An Experimental Evaluation of the Generalizing Capabilities of Process Discovery Techniques and Black-Box Sequence Models

Niek Tax[1], Sebastiaan J. van Zelst[1], and Irene Teinemaa[2]

[1] Eindhoven University of Technology, The Netherlands,
{n.tax,s.j.v.zelst}@tue.nl
[2] University of Tartu, Estonia,
irene.teinemaa@ut.ee

Abstract. A plethora of *automated process discovery* techniques have been developed which aim to discover a process model based on event data originating from the execution of business processes. The aim of the discovered process models is to describe the control-flow of the underlying business process. At the same time, a variety of *sequence modeling* techniques have been developed in the machine learning domain, which aim at finding an *accurate*, not necessarily *interpretable*, model describing sequence data. Both approaches ultimately aim to find a model that *generalizes* the behavior observed, i.e., they describe behavior that is likely to be part of the underlying distribution, whilst disallowing unlikely behavior. While the generalizing capabilities of process discovery algorithms have been studied before, a comparison, in terms of generalization, w.r.t. sequence models is not yet explored. In this paper we present an experimental evaluation of the generalizing capabilities of automated process discovery techniques and black-box sequence models, on the basis of *next activity prediction.* We compare a range of process discovery and sequence modeling techniques on a range of real-life datasets from the business process management domain. Our results indicate that LSTM neural networks more accurately describe previously unseen traces (i.e., test traces) than existing process discovery methods.

Keywords: Process mining, behavioral generalization, next activity prediction, process discovery, sequence modeling

1 Introduction

In recent years, the spectacular off-take of data-driven business process analysis, i.e., *process mining* [1], has lead to the availability of a wide variety of tools and techniques that allow business owners to get a better understanding of the execution of their processes. The vast majority of existing process mining techniques either focuses on *process discovery*, i.e., discovering a process model based on process execution data, or *conformance checking*, i.e., assessing whether a process model indeed is in correspondence with the observed behavior. Process discovery techniques allow a business analyst to get a static view of the process, aiding in restructuring and/or reorganization of the process. In recent years, many algorithms have been developed that automatically discover a model of the process from execution data [3,7,16,22,23,25,38,37]. These algorithms' resulting process models provide a visual and interpretable model of the process. The main challenge of process discovery is to accurately generalize the behavior to unseen but possible executions of the business process.

The data on which process discovery algorithms typically operate, i.e. *event logs*, are typically formalized as a collection of sequences of business process activities, i.e. *traces*. Within machine learning, techniques have been developed for *sequence modeling*, addressing the task of finding a model that accurately describes a collection of sequences. Well-known examples of sequence models include n-gram models [14], Markov models, and Recurrent Neural Networks (RNNs) [18]. In contrast to process discovery techniques, sequence modeling techniques are less domain specific, e.g., they do not explicitly account for the parallel behavior that is often inherently present in business processes. Sequence models have been successfully applied to sequence data in many other application domains, including natural language modeling [14], music sequences [24], and DNA sequences in bioinformatics [33]. Moreover, where process discovery aims to find interpretable models, sequence models are generally *black-box models*, focusing purely on accurately describing the sequences.

While the fields of process discovery and sequence modeling are conceptually very close, the main difference between the two approaches is related to the visual interpretability of the result as well as the explicit assumption on the existence of parallelism in the input data. Generally, interpretability of discovered models is seen as important in the process mining/BPM field. However, we argue that this is not always the case and that in some circumstances the accuracy and generalizing capabilities of the model are more important than its interpretability. In fact, recent efforts in the area of *predictive business process monitoring*, i.e., the prediction of expected properties of process instances at runtime (e.g., the outcome, remaining cycle time) have already applied black-box sequence models in the BPM domain. For example, the recent interest in deep learning [21] motivated several researchers to study the applicability of neural networks for the purpose of predictive business process monitoring [15,28,34,35]. However, to date, it has not been investigated how the capabilities of sequence models to describe the control-flow of a process compares to existing process discovery techniques.

In this paper, we investigate the applicability of both process discovery techniques and black-box sequence models on *next activity prediction*. In particular, we assess to what degree the techniques are able to generalize, i.e. to account for unseen behavior. We additionally present means to use discovered process models as probabilistic classifiers, i.e., to predict a probability distribution over the next activity that follows some prefix of a case. This enables the comparison of existing process discovery and sequence modeling techniques, using standard evaluation methods from the sequence modeling domain. Fig. 1 shows the setup for this comparison, where an event log is first split into a train and a test log. A model (i.e., either a sequence model or a process model) is fitted on the train-

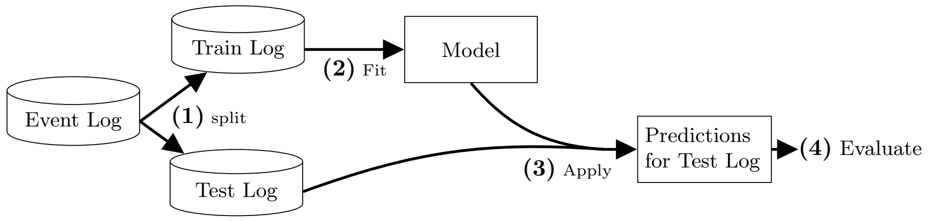

Fig. 1: Overview of the proposed evaluation.

ing log, which is then used to generate probability distributions over the next activity for each prefix of the test log. The quality of the generated probability distributions for the prefixes in the test log indicates the degree to which the model is able to generalize to unseen data. We compare sequence models and process discovery techniques on a range of real-life event logs from the BPM field.

The remainder of this paper is structured as follows. In Section 2, we describe related work. In Section 3, we describe basic concepts and notations that are used throughout the paper. In Section 4 we describe how to use Petri nets as probabilistic classifiers to predict the next activity given a prefix, and we give an overview of sequence modeling techniques. We describe the experimental setup in Section 5 and discuss the results of the experiments in Section 6. Finally, we conclude this paper and identify several interesting areas of future work in Section 7.

2 Related Work

Several approaches have been proposed in the literature to tackle the challenge of predicting the next activity in an ongoing business process instance. Some of these methods encode the available data as a feature vector and apply a classifier to predict the next event [31,36,27]. Others discover a process/sequence model from the control flow using sequential pattern mining [8], Markov models [20] or a Probabilistic Finite Automaton [4]. Often, as a second step after discovering the process model, classifiers are built for each state in the model, enabling to include the data payload of the ongoing case into the prediction process [8,20]. More recently, deep learning approaches (mostly LSTMs) have been applied to the task of next activity prediction [15,28,34]. However, none of these studies compare their method to existing process discovery techniques. In this work we aim to bridge this gap by comparing the most widely used representatives from the sequence modeling field to well-known process discovery techniques. Furthermore, while most of the next activity prediction methods strive for a high accuracy for a given process instance, our focus in this paper is on assessing the generalizing capabilities of the sequence modeling/process discovery techniques in terms of control-flow, i.e. we tackle the question of how good are the states that each technique is able to learn.

Other work in the predictive monitoring field focuses on predicting the entire continuation of the case beyond the immediate next activity [32,34], deadline violations [29], remaining cycle time [34], and the outcome of a case [11,35].

3 Background

In this section, we introduce concepts used in later sections of this paper.

3.1 Events, Sequences, and Sequence Databases

X^* denotes the set of all sequences over a set X and $\sigma{=}\langle a_1, a_2, \dots, a_n\rangle$ a sequence of length n, with $\sigma(i){=}a_i$ and $|\sigma|{=}n$. $\langle\rangle$ is the empty sequence and $\sigma_1{\cdot}\sigma_2$ is the concatenation of sequences σ_1 and σ_2. $hd^k(\sigma){=}\langle a_1, a_2, \dots, a_k\rangle$ is the prefix of length k (with $0{<}k{<}|\sigma|$) of sequence σ, for example, $hd^2(\langle a, b, c, d, e\rangle){=}\langle a, b\rangle$. A multiset (or bag) over X is a function $B : X{\rightarrow}\mathbb{N}$ which we write as $[a_1^{w_1}, a_2^{w_2}, \dots, a_n^{w_n}]$, where for $1{\leq}i{\leq}n$ we have $a_i{\in}X$ and $w_i{\in}\mathbb{N}^+$. The set of all multisets over X is denoted $\mathcal{B}(X)$. A partial function $f\colon X \nrightarrow Y$ is a function which is not necessarily defined on each element in X. A partial function $f{\in}X{\nrightarrow}Y$, can be lifted to sequences over X using: (1) $f(\langle\rangle){=}\langle\rangle$; (2) for any $\sigma{\in}X^*$ and $x{\in}X$:

$$f(\sigma \cdot \langle x\rangle) = \begin{cases} f(\sigma) \cdot \langle f(x)\rangle & \text{if } f(x) \in Y, \\ f(\sigma) & \text{otherwise.} \end{cases}$$

An *event* e denotes the occurrence of a process activity. We write Σ to denote the universe of business process activities. A *trace* is a sequence $\sigma{\in}\Sigma^*$. An *event log* is a finite multiset of sequences, $L{\in}\mathcal{B}(\Sigma^*)$. For example, the event log $L{=}[\langle a, b, c\rangle^2, \langle b, a, c\rangle^3]$ consists of two occurrences of trace $\langle a, b, c\rangle$ and three occurrences of trace $\langle b, a, c\rangle$.

3.2 Process Models and Process Discovery

We use Petri nets to represent process models, as most state-of-the-art process discovery discover Petri net-like models. A Petri net is a directed bipartite graph consisting of places (depicted as circles) and transitions (depicted as rectangles), connected by arcs. A transition describes an activity, while places represent the enabling conditions of transitions. Labels of transitions indicate the type of activity that they represent. Unlabeled transitions (τ-transitions) represent invisible transitions (depicted as gray rectangles), which are only used for routing purposes and are typically unobservable. As an example of a Petri net, consider Fig. 2, which contains 7 places and 6 transitions. The activity corresponding to transition t_1 is activity A, whereas transition t_3 is unobservable.

Definition 1 (Labeled Petri net). *A* labeled Petri net $N = (P, T, F, \ell)$ *is a tuple where P is a finite set of places, T is a finite set of transitions such that $P{\cap}T{=}\emptyset$, $F{\subseteq}(P{\times}T){\cup}(T{\times}P)$ describes the Petri net flow relation (graphically represented by means of arcs), and $\ell{:}T{\nrightarrow}\Sigma$ is a partial labeling function that assigns a label to a transition, or leaves it unlabeled (the τ-transitions).*

We write $\bullet n$ and $n\bullet$ for the input and output nodes of $n \in P \cup T$ (according to F), e.g. in Fig. 2 we have $p\bullet = \{t_1, t_2\}$ and $\bullet t_3 = \{p_2\}$. If the labeling function is undefined for $t \in T$, we write $\ell(t) = \tau$. A state of a Petri net is defined by its

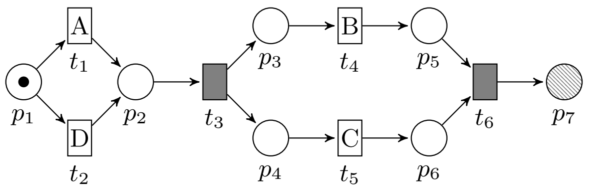

Fig. 2: An example accepting Petri net APN with $\mathfrak{L}(APN) = \{\langle A, B, C\rangle, \langle A, C, B\rangle, \langle D, B, C\rangle, \langle D, C, B\rangle\}$.

marking $m{\in}\mathcal{B}(P)$ being a multiset of places. A marking is graphically denoted by putting $m(p)$ tokens on each place $p{\in}P$. For example, consider the marking of the example Petri net in Fig. 2, i.e. $[p_1]$, represented by the black dot drawn in p_1. State changes of a Petri net occur through *transition firings*. A transition t is enabled (can fire) in a given marking m if each input place $p{\in}{\bullet}t$ contains at least one token. Once t fires, one token is removed from each input place $p{\in}{\bullet}t$ and one token is added to each output place $p'{\in}t{\bullet}$, leading to a new marking $m'{=}m{-}\ {\bullet}t + t{\bullet}$. Firing a transition t in marking m, yielding marking m', is denoted as step $m\xrightarrow{t}m'$. Several subsequent firing steps are lifted to sequences of firing enabled transitions, written $m\xrightarrow{\gamma}m'$ and $\gamma{\in}T^*$ is a *firing sequence*.

Defining an *initial* and *final* markings allows to define the *language* accepted by a Petri net as a set of finite sequences of activities. To this end we define the notion of an *accepting Petri net*, i.e. a Petri net including an explicit initial and final marking.

Definition 2 (Accepting Petri Net). *An* accepting Petri net *is a triplet $APN = (N, m_0, m_f)$, where $N = (P, T, F, \ell)$ is a labeled Petri net, $m_0{\in}\mathcal{B}(P)$ is its initial marking, and $m_f{\in}\mathcal{B}(P)$ its final marking. A sequence $\sigma{\in}\Sigma^*$ is a* trace *of an accepting Petri net APN if there exists a firing sequence $m_0\xrightarrow{\gamma}m_f$, $\gamma{\in}T^*$ and $\ell(\gamma){=}\sigma$.*

In this paper, places that belong to the initial marking contain a token and places belonging to the final marking are marked as ▨.

The *language* $\mathfrak{L}(APN)$ of an accepting Petri net is defined as the set of all its traces, i.e., $\mathfrak{L}(APN) = \{\sigma \in \Sigma^* \mid \exists \gamma \in T^*(\ell(\gamma) = \sigma \wedge m_0\xrightarrow{\gamma}m_f)\}$, which is potentially of infinite size, i.e. when APN contains loops. Fig. 2 shows an example of an accepting Petri net with $\mathfrak{L}(APN) = \{\langle A, B, C\rangle, \langle A, C, B\rangle, \langle D, B, C\rangle, \langle D, C, B\rangle\}$. While we define the language for accepting Petri nets, in theory, $\mathfrak{L}(M)$ can be defined for any process model M with formal semantics. We denote the universe of process models as $\mathcal{M}$ and assume that for each $M{\in}\mathcal{M}$, $\mathfrak{L}(M) \subseteq \Sigma^*$ is defined.

A process discovery method is a function $PD : \mathcal{B}(\Sigma^*) \rightarrow \mathcal{M}$ that produces a process model from an event log. The discovered process model should cover as much as possible the behavior observed in the event log (a property called *fitness*) while it should not allow for too much behavior that is not observed in the event log (called *precision*). For an event log L, $\tilde{L}{=}\{\sigma{\in}\Sigma^* | L(\sigma){>}0\}$ is the *trace set* of L. For example, for event log $L{=}[\langle a, b, c\rangle^2, \langle b, a, c\rangle^3]$, $\tilde{L}{=}\{\langle a, b, c\rangle\langle b, a, c\rangle\}$. For an event log L and a process model M, we say that L is *fitting* on M if $\tilde{L} \subseteq \mathfrak{L}(M)$. *Precision* is related to the behavior that is allowed by a model M that was not observed in event log L, i.e., $\mathfrak{L}(M) \setminus \tilde{L}$.

4 Next Activity Prediction

In this section, we present several means to predict the probability distribution over the next activity following a given prefix of a case. We first introduce a method based on (discovered) Petri nets in Section 4.1 after which we briefly describe sequence models from the machine learning domain in Section 4.2.

4.1 Petri Net Based Prediction

To generate a probability distribution over the set of process activities Σ for the event following the prefix σ of a trace using an accepting Petri net APN, we use a two-step approach:

1. We compute the marking m corresponding to prefix σ in Petri net APN.
2. Based on this derived marking m we consider the subset of the process activities Σ that are reachable from m in APN to have nonzero probability and construct a probability distribution over them.

We now continue by detailing these two steps.

Step 1) Computing Prefix-Based Markings in a Petri net To deduce what activities follow a prefix σ, using Petri net APN as a sequence model, we need to obtain the marking corresponding to σ. A naive approach to this problem is to play the so-called "token-game", to find a firing sequence $\gamma \in T^*$ that projected on ℓ equals σ, and marks some arbitrary marking m' in APN. Subsequently, based on marking m', we determine the probable next activities.

Finding such sequence is however far from trivial. Since ℓ is a partial function, it is possible that multiple transitions have the same label and that some other transitions have no label at all. Hence, finding such $\gamma \in T^*$ is actually a combinatorial problem. Aside from these model-based issues, in real-life scenarios, noise is possibly present in the event data as well and the event data might not be completely fitting on the discovered process model APN. For these reasons, a prefix is potentially not completely explainable in terms of the underlying model. Therefore, we apply a technique called *prefix-alignments* [2] to find the path through the model that most closely resembles the prefix.

A prefix-alignment allows us to explain a partial sequence of observed behavior in context of an accepting Petri net. In particular, such alignment assumes that the behavior starts from the initial marking of the model, yet is incomplete, i.e. future behavior is potentially possible. Essentially, a prefix-alignment maps each activity in a prefix to the execution of a transition in the model. Observe that such mapping manipulates the marking of the corresponding model. If we are able to construct a direct mapping between an observed activity and the

Activity	A	$\gg$	B
Transition	t_1	t_3	t_4
$\ell(t)$	A	τ	B

Activity	A	D	$\gg$	B
Transition	t_1	$\gg$	t_3	t_4
$\ell(t)$	A	$\gg$	τ	B

Fig. 3: Example prefix-alignments for prefixes $\langle A, B\rangle$ and $\langle A, D, B\rangle$ with the example APN of Fig. 2.

execution of a transition, we call such mapping a *synchronous move*. In some cases, we observe behavior in the prefix that we are not able to explain in terms of the model, in such case, we decided not to map an activity on the execution of a transition. Such construct is called a *log move*. The reverse is also possible, i.e. we observe behavior that is likely according to the model, however, we did not observe an activity that actually enables this behavior. In such case we insert the firing of a transition without the explicit binding to an observed activity, which we call a *model move*.

Consider Fig. 3 in which we depict two prefix-alignments of two different prefixes of traces, i.e. $\langle A, B\rangle$ and $\langle A, D, B\rangle$, originating from the process as described by the example accepting Petri net in Fig. 2. The leftmost prefix-alignment refers to a behavioral prefix that fits completely with the model, i.e. we map the A activity in the trace onto an execution of t_1, for which we have $\ell(t_1) = A$, i.e. a synchronous move. Subsequently we observe a model move on t_3, with $\ell(t_3) = \tau$, i.e. inherently unobservable, and finally we map the second activity, i.e. B, on the execution of t_4. The prefix-alignment of prefix $\langle A, D, B\rangle$ depicts some deficiencies. We map the first activity, i.e. A, on the execution of transition t_1. The second activity, i.e. D is not mapped on the execution of any transition, i.e. there is a choice between A and D in the model, and hence we obtain a log move. Again, we use a model move on t_3 to finally obtain a synchronous move on activity B.

In general, a multitude of prefix-alignments exists for a given prefix and a process model. For example, in trace $\langle A, D, B\rangle$, as opposed to the right-most alignment in Fig. 3, we are also able to synchronize on D rather than A. The only requirement is that, after explaining the prefix in terms of the model, the corresponding marking still allows us to reach the final marking, in some way. Often we therefore assign costs to the different types of moves as presented earlier. Typically, synchronous moves have zero-costs, model and log moves usually get a cost of 1 assigned[1]. Computing prefix-alignments is performed by means of solving a shortest path problem on an extended state-space of the accepting Petri net [2]. The costs of a move define the edge costs within such state-space.

For the purpose of this paper, we simply assume the fact that given a prefix and a process model, we are able to obtain the corresponding marking in the accepting Petri net. Observe that, as described, prefix-alignments are the most suitable approach. However, in general any technique that given a prefix and a model results in a corresponding marking is feasible to use.

[1] Except for model moves that relate to unobservable activities, which also get cost 0 assigned.

Step 2) Generating Next Activity Distributions We propose two ways to generate a probability distribution describing the next activity, based on a marking m obtained for a prefix σ in step 1. From a certain marking, several transitions can be enabled at the same time, e.g. consider marking $[p_1]$ in Fig. 2, in which both transition t_1 and t_2 are enabled. Note that the same holds for marking $[p_3, p_4]$ in which both transition t_4 and t_5 are enabled. Both in the case of choice and parallel constructs, multiple transitions are usually enabled. Alternatively, e.g. in marking $[p_2]$, the only transition that is enabled, which is t_3, has no associated label. For a given marking $m \in \mathcal{B}(P)$ in an accepting Petri net APN, $\omega(m) = \{t | t \in T \wedge \bullet t \subseteq m\}$ is the set of enabled, or firable, transitions. Note that $\omega(m)$ can contain labeled as well as invisible transitions. A probability mass function $prob_m : T \to [0, 1]$ assigns a firing probability to each transition that is enabled from marking m, such that $\Sigma_{t \in \omega(m)} prob_m(t) = 1$. In the first approach, we assume this probability distribution over the enabled transitions to be a uniform categorical distribution, i.e., $prob_m^{uniform}(t) = \begin{cases} \frac{1}{|\omega(m)|} & \text{if } t \in \omega(m), \\ 0 & \text{otherwise.} \end{cases}$

To transform the probability distribution $prob_m$ over the next transition to fire into a probability distribution over the next activity, we apply the following procedure. Starting from marking m in Petri net APN we pick an enabled transition at random according to probability distribution $prob_m^{uniform}$. Whenever that transition has a corresponding label, we pick it as the next activity. If it relates to an unobservable transition, we fire it, leading to a new marking m' and apply the same procedure from up-until we select a transition that has a label. This procedure to select the next activity from a marking m is repeated a number of times and we set the probabilities over the activities according to how often they were selected, i.e. by Monte Carlo simulation. For example, in marking p_2 in Fig. 2 we always fire transition t_3 which yields marking $[p_3, p_4]$. In that marking we randomly pick any of the two transitions, i.e. t_4 or t_5, and register their corresponding labels as a next activity, i.e. either B or C. Thus, we obtain $prob_{[p_2]}^{uniform}(B) = prob_{[p_2]}^{uniform}(C) = 0.5$.

In the second approach, after discovering a process model based on the training log, we compute an empirical distribution $prob_m^{empirical}$. This is rather straightforward: for each prefix of each trace in the training log we apply step 1 to obtain the corresponding marking m in the discovered Petri net APN, and we base $prob_m^{empirical}$ on how often each enabled transition $t \in \omega(m)$ was fired when this marking was reached in the training log. This leads to a probability mass function for each marking in the model that is trained/estimated based on the training data. We again apply the same Monte Carlo sampling approach to transform $prob_m^{empirical}$ into a probability distribution over the next activity, instead of over the next transition. For example, in marking p_2 in Fig. 2 we always fire transition t_3 which yields marking $[p_3, p_4]$. From that marking, both B and C are enabled and probability mass values $prob_{[p_2]}^{emperical}(B)$ and $prob_{[p_2]}^{emperical}(C)$ are proportional to how often activities B and C followed marking p_2 in the training log.

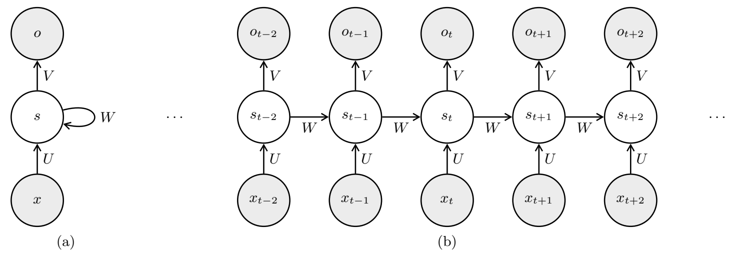

Fig. 4: *(a)* A simple recurrent neural network consisting of a single hidden layer, and *(b)* the recurrent neural network unfolded over time.

Both the Petri-net-based probabilistic classifier based on $prob_m^{uniform}$ and based on the trained $prob_m^{empirical}$ have been implemented and are openly available as part of the ProM process mining toolkit [13] in the package *SequencePredictionWithPetriNets*[2].

4.2 Black-Box Sequence Models

We proceed by introducing sequence models from the machine learning field.

Neural Networks & Recurrent Neural Networks A neural network consists of one layer of *input units*, one layer of output units, and in-between are multiple layers that are referred to as *hidden units.* The outputs of the input units form the inputs for the units of the first *hidden layer* (i.e., the first layer of hidden units), and the outputs of the units of each hidden layer form the input for each subsequent hidden layer. The outputs of the last hidden layer form the input for the output layer. The output of each unit is a function over the weighted sum of its inputs. The weights of this weighted sum performed in each unit are learned through gradient-based optimization from training data that consists of example inputs and desired outputs for those example inputs. Recurrent Neural Networks (RNNs) are a special type of neural networks where the connections between neurons form a directed cycle.

RNNs can be unfolded, as shown in Fig. 4. Each step in the unfolding is referred to as a time step, where x_t is the input at time step t. RNNs can take an arbitrary length sequence as input, by providing the RNN a feature representation of one element of the sequence at each time step. s_t is the hidden state at time step t and contains information extracted from all time steps up to t. The hidden state s is updated with information of the new input x_t after each time step: $s_t = f(Ux_t + Ws_{t-1})$, where U and W are vectors of weights over the new inputs and the hidden state respectively. In practice, either the hyperbolic tangent or the logistic function is generally used for function f, which is referred to as the activation function. The logistic function is defined as: $sigmoid(x) = \frac{1}{1+exp(-x)}$. In neu-

[2] `https://svn.win.tue.nl/repos/prom/Packages/SequencePredictionWithPetriNets/`

ral network literature, the sigmoid function is often represented by σ, however, to avoid confusion with traces, we fully write *sigmoid*. o_t is the output at step t.

Long Short-Term Memory for Sequence Modeling A Long Short-Term Memory (LSTM) model [17] is a special Recurrent Neural Network architecture that has powerful modeling capabilities for long-term dependencies. The main distinction between a regular RNN and an LSTM is that the latter has a more complex memory cell C_t replacing s_t. Where the value of state s_t in an RNN is the result of a function over the weighted average over s_{t-1} and x_t, the LSTM state C_t is accessed, written, and cleared through controlling gates, respectively o_t, i_t, and f_t. Information on a new input will be accumulated to the memory cell if i_t is activated. Additionally, the previous memory cell value C_{t-1} can be "forgotten" if f_t is activated. The information of C_t will be propagated to the output h_t based on the activation of output gate o_t. Combined, the LSTM model can be described by the following formulas:

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

In these formulas all W variables are weights and b variables are biases and both are learned during the training phase.

Gated Recurrent Units Gated Recurrent Units (GRU) were proposed by Cho et al. [9] as a simpler alternative to the LSTM architecture. In comparison to LSTMs, GRUs do not keep a separate memory cell and instead merge the cell state C_t and hidden state h_t. Furthermore, a GRU combines the input gate i_t and the forget gate f_t into a single *update gate*. While the LSTMs and GRUs are identical in the class of functions that they can learn, GRUs are simpler in the sense that they have fewer model parameters. Empirically, GRUs have been found to outperform LSTMs on several sequence prediction tasks [10].

5 Experimental Setup

In process mining, *generalization*, is commonly described as "the likelihood that the process model is able to describe yet unseen behavior of the observed system" [7]. Several measures have been proposed to measure the generalization of a process model [6,12], all of which calculate the generalization of the process model *with respect to the same event log from which the process model was discovered*. In contrast, in the sequence modeling field it is common to measure generalization by splitting the data into a separate *training* set that is used to learn the model and a *test* set on which it is evaluated how well the model fits this data. Because the test set is a separate data set from the training set, the fit between model and test set can be considered to measure the *generalization* of the model to the test data. Fig. 1 describes the experimental setup on a high level.

We apply this experimental setup to a collection of event logs and for a collection of discovery/sequence modeling techniques. For each combination of modeling method and event log we make a random 70/30%-split of the log into training log and test log and after generating the model on the training log we evaluate how well the actual next activity predicted for each prefix in the test log fits the probability distribution over all possible next activities according to the model. For each combination of event log and modeling technique we repeat the experiment three times and calculate the 95% confidence interval around the model performance, to prevent that the results are too dependent on the random sampling of the event log into train and test split. We calculate the Brier score [5] between the probability distribution over the next activity for given a prefix and the actual next activity to express the quality of the prediction. The Brier score is a well-known measure to evaluate a probabilistic classifier and it can intuitively be interpreted as being the mean squared error of the predicted likelihood of a given class (i.e. activity), averaged over all activities and over all prefixes in the test set.

Modeling Methods As modeling techniques we apply the RNN, LSTM, GRU neural networks that are described in Section 4.2. For each of those types of neural networks we explore its performance using only one layer as well as using three layers. We learn the weights of the model with Adam [19]. Furthermore, we first order as well as second order Markov models, i.e. probabilistic models where the state of the model depends on the last k events, with k the order of the Markov model. The predicted probability distribution over the next activity is static for each state. We explore higher order Markov models by including a sequence model called *all k-order Markov models* (AKOM) [30] that fits all Markov models of any order k to the training set. When making an prediction (on the test set), AKOM uses the Markov model with the highest k that has a state that matches the test sequence.

We evaluate the following process discovery techniques: Inductive Miner (IM) [22], the Inductive Miner with infrequency filtering (IMf) [23] (using different thresholds), the Heuristics Miner [37], the Split Miner [3], the Hybrid-ILP Miner [38], and the Evolutionary Tree Miner (ETMd) [7]. Unless parameter values of these process discovery techniques are stated explicitly they are left to their default values. We add the flower model, i.e. the model that allows for all behavior over the set of activities, for comparison. Note that the Brier score of the flower model with a uniform distribution represents random guessing.

Event Logs We evaluate the generalizing capabilities of process discovery techniques and sequence modeling techniques on three real-life event logs. The first log is the receipt phase log from the WABO project[3], containing 8577 events of 27 activities originating from 1434 cases of the receipt phase of the building permit application process at a Dutch municipality. The second log contains cases from a financial loan application process at a large financial institute[4], consisting

[3] `https://doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`
[4] `https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`

Table 1: Experimental Results showing the 95% confidence interval lower bound (LB), upper bound (UB) and the mean of the Brier score.

Method	Receipt Phase			BPI'12			SEPSIS		
	LB	UB	μ	LB	UB	μ	LB	UB	μ
Process Discovery: Uniform distribution per marking									
Flower	0.0371	0.0391	0.0381	0.0417	0.0417	0.0417	0.0675	0.0846	0.0761
IM [22]	0.0321	0.0356	0.0338	0.0301	0.0326	0.0314	0.0559	0.0665	0.0612
IMf 20% [23]	0.0209	0.0239	0.0223	0.0292	0.0295	0.0293	0.0475	0.0497	0.0486
IMf 50% [23]	0.0155	0.0228	0.0191	0.0292	0.0295	0.0293	0.0673	0.0844	0.0759
HM [37]	0.0237	0.0252	0.0245	0.0267	0.0272	0.0269	0.0437	0.0448	0.0442
SM [3]	0.0240	0.0285	0.0262	0.0250	0.0255	0.0252	0.0548	0.0600	0.0574
Hybrid-ILP [38]	0.0155	0.0179	0.0167	0.0227	0.0300	0.0263	0.0413	0.0439	0.0426
ETMd [7]	0.0168	0.0224	0.0196	0.0259	0.0315	0.0287	0.0513	0.0539	0.0526
Process Discovery: Trained distribution per marking									
Flower	0.0327	0.0345	0.0336	0.0402	0.0402	0.0402	0.0540	0.0543	0.0542
IM [22]	0.0228	0.0282	0.0255	0.0276	0.0297	0.0287	0.0420	0.0490	0.0455
IMf 20% [23]	0.0209	0.0239	0.0224	0.0228	0.0255	0.0241	0.0381	0.0409	0.0395
IMf 50% [23]	0.0144	0.0162	0.0153	0.0228	0.0255	0.0241	0.0532	0.0535	0.0534
HM [37]	0.0174	0.0187	0.0181	0.0228	0.0234	0.0231	0.0359	0.0385	0.0372
SM [3]	0.0092	0.0105	0.0099	0.0225	0.0227	0.0226	0.0387	0.0439	0.0413
Hybrid-ILP [38]	0.0155	0.0179	0.0167	0.0286	0.0404	0.0345	0.0399	0.0425	0.0412
ETMd [7]	0.0104	0.0124	0.0114	0.0254	0.0272	0.0263	0.0389	0.0402	0.0396
Neural Network Methods									
RNN (1 layer)	0.0065	0.0080	0.0073	0.0124	0.0127	0.0126	0.0277	0.0282	0.0279
RNN (3 layers)	0.0071	0.0079	0.0075	0.0125	0.0131	0.0128	0.0261	0.0286	0.0273
LSTM (1 layer)	0.0070	0.0075	0.0073	0.0120	0.0123	0.0122	0.0272	0.0273	0.0273
LSTM (3 layers)	0.0066	0.0079	0.0072	0.0123	0.0132	0.0128	0.0283	0.0323	0.0303
GRU (1 layer)	0.0070	0.0081	0.0076	0.0124	0.0126	0.0125	0.0269	0.0280	0.0274
GRU (3 layers)	0.0069	0.0077	0.0073	0.0126	0.0138	0.0132	0.0272	0.0281	0.0276
Markov Model Methods									
1st-order Markov Model	0.0223	0.0245	0.0235	0.0317	0.0318	0.0318	0.0488	0.0493	0.0491
2nd-order Markov Model	0.0216	0.0235	0.0225	0.0256	0.0258	0.0257	0.0468	0.0472	0.0470
AKOM [30]	0.0177	0.0192	0.0185	0.0241	0.0243	0.0242	0.0427	0.0432	0.0429

of 164506 events divided over 13087 cases and 23 activities. As the third log we use the sepsis event log [26], containing medical care pathways of 1050 sepsis patients, for which in total 15214 events were logged from 16 different activities.

6 Results

Tab. 1 shows the results in terms of Brier score on the three datasets for each of the techniques. The worst Brier score value of each column in the table is colored red and the best value is colored green, with the other values taking a color in between. For the process discovery methods it seems that learning the probability distribution on the training data leads to better fitting probability distributions on the test data compared to assuming this probability distribution to be uniform. Note, however, that when using the discovered process model for stakeholder communication about the process there are typically no branching probabilities shown in the model. Therefore, one could say that the uniform distribution matches the graphical representation of the Petri net. The Split Miner [3] is the best performing process discovery technique on two of the three logs when we learn the probability distribution per marking from the training data. The Hybrid-ILP Miner [38] is the best performing process discovery technique on two of the three logs when we consider a uniform distribution per marking, indicating that this miner finds and the most informative states (i.e., markings) and communicates the behavior of the process most effectively to the process analyst.

On all three event logs the Brier scores for the neural network models are considerably better, meaning that they provide considerably more accurate probability distributions over the next event for prefixes from previously unseen traces. These models are black-box and therefore not suitable for communicating the process, however, they seem to be much better generalizing the behavior of the process. We found that it depends per log which neural network architecture performs the best, although the differences between their Brier scores are only minor. Comparing process discovery techniques to Markov models, we find that discovered process models are better at generalizing the behavior when we learn a probability distribution per marking. This holds for 1st-order and 2nd-order Markov models as well as AKOM [30].

7 Conclusions & Future Work

In this paper we have introduced two ways in which (discovered) Petri nets can be used as probabilistic classifiers to predict the next activity for a given prefix of a trace: a uniform distribution approach, which uses only the information that is visually communicated by the graphical representation of the Petri net, and an empirical distribution approach that optimizes a categorical probability distribution per marking using a training log. We have compared how well process models discovered with process discovery techniques are able to generalize by producing accurate probability distributions for a test set of prefixes that were not used to obtain the model. On three real-life business process event logs we have compared process discovery techniques with existing (non-interpretable) sequence modeling techniques from the Machine Learning domain, and have found that machine learning models possess better generalizing properties than discovered process models. This shows that machine learning sequence modeling might be a better choice than process discovery methods to model a business process when interpretability of the model is not a requirement. In future work, we would like to extend our analysis to a larger collection of event logs, process discovery techniques, and sequence modeling methods.

References

1. van der Aalst, W.M.P.: Process mining: data science in action. Springer (2016)
2. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Eindhoven University of Technology (2014)

3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M.: Split miner: Discovering accurate and simple business process models from event logs. In: IEEE International Conference on Data Mining. pp. 1–10. IEEE (2017)
4. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Quarterly 40(4), 1009–1034 (2016)
5. Brier, G.W.: Verification of forecasts expressed in terms of probability. Monthly Weather Review 78(1), 1–3 (1950)
6. 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)
7. Buijs, J.C.A.M., Van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: On the Move to Meaningful Internet Systems. pp. 305–322. Springer (2012)
8. Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D.: Completion time and next activity prediction of processes using sequential pattern mining. In: International Conference on Discovery Science. pp. 49–61. Springer (2014)
9. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing. ACL (2014)
10. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS Deep Learning and Representation Learning Workshop (2014)
11. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Transactions on Services Computing (2016)
12. van Dongen, B.F., Carmona, J., Chatain, T.: A unified approach for measuring precision and generalization based on anti-alignments. In: International Conference on Business Process Management. pp. 39–56. Springer (2016)
13. 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: International Conference on Application and Theory of Petri Nets. pp. 444–454. Springer (2005)
14. Dunning, T.: Statistical identification of language. Computing Research Laboratory, New Mexico State University (1994)
15. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems 100, 129–140 (2017)
16. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. Journal of Machine Learning Research 10, 1305–1340 (2009)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
18. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences 79(8), 2554–2558 (1982)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference for Learning Representations (2015)
20. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowledge and Information Systems 42(1), 97–126 (2015)

21. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436 (2015)
22. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs-a constructive approach. In: International conference on applications and theory of Petri nets and concurrency. pp. 311–329. Springer (2013)
23. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International Conference on Business Process Management. pp. 66–78. Springer (2013)
24. Logan, B., Chu, S.: Music summarization using key phrases. In: IEEE International Conference on Acoustics, Speech, and Signal Processing. vol. 2, pp. II749–II752. IEEE (2000)
25. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: IEEE Symposium on Computational Intelligence and Data Mining. pp. 192–199. IEEE (2011)
26. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+ EMISA. vol. 1859, pp. 72–80. CEUR-ws.org (2017)
27. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A., Toro, M.: Run-time prediction of business process indicators using evolutionary decision rules. Expert Systems with Applications 87, 1–14 (2017)
28. Mehdiyev, N., Evermann, J., Fettke, P.: A multi-stage deep learning approach for business process event prediction. In: IEEE Conference on Business Informatics. vol. 1, pp. 119–128. IEEE (2017)
29. Pika, A., van der Aalst, W.M.P., Fidge, C.J., ter Hofstede, A.H.M., Wynn, M.T.: Predicting deadline transgressions using event logs. In: International Conference on Business Process Management. pp. 211–216. Springer (2012)
30. Pitkow, J., Pirolli, P.: Mining longest repeating subsequences to predict worldwide web surfing. In: USENIX Symposium on Internet Technologies and Systems. pp. 13–26 (1999)
31. Pravilovic, S., Appice, A., Malerba, D.: Process mining to forecast the future of running cases. In: International Workshop on New Frontiers in Mining Complex Patterns. pp. 67–81. Springer (2013)
32. van der Spoel, S., van Keulen, M., Amrit, C.: Process prediction in noisy data sets: a case study in a dutch hospital. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 60–83. Springer (2012)
33. Stanke, M., Waack, S.: Gene prediction with a hidden Markov model and a new intron submodel. Bioinformatics 19(suppl_2), ii215–ii225 (2003)
34. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: International Conference on Advanced Information Systems Engineering. pp. 477–492. Springer (2017)
35. Teinemaa, I., Dumas, M., Maggi, F.M., Di Francescomarino, C.: Predictive business process monitoring with structured and unstructured data. In: International Conference on Business Process Management. pp. 401–417. Springer (2016)
36. Unuvar, M., Lakshmanan, G.T., Doganata, Y.N.: Leveraging path information to generate predictions for parallel business processes. Knowledge and Information Systems 47(2), 433–461 (2016)
37. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: IEEE Symposium on Computational Intelligence and Data Mining. pp. 310–317. IEEE (2011)
38. van Zelst, S.J., van Dongen, B.F., vander Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing (2017)