinsic probabilistic nature of the technique, doing this theoretically might be hard and require empirical

backing. Another interesting avenue would be to directly rely on the given process model, omitting the model log generation step, although this would require serious alterations to the technique. A less drastic improvement could be found in a more sophisticated antilog generation, though the simplicity combined with good results obtained by the current approach is nonetheless appealing. Another method could e.g. use negative events [8,13,27] or other smart usage of different data features. Next, it could be interesting to use the output at each time step (see Figure 2) for explanatory purposes. Since we could then inspect at which activity exactly the model starts to classify the instance as being nonconforming. Preliminary results show that such an approach is viable. Finally, since the model could in theory be extended to include additional data attributes (e.g. the resource or other perspectives), including this in future implementations might provide us with additional advantages compared to competitive conformance checking techniques.

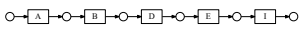
Acknowledgement. *This research has been financed in part by the NeEDS research project, an EC H2020 MSCA RISE project with Grant agreement No. 822214.*

References

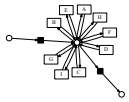
1. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery **2**(2), 182–192 (2012). https://doi.org/10.1002/widm.1045
2. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: Business Process Management Workshops. pp. 137–149 (2013)
3. Augusto, A., Conforti, R., Armas-Cervantes, A., Dumas, M., La Rosa, M.: Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. IEEE Transactions on Knowledge and Data Engineering pp. 1–1 (2020)
4. Berti, A., van Zelst, S.J., van der Aalst, W.: Process mining for python (PM4Py): Bridging the gap between process-and data science. In: Proceedings of the ICPM Demo Track 2019, co-located with 1st International Conference on Process Mining (ICPM 2019), Aachen, Germany, June 24-26, 2019. p. 13–16 (2019)
5. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR **abs/1406.1078** (2014)
6. Cook, J.E., Wolf, A.L.: Software process validation: Quantitatively measuring the correspondence of a process to a model. ACM Trans. Softw. Eng. Methodol. **8**(2), 147–176 (Apr 1999)
7. De Koninck, P., vanden Broucke, S., De Weerdt, J.: act2vec, trace2vec, log2vec, and emodel2vec: Representation learning for business processes. In: Business Process Management. pp. 305–321 (2018)
8. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A robust f-measure for evaluating discovered process models. In: IEEE Symposium series on computational intelligence. pp. 148–155 (2011)



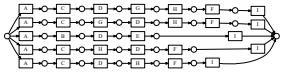
Model 1: good recall and good precision [9].



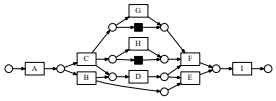
Model 2: single most frequent trace [9].



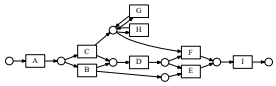
Model 3: flower model [9].



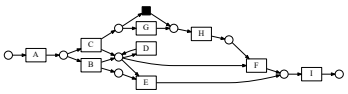
Model 4: all traces in parallel [9].



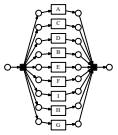
Model 5: G and H in parallel [9].



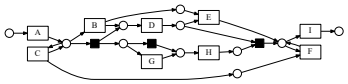
Model 6: G and H in self-loops [9].



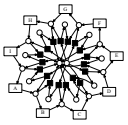
Model 7: D in a self-loop [9].



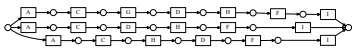
Model 8: all transitions in parallel [9].



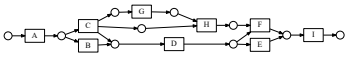
Model 9: C and F are in a loop and need to be executed the same amount to reach the final marking [9].



Model 10: Round-robin model [9].

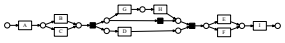


Model 11: 3 least common traces in parallel.



Model alpha: discovered by alpha miner [25] using the log in Table 1.

Model Ind. 0: discovered by inductive miner with noise set to 0 [17] using the log in Table 1.



Model Ind. 0.5: discovered by inductive miner with noise set to 0.5 [17] using the log in Table 1.

Model Ind. 1: discovered by inductive miner with noise set to 1 [17] using the log in Table 1.

Fig. 3: Models used for the experimental setup.

9. van Dongen, B.F., Carmona, J., Chatain, T.: A unified approach for measuring precision and generalization based on anti-alignments. In: Business Process Management. pp. 39–56. Springer International Publishing, Cham (2016)
10. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The prom framework: A new era in process mining tool support. In: Applications and Theory of Petri Nets 2005. pp. 444–454 (2005)

11. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)
12. Friedland, L., Gentzel, A., Jensen, D.: Classifier-adjusted density estimation for anomaly detection and one-class classification. In: SDM. pp. 578–586 (04 2014). https://doi.org/10.1137/1.9781611973440.67
13. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. Journal of Machine Learning Research **10**, 1305–1340 (2009)
14. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering **18**(8), 1010–1027 (2006)
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735
16. Leemans, S.J., Fahland, D., Aalst, W.M.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2) (2018)
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Application and Theory of Petri Nets and Concurrency. pp. 91–110 (2014)
18. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: 1st International Conference on Learning Representations, ICLR 2013 (2013)
19. Muñoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Business Process Management. pp. 211–226 (2010)
20. Rozinat, A., Aalst, van der, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
21. Syring, A.F., Tax, N., van der Aalst, W.M.P.: Evaluating Conformance Measures in Process Mining Using Conformance Propositions, pp. 192–221. Springer Berlin Heidelberg (2019)
22. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Advanced Information Systems Engineering. pp. 477–492. Springer International Publishing, Cham (2017)
23. Taymouri, F., Rosa, M.L., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management. pp. 237–256. Springer International Publishing, Cham (2020)
24. Tieleman, T., Hinton, G.: Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning (2012)
25. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1128–1142 (Sep 2004)
26. vanden Broucke, S.K.L.M., De Weerdt, J., Vanthienen, J., Baesens, B.: A comprehensive benchmarking framework (cobefra) for conformance analysis between procedural process models and event logs in prom. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 254–261 (2013)
27. vanden Broucke, S.K.L.M., De Weerdt, J., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. IEEE Transactions on Knowledge and Data Engineering **26**(8), 1877–1889 (2014)
28. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**, 1009–1025 (11 2011)