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.

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

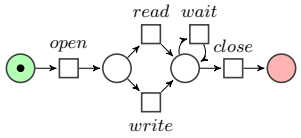


Fig. 1: A log L and an associated process model M

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

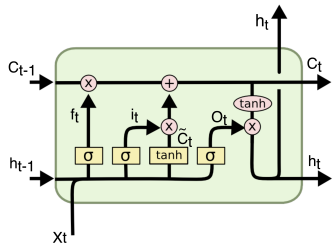
There are several approaches towards training classification models from sequential data in a supervised way. They have in common that they must encode sequences of variable lengths as fixed-size vectors; these vectors are subsequently used as training examples for the classifier, which learns a classification 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.

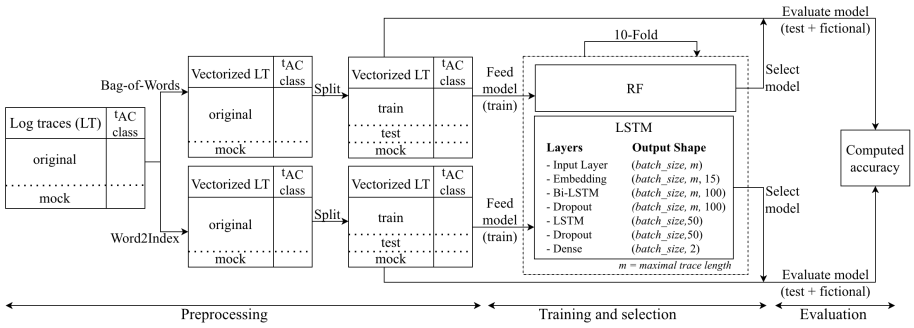


Fig. 3: Overview of the experimental setup

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)