Noname manuscript No.
(will be inserted by the editor)

An Interdisciplinary Comparison of Sequence Modeling Methods for Next-Element Prediction

Niek Tax · Irene Teinemaa ·
Sebastiaan J. van Zelst

Received: date / Accepted: date

Abstract Data of sequential nature arise in many application domains in the form of, e.g., textual data, DNA sequences and software execution traces. Different research disciplines have developed methods to learn *sequence models* from such datasets: (i) in the *machine learning* field methods such as (hidden) Markov models and recurrent neural networks have been developed and successfully applied to a wide-range of tasks, (ii) in *process mining* process discovery methods aim to generate human-interpretable descriptive models, and (iii) in the *grammar inference* field the focus is on finding descriptive models in the form of formal grammars. Despite their different focuses, these fields share a common goal: learning a model that accurately captures the sequential behavior in the underlying data. Those sequence models are *generative*, i.e, they are able to predict what elements are likely to occur after a given incomplete sequence. So far, these fields have developed mainly in isolation from each other and no comparison exists. This paper presents an interdisciplinary experimental evaluation that compares sequence modeling methods on the task of *next-element prediction* on four real-life sequence datasets. The results indicate that machine learning methods, which generally do not aim at model interpretability, tend to outperform methods from the process mining and grammar inference fields in terms of accuracy.

Niek Tax
Eindhoven University of Technology
E-mail: n.tax@tue.nl

Irene Teinemaa
University of Tartu
E-mail: irene.teinemaa@ut.ee

Sebastiaan J. van Zelst
Fraunhofer Institute for Applied Information Technology, FIT,
RWTH Aachen University
E-mail: sebastiaan.van.zelst@fit.fraunhofer.de

1 Introduction

A large share of the world's data naturally occurs in sequences. Examples thereof include textual data, e.g., sequences of letters/words, DNA sequences, web browsing behavior, and execution traces of business processes or of software systems. Several different research fields have focused on the development of tools and methods to model and describe such *sequence data*. However, these research fields mostly operate independently from each other, and, with little knowledge transfer between them. Nonetheless, the different methods from the different research fields generally share the same common goal: learning a descriptive model from a dataset of sequences such that the model accurately *generalizes* from the sequences that are present in the dataset. The three research communities that developed sequence modeling methods include *machine learning*, *grammar inference*, and *process mining*.

In the *machine learning* research community, several methods for modeling sequences have been developed in the *sequence modeling* subfield. Well-known examples of such sequence models include n-gram models [36], Markov chains [38], Hidden Markov Models (HMMs) [62] and Recurrent Neural Networks (RNNs) [45]. Sequence modeling methods have been successfully applied in many application domains, including modeling of natural language [36], music sequences [54], and DNA sequences [69]. Typically, the end-goal of these methods is to automate a certain task and therefore the focus is mostly on the accuracy of the models, with little emphasis on producing human-interpretable models. Examples of successfully automated tasks within the aforementioned application domains include automated translation of text documents [27], as well as automatic music generation or transcription [18].

In the field of *grammar inference* [30, 43], it is typically assumed that there exists an unknown formal language that generates the sequential dataset. As such, the sequences in the dataset are considered *positive examples*, on the basis of which the resulting formal language is to be inferred. The learned language is represented as an automaton (in case the language is regular) or as a context-free grammar, thereby contrasting with the machine learning field by creating human-interpretable models.

The *process mining* [1] field, originating from the research domain of *business process management*, is concerned with finding an accurate description of a business process from a dataset of logged execution sequences, captured during the execution of the process. The result of a process mining algorithm is usually a process model, i.e., a model in a graphical form with corresponding formal execution semantics. Some of the process model notations that are generated by process mining methods are heavily used in industry, e.g., BPMN [58]. As such, in the process mining field, there is a strong emphasis on the human-interpretability of the discovered models. Another feature that differentiates process mining from both the machine learning and the grammar inference methods is its focus on explicitly modeling concurrent behavior.

There have been efforts to systematically compare and benchmark the accuracy of different methods within each of these research fields independently.

In the machine learning field, this often occurs through task-specific benchmark datasets, such as the popular WMT'14 dataset for machine translation [17]. In grammar inference, this often happens through competitions with standardized evaluations [13, 29, 49]. In the process mining field, the work of [11] compares 35 process discovery algorithms on a collection of datasets. Additionally, competitions with standardized evaluations have also emerged in the process mining field as well [25].

While there have been many efforts to compare sequence modeling methods *within* each research field, to the best of our knowledge, there has been little to no work in the evaluating the accuracy of sequence models *between* the different research fields. In order to enable a comparison between sequence models from the different research fields, we focus on a single task to which such models can be applied: predicting the next element of an unfinished/incomplete sequence. More specifically, we focus on generating the whole probability distribution over possible next elements of the sequence, instead of just predicting the single most likely next element. Machine learning approaches are generally already capable of generating the whole probability distribution over possible next elements and the same holds for a subset of grammar inference methods, called *probabilistic grammar inference*. In earlier work [73], we presented a method to use a process model as a *probabilistic sequence classifier*, thereby enabling the comparison with other probabilistic sequence models. Furthermore, [73] presented a preliminary comparison between machine learning and process discovery methods, i.e., excluding grammar inference.

This paper extends the work started in [73] in several ways. First, we have expanded the scope of the paper to include a new research community that was not yet represented in the initial study: the field of *grammar inference*. Secondly, we expanded our experimental setup by experimenting on a larger number of datasets and covering a larger set of methods, i.e., additionally covering *hidden Markov models* [62] and Active LeZi [40] in the machine learning category and in the process mining category adding the Indulpet Miner [50] process discovery algorithm as well as a class of automaton-based prediction methods. Finally, in an attempt to bring the three research communities together and to make this article useful for researchers from the machine learning, process mining and grammar inference domain, we have added considerable detail to the description of process-model-based predictions.

The remainder of this paper is structured as follows. In Section 2, we describe basic concepts and notations that are used throughout the paper. In Section 3, we introduce several existing next-element prediction methods from the machine learning, grammar inference, and process mining domain. In Section 4, we introduce how process mining methods can be used as a sequence predictor. We describe the experimental setup in Section 5 and discuss the results of the experiments in Section 6. In Section 7, we present an overview of related work. Section 8 discusses threats to validity and limitations. Finally, we conclude this paper and identify several interesting areas of future work in Section 9.

2 Preliminaries

In this section, we introduce preliminary concepts used in later sections of this paper. We cover the fundamental basis of sequential data and formalize the notion of sequence databases. Furthermore, we briefly introduce Petri nets, which play an important role in the field of process mining.

2.1 Sequences and Multisets

Sequences relate positions to elements, i.e., they allow us to order elements. A sequence of length n over a set X is a function $\sigma\colon \{1,...,n\}\to X$ and is alternatively written as $\sigma=\langle\sigma(1),\sigma(2),\dots,\sigma(n)\rangle$. X^* denotes the set of all possible finite sequences over a set X. Given a sequence $\sigma\in X^*$, $|\sigma|$ denotes its length, e.g., $|\langle x,y,z\rangle|=3$. We represent the empty sequence by ϵ, i.e., $|\epsilon|=0$, and, $\sigma_1\cdot\sigma_2$ denotes the concatenation of sequences σ_1 and σ_2. $hd^k(\sigma)=\langle\sigma(1),\sigma(2),\dots,\sigma(k)\rangle$ is the prefix (or head) of length k of sequence σ (with $0<k<|\sigma|$), e.g., $hd^2(\langle a,b,c,d,e\rangle)=\langle a,b\rangle$. Similarly, $tl^k(\sigma)=\langle\sigma(|\sigma|-k+1),...,\sigma(|\sigma|-1),\sigma(|\sigma|)\rangle$ is the postfix (or tail) of length k (with $0<k<|\sigma|$). Given $\sigma\in X^*$ and $1\leq i\leq j\leq|\sigma|$, we let $\sigma_{(i,j)}=\langle\sigma(i),...,\sigma(j)\rangle$. We say that one sequence σ is a prefix of another sequence σ' if and only if $hd^{|\sigma|}(\sigma')=\sigma$.

Given a function $f\colon X\to Y$ and a sequence $\sigma=\langle\sigma(1),...,\sigma(n)\rangle\in X^*$, we lift function application to sequences, i.e., $f(\sigma)\in Y^*$, where $f(\sigma)=\langle f(\sigma(1)),...,f(\sigma(n))\rangle$. Furthermore, given $\sigma\in X^*$ and $X'\subseteq X$, we define $\sigma_{\downarrow_{X'}}\in X'^*$, with (1) $\epsilon_{\downarrow_{X'}}=\epsilon$ and (2) for any $\sigma\in X^*$ and $x\in X$:

$$(\sigma\cdot\langle x\rangle)_{\downarrow_{X'}}=\begin{cases}\sigma\cdot\langle x\rangle & \text{if } x\in X',\\ \sigma & \text{otherwise.}\end{cases}$$

A multiset (or bag) over X is a function $B\colon X\to\mathbb{N}$ which we write as $[x_1^{w_1},x_2^{w_2},\dots,x_n^{w_n}]$, where for $1\leq i\leq n$ we have $x_i\in X$ and $w_i\in\mathbb{N}^+$. Here, w_i represents the value of B for x_i, i.e., $B(x_i)=w_i$. In case $w_i=0$, we omit x_i from multiset notation, and, in case $w_i=1$, we simply write x_i, i.e., without w_i as its superscript. For example, for multiset $B_1=[a,c^2]$ over set $X=\{a,b,c\}$, we have $B_1(a)=1$, $B_1(b)=0$ and $B_1(c)=2$. The empty multiset is written as $[\,]$. The set of all multisets over X is denoted $\mathcal{B}(X)$. Given $B\in\mathcal{B}(X)$, we let $\tilde{B}=\{x\in X\mid B(x)>0\}$.

Finally, given $\sigma\in X^*$, we let $\overline{\sigma}=\{x\in X\mid\exists i\in\{1,...,|\sigma|\}(\sigma(i)=x)\}$ and $\overrightarrow{\sigma}=\left[x^k\in X\mid k=|\{i\in\{1,...,|\sigma|\}\mid\sigma(i)=x\}|\right]$ (also known as the Parikh representation of the sequence), e.g., $\overline{\langle a,b,b,c\rangle}=\{a,b,c\}$ and $\overrightarrow{\langle a,b,b,c\rangle}=[a,b^2,c]$.

2.2 Sequence Databases

As indicated, in this paper we study next-element prediction on the basis of *sequential data*. As an example introduction to this type of data, we present

Table 1: A fictional example sequence database, adopted from [1], describing behavior related to a compensation request process for concert tickets.

Case id	*Event id*	*Timestamp*	*Activity*	*Resource*	*Cost*	⋯
⋮	⋮	⋮	⋮	⋮	⋮	...
1337	745632	30-7-2018 11.02	register request (a)	Barbara	50	⋯
1338	745633	30-7-2018 11.32	register request (a)	Jan	50	⋯
1337	745634	30-7-2018 12.12	check ticket (d)	Stefanie	100	⋯
1338	745635	30-7-2018 14.16	examine casually (c)	Jorge	400	⋯
1339	745636	30-7-2018 14.32	register request (a)	Josep	50	⋯
1339	745637	30-7-2018 15.42	examine thoroughly (b)	Marlon	600	⋯
1337	745638	3-8-2018 11.18	examine thoroughly (b)	Barbara	600	⋯
1337	745639	3-8-2018 15.34	decide (e)	Wil	700	⋯
1338	745640	3-8-2018 15.50	check ticket	Marcello	100	⋯
1337	745641	3-8-2018 16.50	reject request (h)	Arthur	25	⋯
1340	745642	3-8-2018 16.58	register request (a)	Hajo	50	⋯
1338	745643	3-8-2018 16.59	pay compensation (g)	Boudewijn	75	⋯
⋮	⋮	⋮	⋮	⋮	⋮	...

fictional data, which is assumed to be generated and captured during the execution of a *(business) process*. Consider Tab. 1, adopted from [1], in which we depict example data related to the process of handling a compensation request for concert tickets. Each row in the table corresponds to a single recorded data point, in this case representing the execution of an activity, in the context of an instance of the process. We are able to relate the different data points by means of the *Case id* column, which allows us to identify the underlying process instance. Observe that, the data elements describe several data attributes, e.g., the timestamp of the activity, the resource that executed the activity, etc., illustrating the practical relevance of this type of data. However, for the purpose of this paper, we primarily focus on sequences of data items that we are able to represent as a single symbol. For example, when using the short-hand activity notation of the activities listed in Tab. 1, we obtain the sequence $\langle a, d, b, e, h\rangle$, for the process instance with case-id 1337.[1]

In the remainder, we let Σ to denote the *set of symbols*. In the context of this paper, a *sequence database* (often called *event log* in process mining) is defined as a finite multiset of sequences, $L{\in}\mathcal{B}(\Sigma^*)$. A *word* is a sequence $\sigma{\in}L$, i.e., a sequence of symbols in a sequence database. For example, the sequence database $L{=}[\langle a, b, c\rangle^2, \langle b, a, c\rangle^3]$ consists of two occurrences of the word $\langle a, b, c\rangle$ and three occurrences of word $\langle b, a, c\rangle$.

[1] Note that, in the general sense, we are always able to map complex alpha-numerical string data into single characters.

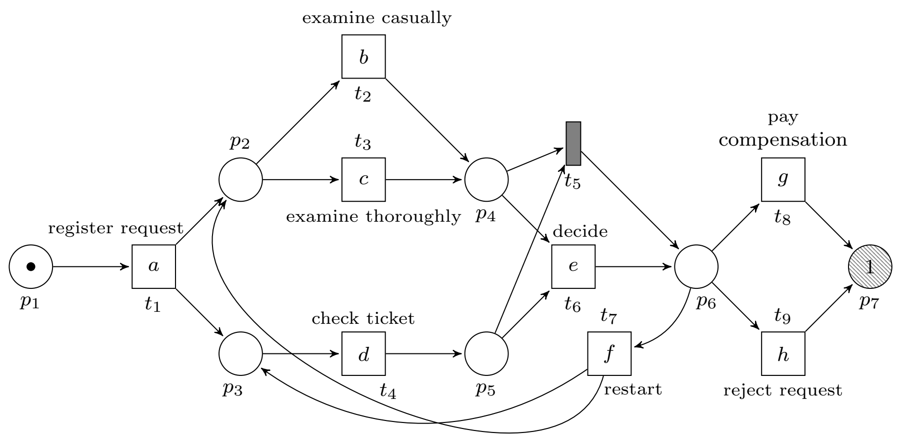

Fig. 1: An example accepting Petri net APN, adopted from [1]. The initial marking of the APN is $[p_1]$, the final marking is $[p_7]$

2.3 Petri nets

Petri nets are a commonly used process modeling formalism to represent processes in process mining. They allow for explicit modeling of concurrent (i.e., parallel) behavior in a relatively compact manner. Additionally, most high-level, business-oriented, process modeling notations, e.g., BPMN [58], are often translatable into Petri nets.

A Petri net is a directed bipartite graph consisting of places (depicted as circles) and transitions (depicted as rectangles), connected by arcs. The transitions allow us to describe the possible activities/symbols of the process, whereas the places represent the enabling conditions of transitions. The label of a transition indicates the symbol that the transition represents. Unlabelled transitions (τ-transitions) represent invisible transitions (depicted as grey rectangles), which are used for routing purposes and are unobservable. As an example of a Petri net, consider Fig. 1, which contains 7 places and 9 transitions. The symbol corresponding to transition t_1 is symbol a, whereas transition t_5 is unobservable.

The state of a Petri net is defined by its *marking* $m \in \mathcal{B}(P)$, i.e., a multiset of places. A marking is graphically denoted by drawing $m(p)$ tokens in each place $p \in P$. For example, consider the marking of the example Petri net in Fig. 1, 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 of transition t contains at least one token. Once t fires, one token is removed from each input place of t and one token is added to each output place of t, leading to a new marking. Firing a transition t in marking m, yielding marking m', is denoted as step $m \xrightarrow{t} m'$. For example, in Fig. 1, in marking $[p_1]$, the only enabled transition is t_1, with label a. If we fire t_1, we obtain a new marking, i.e., $[p_2, p_3]$. In the remainder of this paper, we consider *Accepting Petri nets*, i.e., Petri nets with a designated *initial* and *final* marking. In the example Petri net of Fig. 1, these markings are $[p_1]$ and $[p_7]$ respectively.

The markings of a Petri net allow us to describe sequences of transition firings. For example, starting from marking $[p_1]$, the sequence of transitions $\langle t_1, t_2, t_4 \rangle$ results in marking $[p_4, p_5]$. Observe that the sequence $\langle t_1, t_4, t_3 \rangle$ (and many other sequences) also result in marking $[p_4, p_5]$. This is due to the fact that we are able to fire t_4 and t_2/t_3 (either one of the two but not both) independently. Furthermore, due to the *loop structure* in the model, i.e., t_7 we can again reach marking $[p_2, p_3]$ (from $[p_6]$). We are able to project each sequence of transitions on the labels of the corresponding transitions, e.g., $\langle t_1, t_4 \rangle$ translates to $\langle a, d \rangle$. Moreover, in the corresponding marking, i.e., $[p_2, p_5]$, only transitions t_2 and t_3 are enabled. Observe that, firing t_5 is not observable, i.e., sequence $\langle t_1, t_2, t_4, t_5 \rangle$, translates to $\langle a, b, d \rangle$, yielding marking $[p_6]$.

3 Next Element Prediction Methods

In this section, we present several methods to predict the probability distribution over the next element following a given incomplete sequence. These methods originate from different research fields. We start by introducing several sequence models from the *machine learning domain*: neural networks in Section 3.1.1, Markov models in Section 3.1.2, and Active LeZi in Section 3.1.3. We continue this section by introducing *grammar inference* in Section 3.2. In Section 3.3.1 we introduce some automaton-based methods for next-element prediction that originate from the *process mining* domain. Another class of methods from the process mining domain is presented in Section 3.3.2, where we discuss so-called *process discovery* methods, which address the task of inferring a Petri net from a sequence database.[2] We conclude this section on sequence modeling methods in Section 3.4, where we discuss the similarities and differences between the methods presented in this section.

3.1 Machine Learning

In this section, we present methods for next-element prediction, originating from the machine learning domain. We cover neural networks, Markov models, and Active LeZi.

3.1.1 Neural Networks & Recurrent Neural Networks

A neural network consists of one layer of *input units*, one layer of output units, and in-between are one or more layers that are referred to as *hidden units*.

[2] Note that we postpone methods for actually making predictions with a Petri net to Section 4. This allows us to separate the discussion of existing algorithms (this section) from the part where we introduce novel methods (Section 4)

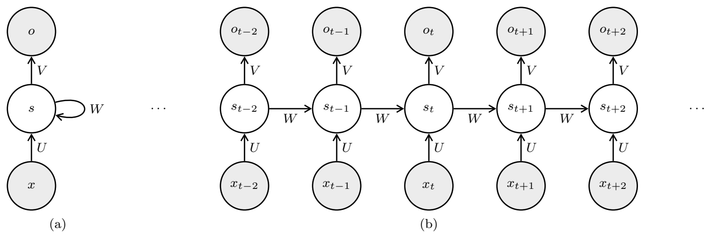

Fig. 2: *(a)* A simple recurrent neural network consisting of a single hidden layer, and *(b)* the recurrent neural network unfolded over time.

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 optimized iteratively by applying the current weights to some training sequences and back-propagating the partial derivative of the weights with respect to the error back through the network and adjusting the weights accordingly. 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. 2. 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 neural network literature, the sigmoid function is often represented by σ, however, to avoid confusion with sequences, we fully write *sigmoid*. o_t is the output at step t.

Long Short-Term Memory A Long Short-Term Memory (LSTM) model [44] 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 is 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 is 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. [27] 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 THE 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 [28].

3.1.2 Markov Models

A Markov model is a stochastic model commonly used in probability theory in order to model randomly changing systems. The assumes the *Markov property*, i.e., the future state of a system only depends on its present state. The simplest type of Markov models are *Markov chains*, which can be used when the states of the system are fully observable. The parameters of a Markov chain consist of a matrix of *transition probabilities* expressing the likelihood of transitioning from any given state to any other state. In a 1st-order Markov chain, the state represents the last observed symbol of the sequence. In Markov chains of higher order, the state represents a longer window of observed symbols, i.e., in a k^{th}-order Markov model, the state represents the last k symbols.

A sequence model called *all k-order Markov models* (AKOM) [60] is an extension of Markov chains that fits all models up to an order k to the training sequences. When performing a prediction, i.e., estimating the transition probability from a given state in a test sequence, AKOM uses the Markov model with the highest k that has a state that matches the test sequence.

Hidden Markov models (HMM) are a type of Markov model where the states represent *latent variables* that do not represent some window over the sequences directly but instead represent some unobserved property that is inferred from the sequence. An HMM assumes that the system can be described in terms of a number of *hidden states* that are not directly observable in the data. Therefore, the next observation in the sequence depends not only on the most recent observation, but also on the likelihood of being in a particular hidden state.

The parameters of an HMM include the transition probabilities, i.e., transitioning from a hidden state to another hidden state, and the *emission probabilities*, expressing the likelihood of a particular observation while being in a given hidden state. The transition probabilities between states and the emission probabilities from states to symbols are learned from a training dataset using the Baum-Welch algorithm. When predicting the next symbol for a sequence with an HMM, one can either 1) apply the well-known Viterbi algorithm to extract the most likely sequence of hidden states for the given sequence and make the prediction according to the emission probabilities of the final hidden state, or 2) apply the forward algorithm to obtain a probability distribution over the likelihoods of the hidden states given the sequence and make the prediction by weighting the emission probabilities of those states by their hidden state likelihood.

3.1.3 Active LeZi

It has been shown in the information theory field [78], that the tasks of prediction and compression are closely related, and that good compression methods are also good prediction methods. The LZ78 algorithm [84] is a widely used dictionary-based text compression algorithm that incrementally parses an input sequence $\langle x_1, x_2, \ldots, x_i \rangle$ by parsing it into a set of subsequences $W{=}\{w_1, w_2, \ldots, w_k\}$ such that for each $w_i \in W$ with $|w_i| > 1$ it holds that $\exists w_j \in W$ such that w_j is a prefix of w_i (the *prefix property*).

As a result of this prefix property, subsequences W and their occurrence counts in the input sequence can be efficiently stored in a trie datastructure. Based on these occurrence counts, the transition probabilities from one state in the trie $w_i \in W$ to all of its possible succeeding states in the trie (all representing subsequences of which w_i is a subsequence) can be calculated.

It can be shown that the trie data structure generated by the LZ78 algorithm is equivalent to a Markov model. Gopalratnam and Cook [40] developed the active LeZi algorithm as an enhancement of LZ78, which addresses the slow convergence of the original LZ78 algorithm, by making use of a sliding window.

3.2 Grammar Inference

The research field of *grammar inference*, also called *grammar induction*, is concerned with learning a grammar that describes a language based on a collection of positive examples of elements from this language. Observe that, the grammar inference field closely links to the area of process discovery, where the dataset of positive examples is called an event log and the language that is learned is represented as a process model instead of a formal grammar. A variety of grammar inference methods exists. *Automaton learning* methods focus on learning a *deterministic finite automaton* (DFA) that describes the language and can be used when the language is assumed to belong to the class of *regular languages*. Other grammar inference methods assume the language to belong to the class of *context-free languages* and focus on extracting a context-free language in extended Backus-Naur form [80]. We refer to [30] and [43] for an extensive overview of the grammar inference field.

The grammar inference field puts special emphasis on formal analysis of the learnability of languages. Early work by Gold [39], who proved that the problem of finding the smallest DFA consistent with a given set of strings is NP-hard, plays a central role in the grammar inference field. Angluin [8] proposed an *active learning* setting of grammar inference where the algorithm does not learn the grammar from a fixed set of positive samples, but instead iteratively queries the world with strings for which it wants to obtain whether or not this string is or is not part of the language. Angluin [8] proved that regular languages can be identified in a polynomial amount of queries using active learning and proposed an algorithm called L* that is able to do so. In this paper, we assume a dataset of positive examples of sequences from a language to be given and therefore we leave active learning algorithms to grammar inference out of scope.

Some grammar inference methods focus on learning probabilistic grammars, i.e., they do not just specify which sequences are included in and which are excluded from the language, but they additionally specify the likelihoods of each word of the language.

Several competitions have been organized in the grammar inference field that aimed at benchmarking grammar inference methods and tools. The Abbadingo challenge (1998) [49] focused on learning DFAs, Omphalos (2004) [29] focused on learning context-free grammars, and PAutomaC (2014) [76] on learning probabilistic finite state machines. The $\underline{\text{S}}$equence $\underline{\text{P}}$red$\underline{\text{I}}$ction $\underline{\text{C}}$halleng$\underline{\text{E}}$ (SPiCe) [13] was a recent challenge from the grammar inference field that defined the challenge task similar to the focus of this paper: predicting the next symbol in a sequence. The SPiCE competition used a well-known probabilistic automaton learning algorithm from the grammar inference field as a baseline method, which is called *spectral learning* [12].

A central concept in the spectral learning approach to grammar inference is the so-called Hankel matrix, which is a bi-infinite matrix in which the rows correspond to the prefixes of the sequences in the dataset and columns its suffixes. The value in a cell of the Hankel matrix represents the weight of the corresponding sequence in the corresponding weighted automaton. The rank of this matrix is the number of states in the minimal weighted automaton. Balle et al. [12] showed that in this way the weighted automaton can be constructed through a rank factorization of the Hankel matrix. Spectral learning relies on constructing a finite sub-block approximation of the Hankel matrix and using a Singular Value Decomposition on the resulting matrix to obtain a rank factorization, and thus, a weighted automaton.

3.3 Process Mining

In this section we discuss next-element prediction on the basis of process mining methods and/or commonly used sequence models in the process mining domain. We cover automata-based prediction and Petri net based prediction.

3.3.1 Automaton Based Prediction

Automata are a well defined mathematical model for describing discrete sequential data, i.e., an automaton allows us to describe the possible states of a system, as well as the ways in which the system is able to change its state. In the context of next-element prediction, we are specifically interested in probabilistic automata, which not only allow us to inspect a state of the system and its possible actions, but also to quantify the probabilities of these actions.

When constructing a probabilistic automaton for the purpose of next element prediction, we perform two steps, i.e., 1) automaton construction and 2) transition probability computation. In automaton construction, each word is transformed into a sequence of automaton transitions, which we represent by tuples of the form $(\texttt{from state}, \texttt{event label}, \texttt{to state})$.

In process mining, the automata are commonly constructed based on *abstraction functions* [2]. In particular, we first extract subsequences of at most size k from the original sequences. Then, we either keep the subsequences as-is (i.e. use the sequence abstraction), or project them onto their set or multiset representation. For example, consider applying the different abstractions, written as $\pi^k_{\texttt{seq}}$, $\pi^k_{\texttt{set}}$, and $\pi^k_{\texttt{mul}}$, respectively, on the word $\langle a, b, b, c\rangle$, for $k = 2$:

- $\pi^2_{\texttt{seq}}(\langle a, b, b, c\rangle) = \langle(\epsilon, a, \langle a\rangle), (\langle a\rangle, b, \langle a, b\rangle), (\langle a, b\rangle, b, \langle b, b\rangle), (\langle b, b\rangle, c, \langle b, c\rangle)\rangle$
- $\pi^2_{\texttt{set}}(\langle a, b, b, c\rangle) = \langle(\emptyset, a, \{a\}), (\{a\}, b, \{a, b\}), (\{a, b\}, b, \{b\}), (\{b\}, c, \{b, c\})\rangle$
- $\pi^2_{\texttt{mul}}(\langle a, b, b, c\rangle) = \big\langle([\,], a, [a]), ([a], b, [a, b]), \big([a, b], b, [b^2]\big), \big([b^2], c, [b, c]\big)\big\rangle$

Hence, given an abstraction of choice, i.e., either *set*, *multiset*, or *sequence*, and a value for k, we are able to transform a sequence database into a multiset of abstraction sequences. Translating such a multiset to an automaton is trivial, i.e., each first and third element of the tuples present in the different abstraction sequences, as defined by the sequence database, represents a state. The second element of each tuple present in the different abstraction sequences represents a transition.

As the final step, we learn the transition probabilities. We simply count the number of times a certain abstraction (written as (q, a, q')) occurs in all the calculated abstraction sequences of the sequence database. To determine the probability of label a occurring in a state q, written as $P(a \mid q)$, we sum up all the values counted for (q, a, q') (for all possible values for q'). We divide this number over the sum of all the counted values (q, a', q') (for all possible values for $a' \neq a$ and q').

Using such an automaton for the purpose of next-element prediction is straightforward. We apply the chosen abstraction function to the given incomplete sequence, determine the corresponding state in the automaton, and output the outgoing event distribution of that state.

3.3.2 Process Discovery: Discovering Petri Nets from Sequence Databases

Several approaches have been introduced in the process mining field to algorithmically extract a process model from a sequence database, i.e., commonly referred to as *process discovery algorithms* (See [1] and [11] for an overview). Here, we introduce the key concepts and ideas behind the process discovery algorithms that we consider in the experiments conducted for this paper.

In contrast to machine learning methods, the process models generated by process discovery algorithms are often not inherently probabilistic, but merely model what behavior is allowed and what behavior is not allowed in a discrete manner by identifying states (e.g. *reachable markings* in the case of Petri nets) in the process. The main question here, is whether these states accurately describe which activities are possible at which point in the process. From a process mining point-of-view, these states should allow for the possible next activities that are likely to follow according to the data (referred to as *fitness*), but not many more than those (referred to as *precision*).

A precise process model disallows activities that are never observed as next activity from a given state anywhere in the data, which from a probabilistic point-of-view means assigning a zero probability to these activities from such a state. A fitting process model allows activities that are observed as a next activity from a given state somewhere in the data, which from a probabilistic point-of-view means assigning a nonzero probability to these activities from such a state. Therefore, a fitting and precise process model is better w.r.t. another process model that is either non-fitting or non-precise in terms of probabilistic machine learning measures as well, simply by assigning zero probability to the activities that actually have zero probability and by assigning non-zero probability to the activities that should have non-zero probability.

Petri nets, with the exception of specific extensions, such as Generalized Stochastic Petri nets (GSPNs), do not assign a probability to the activities that are allowed by the model from a given state. However, using the same data from which the process model was discovered by the process discovery algorithm we are estimate a probability distribution for each of these states, therefore yielding probabilities over the next activities. Note that, this can be estimated using a maximum likelihood estimator. As no common approach is established for this in prior work, in Section 4, we discuss such methods in more detail, i.e., using Petri nets as a sequence predictor.

The Inductive Miner (IM) [51, 52] is a process discovery algorithm that in a first step extracts a so-called *directly-follows graph* from the sequence database. This directed graph consists of vertices that represent symbols from the database and edges that indicate whether two labels directly follow each other in one of the sequences of the dataset. Edges are annotated with frequency information, i.e., the edge weight corresponds to the number of times one symbol directly follows another symbol. This closely links the directly-follows graph to a 1st-order Markov chain. In a second step, so-called cuts are detected by detecting groups of symbols such that all their connecting edges have the same direction. Based on these cuts, a process model in a tree-based process model notation called a *process tree* [23] is extracted from the directly-follows graph. Translation of a process tree to a Petri net is straightforward. Moreover, the obtained Petri net is deadlock and livelock free.

The Inductive Miner infrequent (IMf) [52] is a variant of the IM algorithm that is designed to be able to deal with noisy sequence databases. The IMf algorithm follows the same cut detection procedure as the IM algorithm, but it first filters the directly-follows graph by removing the edges of which the corresponding frequency is less than a certain threshold ratio of the number of sequences, where this threshold is referred to as the *noise threshold*.

The Heuristics Miner (HM) [77] defines a set of heuristics to deduce sequential relations, loop relations, long-term relations, and concurrency relations between symbols in the sequence database. These heuristics are defined in terms of the frequency of certain patterns in the data. The extracted set of sequential, loop, long-term, and concurrency relations are transformed into a process model notation that is called a *heuristics net*. A heuristics net can be transformed into a Petri net, however the resulting Petri net potentially contains some behavioral problems: it may contain deadlocks as well as improper completion.

The Split Miner (SM) [10] is similar to the HM algorithm in the sense that it defines a set of heuristics to extract a set of relations between symbols. The name of the algorithm originates from the fact that the extracted relations are used to create a process model in BPMN [58] notation where the extracted relations are used to determine where in the model the AND-splits (concurrency) and the XOR splits (exclusive choice) should be positioned. The authors show that it is a difficult problem to determine the types of joins corresponding to these splits. Therefore, as a pragmatic solution, the OR-join is used to join all types of splits. This yields a valid BPMN model, but may lead to a model with improper completion when the resulting BPMN models are transformed to Petri nets.

ILP-based process discovery [79, 83] works by translating the sequence database into a prefix-closure, i.e., a set containing all sequences from the database as well as all their prefixes. The prefix-closure forms the basic set of constraints of an Integer Linear Program (ILP). The body of constraints makes sure that each solution of such an ILP, corresponds to a place in the resulting Petri net, that allows for all the behaviour observed in the sequence database. In this way, the process discovery problem is turned into a mathematical optimization problem to find the minimal set of places that are needed to constrain the behavior of the process model to the behavior that was observed in the sequence database. Several variations of the basic scheme exist, that allow us to pose a variety of formal properties w.r.t. the discovered models.

The Evolutionary Tree Miner (ETM) [23, 24] is a process-tree-based process discovery algorithm, like the IM algorithm. The ETM uses an multi-

Table 2: An overview of several tasks related to sequence models.

Sequence model	Model structure	State-to-output (training time)	Prefix-to-state (prediction time)
RNN	Hyper-parameter opt.	SGD + backpropagation	Forward pass
HMM	Hyper-parameter opt.	BaumWelch algorithm	Viterbi algorithm
Process model	Process discovery	Section 3.3.2	Prefix-alignments

criteria evolutionary algorithm to optimize towards a process model that scores well on a set of quality criteria. These quality criteria, amongst others include *fitness* (recall) and *precision* of the discovered models.

The Indulpet miner [53] is another process-tree-based process discovery algorithm that aims to address the shortcomings of the IM algorithm by combining several existing process tree mining algorithms. The Indulpet miner starts with applying the IM algorithm. For local parts of the process where the IM algorithm fails to deduce any precise process fragment, a novel bottom-up recursion approach is applied, but only for the part of the parts of the dataset that were not already described by the IM. A pattern mining algorithm called the *local process model* (LPM) miner [70, 72] is applied to those parts of the dataset that are still not described in a precise manner after the bottom-up procedure. The LPM miner mines frequent patterns of behavior that are expressed in the form of process trees. The ETM algorithm is then used, seeded with the mined set of frequent LPM patterns, to stitch together the patterns into a single model. The process tree notation makes it easy to combine the initial IM model with the models that are created in the later stages: the later models simply become a subtree of the model of the previous stage.

3.4 Summary of Sequence Modeling Methods and Discussion

Many of the sequence models that we discussed in the previous sections conceptually consist of several common sub-procedures: 1) *determining the structure* of the model, thereby creating a set of model states, 2) *optimizing weights* or *parameters* that determine how these model states map to prediction outputs, and 3) when making prediction for a prefix, map the *prefix to a state* and make the prediction using that state. Different sequence models often differ in the exact algorithms that are used to execute these three sub-procedures. Tab. 2 gives an overview of the algorithms for these three sub-procedures for several sequence models.

The model structure in the case of RNNs consists of the number of layers, the number of units per layer, and the architecture of the network. These elements are often decided through careful hand-tuning or can alternatively be automated using a hyper-parameter optimization method. In an HMM this concerns the number of latent variables, which likewise is often selected through hyper-parameter optimization. In the case of a Petri net, the states of the model are its markings. Therefore, *process discovery* can be seen as an automated approach to determine the model structure for a process-model-based predictor. Process discovery can, therefore, be seen as analogous to hyper-parameter optimization in RNNs and HMMs. Like RNNs and HMMs can be hand-tuned instead of automated hyper-parameter optimization, process models can be hand-modeled instead of discovered automatically.

The Baum-Welch algorithm is a well-known algorithm to learn the transition and emission probability parameters of an HMM based on some training data. By determining these parameters, it becomes fixed how a certain state maps to a certain prediction, i.e., according to its emission probabilities. That means that all that is left in order to make a prediction for a given prefix with an HMM is to determine the most likely state for that prefix. As indicated, for process-model-based approaches, no common approach is established to do this, hence, in Section 4, we discuss such methods in more detail.

The Viterbi algorithm is a well-known algorithm to determine the most likely sequence of hidden states in an HMM given a prefix or sequence. The last hidden state of such a sequence of hidden states that the Viterbi algorithm returns for a prefix can, therefore, be seen as the most likely state for that prefix. Predictions can be made with an HMM by determining the most likely state for a prefix using Viterbi and then predicting according to the emission probabilities that were provided for that state by the Baum-Welch algorithm during training. In process models, *(prefix-)alignments* are the common way to determine a sequence of model steps given a prefix or sequence. This creates an analogy between the Viterbi algorithm for HMMs and the alignment algorithm for process models.

It is important to highlight one crucial difference between Viterbi and the previously mentioned alignments: while the Viterbi algorithm provides the *most likely* sequence of hidden states in an HMM given sequence σ, alignments only provide a sequence of steps in a process model that *deviates the least* from σ, i.e., no probability information is used to determine the sequence of steps in an alignment. In order to be truly analogous to Viterbi, alignments would need to return not only the sequence of steps through the process model that deviates the least from σ, but it should additionally need to provide the *most likely* one according to model probabilities when there exist multiple least-deviating sequences of model steps. Several steps have been made towards computing probabilistic alignments [6, 7, 47]. However, the approaches in [6, 7] are heuristic-based and they have been shown to fail to produce the most probable alignment under certain conditions [47]. The probabilistic alignment approach of [47] requires not one optimal alignment but instead requires the computation of all optimal alignments, which is computationally intractable. Furthermore, the probabilistic alignment approaches of [6, 7, 47] have so far not been generalized from alignments to prefix-alignments. Therefore, we resort to non-probabilistic alignments in the experimental comparison in this paper. In the same time, probabilistic prefix-alignments form a relevant research direction in order to develop a true Viterbi-equivalent for process models and to overcome this limitation in the future.

4 Using a Petri net as a Sequence Predictor

In this section, we describe how to use a Petri net as a sequence prediction method. This enables the use of the process discovery algorithms that we have introduced in the previous section as sequence prediction algorithms, by combining them with the methods that we present in this section.

We assume that we have a method f to map a prefix to a marking m. We start with a training phase in which we deduce a probability distribution over symbols Σ based on sequence database L for each marking m of APN that is reached when replaying L (using f). After the training phase, when making a prediction with Petri net APN for a given prefix σ, we again map the prefix to a marking m in APN and predict the next symbol according to the probability distribution that we learned for m. We propose a two-step approach for the training-phase:

1. We **compute the most likely markings** in Petri net APN for all observed prefixes of training sequences L.
2. For each marking reached we **compute a probability distribution** describing the possible next elements $\Sigma \to [0, 1]$.

We now proceed by detailing these two steps.

4.1 Estimating Markings

To deduce what symbols are able to follow a prefix σ, using a Petri net APN as a sequence model, we obtain a marking of the model that corresponds to firing the given prefix σ in the Petri net. A naive approach to this problem is to play the so-called "token-game". In the token-game, starting from the initial marking, we simply fire enabled transitions in such a way that we obtain a firing sequence that projected on its labels equals σ, and marks some arbitrary marking m in APN. Such an approach works, as long as the observed prefix actually allows us to reach a marking m, i.e., the prefix should fit the model. Furthermore, in case the Petri net contains multiple transitions describing the same label, such a strategy becomes more complex and potentially leads to ambiguous results. Therefore, we propose to calculate prefix-alignments [4, 82] as they provide a natural solution to the two aforementioned problems. In essence, a prefix-alignment provides us with the *most likely marking* corresponding to an observed behavioral sequence. However, it is able to identify whether certain activities are duplicated, missing, etc. Moreover, it is able to handle duplicated labels and invisible transitions within computing the most likely marking.

Consider Fig. 3 in which we depict two prefix-alignments of two different prefixes of words, i.e., $\langle a, b \rangle$ and $\langle a, b, e \rangle$ in the context of the example accepting Petri net, depicted in Fig. 1. In each alignment, the top-row represents the observed behavior, whereas the bottom row descirbes a firing sequence of the Petri net. It is easy to see that the word $\langle a, b \rangle$ complies with a prefix as

α_3 :

a	b
t_1	t_2

α_4 :

a	b	$\gg$	e
t_1	t_2	t_4	t_6

Fig. 3: Two prefix-alignments (α_3 and α_4) w.r.t. the accepting Petri net of Fig. 1 for the prefixes $\langle a, b\rangle$ and $\langle a, b, e\rangle$ respectively.

described by the model. Hence, the leftmost prefix-alignment describes that sequence $\langle t_1, t_2\rangle$ is a firing sequence explaining the observed activity sequence $\langle a, b\rangle$. Observe that, indeed, $\ell(t_1){=}a$ and $\ell(t_2){=}b$, and $[p_1] \xrightarrow{\langle t_1,t_2\rangle} [p_3, p_4]$.

For sequence $\langle a, b, e\rangle$ however, we observe that it does not directly comply with a prefix as described by the model. The rightmost prefix-alignment describes that sequence $\langle t_1, t_2, t_4, t_6\rangle$ is a firing sequence that most accurately describes the observed behavioral sequence in terms of the model. In this case, it states that the best possible way to explain the observed behavioral sequence, by means of assuming that label d was not observed, whereas it was supposed to happen (as represented by the $(\gg, t_4)$ cell). Nonetheless, like in the case of prefix $\langle a, b\rangle$, we obtain a valid firing sequence in the model, that yields us with a marking of the model, i.e., in this case $[p_1] \xrightarrow{\langle t_1,t_2,t_4,t_6\rangle} [p_6]$.

4.2 Computing a Probability Distribution

Recall the example alignment depicted in Fig. 3, related to prefix $\langle a, b\rangle$, i.e., the leftmost prefix-alignment. In marking $[p_3, p_4]$ that corresponds to the prefix-alignment we observe that only transition t_4 is enabled, which is labeled d. Therefore, when making a prediction for $\langle a, b\rangle$ on this Petri net we predict next symbol d with probability 1. However, the task becomes non-trivial when making a prediction for a prefix that corresponds to a marking from which more than one transition in enabled. We propose two ways to generate a probability distribution describing the next element for each marking, that deal with the problem of multiple enabled transitions in different ways:

1. ***Model-driven probability distribution generation*** For a marking m, we investigate which transitions are enabled in the model, using a uniform distribution over the enabled transitions. Based on this uniform distribution over the transitions we calculate the corresponding (not necessarily uniform) categorical distribution over the symbols.
2. ***Data-driven probability distribution generation*** For a marking m, we determine (not necessarily uniform) categorical distribution over the enabled transitions. Based on this categorical distribution over the transitions we calculate the corresponding categorical distribution over the symbols.

4.2.1 Model-Driven Probability Distribution Generation.

Consider prefix $\langle a, b, d\rangle$ and the same Petri net, leading to prefix-alignment $\langle (a, t_1), (b, t_2), (d, t_4)\rangle$. Observe that we fetch corresponding marking $[p_4, p_5]$.

We observe two enabled transitions from this marking: t_5 and t_6, Therefore, the uniform distribution over these two transitions describes firing either one of the two with probability $\frac{1}{2}$. However, t_5 is not observable. In fact, after firing t_5, yielding marking $[p_6]$, we observe that transitions t_7, t_8 and t_9 are enabled, all of which do have an observable label. Hence, the true set of symbols that can be observed after prefix $\langle a, b, d \rangle$ with non-zero probability is e, f, g and h. Since we assume the probability distribution over the transitions to be uniform, we observe label e with probability $\frac{1}{2}$, and labels f, g and h, each with probability $\frac{1}{2} \cdot \frac{1}{3}=\frac{1}{6}$ (i.e., probability $\frac{1}{2}$ to fire t_5 from $[p_4, p_5]$ and reach $[p_6]$ and probability $\frac{1}{3}$ for each of the three labels from $[p_6]$). In the previous example, deriving the exact occurrence probabilities of the different labels is easy, however, in general, it is possible to generate longer transition sequences solely consisting of unobservable transitions. In some cases, such sequences can even be of arbitrary length. We therefore resort to Monte Carlo simulation to approximate the corresponding categorical distribution over the symbols Σ.

For a given marking $m \in \mathcal{B}(P)$ in an accepting Petri net APN, $\omega(m) = \{t | t \in T \wedge \bullet t \subseteq m\}$ denotes the set of enabled transitions. We, correspondingly, let probability mass function $prob_m : T \to [0, 1]$ assign a firing probability to each transition that is enabled from marking m, such that $\Sigma_{t \in \omega(m)} prob_m(t) = 1$. We assume this probability distribution over the enabled transitions to be a uniform categorical distribution, i.e., $prob_m^{uniform}(t) = \begin{cases} \frac{1}{|\omega(m)|} & \text{if } t \in \omega(m), \\ 0 & \text{otherwise.} \end{cases}$
We maintain a counter c: $\Sigma \to \mathbb{N}$, with initially $c(a) = 0, \forall a \in \Sigma$. 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 visible label, we count it as the next element, i.e., if $\ell(t) = a$, then $c(a) \leftarrow c(a)+1$. If it relates to an unobservable transition, we fire it, leading to a new marking m' and apply the same procedure, i.e., picking a new enabled transition from $prob_{m'}^{uniform}$, up-until we select a transition that has a visible label. Assume we apply the aforementioned procedure K times (with K the number of Monte Carlo iterations) then the probability of observing a certain label a is equal to $\frac{c(a)}{K}$.

4.2.2 Data-Driven Probability Distribution Generation.

We consider the model-driven probability distribution to be the probability distribution that is visually implied by the process model, i.e., the information that the process model visually suggests to the user. However, from an accuracy point-of-view it might be better to fit the categorical distribution over the enabled transitions for each marking, instead of assuming this distribution to be uniform. Therefore, we present a data-driven approach in which we compute an empirical distribution $prob_m^{empirical}$ after discovering a process model based on the training sequences. This is rather straightforward: for each prefix of each word in the training log we compute a prefix-alignment to obtain

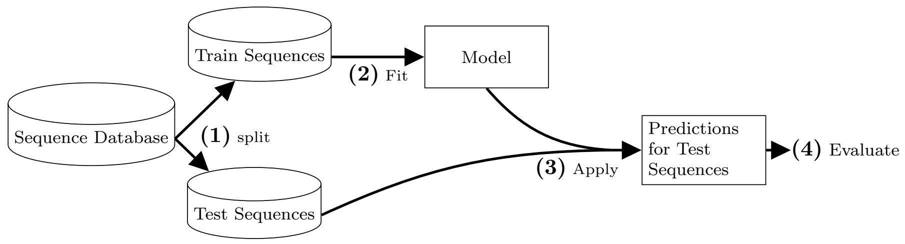

Fig. 4: An overview of the experimental setup.

the corresponding marking m in the discovered Petri net APN. Subsequently, we investigate the transition that we need to fire next, in order to explain the next character observed in the word. Hence, 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.[3] 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 element, instead of over the next transition.

We have implemented both the Petri-net-based probabilistic classifier based on $prob_m^{uniform}$ and the one based on the trained $prob_m^{empirical}$ and they are openly available as part of the ProM process mining toolkit [33] in the package *SequencePredictionWithPetriNets*[4].

Finally, note that, for the purpose of this paper, we assume the fact that given a prefix and a process model, we are able to obtain the corresponding marking in the accepting Petri net. However, the prediction method itself is more general and could be used with any alternative approach to obtain a marking in a Petri net given a given a prefix.

5 Experimental Setup

Fig. 4 depicts a high-level overview of the experimental setup that we employ to compare the sequence modeling methods. First, for each combination of modeling method and sequence database we make a sequence-level random split into $\frac{2}{3}$ training sequences and $\frac{1}{3}$ test sequences. After generating the model on the training sequences we evaluate how well the actual next element predicted for each prefix in the test sequences fits the probability distribution over all possible next elements according to the model. For each combination of sequence database and modeling method we repeat the experiment three times (with different random splits) to prevent that the results are too dependent

[3] Alternatively we are able to store a distribution of labels directly in correspondence with marking m. However, in such case, the predictor allows us to predict labels which are in fact not described by the process model in the corresponding marking.

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

on the random sampling of the sequence database into train and test split and over these three results we calculate the 95% confidence interval around the model performance. The mean value of the three results that is obtained for each combination of data set and modeling method is an unbiased point estimator of the actual accuracy of that method on that data.[5]

This experimental setup consisting of repeated generation of train/test splits is commonly known in the machine learning literature as *repeated holdout validation* and is sometimes alternatively referred to as *Monte Carlo cross-validation.* Repeated holdout validation has been proven to have the property that in the limit, when the number of repetitions goes to infinity, the probability of identifying the truly best method out of a set of candidate methods equals one [67].

A common experimental setup for evaluating machine learning methods is *k-fold cross validation*, where the data set is divided into k non-overlapping subsets after which k experimental runs are performed: each using $k-1$ subsets for training and one subset for testing. As a result, all k experimental runs are completely determined by a single random allocation of the original dataset into k subsets. While this approach allows for an unbiased estimate of the mean accuracy, it does not allow for unbiased estimates of the variance of the accuracy without multiple times repeating the *k-fold cross validation* procedure itself [14], resulting in an experimental setup that is refered to as *repeated k-fold cross validation.* Repeated k-fold cross validation is however prohibitively computationally expensive (e.g. 3 repetitions of 3-fold cross validation would already result in 9 experimental runs) in combination with the computational complexity of some of the sequence modeling methods[6]. In this study, we therefore choose the repeated holdout validation setup as it contrasts k-fold cross validation by having independent allocations of data points into train and test set between the consecutive experimental runs, allowing the estimation of confidence intervals of the accuracy.

The performance measure that we use to assess the model performance is the Brier score [20], which is a widely used measure to evaluate a probabilistic classifier. Intuitively, it can be interpreted as being the mean squared error of the predicted likelihoods over all symbols.

We now continue by giving an overview of the sequence databases used for the evaluation in Section 5.1 and describing the configurations and the implementations that were used for the sequence models in Section 5.2.

5.1 Sequence Databases

We evaluate the generalizing capabilities of process discovery methods and sequence modeling methods on four real-life sequence databases:

[5] As proven in [81]

[6] Especially the predictors based on the process discovery methods are computationally expensive, as each prediction requires computing a prefix-alignment on a Petri net.

- The **Receipt phase**[7] sequence database [22] from the WABO project, containing 8577 events of 27 symbols originating from 1434 cases of the receipt phase of the building permit application process at a Dutch municipality.
- The **BPI'12**[8] sequence database [32] which contains cases from a financial loan application process at a large financial institute, consisting of 164506 events divided over 13087 sequences and 23 symbols
- The **SEPSIS**[9] sequence database [55], containing medical care pathways of 1050 sepsis patients, for which in total 15214 events were logged from 16 different symbols.
- The **NASA**[10] sequence database [50], which contains method-call-level events that each describe a single run of an exhaustive unit test suite for the NASA Crew Exploration Vehicle (CEV)[11]. The dataset consists of 2566 sequences consisting of 36819 events in total over 47 symbols.

5.2 Configurations and Implementations

We apply all the sequence modeling methods that we have introduced in Section 3. Most of these sequence modeling methods have several hyper-parameters that can be manually selected or automatically tuned in order to achieve good performance on a given dataset. For instance, in the case of HMM, the number of hidden states needs to be selected. The performance of the sequence modeling methods (AKOM, HMM, RNN, GRU, and LSTM) can highly depend on the chosen configuration of hyper-parameters. Therefore, we conduct an optimization procedure for these methods to find the best-performing parameter setting before building the final model. To this end, we split the set of training sequences further into two parts using a random split, resulting in a 80% inner training set (53% of original sequences) and a 20% validation set (13% of original sequences). We then test multiple parameter configurations by training the model on the inner training set and measuring the performance on the validation set. We choose the configuration that achieved the best Brier score on the validation set and use these parameter settings to build the final model on all training sequences.

The hyper-parameters included in the optimization procedure and the considered values for these parameters are shown in Tab. 3. As the parameter space for AKOM, HMM, and the abstraction-based approach is rather small, we perform a grid search for these methods, testing all the possible combinations of the considered values. In case of the neural-network-based models we employ a state-of-the-art hyper-parameter optimization that is called tree-structured Parzen estimator (TPE) [15]. The TPE optimizer is necessary

[7] `https://doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`
[8] `https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
[9] `https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`
[10] `https://doi.org/10.4121/uuid:60383406-ffcd-441f-aa5e-4ec763426b76`
[11] `http://babelfish.arc.nasa.gov/hg/jpf/jpf-statechart`

Table 3: Definition of the hyper-parameter search spaces for the sequence modeling methods.

Method	Procedure	Parameter	Considered values
RNN, GRU, LSTM	TPE	*n_layers*	$\{1,2,3\}$
		layer_size	$[10,150]$
		batch_size	$\{2^3,2^4,2^5,2^6\}$
		dropout	$[0,0.5]$
		l_1	$[0.00001,0.1]$
		l_2	$[0.00001,0.1]$
AKOM	grid search	k	$\{1,2,...,19\}$
HMM	grid search	*n_states* to $\|\Sigma\|$ ratio	$\{0.1,0.5,1.0,1.5,2.0,3.0,5.0\}$
		regularizer	$\{None,l_2,l_\infty\}$
Automaton-based	grid search	*type*	$\{seq, mult, set\}$
		k	$\{1,2,...,19\}$

for the case of neural networks since the high dimensionality of its hyper-parameter search space makes a grid search inefficient. TPE is a sequential model-based optimization procedure that in each iteration selects a parameter configuration for testing according to predefined distributions for each parameter.

5.2.1 Neural Networks

For the neural-network-based approaches (RNN, GRU, and LSTM) we use their respective implementations provided in the Keras[12] Python deep learning library. For each of these types of neural networks we explore multiple architectures described later in this subsection. We optimize the weights of the models using Adam [46], which is a recent variant of stochastic gradient descent that has empirically shown to perform well. Furthermore, we apply early stopping, i.e., we stop training the neural network if no performance improvement has been seen for 20 training iterations in a row. Early stopping can help to reduce or prevent overfitting by preventing the model weights to take values that represent specific characteristics of the training data that do not generalize to the test data. Architecture choices such as the number of layers of the neural network and the number of units per layer are left to the hyper-parameter optimization procedure (as shown in Tab. 3, together with the learning rate parameter, which determines the step-size of the gradient descent weight optimization procedure, in the direction of the gradient. Finally, the hyper-parameter optimization procedure includes dropout, l_1 and l_2, which are three types of model regularization that can help to prevent overfitting by punishing overly complex weight structures of the neural network, thereby applying Occam's razor and steering the neural network towards simpler so-

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

lutions. The code to train and evaluate the neural network architectures is available in a Github repository that accompanies this paper.[13]

5.2.2 Markov Models

For the Markov chains we apply a first-order as well as a second-order Markov chain. We have implemented the Markov chain predictors and the AKOM model ourselves and the code is publicly available in a Github repository.[13] For the AKOM model, we optimize the hyper-parameter k using grid search. For Hidden Markov Models (HMM), we use the `hmm.discnp`[14] library in R. We perform a grid search hyper-parameter optimization to select the number of hidden states of the HMM as well as the type of regularization that is used to prevent the model from overfitting.

5.2.3 Grammar Inference

For the spectral learning grammar inference method we use the implementation that is provided in the Python spectral learning toolkit Scikit-SpLearn [9]. The implementation of train-test procedure that makes use of this code is included in the Github repository that accompanies this paper. [13]

5.2.4 Automaton-based Approaches

The automaton-based next-element prediction methods have been implemented in the Python process mining library PM4Py [16][15]. We performed a grid search hyper-parameter optimization to select the type of abstraction that is used for the automaton (i.e., set, multiset, or sequence abstraction) and the window size parameter k.

5.2.5 Process Model Approaches

For the next-element predictors based on Petri nets, we apply the process discovery methods that were described in Section 3.3.2 with their default parameter settings, unless in cases where we explicitly state a different parameter value. Hyper-parameter optimization of the process discovery methods for process-model-based prediction is not possible due to the computational time needed to make process-model-based prediction (mainly due to the computation time needed to calculate prefix-alignments). However, in general, hyper-parameters are not as omnipresent for process discovery approaches as opposed to machine learning methods. For hyper-parameters that are of vital importance to the success of process discovery, such as the frequency threshold of the Inductive Miner, we report the performance using several settings.

[13] `https://github.com/TaXxER/rnnalpha`
[14] `https://CRAN.R-project.org/package=hmm.discnp`
[15] `http://pm4py.org/`

Table 4: The mean of Brier score and the 95% confidence interval (ranging from $\mu \pm CI$) for each combination of method and dataset.

Method	Receipt Phase		BPI'12		SEPSIS		NASA	
	μ	CI	μ	CI	μ	CI	μ	CI
Baselines Methods								
Random guessing	0.0381	0.0010	0.0417	0.0000	0.0620	0.0061	0.0213	0.0001
Proportional guessing	0.0336	0.0009	0.0402	0.0000	0.0542	0.0002	0.0209	0.0001
Process Mining: Process Discovery with Uniform Distribution per Marking								
IM [51]	0.0338	0.0018	0.0314	0.0013	0.0612	0.0053	0.0207	0.0003
IMf 20% [52]	0.0224	0.0015	0.0293	0.0001	0.0486	0.0011	0.0152	0.0001
IMf 50% [52]	0.0191	0.0037	0.0390	0.0005	0.0759	0.0085	0.0211	0.0006
HM [77]	0.0245	0.0008	0.0258	0.0004	0.0442	0.0005	0.0177	0.0004
SM [10]	0.0262	0.0022	0.0252	0.0002	0.0574	0.0026	0.0160	0.0002
ILP [83]	0.0167	0.0012	0.0413	0.0037	0.0526	0.0013	0.0232	0.0013
ETMd [23]	0.0196	0.0028	0.0287	0.0028	0.0526	0.0013	0.0197	0.0004
Indulpet [53]	0.0249	0.0010	0.0429	0.0059	0.0578	0.0024	0.0205	0.0043
Process Mining: Process Discovery with Trained Distribution per Marking								
IM [51]	0.0255	0.0027	0.0287	0.0011	0.0455	0.0035	0.0202	0.0002
IMf 20% [52]	0.0152	0.0015	0.0293	0.0001	0.0395	0.0014	0.0106	0.0003
IMf 50% [52]	0.0153	0.0009	0.0347	0.0009	0.0664	0.0017	0.0207	0.0005
HM [77]	0.0181	0.0007	0.0231	0.0003	0.0372	0.0013	0.0159	0.0006
SM [10]	0.0099	0.0006	0.0226	0.0001	0.0513	0.0026	0.0155	0.0002
ILP [83]	0.0167	0.0012	0.0445	0.0059	0.0512	0.0013	0.0232	0.0012
ETMd [23]	0.0114	0.0010	0.0263	0.0059	0.0396	0.0013	0.0195	0.0002
Indulpet [53]	0.0153	0.0041	0.0441	0.0050	0.0451	0.0102	0.0195	0.0025
Process Mining: Automata Based prediction								
Automaton-based (Section 3.3.1)	0.0072	0.0002	0.0120	0.0000	0.0283	0.0004	0.0052	0.0000
Machine Learning: Neural Networks								
RNN	0.0072	0.0007	0.0159	0.0003	0.0277	0.0000	0.0048	0.0001
LSTM	0.0075	0.0012	0.0122	0.0000	0.0277	0.0008	0.0049	0.0002
GRU	0.0073	0.0008	0.0127	0.0001	0.0277	0.0004	0.0048	0.0000
Machine Learning: Compression								
Active LeZi [40]	0.0128	0.0004	0.0182	0.0011	0.0331	0.0002	0.0088	0.0004
Machine Learning: Markov Models								
1st-order Markov chain	0.0114	0.0007	0.0207	0.0000	0.0342	0.0002	0.0066	0.0000
2nd-order Markov chain	0.0110	0.0001	0.0135	0.0000	0.0313	0.0003	0.0052	0.0000
AKOM [60]	0.0070	0.0000	0.0119	0.0000	0.0262	0.0003	0.0049	0.0000
Hidden Markov Model	0.0184	0.0019	0.0188	0.0017	0.0340	0.0009	0.0081	0.0009
Grammar Inference								
Spectral Learning [12]	0.0195	0.0001	0.0370	0.0004	0.0480	0.0010	0.0207	0.0002

The experiments with the process-model-based predictors are performed using their implementation in the *SequencePredictionWithPetriNets*[16] package of the ProM process mining toolkit [33].

6 Results

Tab. 4 shows the results in terms of Brier score on the four datasets for each of the methods. The worst Brier score value of each μ-column in the table is colored in red and the best value is colored green, with the other values taking

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

an intermediate color. In the CI columns, the color represents the consistency of the approach: if the 95%-CI range has a small width (i.e., if the method performed very consistently amongst the three runs), then the cell is colored in green and otherwise in red. Two baseline methods are included in the table: the *random guessing* baseline corresponds to predicting the equal probability to each symbol (i.e., predicting according to a uniform categorical distribution), while the *proportional guessing* baseline corresponds to predicting according to the frequency distribution of symbols in the training sequences.

Overall, the three neural network types, AKOM, and the automaton-based approach have the lowest error in terms of Brier score, with only very small differences between their accuracies. AKOM is the best performing sequence model on average on three of the four datasets, with GRU being the best performing sequence model on average on the NASA software log. The confidence interval around the mean Brier score for the neural network methods turns out to be wider than for AKOM and the automaton-based predictor, meaning that while their mean Brier scores are similar, the neural networks were impacted to a larger degree by the random splits into training and test data. This might indicate that the neural network approaches are more prone to overfitting the training data, even though we applied regularization to prevent overfitting and used hyper-parameter optimization to select the degree of regularization.

On all four datasets, the Brier scores for the aforementioned methods are considerably better than for the process-model-based approaches and than grammar inference, meaning that they provide considerably more accurate probability distributions over the next event for prefixes from previously unseen sequences. This finding is independent of whether uniform categorical distributions or trained categorical distributions were used for the process models. HMMs and Active Lezi generally perform better than the process-model-based and grammar inference approaches, but do not reach top performance in any of the four datasets.

The results also show that learning a categorical probability distribution over the enabled transitions for each marking from the training data leads to more accurate predictions on the test data compared to the approach where we assumed the categorical probability distribution over the enabled transitions to be uniform. Note, however, that in the process mining field the discovered process models are often used to communicate with process stakeholder about the business process, and that the process model discovered with process discovery typically have 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 [10] is the best performing process discovery method on two of the four logs when we learn the probability distribution per marking from the training data, with on the other logs the Heuristics Miner [77] and the Inductive Miner with 20% filtering [52] being the best approach. Interestingly, the Indulpet Miner [53] does not perform well on average, yet, has a very large 95%-CI for all logs, indicating that for some of the random train/test-splits the method generates quite accurate predictions but for others very inaccurate ones.

The spectral learning grammar inference performs similarly to the process discovery methods when training the distribution per marking. This shows that the methods from the grammar inference field and from the process mining field, which both have the aim to generate human-interpretable sequence models, are less accurate than the machine learning methods that have no goal of interpretability and focus solely on accuracy.

The prediction accuracy of the automaton-based methods from the process mining field is remarkable, since these methods are still somewhat interpretable: the simple nature of the abstraction functions that we introduced in Section 3.3.1 imply that the resulting automaton has interpretable states. Furthermore, methods exists to transform this automaton into a Petri net (see [2]). Note that in some sense, AKOM could also be argued to be an interpretable model in the sense that for a given prediction it can easily be traced back what caused the model to make this prediction by looking up the state in the Markov model that was used to make the prediction. However, if we put the threshold for interpretability at a stronger notion of interpretability: "can we get insight into the model behavior by looking at the model" instead of "can we trace back the reason why a model made a certain prediction", then AKOM does not yield an interpretable model since it consists of k individual Markov models that would each need to be comprehended to understand the model behavior. The automaton-based predictors based on set, multiset, and sequence abstraction fit both notions of interpretability.

7 Related Work

We group related work into several directions of related work. Measures for generalization from the process mining field are one area of related work, which we discuss in Section 7.1. Another area of related work is predictive business process monitoring, which we discuss in Section 7.2. Finally, in Section 7.3 we discuss several sequence prediction methods that predict only the single most likely next element instead of predicting a probability distribution over all possible next elements.

7.1 Measuring Generalization

The work presented in this paper is closely related to the challenge of measuring precision in process mining. In the process mining field, generalization is often defined as "the likelihood that the process model is able to describe yet unseen behavior of the observed system" [23]. This definition is noticeably different from the definition of generalization in the machine learning field. The process mining definition of generalization is an asymmetric one: it specifies that the model should ideally allow for sequences from the test set, but it does not specify that the models should *not* allow for sequences that are *not* in the test set. A consequence of this definition is that a model that allows for all behavior is the most generalizing one. In contrast, in the machine learning field it is common to have a more probabilistic notion of generalization: the model should specify a probability distribution over sequences that make sequences in the test set likely (and as a probability distribution has to sum to 1, an effect is that other sequences should be unlikely).

Several generalization measures have been proposed in the process mining field that quantify the degree to which a given process model generalizes the behavior that is observed in a given sequence dataset [3, 21, 35]. All of these measures calculate the generalization of the process model *with respect to the same sequence database from which the process model was discovered.* In contrast, in the machine learning 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 disjoint 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.

7.2 Predictive Business Process Monitoring

Predictive business process monitoring is a research field that is concerned with sequence predictions within the application domain of *business process management.* The field focuses on several prediction tasks for ongoing instances of a business process, including prediction of an outcome of the process instance [31, 74], prediction of the remaining time of a process instance [34, 64, 71], predicting deadline violations [59], and, most relevant to this work, prediction of the coming business activities with a running instance of a business process [68, 71].

Several approaches have been proposed in the literature to tackle the challenge of predicting the next business activity in ongoing instances of a business process. Often, events are considered to be multi-dimensional, i.e., there are additional attributes that describe the events in addition to only the business activity. Some of the methods encode the available data as a feature vector, and subsequently apply a classifier to predict the next event [61, 75, 56]. Other methods discover a process/sequence model from the control-flow using sequential pattern mining [26], Markov models [48] or a Probabilistic Finite Automaton [19]. Often, as a second step after discovering the process model, classifiers are built for each state in the model, enabling the inclusion the data payload of the ongoing case into the prediction process [26, 48]. In [66], the authors propose a recommendation engine that allows to predict the business activity based on the assumption that a process model of the underlying process is known. As a consequence, a collection of possible next elements is assumed to be known/given. The recommendation engine allows the user to compute the best possible next element from the given collection, that is expected to most positively impact a user-specified KPI.

Rogge-Solti developed several methods [63, 65, 64] to predict the remaining cycle time using stochastic Petri nets (SPN) and Generalized Stochastic Petri nets (GSPN) [5]. These SPNs and GSPNs closely link to the Petri nets with probability that we introduce in this work, however, there is an important difference: where the Petri nets in our work define a categorical probability distribution over the next transition, SPNs and GSPNs define a continuous probability distribution that specifies the *timing* of transition firings. SPNs and GSPNs thereby only implicitly specify an ordering: transitions that are likely to fire soon are more likely to be the next transition to fire. Furthermore, the work of Rogge-Solti applies these SPN and GSPN models only for the prediction of the remaining cycle time and does not address next-element prediction for unfinished sequences.

More recently, deep learning approaches, i.e., in particular Long Short-Term Memory (LSTM) unit based networks, have been applied in the context of next element prediction [37, 57, 71]. However, none of these studies compare their method to existing process discovery methods. Therefore, 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 methods. Furthermore, while most of the next element 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 methods in terms of control-flow.

7.3 Non-probabilistic Sequence Classification

In this paper we have focused on sequence models where the next-element predictions are probabilistic, i.e., that provide the probability distribution over the set of possible outcomes instead of simply giving the single most likely outcome. Several methods from the data mining community focus solely on predicting the single most likely next element of a sequence without generating the whole probability distribution, i.e., non-probabilistic sequence models.

One of such non-probabilistic sequence prediction algorithms is the *compact prediction tree* (CPT) [41] algorithm, which was later improved to the computationally more efficient CPT+ algorithm [42]. The CPT and CPT+ algorithms generate a tree-based data structure that makes a lossless compression from the training data which can be used at prediction time to efficiently search for the most likely next element.

8 Threats to Validity & Limitations

One threat to validity of the performed experimental evaluation is the limited number of datasets that are included in the experiment. The main reason for this is of computational nature: computing prefix-alignments for each prefix in a training and a test set is computationally expensive and process discovery methods are therefore not very practically applicable for next-element prediction. We have aimed at mitigating this limitation by using publicly available datasets from a range of different application domains and by making the code for the experiments available, so that it can be used by other researchers to expand the evaluation to new methods and datasets.

Another threat to validity with respect to the results of the process-discovery-based next-element prediction methods is the fact that the prefix-alignments that were used in the experiments were not probabilistic in nature. Currently, no tractable approach to alignments exists that is also able to deal with prefix-alignments. However, we estimate the impact of this validity threat to be limited, as it only affects predictions for prefixes that are non-fitting in the Petri net.

One limitation of the study is the fact that our findings only generalize to the set of methods that are included in the comparison. For every process discovery, every grammar inference, and every machine learning algorithm that will be newly developed in the coming years we cannot judge its performance until the experiments are repeated for this new method. We have aimed at mitigating this limitation by using public datasets and making the code for our experimental setup available.

A threat to validity that follows from the above-mentioned limitation is related to the selection of algorithms: our selection of grammar inference, of process discovery, and of machine learning algorithms might have impacted the overarching conclusions related to the relative accuracy of process discovery, of machine learning, and of grammar inference methods. We have aimed to mitigate this limitation by focusing on the state-of-the-art algorithms from each of the three research fields.

9 Conclusions & Future Work

In this work we performed an experimental evaluation of 26 sequence modeling methods spanning three research fields: *grammar inference*, *process mining*, and *machine learning*. To the best of our knowledge, there has so far been no comparative evaluation that compares sequence modeling methods from these different fields. Methods from the grammar inference field and from the process mining field have an aim to generate human-interpretable sequence models, whereas machine learning methods often have no aim to be interpretable and focus solely on accurate predictions.

In order to exploit process mining methods as generative sequence models we have introduced two ways in which (discovered) Petri nets can be used as probabilistic classifiers to predict the next element for a given prefix of a sequence: 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 applied these two approaches, along with representative methods from the grammar inference and machine learning fields, to a collection of four publicly available real-life sequence databases. We have found that overall, the black-box methods from the machine learning field generate more accurate predictions than the interpretable models from the grammar inference and from the process mining fields. 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 particular, the best performance is achieved by AKOM and recurrent neural networks. However, in cases where interpretability is a priority, automaton-based prediction can serve as a good alternative. Furthermore, if it is important to have a formal model that explicitly models concurrency, the best choices would be Split Miner and IMf 20%. Note that out of these two methods, only IMf 20% guarantees a sound Petri net.

An interesting direction for future work arises with regard to process-model-based predictions. Namely, the approaches proposed in this paper are limited by the non-probabilistic nature of prefix-alignments. We see probabilistic prefix-alignments as a vital future area of research in order to more accurately infer the categorical probability distributions over the next symbols for each marking of the process model. Such extensions could potentially lead to interpretable sequence modeling methods that are competitive with machine learning methods in terms of accuracy.

References

1. van der Aalst WMP (2016) Process mining: data science in action. Springer
2. van der Aalst WMP, Rubin VA, Verbeek HMW, van Dongen BF, Kindler E, Günther CW (2010) Process mining: a two-step approach to balance between underfitting and overfitting. Software and System Modeling 9(1):87–111
3. van der Aalst WMP, Adriansyah A, van Dongen BF (2012) Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2):182–192
4. Adriansyah A (2014) Aligning observed and modeled behavior. PhD thesis, Eindhoven University of Technology
5. Ajmone Marsan M, Conte G, Balbo G (1984) A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2(2):93–122
6. Alizadeh M, de Leoni M, Zannone N (2014) History-based construction of log-process alignments for conformance checking: Discovering what really went wrong. In: Data-Driven Process Discovery and Analysis, Springer, pp 1–15

7. Alizadeh M, de Leoni M, Zannone N (2015) Constructing probable explanations of nonconformity: A data-aware and history-based approach. In: Proceedings of the IEEE Symposium Series on Computational Intelligence, IEEE, pp 1358–1365
8. Angluin D (1987) Learning regular sets from queries and counterexamples. Information and computation 75(2):87–106
9. Arrivault D, Benielli D, Denis F, Eyraud R (2017) Scikit-SpLearn: a toolbox for the spectral learning of weighted automata compatible with scikit-learn. In: Conférence francophone sur l'Apprentissage Aurtomatique
10. Augusto A, Conforti R, Dumas M, La Rosa M (2017) Split miner: Discovering accurate and simple business process models from event logs. In: IEEE International Conference on Data Mining (ICDM), IEEE, pp 1–10
11. Augusto A, Conforti R, Dumas M, La Rosa M, Maggi FM, Marrella A, Mecella M, Soo A (2018) Automated discovery of process models from event logs: Review and benchmark. IEEE Transactions on Knowledge and Data Engineering
12. Balle B, Carreras X, Luque FM, Quattoni A (2014) Spectral learning of weighted automata. Machine learning 96(1-2):33–63
13. Balle B, Eyraud R, Luque FM, Quattoni A, Verwer S (2017) Results of the sequence prediction challenge (SPiCe): a competition on learning the next symbol in a sequence. In: International Conference on Grammatical Inference (ICGI), Springer, pp 132–136
14. Bengio Y, Grandvalet Y (2004) No unbiased estimator of the variance of k-fold cross-validation. Journal of machine learning research 5(Sep):1089–1105
15. Bergstra JS, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems (NIPS), pp 2546–2554
16. Berti A, van Zelst SJ, van der Aalst WMP (2019) 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 1316, URL `http://ceur-ws.org/Vol-2374/`
17. Bojar O, Buck C, Federmann C, Haddow B, Koehn P, Leveling J, Monz C, Pecina P, Post M, Saint-Amand H, et al (2014) Findings of the workshop on statistical machine translation. In: Proceedings of the ninth workshop on statistical machine translation, pp 12–58
18. Boulanger-Lewandowski N, Bengio Y, Vincent P (2012) Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In: Proceedings of the International Conference on Machine Learning (ICML)
19. Breuker D, Matzner M, Delfmann P, Becker J (2016) Comprehensible predictive models for business processes. MIS Quarterly 40(4):1009–1034
20. Brier GW (1950) Verification of forecasts expressed in terms of probability. Monthly Weather Review 78(1):1–3

21. vanden Broucke SKLM, De Weerdt J, Vanthienen J, Baesens B (2014) Determining process model precision and generalization with weighted artificial negative events. IEEE Transactions on Knowledge and Data Engineering 26(8):1877–1889
22. Buijs JCAM (2014) Receipt phase of an environmental permit application process (wabo), coselog project. DOI 10.4121/UUID:A07386A5-7BE3-4367-9535-70BC9E77DBE6
23. Buijs JCAM, Van Dongen BF, van der Aalst WMP (2012) On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, pp 305–322
24. Buijs JCAM, van Dongen BF, van der Aalst WMP (2014) Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems 23(01):1440,001
25. Carmona J, de Leoni M, Depaire B, Jouck T (2016) Summary of the process discovery contest 2016. In: Proceedings of the Business Process Management Workshops, Springer
26. Ceci M, Lanotte PF, Fumarola F, Cavallo DP, Malerba D (2014) Completion time and next activity prediction of processes using sequential pattern mining. In: International Conference on Discovery Science, Springer, pp 49–61
27. Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing (EMNLP), ACL
28. Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS Deep Learning and Representation Learning Workshop
29. Clark A (2007) Learning deterministic context free grammars: The omphalos competition. Machine Learning 66(1):93–110
30. De La Higuera C (2005) A bibliographical study of grammatical inference. Pattern recognition 38(9):1332–1348
31. Di Francescomarino C, Dumas M, Maggi FM, Teinemaa I (2016) Clustering-based predictive process monitoring. IEEE Transactions on Services Computing
32. van Dongen BF (2012) Bpi challenge 2012. DOI 10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F, URL `https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f`
33. van Dongen BF, de Medeiros AKA, Verbeek HMW, Weijters AJMM, van der Aalst WMP (2005) The ProM framework: A new era in process mining tool support. In: International Conference on Application and Theory of Petri Nets (PETRI NETS), Springer, pp 444–454
34. van Dongen BF, Crooy RA, van der Aalst WMP (2008) Cycle time prediction: When will this case finally be finished? In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems",

Springer, pp 319–336

35. van Dongen BF, Carmona J, Chatain T (2016) A unified approach for measuring precision and generalization based on anti-alignments. In: International Conference on Business Process Management (BPM), Springer, pp 39–56
36. Dunning T (1994) Statistical identification of language. Computing Research Laboratory, New Mexico State University
37. Evermann J, Rehse JR, Fettke P (2017) Predicting process behaviour using deep learning. Decision Support Systems 100:129–140
38. Gagniuc PA (2017) Markov Chains: From Theory to Implementation and Experimentation. John Wiley & Sons
39. Gold EM (1978) Complexity of automaton identification from given data. Information and control 37(3):302–320
40. Gopalratnam K, Cook DJ (2007) Online sequential prediction via incremental parsing: The active LeZi algorithm. IEEE Intelligent Systems (1):52–58
41. Gueniche T, Fournier-Viger P, Tseng VS (2013) Compact prediction tree: A lossless model for accurate sequence prediction. In: Proceedings of the International Conference on Advanced Data Mining and Applications (ADMA), Springer, pp 177–188
42. Gueniche T, Fournier-Viger P, Raman R, Tseng VS (2015) Cpt+: Decreasing the time/space complexity of the compact prediction tree. In: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Springer, pp 625–636
43. De la Higuera C (2010) Grammatical inference: learning automata and grammars. Cambridge University Press
44. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780
45. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences 79(8):2554–2558
46. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: International Conference for Learning Representations (ICLR)
47. Koorneef M, Solti A, Leopold H, Reijers HA (2017) Automatic root cause identification using most probable alignments. In: Proceedings of the Business Process Management Workshops, Springer, pp 204–215
48. Lakshmanan GT, Shamsi D, Doganata YN, Unuvar M, Khalaf R (2015) A markov prediction model for data-driven semi-structured business processes. Knowledge and Information Systems 42(1):97–126
49. Lang KJ, Pearlmutter BA, Price RA (1998) Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: International Colloquium on Grammatical Inference (ICGI), Springer, pp 1–12
50. Leemans M, van der Aalst WMP, van den Brand MGJ (2018) Recursion aware modeling and discovery for hierarchical software event log analysis. In: 2018 IEEE 25th International Conference on Software Analysis,

Evolution and Reengineering (SANER), IEEE, pp 185–196
51. Leemans SJJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs-a constructive approach. In: International conference on applications and theory of Petri nets and concurrency (PETRI NETS), Springer, pp 311–329
52. Leemans SJJ, Fahland D, van der Aalst WMP (2013) Discovering block-structured process models from event logs containing infrequent behaviour. In: International Conference on Business Process Management (BPM), Springer, pp 66–78
53. Leemans SJJ, Tax N, ter Hofstede AHM (2018) Indulpet miner: Combining discovery algorithms. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, pp 97–115
54. Logan B, Chu S (2000) Music summarization using key phrases. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE, vol 2, pp II749–II752
55. Mannhardt F, Blinde D (2017) Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+ EMISA, CEUR-ws.org, vol 1859, pp 72–80
56. Márquez-Chamorro AE, Resinas M, Ruiz-Cortés A, Toro M (2017) Run-time prediction of business process indicators using evolutionary decision rules. Expert Systems with Applications 87:1–14
57. Mehdiyev N, Evermann J, Fettke P (2017) A multi-stage deep learning approach for business process event prediction. In: IEEE Conference on Business Informatics (CBI), IEEE, vol 1, pp 119–128
58. Object Management Group (2011) Business process model and notation. Tech. Rep. formal/2011-01-03, Object Management Group, URL `https://www.omg.org/spec/BPMN/2.0/`
59. Pika A, van der Aalst WMP, Fidge CJ, ter Hofstede AHM, Wynn MT (2012) Predicting deadline transgressions using event logs. In: International Conference on Business Process Management (BPM), Springer, pp 211–216
60. Pitkow J, Pirolli P (1999) Mining longest repeating subsequences to predict worldwide web surfing. In: USENIX Symposium on Internet Technologies and Systems, pp 13–26
61. Pravilovic S, Appice A, Malerba D (2013) Process mining to forecast the future of running cases. In: International Workshop on New Frontiers in Mining Complex Patterns, Springer, pp 67–81
62. Rabiner LR (1989) A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2):257–286
63. Rogge-Solti A, Weske M (2013) Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Proceedings of the International Conference on Service-Oriented Computing (ICSOC), Springer, pp 389–403
64. Rogge-Solti A, Weske M (2015) Prediction of business process durations using non-markovian stochastic petri nets. Information Systems 54:1–14

65. Rogge-Solti A, van der Aalst WM, Weske M (2013) Discovering stochastic petri nets with arbitrary delay distributions from event logs. In: Proceedings of the International Conference on Business Process Management (BPM), Springer, pp 15–27
66. Schonenberg H, Weber B, van Dongen BF, van der Aalst WMP (2008) Supporting flexible processes through recommendations based on history. In: Proceedings of the International Conference on Business Process Management (BPM), pp 51–66
67. Shao J (1993) Linear model selection by cross-validation. Journal of the American statistical Association 88(422):486–494
68. van der Spoel S, van Keulen M, Amrit C (2012) Process prediction in noisy data sets: a case study in a dutch hospital. In: International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA), Springer, pp 60–83
69. Stanke M, Waack S (2003) Gene prediction with a hidden Markov model and a new intron submodel. Bioinformatics 19(suppl_2):ii215–ii225
70. Tax N, Sidorova N, Haakma R, van der Aalst WMP (2016) Mining local process models. Journal of Innovation in Digital Ecosystems 3(2):183–196
71. Tax N, Verenich I, La Rosa M, Dumas M (2017) Predictive business process monitoring with lstm neural networks. In: International Conference on Advanced Information Systems Engineering (CAiSE), Springer, pp 477–492
72. Tax N, Sidorova N, van der Aalst WMP, Haakma R (2018) LocalProcessModelDiscovery: Bringing Petri nets to the pattern mining world. In: Proceedings of the International Conference on Applications and Theory of Petri Nets and Concurrency (PETRI NETS), Springer, pp 374–384
73. Tax N, van Zelst SJ, Teinemaa I (2018) An experimental evaluation of the generalizing capabilities of process discovery techniques and black-box sequence models. In: Proceedings of the International Conference on Enterprise, Business-Process and Information Systems Modeling (BPMDS), pp 165–180
74. Teinemaa I, Dumas M, Maggi FM, Di Francescomarino C (2016) Predictive business process monitoring with structured and unstructured data. In: International Conference on Business Process Management (BPM), Springer, pp 401–417
75. Unuvar M, Lakshmanan GT, Doganata YN (2016) Leveraging path information to generate predictions for parallel business processes. Knowledge and Information Systems 47(2):433–461
76. Verwer S, Eyraud R, De La Higuera C (2014) PautomaC: a probabilistic automata and hidden markov models learning competition. Machine learning 96(1-2):129–154
77. Weijters AJMM, Ribeiro JTS (2011) Flexible heuristics miner (FHM). In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, pp 310–317
78. Weinberger MJ, Seroussi G (1997) Sequential prediction and ranking in universal context modeling and data compression. IEEE Transactions on

Information Theory 43(5):1697–1706
79. van der Werf JMEM, van Dongen BF, Hurkens CAJ, Serebrenik A (2009) Process discovery using integer linear programming. Fundam Inform 94(3-4):387–412
80. Wirth N (1977) What can we do about the unnecessary diversity of notation for syntactic definitions? Communications of the ACM 20(11):822–823
81. Wong TT (2015) Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. Pattern Recognition 48(9):2839–2846
82. van Zelst SJ, Bolt A, van Dongen BF (2017) Tuning alignment computation: An experimental evaluation. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data (ATAED), pp 6–20
83. van Zelst SJ, van Dongen BF, van der Aalst WMP, Verbeek HMW (2017) Discovering workflow nets using integer linear programming. Computing
84. Ziv J, Lempel A (1978) Compression of individual sequences via variable-rate coding. IEEE transactions on Information Theory 24(5):530–536