Tell me what's ahead? Predicting remaining activity sequences of business process instances

Ilya Verenich[1,2], Dmytro Chasovskyi[2], Marlon Dumas[2], Marcello La Rosa[1], Fabrizio Maria Maggi[2], and Andrii Rozumnyi[2]

[1] Queensland University of Technology, Australia
{ilya.verenich,m.larosa}qut.edu.au
[2] University of Tartu, Estonia
{chasovsk,marlon.dumas,andriiro,f.m.maggi}@ut.ee

Abstract. Predictive business process monitoring is a family of techniques to determine how running instances of a business process are likely to unfold in the future. Techniques in this space differ according to their object of prediction. Some predict whether or not a running process instance will fulfill a compliance rule, others predict whether or not a given activity will occur, while others predict the remaining execution time. These and other predictive process monitoring problems are subsumed by the problem of predicting the remaining sequence of activities of a given process instance. In this paper, we tackle this latter problem using two alternative approaches. In the first one, we statically construct a transition system from an event log of completed process instances and annotate each transition with a probability calculated using a k-nearest neighbors classifier. At runtime, we map the (incomplete) trace of a process instance to a state in the transition system. To predict the remaining activity sequence of a process instance, we calculate a highest-probability path starting from the current state. In the second approach, we treat the problem of activity sequence prediction as a structured output prediction problem and apply recurrent neural networks. The accuracy of the two proposed approaches is evaluated on real-life and synthetic datasets and compared against an existing baseline technique.

1 Introduction

Predictive business process monitoring is concerned with determining how running instances of a business process will unfold up to their completion based on models extracted from historical event logs. A range of such techniques have been proposed in the past decade, which differ widely depending on the object of prediction: predicting the next activity [1], predicting a deadline violation [2], predicting the remaining execution time [3,4] or predicting whether or not a process instance will fulfill a property upon completion [5].

Several of the above prediction problems are subsumed by a more general problem, namely that of predicting the remaining sequence of activities of a process instance. For example, predicting the next activity, the remaining execution time, a deadline violation or the fulfillment of a compliance rule can be addressed

if we can accurately predict the remaining activity sequence. Predicting the remaining activity sequence of running process instances also assist in assessing their future resource requirements and scheduling their efficient execution [6].

In this paper, we propose two approaches to predict the future sequence of activities of a running process instance starting from its current state and up to its completion. We take as input a log consisting of traces of completed process instances; a trace is a sequence of events, each recording the occurrence of an activity. Given such a log, in the first approach, we construct (offline) a transition system wherein each state represents a set of possible prefixes. Each transition is annotated with a firing probability estimated based on a classifier. Given an incomplete trace of a running process instance, we map this trace to a state and predict the remaining activity sequence by computing a highest-probability path from this state to the sink state of the transition system.

In the second approach, we treat the problem of activity sequence prediction as a structured output prediction problem and apply artificial recurrent neural networks. Recurrent neural networks have been applied in different domains, e.g. for predicting continuous spatiotemporal patterns, or perceptual sequences of robots [7]. However, to the best of our knowledge, they have never been used in the context of predictive business process monitoring.

We conduct an evaluation with real-life and synthetic logs to assess the accuracy of our two approaches against an existing baseline technique addressing the same problem, and discuss the relative strengths and weaknesses.

The rest of the paper is structured as follows. In Section 2 we briefly review related work in the area of predictive process monitoring. Section 3 introduces machine learning concepts and techniques used throughout the paper. Section 4 describes the two proposed approaches, while Section 5 presents their evaluation. Finally, Section 6 draws conclusions and outlines future work directions.

2 Related Work

Predictive process monitoring techniques differ according to their object of prediction. For example, Maggi et al. [5] focus on predicting boolean properties of process instances (e.g. fulfillment of compliance rules). Meanwhile, Lakshmanan et al. [8] propose a technique to estimate the probability of future execution of a given task in an ongoing process instance using extended Markov chains. Becker et al. [1] use probabilistic finite automaton to predict the most likely type of the next event to be performed in an ongoing process instance. Pravilovic et al. [9] propose a framework to predict future events or their properties, such as the resource that will trigger the next event. Their approach transforms event-forecasting tasks into sliding windows of time intervals, which are then processed in a predictive clustering tree. Finally, a range of approaches [3,4] aims at predicting the remaining execution time of a process instance.

Motivated by the observation that different approaches in this space target different but related problems, De Leoni et al. [10] propose a general framework for predicting characteristics of process instances (dependent variables) based on correlations with other characteristics (independent variables). This approach is targeted at scenarios where a set of (discrete) variables of interest is given.

Subramaniam et al. [6] go a step further by predicting multiple possible suffixes of a running process instance. For this, they generate classification rules using a decision trees algorithm. Van der Aalst et al. [11] utilize annotated transition systems, which provide recommendations regarding the future course of actions to process workers to ensure conformance with time objectives. Polato et al. [12,13] refine this approach by adding classifiers and regressors (which make use of event attributes) as annotations. Specifically, they use the naïve Bayes classifier to estimate the transition probability between two states and support vector regression to predict the remaining cycle time from a given state. In this paper, we take this latter approach as a baseline and explore alternatives based on k-nearest neighbor classification and recurrent neural networks.

3 Background

This section formulates the activity sequence prediction problem as a structured prediction problem. Next, we introduce transition systems and how they can be constructed from a log. Then, we briefly introduce the k-nearest neighbors algorithm and recurrent neural networks, which are used later in the paper.

3.1 Structured prediction

A supervised machine learning system is characterized by a learning algorithm and labeled training data. Training data is typically of the form:

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) : m, n \in \mathbb{N}\}, \tag{1}$$

where $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^m$ are the feature vectors and $y_i \in \mathcal{Y} = \mathbb{N}$ are the corresponding class labels. A learning framework uses the feature vectors extracted from the labeled training data to build a *classifier* for classification. The task of classification is to assign labels on new data given labeled training data.

Structured prediction is a generalization of the standard paradigms of supervised learning. Rather than predicting discrete class labels, it aims at predicting complex objects such as sequences and graphs. Consider an event stream for which we want to predict the next few events. Using classification, we could predict each future event individually by training a classifier to predict the next event, then another classifier to predict the second next event, and so on. This approach has clear drawbacks: to do the correct predictions, for each future event to be predicted, we need to have all possible activities in the training set. Furthermore, each future event is predicted independently of its neighbors, even though the output is a sequence of interrelated events. Structured prediction overcomes these problems by considering the output as a whole.

3.2 Transition system

A transition system is an abstract machine that can be in one of a finite number of *states*. The machine can move from one state to another by triggering a *transition*. The machine can be in only one state at a time. A transition system

can also be represented by a directed graph, wherein each node corresponds to a state, while edges show transitions from one state to another. Formally, a transition system is defined as follows:

Definition 1 (Transition system (TS)). *A transition system is a triplet $TS = (S, A, T)$, where S is the set of possible state of the process, E is the set of events (i.e. transition labels), and $T \subseteq S \times E \times S$ is the set of transitions which describe how a system can move from one state to another. $S^{start} \subseteq S$ is the set of initial states, and $S^{end} \subseteq S$ is the set of final (accepting) states.*

A transition system can be constructed using *state* and *event representation functions*, also referred to as *abstractions* [11].

Definition 2 (State representation function). *Let R_S be the set of possible state representations, a state representation function $f^{state} \in \Sigma \to R_S$ is a function that, given a (partial) trace σ returns some representation of it (e.g., sequences, sets, multiset over some event properties) [12].*

Here a trace corresponds to a *completed* process instance, while a *partial* trace represents an ongoing process instance which is still being executed.

According to [14] three common state abstractions are:

- *sequence abstraction*, wherein the order of activities is recorded in the state,
- *multiset abstraction*, which only includes the number of times each activity is executed ignoring their order,
- *set abstraction*, which keeps track of the mere presence of activities.

Definition 3 (Event representation function). *Let R_e be the set of possible event representations, an event representation function $f^{event} \in E \to R_e$ is a function that, given an event e produces some representation of it (e.g., projection functions over $e \in E$) [12].*

Using functions f^{event} and f^{state}, we define a transition system where states correspond to prefixes of traces in the log mapped to some representations using f^{state}, and transitions correspond to representation of events through f^{event} [12]. The exact algorithm for the construction of a transition system can be found in [12], while here we provide an example of an event log and transition system extracted from it (Figure 1). Each state $s_0, \ldots, s_5$ is labeled by the corresponding state representation function and each transition by the corresponding event representation function.

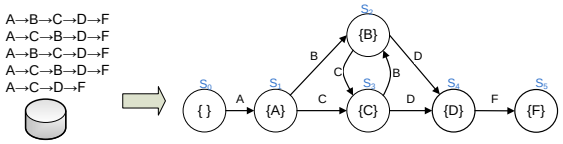


Fig. 1: A transition system constructed from an event log.

3.3 K-nearest neighbors

K-nearest neighbors algorithm (KNN) is one of the most well-known supervised learning algorithms in pattern classification. The input consists of the k closest training examples in the feature space and the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the most common class among its k nearest neighbors. KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification [15].

3.4 Neural networks

Artificial neural networks represent a diverse class of computational models that are designed by analogy with biological brain modules. They consist of a large number of simple neuron-like processing units, organized in layers. Every unit in a layer is connected with all the units in the previous layer. Each connection may have a different strength or weight. The weights on these connections encode the knowledge of a network. Data enters at the inputs and passes through the network, layer by layer, until the outputs are identified.

Neural networks are known to be a powerful tool for predicting various kinds of temporal sequences. However, the neural networks produce black-box patterns which are effectively incomprehensible as they do not provide a structural description. To the best of our knowledge, they have not been applied in the context of predictive business process monitoring.

4 Proposed Approaches

In this section, we describe the two proposed approaches to future path prediction: i) annotated transition systems enriched with k-nearest neighbors classifiers (ATS+KNN), and ii) recurrent neural networks (RNN).

4.1 Annotated Transition Systems with KNN Classifiers

Given an event log, containing completed traces of business process execution, and a partial trace, corresponding to an ongoing process instance, the goal of our approach is to predict how this instance will evolve in future, specifically what events will be performed next. Our approach is a refinement of the work by Polato et al. [13] who applied the naïve Bayes classifier to estimate the probability of transition from a current state to the next one. Naïve Bayes algorithm makes a rather strong assumption of the conditional independence of predictors, that is the data attributes of the running case. However, in real-life business processes, these attributes are usually interdependent. Therefore, in our work, we replace it with the k-nearest neighbors classifier.

We start by extracting a transition system from an event log, according to the algorithm described in [13]. To create a transition system, we use sequence abstraction, as in a log activities may repeat, and the order of activities is important. Next, we annotate each transition with a probability of firing, estimated

from KNN classifiers using the historical data of completed process instances that are similar to the ongoing instance. Having found the transition probabilities, we can calculate the most likely sequence from a current state to the terminating state. Then we compare the predicted sequence with the actual instance continuation to assess the accuracy of our technique.

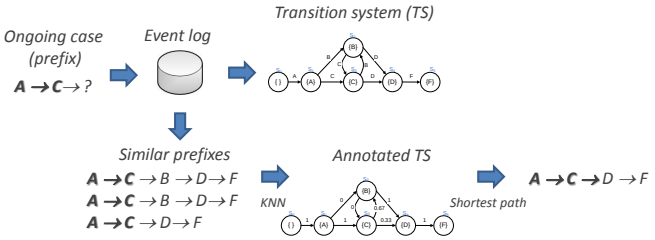


Fig. 2: Overview of the proposed approach for annotating transition systems.

Estimation of transition probabilities. A core part of the approach is the estimation of transition probabilities. This is done via KNN classifiers, which take as input the attributes of a running instance and search for instances with a similar prefix in the training set. In this work, we only consider control-flow attributes to find similar prefixes.

We define similarity metrics between two sequences x_1 and x_2 based on the Damerau-Levenshtein distance. The latter is computed as the minimum number of edit operations required to transform one string into the other, where an operation can be insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters. The Damerau-Levenshtein distance is defined for strings. To use it for event sequences, we need to convert them into strings so that each event is mapped onto a character. However, since the sequences can be of very different length, we normalize the distance by the length of the longer sequence and, for convenience, inverse it. We will refer to this metrics as Damerau-Levenshtein similarity $f_{sim}(x_1, x_2)$:

$$0 \leq f_{sim}(x_1, x_2) = 1 - \frac{d(x_1, x_2)}{max(|x_1|, |x_2|)} \leq 1. \tag{2}$$

Additionally, due to the frequent occurrence of ties (i.e. sequences having the same similarity), instead of fixing parameter k in KNN, we fix a certain similarity threshold f^0_{sim} and assume all the training prefixes with a similarity with a test prefix above the threshold are its nearest neighbors.

Having found similar prefixes, based on their continuations (i.e. suffixes), we compute the transition probabilities starting from a current state until the process ends. Consider the event log in Figure 1. Assuming $f^0_{sim} = 1$, there are three nearest neighbors in the log for a partial test trace $\langle A, C \rangle$. Their continuations are $\langle B, D, F \rangle$ and $\langle D, F \rangle$, the former being twice as likely. Consequently, Figure 3 shows a transition system annotated with transition probabilities, starting from the current state s_3. The probability of moving from s_2 to s_3 is 0, since,

even though the corresponding transition is present in the training set, it is not found in the closest neighbors of $\langle A, C \rangle$.

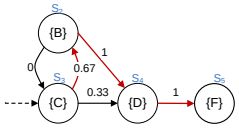


Fig. 3: Annotated transition system, with a current state s_3. $f^0_{sim} = 1$

Prediction of the most likely path. Given a transition system annotated with the transition probabilities, we can find a probability of any sequence of activities $S = \langle s_1, s_2, \ldots, s_n \rangle$ by taking the product of transition probabilities between every pair of connecting states:

$$P_S = \prod_{i=1}^{n-1} p_i \tag{3}$$

At runtime, we map a partial trace of an ongoing instance to the current state in the transition system. To find the most likely future sequence from its current state until its completion, we use the idea by Polato et al. [13], who transform the problem of finding the sequence with the highest probability into the shortest path problem. Indeed, an annotated transition system can be viewed as a weighted directed graph, where nodes are states, edges are possible transitions, and weights are transition probabilities. Consequently, we would like to find the shortest path between a current node and the target node, which corresponds to the final state of the process. However, in the shortest path problem, the *sum* of edge weights, rather than the product, is to be minimized. Thus, instead of using transition probabilities as the edge weights, we use their logarithm. Furthermore, to ensure that the transitions with low probability correspond to edges with high weights and vice versa, we take the absolute value of the logarithm of transition probability. Then we have to minimize the sum:

$$\sum_{i=1}^{n-1} |\log p_i| \tag{4}$$

It should be noted that in the shortest path problem we have to explicitly specify current node and target node. To account for the fact that processes may have multiple final states, we add a single "exit" state, and connect each possible final state to the "exit". Then the transition probability from each final state to the "exit" will be the probability of an instance ending with this state.

4.2 Recurrent Neural Networks

In traditional, feed-forward, networks, all inputs and outputs are independent of each other. However, for many kinds of temporal sequences, including business processes, this is not an accurate representation, as sequence elements (events) are interdependent. Therefore, in this work, we use *recurrent* neural networks

(RNN). In RNNs, connections between neurons form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Thus, RNNs can use their internal memory to process arbitrary sequences of inputs [7].

RNNs, once unfolded in time (Figure 4), can be viewed as very deep feed-forward networks. x_t is the input at time step t. For example, x_1 can be a vector corresponding to the second event of a sequence. Then, s_t is the hidden state at time step t responsible for the memory capability of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. Function f, known as activation function, is usually nonlinear, such as hyperbolic tangent. o_t is the output at step t. Parameters U, V, W are fixed across all steps.

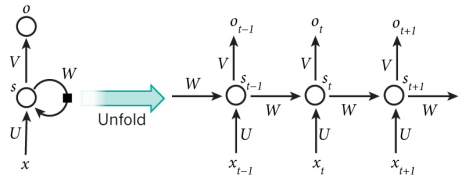


Fig. 4: A simple recurrent neural network [16].

Similarly to feed-forward networks, to train RNNs, we use backpropagation algorithm. We apply neural networks in a sliding window fashion, such that the data attributes events $1 \dots n$ are used to predict the most likely $(n+1)$-th event. Then based on the attributes of events $2 \dots (n+1)$, we predict $(n+2)$-th event and so on. To tune the neural network's hyperparameters we apply a grid search.

5 Evaluation

We implemented the two proposed future path prediction approaches (ATS-KNN and RNN) as a set of Python scripts packaged under the name of *ProSeqPredict*,[1] and applied them to four event logs. In this section, we describe the characteristics of the logs; we report on the accuracy of the predictive models and compare the results with the baseline approach by Polato et al. [13].

5.1 Event logs

To evaluate our approaches, we selected three real-life event logs and one synthetic log. Table 1 summarizes the main characteristics of the logs.

The first event log (*Env permit*) originates from an environmental permit request process carried out by a Dutch municipality, available as part of the *CoSeLoG* project [17]. The second log (*Hospital*) is taken from the BPI 2011

[1] Available at `http://apromore.org/platform/tools`

Dataset	Number of cases	Median trace length	Number of trace variants	Number of events	Event classes
Env permit	937	43	916	38,944	381
Hospital	225	93	217	29,694	316
Insurance	1065	12	925	16,869	9
Synthetic	2000	11	19	26,169	25

Table 1: Summary of the event logs.

challenge and contains events related to treatment and diagnosis steps for patients diagnosed with cancer in a Dutch Academic Hospital [18]. For faster processing, we took a random subset of this log containing 20% of the traces. The third log (*Insurance*) is taken from a large Australian insurance company and contains instances of an insurance claims handling process. All three logs have a high degree of variability between cases – the number of trace variants is almost equal to the number of cases (Table 1). Finally, the fourth log has been synthetically generated and contains only a limited number of trace variants.

5.2 Evaluation procedure

To evaluate the two proposed approaches we trained the models and measure their predictive power on each event log. As a baseline to compare against, we used the technique developed by Polato et al. in [13].

For the ATS-KNN approach, we randomly split each log such that 80% of the cases are used as a training set and the remaining 20% are used to test the model performance. Then we construct a transition system using the Fuzzy miner [19] which works well with complex and highly variable logs. Next, to estimate the optimal value of the similarity threshold f^0_{sim} for KNN, we perform 3-fold cross-validation on the training set. In this paper, we used the Python implementation of RNNs available in the `Keras` library.

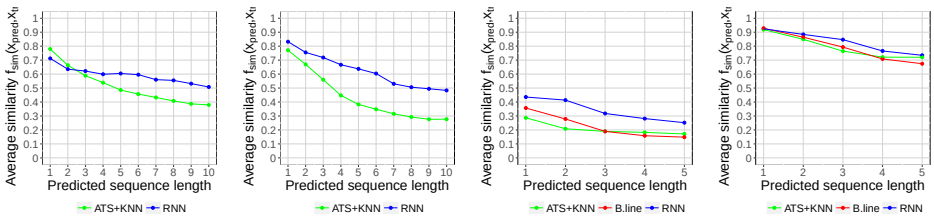
At runtime, for each instance in the test set, we chose a random state ("current state") and predict the future sequences of states up to the instance completion. However, for the sake of the evaluation and given the sensitivity of the evaluation measure to the length of the sequences, we limited predictions to a specific horizon by truncating the predicted sequence to the specific length N. For the event logs with longer traces (Environmental permit and Hospital), we evaluated predictions for sequence lengths from 1 to 10, while for the other logs we chose the lengths from 1 to 5.

We assessed the prediction accuracy using the previously defined Damerau-Levenshtein similarity between predicted x_{pred} and the actual sequences x_{true}. For perfectly matching sequences, i.e. in case of ideal prediction, $f_{sim}(x_{pred}, x_{true}) = 1$.

5.3 Results and discussion

Figure 5 reports the similarity between predicted and true sequences of future events averaged over all test cases. The technique in [13], provided as a baseline, was evaluated only on the Insurance and Synthetic event logs, since we were not able to run it on the two other logs due to the higher number of state transitions

in those logs. Nevertheless, we notice that for longer predicted sequences, the ATS-KNN approach slightly outperforms the baseline. However, for shorter sequences, the baseline is more accurate. Meantime, the RNN approach provides the most accurate predictions across all logs, except for very short sequence lengths (up to two events) in the case of the Hospital log, where ATS-KNN slightly outperforms RNN.



(a) (b) (c) (d)

Fig. 5: Average Damerau-Levenshtein similarity for the predicted sequences of different lengths using the Environmental permit (a), Hospital (b), Insurance (c) and Synthetic (d) datasets, for ATN+KNN, RNN and the baseline approach.

Our experiments also highlight that prediction accuracy heavily depends on the underlying business process – complex and highly variable processes are much more difficult to predict. For example, in the Insurance log, the next predicted event matches the actual one only in 30% of cases. As a further example, consider the Fuzzy net of the synthetic event log shown in Figure 6. In this model the first seven events are always the same for all the process instances, i.e. the process starts with a sequence of seven activities. Thus, the prediction of the next event until after the first six events will always be 100% accurate, but the accuracy will drop after the seventh event as there are multiple outgoing paths in the corresponding process model. This is evident from Table 2, which lists the prediction accuracy of the next N events given different prefix sizes, for the synthetic log. The reported results are for the ATS+KNN approach; for RNN we obtained qualitatively the same results. In this table, we can see that the prediction accuracy of 1 drops to 0.496 at prefix length of 7, due to the branching point after the seventh event in the model of Figure 6. The accuracy goes back to 1 after the eight event because there are no branching points at this event, and drops again below 1 at the ninth event, due to the existence of a second branching point. These results clearly also affect the prediction accuracy for suffixes of longer lengths.

Finally, one can also notice that the prediction accuracy is dependent on the length of the sequence to be predicted. Naturally, the complexity of the prediction task grows with the length of the sequence to be predicted. However, the accuracy eventually plateaus, as the number of possible events to choose the predictions from is limited. This is for example visible in the case of the Insurance and synthetic datasets in Figure 5.

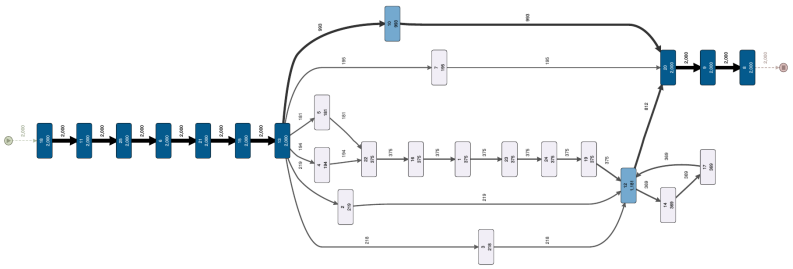


Fig. 6: Fuzzy net of the synthetic log.

Table 2: ATS+KNN: avg. similarity between predicted and true sequences based on difference combinations of prefix and suffix lengths, for the synthetic log.

Prefix length	**Predicted suffix length,** N				
	1	2	3	4	5
3	1.000	1.000	1.000	1.000	0.899
4	1.000	1.000	1.000	0.874	0.818
5	1.000	1.000	0.832	0.773	0.737
6	1.000	0.748	0.697	0.671	0.679
7	0.496	0.545	0.561	0.599	0.174
8	1.000	0.949	0.933	0.813	0.720
9	0.899	0.899	0.751	0.650	0.689

6 Conclusion and Future Work

In this paper, we tackled the problem of predicting the most likely future activity sequence of a running process instance starting from its current state and up to its completion. We considered two approaches to this problem. The first one is based on the construction of transition systems annotated with transition probabilities learned from k-nearest neighbors classifiers. The second stems from casting the problem of activity sequence prediction as a structured output prediction problem and applying a general-purpose technique for this purpose, namely recurrent neural networks.

The evaluation on real-life and synthetic event logs showed that the approach based on recurrent neural networks outperforms the one based on transition systems and k-nearest neighbors. The latter in turn slightly outperforms the approach of Polato et al. [13], which constructs an annotated transition system using a naive Bayes classifier instead of a k-nearest neighbors one. This latter observation applies when predicting longer activity sequences.

An avenue for future work is to take advantage of event attributes, such as timestamps and resource information. Another avenue for future work is to apply alternative techniques from the field of Natural Language Processing (NLP). It has been shown that methods developed in the NLP domain, specifically n-gram-based probabilistic models and spectral learning of weighted finite automata, can be adapted to the problem of predicting future sequences of tokens [20]. There is potential in adapting these techniques to the activity sequence prediction.

Acknowledgments. This research is funded by the Australian Research Council Discovery Project DP150103356 and the Estonian Research Council.

References

1. Becker, J., Breuker, D., Delfmann, P., Matzner, M.: Designing and implementing a framework for event-based predictive modelling of business processes. Proc. of EMISA (2014) 71–84
2. Ferreira, D.R., Vasilyev, E.: Using logical decision trees to discover the cause of process delays from event logs. Computers in Industry **70** (2015) 194–207
3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
4. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Proc. of ICSOC. Springer (2013) 389–403
5. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Proc. of CAiSE, Springer (2014) 457–472
6. Subramaniam, S., Kalogeraki, V., Gunopulos, D.: Business Processes: Behavior Prediction and Capturing Reasons for Evolution. In: Proc. of ICEIS (vol. 3). (2006) 3–10
7. Ioannou, P., Casey, M., Grüning, A.: Factors Influencing Polychronous Group Sustainability as a Model of Working Memory. Springer (2014)
8. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowledge and Information Systems **42**(1) (2013) 97–126
9. Pravilovic, S., Appice, A., Malerba, D.: Process mining to forecast the future of running cases. In: Proc. of the 2nd International Workshop on New Frontiers in Mining Complex Patterns, Springer (2014) 67–81
10. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems **56** (2016) 235–257
11. van der Aalst, W.M.P., Schonenberg, M.H., Song, M., der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems **36**(2) (2011) 450–475
12. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-aware remaining time prediction of business process instances. In: Proc. of IJCNN, IEEE (2014) 816–823
13. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and Activity Sequence Prediction of Business Process Instances. arXiv preprint arXiv:1602.07566 (2016)
14. Van Der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., Van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: A two-step approach to balance between underfitting and overfitting. Software and Systems Modeling **9**(1) (2010) 87–111
15. Gou, J., Du, L., Zhang, Y., Xiong, T., Others: A new distance-weighted k-nearest neighbor classifier. J. Inf. Comput. Sci **9**(6) (2012) 1429–1436
16. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553) (2015) 436–444
17. Buijs, J.: 3TU.DC Dataset: Receipt phase of an environmental permit application process (WABO)
18. van Dongen, B.: 4TU Dataset: Real-life event logs - Hospital log `https://data.4tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`. Accessed: 2016-05-30.
19. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: Proc. of BPM. Volume 4714 of LNCS., Springer (2007) 328–343
20. SPiCe: Sequence PredIction ChallengE `http://spice.lif.univ-mrs.fr`.