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

Time and Activity Sequence Prediction of Business Process Instances

Mirko Polato · Alessandro Sperduti · Andrea Burattin · Massimiliano de Leoni

the date of receipt and acceptance should be inserted later

Abstract The ability to know in advance the trend of running process instances, with respect to different features, such as the expected completion time, would allow business managers to timely counteract to undesired situations, in order to prevent losses. Therefore, the ability to accurately predict future features of running business process instances would be a very helpful aid when managing processes, especially under service level agreement constraints. However, making such accurate forecasts is not easy: many factors may influence the predicted features. Many approaches have been proposed to cope with this problem but, generally, they assume that the underlying process is stationary. However, in real cases this assumption is not always true. In this work we present new methods for predicting the remaining time of running cases. In particular we propose a method, assuming process stationarity, which achieves state-of-the-art performances and two other methods which are able to make predictions even with non-stationary processes. We also describe an approach able to predict the full sequence of activities that a running case is going to take. All these methods are extensively evaluated on different real case studies.

Keywords process mining · prediction · remaining time · machine learning

M. Polato
University of Padua, Torre Archimede, via Trieste 63, 35121, Padova, Italy
Tel.:(+39) 049 827 1437
E-mail: mpolato@math.unipd.it

A. Sperduti
University of Padua, Torre Archimede, via Trieste 63, 35121, Padova, Italy
E-mail: sperduti@math.unipd.it

A. Burattin
DTU, Technical University of Denmark, Kgs. Lyngby, Denmark
E-mail: andbur@dtu.dk

M. de Leoni
Eindhoven University of Technology, Eindhoven, The Netherlands
E-mail: m.d.leoni@tue.nl

1 Introduction

An increasing number of companies are using Process Aware Information Systems (PAIS) to support their business. All these systems record execution traces, called *event logs*, with information regarding the executed activities. In typical scenarios, these traces do not only contain which activities have been performed, but also additional *attributes*. The extraction of useful information out of large amount of data is the main challenge of the *data mining* field. However, recently, a new topic branched off data mining: *process mining* [1,18]. This latter case expects that the analyzed data are specifically referring to executions of business processes, and this assumption is made throughout the analysis phases. Frequently, under the umbrella of the term "process mining", three main activities are distinguished: the discovery of models, such as control-flow, describing the actual process undergoing (namely, *process discovery*); the assessment of the conformance of a given event log with respect to a given process model (namely, *conformance checking*); and the extension of existing process models with additional information (namely, *process enhancement*). However, apart from these classical activities, it is also possible to exploit the event log in order to create *predictive models*, i.e., models that are useful to predict future characteristics of incomplete process instances. In general, it is useful to distinguish two types of process mining techniques, according to *when* the analysis takes place: *(i)* a-posteriori; and *(ii)* at runtime. A-posteriori, or "off-line", techniques use a finite portion of historical data to extract knowledge out of it. Runtime, or "on-line", approaches, on the other hand, give information as long as the business process is running. It is more difficult to define approaches belonging to the latter category: the process generating the data or the frequency of events emission, may be subject to modifications or drifts (which may also be seasonal). To tackle these on-line problems, it is therefore necessary to use tools able to adapt to new or unobserved scenarios. One of the most challenging task in process mining is *prediction*. Obviously, predictions are useful only if the object of such predictions have not been observed yet. For this reason, prediction *per se* is, inherently, a task performed on incomplete traces, at runtime. Moreover, the ability to predict the evolution of a running case is difficult also due to the peculiarities of each process instance and, because of the "human factor", which might introduce strong flexibility.

In this work, we focus on predicting the remaining time of running cases. We decided to ground our approach not only on the control flow, but also on additional data that we could observe. Moreover, the system we present is also capable of dealing with unexpected scenarios and evolving conditions. With respect to our seminal work [28], here we propose a slightly modified version of that method and we also propose two novel approaches able to overcome its limitations. In particular, we are going to define two different scenarios: in the first one we assume the process has a well defined static workflow, the same assumption made in [28], while in the latter we remove such assumption and the process is considered dynamic, e.g., the process has seasonal drift. We will show that the proposed approaches improve state-of-the art performances in the first scenario and one of them is also able to deal with dynamic processes. We also leverage one of these models to predict the future sequence of activities of a running case. We assess the remaining time prediction accuracy of our methods against the state-of-the art ones using three real-life case studies concerning a process of an Italian software company, the

management of road-traffic fines by a local police office of an Italian municipality and the BPI 12 challenge. The remainder of this paper is structured as follows: Section 2 reviews recent works concerning prediction tasks in the framework of process mining. Section 3 gives some essential background on process mining and machine learning concepts used throughout the paper, while Section 4 describes the prediction approaches. Section 5 discusses the implementation and the experimental results and finally Section 6 briefly summarizes the presented content and concludes the paper.

2 Related Work

The first framework focused on the time perspective has been proposed by van der Aalst et al., in [3]. They describe a prediction approach which extracts a transition system from the event log and decorates it with time information extracted from historical cases. The transition system consists in a finite state machine built with respect to a given abstraction of the events inside the event log. The time prediction is made using the time information (e.g., mean duration of specific class of cases) inside the state of the transition system corresponding to the current running instance. A closely related approach is presented in [7], where the same idea of a decorated (a.k.a. annotated) transition system is used. In this approach the annotations are decision trees used to predict the completion time and also the next future activity of the process instance. The work presented in [20] considers the data perspective in order to identify SLAs (Service Level Agreement) violations. In this work, authors try to estimate the amount of unknown data, in order to improve the quality of the final prediction. The actual forecast is built using a multilayer perceptron, trained with the backpropagation algorithm. Two works by Folino et al. [12, 13] report an extended version of the technique described in [3]. In particular, they cluster the log traces according to the corresponding "context features" and then, for each cluster, they create a predictive model using the same method as in [3]. The clustering phase is performed using predictive clustering trees. In order to propose a forecast, the approach clusters the new running instance and then uses the model belonging to the specific cluster. One of the weaknesses of these methods based on [3] is that they assume a static process, where the event log used for the training phase contains all the possible process behaviours. Unfortunately, in real life cases this assumption is usually not valid. [19] reports an approach which uses the *Instance-specific Probabilistic Process Models* (PPM) and is able to predict the likelihood of future activities. Even though the method does not provide a time prediction, it gives to the business managers useful information regarding the progress of the process. This work also shows that the PPM built is actually Markovian. A probabilistic method, based on Hidden Markov Model (HMM), is also proposed in [27]. Results show how the proposed method outperforms the state-of-the-art. However, the achieved results seem to be not much different from the ones achieved by [3]. Ghattas et al., in a recent work [15], exploit *Generic Process Model* and decision trees, based on the process context, to provide decision criteria defined according to the actual process goals. In [21], de Leoni et al. propose a general framework able to find correlation between business process characteristics. They manipulate the event log in order to enrich it with derived information, and then generate a decision tree in order to discover

correlations. In particular, one of the correlation problem suggested here is the forecast of the remaining time of running cases. Being based on decision trees, numeric values need to be discretized and this lower the accuracy of the method. For this reason, they do not provide any prediction example. In [28], Polato et al. show an approach based on [3] in which the additional attributes of the events are taken into account in order to refine the prediction quality. This method exploits the idea of annotating a transition system adding machine learning models, such as Naïve Bayes and Support Vector Regressors. The experimental results show how the additional attributes can influence positively the prediction quality.

In this paper we propose two new approaches based on Support Vector Regression and we discuss their strengths and their weaknesses compared with the approaches presented in [3] and [26]. In particular we emphasize in which scenarios an approach is better than the others and why. A similar approach is presented in [7], in which authors create a process model similar to a transition system, in which non frequent behaviours of the process are discarded. Predictions are made on top of this model using decision trees. This method is very similar to the one presented in [28], the main difference is the process model which is a kind of reduced transition system. More recently, deep neural network based approaches have been proposed. In [11], authors present a recurrent neural network (RNN) for the prediction of the next event, however they do not predict the remaining time. Their network is composed by two hidden RNN layers using basic LSTM cells (Long-short term memory). To train the model, GPUs have been used because of the size of the deep network. Another recent approach based on LSTM is the one by Tax et al. [26]. In that work authors propose a LSTM neural network architecture for predicting the next activity and its execution time, from a partial trace. Such a network can be adopted for remaining time prediction by iteratively predicting the next activity and its duration, until a special *end of case* activity is predicted. Both the cited deep NN based methods do not take into account additional attributes. A known problem of the deep NN based approaches is their training time and their sensitivity to the hyperparameters choice [9].

Approaches coming from different areas can also be used to achieve similar results. For example, queue theory [17, 5] and queue mining can be seen as a very specific type of process mining, and recent works are starting to aim for similar prediction purposes [30]. In this particular case, authors decided to focus on the delay prediction problem (i.e., providing information to user waiting in lines, in order to improve customer satisfactions). The method presented here is based on the construction of an annotated transition system. Such model is then used to make delay predictions using simple averages or non-linear regression algorithms. They also show other methods based on queue mining techniques. Even though this approach concerns making predictions on the time perspective, the slant is totally different and the goal is more narrow with respect to the aim of the approach we propose in this work. Another example of prediction-focused paper has recently been published by Rogge-Solti and Weske [29]. In this case the focus is on the prediction of the remaining time, in order to avoid missing deadlines. However, only information about the workflow are used to build the model and to make prediction.

Remaining time prediction is a particular instantiation of the so-called predictive monitoring task, in which the goal is to provide early advice so that users can steer ongoing process executions towards the achievement of business constraints.

Maggi et al. [23] proposed a decision tree based predictor model. The presented framework continuously estimate how likely is that a user defined constraint it will be fulfilled by the ongoing process instances. The same task is faced in [22]. Here authors present an alternative approach where traces are treated as complex symbolic sequences. They propose two possible encodings: *(i)* index-based and *(ii)* a combination of the first one and an HMM based one. From an Area Under the Roc Curve perspective the proposed encoding outperforms the simple baselines (i.e., boolean, frequency-based). Using the just mentioned encoding, Verenich et al. [35] proposes a two phases approach: in the first phase the dataset is divided into groups by means of unsupervised clustering; in the second phase a predictor is trained for each cluster. In particular they use random forests as the predictor algorithm. A very similar approach is presented in [14]. Finally, Teinemaa et al. [33] show how textual information, when sufficiently large, can be useful to improve accuracy and earliness. Even in this work the best performing predictors are random forests.

3 Background

In this section we define some useful process mining concepts which we will use throughout the paper. First of all we give the concept of sequence and then the basic definition of *event*, *trace* and *event log*.

Case Id	Timestamp	Resource	Activity	Category	Amount
65923	20-02-2002:11.11	Jack	A	-	1000
65923	20-02-2002:13.31	Jack	B	Gold	1000
65923	21-02-2002:08.40	John	C	Gold	900
65923	22-02-2002:15.51	Joe	F	Gold	900
65924	19-02-2002:09.10	Jack	A	-	200
65924	19-02-2002:13.22	John	B	Standard	200
65924	20-02-2002:17.17	John	D	Standard	200
65924	21-02-2002:10.38	Joe	F	Standard	200

Table 1 Example of an event log fragment with events sorted using their Timestamp and grouped by Case Id.

Given a set A, a finite sequence over A of length n is a mapping $s \in \mathbb{S} : ([1, n] \subset \mathbb{N}) \to A$, and it is represented by a string, i.e., $s = \langle s_1, s_2, \ldots, s_n \rangle$, $|s| = n$. Over a sequence s we define the following functions:

- *selection operator* $(\cdot)$: $s(i) = s_i$, $\forall\, 1 \leq i \leq n$;
- $hd^k(s) = \langle s_1, s_2, \ldots, s_{\min(k,n)} \rangle$;
- $tl^k(s) = \langle s_w, s_{w+1}, \ldots, s_n \rangle$ where $w = \max(n - k + 1, 1)$;
- $set(s) = \{s_i \mid s_i \in s\}$, e.g., $set(\langle a, b, b, c, d, d, d, e \rangle) = \{a, b, c, d, e\}$.

Definition 1 (Event) An *event* is a tuple $e = (a, c, t, d_1, \ldots, d_m)$, where $a \in \mathcal{A}$ is the process activity associated with the event, $c \in \mathcal{C}$ is the *case id*, $t \in \mathbb{N}$ is the event timestamp (seconds since 1/1/1970[1]) and $d_1, \ldots, d_m$ is a list of additional attributes values, where $\forall 1 \leq i \leq m$, $d_i \in \mathcal{D}_i$, with $\mathcal{D}_i$ the domains of the additional attributes. We call $\mathcal{E} = \mathcal{A} \times \mathcal{C} \times \mathcal{T} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$ the *event universe*.

[1] We assume this representation according to the Unix epoch time.

Over an event e we define the following *projection* functions: $\pi_{\mathcal{A}}(e) = a$, $\pi_{\mathcal{C}}(e) = c$, $\pi_{\mathcal{T}}(e) = t$ and $\pi_{\mathcal{D}_i}(e) = d_i, \forall\, 1 \leq i \leq m$. If e does not contain the attribute value d_i for some $i \in [1, m] \subset \mathbb{N}$, $\pi_{\mathcal{D}_i}(e) = \perp$. In the remainder of this paper we will call the number of additional attribute m as $|\mathcal{D}|$.

Definition 2 (Trace, Partial Trace) A *trace* is a finite sequence of events $\sigma_c = \langle e_1, e_2, \ldots, e_{|\sigma_c|} \rangle \in \mathcal{E}^*$ such that $\forall\, 1 \leq i \leq |\sigma_c|, \pi_{\mathcal{C}}(e_i) = c \wedge \forall\, 1 \leq j < |\sigma_c|$, $\pi_{\mathcal{T}}(\sigma_c(j)) \leq \pi_{\mathcal{T}}(\sigma_c(j+1))$. We define a *partial trace* of length k as $\sigma_c^k = hd^k(\sigma_c)$, for some $k \in [1, |\sigma_c|] \subset \mathbb{N}$. We call Σ the set of all possible (partial) traces.

While a trace corresponds to a complete process instance, i.e., an instance which is both started and completed; a partial trace represents a process instance which is still in execution and hence it has not completed yet. Over a trace $\sigma_c = \langle e_1, e_2, \ldots, e_{|\sigma_c|} \rangle$ we define the following projection functions: $\mathit{\Pi}_{\mathcal{A}}(\sigma_c) = \langle \pi_{\mathcal{A}}(e_1), \pi_{\mathcal{A}}(e_2), \ldots, \pi_{\mathcal{A}}(e_{|\sigma_c|}) \rangle$, $\mathit{\Pi}_{\mathcal{T}}(\sigma_c) = \langle \pi_{\mathcal{T}}(e_1), \pi_{\mathcal{T}}(e_2), \ldots, \pi_{\mathcal{T}}(e_{|\sigma_c|}) \rangle$ and $\mathit{\Pi}_{\mathcal{D}_i}(\sigma_c) = \langle \pi_{\mathcal{D}_i}(e_1), \pi_{\mathcal{D}_i}(e_2), \ldots, \pi_{\mathcal{D}_i}(e_{|\sigma_c|}) \rangle$ for all $1 \leq i \leq |\mathcal{D}|$.

Definition 3 (Event log[3]) An *event log* L is a set of traces, $L = \{\sigma_c \mid c \in \mathcal{C}\}$ such that each event appears at most once in the entire log, i.e., $\forall \sigma_1, \sigma_2 \in L, \sigma_1 \neq \sigma_2 : \mathit{set}(\sigma_1) \cap \mathit{set}(\sigma_2) = \emptyset$.

A transition system is one of the simplest process modelling notations, it consists of *states* and *transitions* where each transition connects two states (not necessarily different). Formally, it is defined as follows:

Definition 4 (Transition System (TS)) A *transition system* is a triplet $TS = (S, A, T)$, where S is the set of *states*, $A \subseteq \mathcal{A}$ is the set of activities and $T \subseteq S \times A \times S$ is the set of *transitions*. $S^{start} \subseteq S$ is the set of *initial states*, and $S^{end} \subseteq S$ is the set of *final (accepting) states*.

Given a state $s \in S$, it is possible to define the set of reachable states from s as: $s\bullet = \{s' \in S \mid \exists t \in T, \exists a \in A \text{ s.t. } t = (s, e, s')\}$. According to [3], to construct a transition system which maps each partial trace in the log to a state, we need the so called *event* and *state representation functions*.

Definition 5 (Event representation function) Let $\mathcal{R}_e$ be the set of possible event representations. An event representation function $f^{event} \in \mathcal{E} \to \mathcal{R}_e$ is a function that, given an event e produces some representation of it (e.g., projection functions over $e \in \mathcal{E}$: $\pi_{\mathcal{A}}(e), \pi_{\mathcal{T}}(e)$).

Definition 6 (State representation function) Let $\mathcal{R}_s$ be the set of possible state representations, a state representation function $f^{state} \in \Sigma \to \mathcal{R}_s$ is a function that, given a (partial) trace σ returns some representation of it (e.g., sequences, sets, multiset over some event properties).

Even though conceptually these representation functions are not directly correlated, in practice, f^{state} needs an f^{event} in order to create a meaningful state representation. Choosing the right functions f^{state} and f^{event}, also referred to as *abstractions*, is not a trivial task. The main issue with the representation choice is the overfitting vs. underfitting problem as discussed in [2,3] where authors also propose possible good choices for f^{state} and f^{event}. A common event abstraction is

$f^{event}(e) = \pi_{\mathcal{A}}(e)$, which maps an event onto the name of the activity, while commons state abstractions are: the *set abstraction*, i.e., $f^{state}(\sigma_c) = \{\pi_{\mathcal{A}}(e) \mid e \in \sigma_c\}$, the *multiset abstraction*, i.e., $f^{state}(\sigma_c) = \{(a, m) \mid a = \pi_{\mathcal{A}}(e) \wedge m = |\Pi_{\mathcal{A}}(\sigma_c) \uparrow \{a\}|\}$ and the *list abstraction*, i.e., $f^{state}(\sigma_c) = \langle \pi_{\mathcal{A}}(\sigma_c(1)), \ldots, \pi_{\mathcal{A}}(\sigma_c(|\sigma_c|))\rangle$.

Using these two functions f^{event} and f^{state}, it is possible to define a (labeled) 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}.

Definition 7 (Labeled Transition System (LTS)) Given a state representation function f^{state}, an event representation function f^{event} and an event log L, we define a Transition System as $LTS = (S, E, T)$, where:

- $S = \{f^{state}(hd^k(\sigma)) \mid \sigma \in L \wedge 0 \leq k \leq |\sigma|\}$ is the state space;
- $E = \{f^{event}(\sigma(k)) \mid \sigma \in L \wedge 1 \leq k \leq |\sigma|\}$ is the set of event labels;
- $T(\subseteq S \times E \times S) = \{(f^{state}(hd^k(\sigma)),\ f^{event}(\sigma(k+1)),\ f^{state}(hd^{k+1}(\sigma))) \mid \sigma \in L \wedge 0 \leq k < |\sigma|\}$ is the transition relation.

$S^{start} = \{f^{state}(\langle\rangle)\}$ is the set with the unique initial state, and $S^{end} = \{f^{state}(\sigma) \mid \sigma \in L\}$ is the set of final (accepting) states.

We say that a trace is *compliant* with the transition system if it corresponds to a walk from $s_i \in S^{start}$ to $s_e \in S^{end}$. We also call a trace σ *non-fitting* with respect to a transition system if $f^{state}(\sigma) \notin S$.

Figure 1 depicts an example of a transition system extracted from the log fragment reported in Table 1.

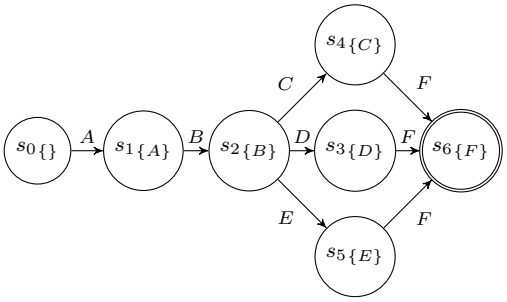

Fig. 1 Example of a transition system extracted from a log containing three trace types $\langle A, B, C, F\rangle$, $\langle A, B, D, F\rangle$ and $\langle A, B, E, F\rangle$, with $f^{event}(e) = \pi_{\mathcal{A}}(e)$ and $f^{state}(\sigma) = \{f^{event}(\sigma(|\sigma|))\}$. The state s_0 is the initial state, while s_6 is the accepting (i.e., final) state. The notation $s_{1\{A\}}$ means that the state s_1 has a state representation equals to $\{A\}$. Each transition is labeled by the corresponding event representation value.

4 Remaining Time Prediction

In this section we show a spectrum of different approaches able to predict the remaining time of running business process instances. In particular, we will emphasize the pros and cons of each approach and which are the situations in which an approach should be preferred among the others.

The problem of predicting the remaining time of running process instances can be summarized as follows: given an event log, containing historical traces

about the execution of a business process, we want to predict, for an ongoing process instance, how much time remains until its completion. All the approaches described in this work are based on the idea of making predictions using a model constructed (i.e., learned) using the information collected so far (i.e., the event log). The obtained model takes the partial trace, which represents the running process instance, as input and returns the remaining time forecast. The remaining time of a case consists of the time that will be spent from now up to the end of the execution of the last activity of the process. This amount of time, given a complete trace σ_c, is easily computable for each event $e_i \in \sigma_c$. We define the function $rem : \Sigma \times \mathbb{N} \to \mathbb{N}$ as follows: $rem(\sigma_c, i) = \pi_{\mathcal{T}}(\sigma_c(|\sigma_c|)) - \pi_{\mathcal{T}}(\sigma_c(i))$, where i is an event index. If $\sigma_c = \langle\rangle$ or $i \notin \{1, 2, \ldots, |\sigma_c|\}$ then $rem(\sigma_c, i) = 0$. This function calculates the time difference between the last event in the trace and the i-th event. In the remainder of this section we present our new time prediction approaches based on the application of machine learning models. Moreover, we discuss how to exploit one of the proposed model to predict also the future sequence of activities.

4.1 **Approach 1**: Simple Regression

This prediction task has all the characteristics to be faced as a regression problem. (Partial) Traces in the event log are the input vectors and the remaining time at each event is the target value. In this section we present a direct application of ϵ-SVR algorithm. Despite the simplicity of this method, there are some aspects which need particular attention. First of all we describe how to switch from the event log to a representation suitable for the ϵ-SVR algorithm.

In our setting, the input consists of traces and, in particular, the attributes of the corresponding events. Whereas SVR takes as input vectors $\mathbf{x} \in \mathbb{R}^l$, for some $l \in \mathbb{N}^+$, we have to convert sequences of events into some representation in $\mathbb{R}^l$. Let us consider a (partial) trace $\sigma_c = \langle e_1, e_2, \ldots, e_n \rangle$ of length $n \in \mathbb{N}^+$. For each event $e_i = (a^i, c, t^i, d_1^i, \ldots, d_m^i) \in \sigma_c^k$, the attributes $d_1^i, \ldots, d_m^i$ may have different values, because they can change as the process instance evolves. We consider as additional attributes values the last values chronologically observed. Formally, we define the function $last : \Sigma \times \mathbb{N} \to \mathcal{D} \cup \{\perp\}$ as:

$last(\sigma_c, i) = \left\{ \pi_{\mathcal{D}_i}(e_j) \mid j = \arg\max_{1 \leq h \leq |\sigma_c|} \pi_{\mathcal{D}_i}(e_h) \neq \perp \right\}$, where i is the index of the attribute. If there is no index j such that $\pi_{\mathcal{D}_i}(\sigma_c(j)) \neq \perp$ then $last(\sigma_c, i) = \perp$. What we need to do next is to transform the domains $\mathcal{D}_i$ into a numerical representation that can be given as input to the SVR. In order to do that, we use the *one-hot* encoding for all nominal attributes and for the activity. This encoding converts the nominal data value d_i into a binary vector $v \in \{0, 1\}^{|\mathcal{D}_i|}$, with $|\mathcal{D}_i|$ components and with all values set to 0 except for the component referring to d_i, which is set to 1. For example, given $\mathcal{D}_j = \{a, b, c, d\}$ and $\pi_{\mathcal{D}_j}(e) = b$, the one-hot encoding would return the vector $\mathbf{v} = [0, 1, 0, 0] \in \{0, 1\}^4$, where $\mathbf{v}_1$ corresponds to the value a, $\mathbf{v}_2$ to the value b, and so on[2]. Instead, all the other attributes values d_j such that $\mathcal{D}_j \subseteq \mathbb{R}$ are simply put in a single component vector, e.g., let $\mathcal{D}_i \equiv \mathbb{N} \subset \mathbb{R}$ and $\pi_{\mathcal{D}}(e) = 17$, the output vector is $\mathbf{u} = [17] \in \mathbb{R}$. In the remainder of this paper

[2] We assume that a fixed order is always available for attribute's values (for example, the lexicographical order)

we will call $\mathbb{1} : A \to \mathbb{R}^n$, for some $n \in \mathbb{N}^+$, the function which maps an attribute value to its one-hot encoded vector.

After the just described transformation, all vectors are concatenated (keeping the same fixed order) together. We use the symbol $\mathbf{v}||\mathbf{w}$ as the concatenation operator between the vectors $\mathbf{v}$ and $\mathbf{w}$. , e.g., recalling $v \in \mathbb{R}^4$ and $u \in \mathbb{R}$ from the previous examples, their concatenation is equal to $\mathbf{z} = \mathbf{v} \,||\, \mathbf{u} = [0,1,0,0]||[17] = [0,1,0,0,17] \in \mathbb{R}^5$. Note that if $\pi_{\mathcal{D}_i}(e) = \perp$ and $\mathcal{D}_i$ is nominal, then $\perp$ is projected to a vector ($\in \{0,1\}^{|\mathcal{D}_i|}$) with all components set to zero. Otherwise, if $\mathcal{D}_i \subseteq \mathbb{R}$ the value $\perp$ is simply interpreted as a zero. Eventually, the concatenation of all vectors constitutes an input vector $\mathbf{x} \in \mathbb{R}^l$ for the ϵ-SVR. We summarize all of these steps with the function $\gamma^* : \Sigma \to \mathbb{R}^l$, i.e., $\forall i, \gamma^*(\sigma_c^i) = \mathbf{x}_i$, such that $\gamma^*(\sigma) = ||_j \, \mathbb{1}(last(\sigma, j))$. Finally, the corresponding target value is calculated using the function *rem*, e.g., $y = rem(\sigma_c, i)$ for some $1 \leq i \leq |\sigma_c|$. Hence, starting from a trace $\sigma_c = \langle e_1, e_2, \ldots, e_n \rangle$, the corresponding set of n training examples $(\mathbf{x}, y)$ will be $(\gamma^*(\sigma_c^i), rem(\sigma_c, i)), \forall i \in \{1, \ldots, n\}$.

Training: A training set for a ϵ-SVR algorithm is defined as $Tr = \{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ where $\mathbf{x} \in \mathbb{R}^n$, for some $n \in \mathbb{N}^+$, and $y \in \mathbb{R}$. In order to map the data contained in an event log L, we exploit the transformations described in the previous section. In particular, the training set is created by the Algorithm 1.

Algorithm 1: Training set construction

```
   Input: L: event log
   Output: Tr: training set
 1 Tr ← ∅
 2 foreach σ ∈ L do
 3 |   for k ← 1 to |σ| do
 4 |   |   x ← 𝟙(π_A(σ(k)))
 5 |   |   for i ← 1 to |D| do
 6 |   |   |   x ← x || 𝟙(last(σ^k, i))
 7 |   |   end
 8 |   |   Tr ← Tr ∪ (x, rem(σ, k))
 9 |   end
10 end
11 return TS
```

Algorithm 2: Prediction

```
   Input: σ_p: (partial) trace, f*:
          ε-SVR model
   Output: remaining time
           prediction
1 x ← 𝟙(π_A(σ_p(|σ_p|)))
2 for i ← 1 to |D| do
3 |   x ← x || 𝟙(last(σ_p^k, i))
4 end
5 return f*(x)
```

The value returned by the function *rem* depends on the time granularity (e.g., hours, minutes, seconds). It is important to keep the same granularity for all the instances. Once the training set Tr is constructed, the training phase learns the parameters of the ϵ-SVR.

Prediction: After the training phase, the ϵ-SVR model is created and it can be directly used to predict the remaining time of partial traces. First of all, the new trace is converted to a vector $\mathbf{x}$ suitable for the ϵ-SVR, applying the same procedure illustrated in Algorithm 1, in particular from line 4 to line 7. In line 4 the information about the activity is encoded in the vector $\mathbf{x}$ and the *for* loop appends to it the encoding of the additional attributes. Then this vector $\mathbf{x}$ is given as input to f^*, i.e., the ϵ-SVR model, which produces the time prediction. This prediction value has to be interpreted with the same granularity used in the creation of the training instances. Algorithm 2 shows the prediction algorithm.

4.2 **Approach 2**: Regression with Contextual Information

This approach differs from the previous one since it makes use of control-flow information in order to add contextual knowledge. The basic idea consists of adding a limited set of features able to encapsulate the control-flow path followed by a partial trace. We chose as control-flow model a transition system because it generally represents a good trade-off between expressivity and compactness. Given a labelled transition system $TS = (S, E, T)$ starting from an event log L we have to transform it into a series of features and encode it into a proper form applicable to the ϵ-SVR algorithm. As for the literal attributes, we use the one-hot encoding: the set $S \setminus S^{start}$ is treated as a literal domain, where the possible values are the states $s \in S$ excluding the initial state because a non-empty trace always maps onto a state not included in S^{start}. So, we enumerate the states, $s_1, s_2, \ldots, s_n \in S \setminus S^{start}$, and we map a state s_i into a vector $\mathbf{v} \in \{0,1\}^n$ by setting to 1 the i-th component in the n-dimensional vector, and to 0 all the others. For example, given the states set $S \setminus S^{start} = \{s_1, s_2, s_3, s_4\}$ we encode the state s_3 into $\mathbf{v} \in \{0,1\}^4$ such that $\mathbf{v} = [0, 0, 1, 0]$. In the following we will refer to this method as *SVR+TS*.

4.2.1 Non-fitting Traces

As for the previous approach, even this one is able to handle non-fitting traces. However, without any adjustment, the prediction would be calculated overlooking the control-flow. Indeed, using the encoding described above, if $f^{state}(\sigma) \notin S$ then the corresponding vector would be null (i.e., $\mathbf{v} = [0, 0, \ldots, 0]$). We cope with this problem by mapping the *non-compliant* state $s = f^{state}(\sigma) \notin S$, into a set of lawful states $s_i \in S$. The idea is to associate the non-fitting trace with states that are, within some degree, similar. Then the vector $\mathbf{v}$ will contain for each state the normalized similarity value. It is very important to define a thoughtful similarity function: we assume that two states are similar if their representations are similar. In particular, since we are focusing on control-flow, we use as event representation function something like $f^{event}(e) = \pi_{\mathcal{A}}(e)$. This implies that abstractions are aggregate representations of a set of activities. Let us define a similarity function for each abstraction considered in Section 3 (i.e., set, bag and sequence):

Definition 8 (Set Similarity Function) Given two sets $x_1, x_2 \subseteq \mathcal{X}$, with $\mathcal{X}$ the set of all possible values, we define the similarity function $f^{sim}_{set} \in 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to [0, 1]$ as the Jaccard similarity [24].

Definition 9 (Bag Similarity Function) Given two multi-sets over a root set $\mathcal{X}$, $x_1, x_2 \in \mathbb{B}(\mathcal{X})$, we define the similarity function $f^{sim}_{bag} \in \mathbb{B}(\mathcal{X}) \times \mathbb{B}(\mathcal{X}) \to [0, 1]$ as the Jaccard similarity [24].

Definition 10 (List Similarity Function) Given two finite sequences over $\mathcal{X}$, $x_1, x_2 \in \mathbb{S}(\mathcal{X})$, we define the similarity function $f^{sim}_{list} \in \mathbb{S}(\mathcal{X}) \times \mathbb{S}(\mathcal{X}) \to [0, 1] \subset \mathbb{R}$ as the Damerau-Levenhstein similarity [8].

The computational complexity of these similarities is quadratic in the size of the abstractions. On the basis of the abstraction used for constructing the transition system, the corresponding similarity function is chosen. Other similarity functions can be used, e.g. Levenshtein similarity. A reasonable choice should take into

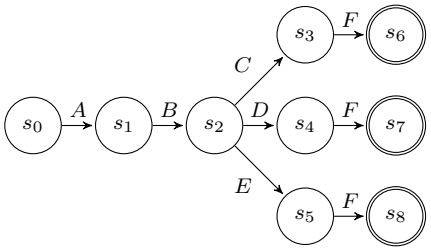

Fig. 2 Example of a transition system extracted from a log containing three trace types $\langle A, B, C, F\rangle$, $\langle A, B, D, F\rangle$ and $\langle A, B, E, F\rangle$, with $f^{event}(e) = \pi_{\mathcal{A}}(e)$ and $f^{state}(\sigma) = \{f^{event}(e) \mid e \in \sigma\}$. The state s_0 is the initial state, while s_6, s_7, s_8 are the accepting (i.e., final) states. Each transition is labeled by the corresponding event representation value.

account how the state representation is defined. In our case, in particular with sequences, we consider the analogy of this abstraction with strings. Using more in depth knowledge about the process, other (custom) similarities could be used.

Every time a non-fitting trace comes into play, its representation is compared with all the state representations of the TS (excluding the initial state). So, given a transition system $TS = (S, E, T)$ (created using f^{event} and f^{state}), a similarity function f^{sim} and a trace σ (such that $s' = f^{state}(\sigma) \notin S$) $f^{sim}(s, s')$ is computed for each state $s \in S \setminus S^{start}$. After that, each similarity value is normalized and finally put into the vector $\mathbf{v}$. We will call this kind of TS *similarity-based* transition system.

Training: The training phase of this method is almost the same of the preceding one. The main difference lays on the introduction of a new derived feature to the training set. This can be done by making some minor changes to the Algorithm 1. Since we assume the construction of *TS*, we need it as input along with the state representation function f^{state} and the similarity function f^{sim}. We calculate the state associated with each partial trace and we encode it into a one-hot vector or into the normalized similarity vector if it is a non-fitting trace. Finally, we construct the rest of the training instances as in Algorithm 1.

Prediction: In this phase, as for the Simple Regression approach, the ϵ-SVR model created in the previous step is used to forecast the remaining time of running process instances. The novelty of the method just described consists of the resulting model, which is obtained from the training phase. The introduction of contextual information, generally, leads to a different optimization problem and consequently to a different final model. The only changes to make in Algorithm 2 are the addition of f^{state} as input, and the substitution of the right side of line 1 with the one-hot (or the non-fitting) encoding of $f^{state}(\sigma_p)$.

4.3 **Approach 3**: Data-aware Transition System (DATS)

The approach presented in this section is a refinement of [28] which exploits the same idea described in [3]. The main difference from [28] is the removal of the *soujourn time* expected value in the prediction because now that time is included in the SVR estimation which takes into account also the additional data. Let us recall the main characteristics of the method presented in [3]. In their work van der Aalst et al. introduced the concept of annotated transition system: each state of the

transition system is "decorated" with predictive information, called *measurements*. Since we want to predict the remaining time, a possible set of measurements collected in a state might be the remaining time starting from the state itself. Formally, in [3] a measurement is defined as:

Definition 11 (Measurement) A measurement function $f^{measure}$ is a function that, given a trace σ and an event index $i \in [1, 2, \dots, |\sigma|]$ produces some measurement. Formally, $f^{measure} \in \Sigma \times \mathbb{N} \to \mathcal{M}$, where $\mathcal{M}$ is the set of possible measurement values (e.g., remaining time).

In [3] different kinds of measurements are proposed. Once the suitable measurement is chosen, an annotated transition system is created according to the event log:

Definition 12 (Annotated transition system) Let L be an event log and $TS = (S, E, T)$ a labeled transition system constructed over L using the representation functions f^{event} and f^{state}. Given a particular measurement function $f^{measure} : \Sigma \times \mathbb{N} \to \mathcal{M}$, we define an annotation $A \in S \to \mathbb{B}(\mathcal{M})$, such that $\forall s \in S$:

$$A(s) = \biguplus_{\sigma \in L} \biguplus_{\substack{1 \le k \le |\sigma| \\ s = f^{state}(\sigma^k)}} f^{measure}(\sigma, k),$$

where $\mathbb{B}(A) : A \to \mathbb{N}$ is the set of multiset over a finite set A and $\uplus$ is the *disjoint union* defined as the multiset $X \uplus X' = \{(A \cup B) \to \mathbb{N}^+ \mid \forall\, c \in A \cup B,\ M(c) = X(c) + X'(c)\}$. An annotated transition system is the tuple (S, E, T, A).

Since we are facing the remaining time prediction problem, our $f^{measure}$ function coincides with the function *rem* defined in Section 4. The last step consists of the definition of a prediction function $\in \mathbb{B}(\mathcal{M}) \to \mathcal{M}$, such that, given a multiset of measurements, it produces some prediction, e.g., the average or the median value. So, in the operative setting we have an annotated transition system (S, E, T, A) constructed over L and a prediction function $\mathcal{P} : \mathbb{B}(\mathcal{M}) \to \mathcal{M}$. Using these tools a prediction is made in a straightforward way: given a partial trace σ_p observed so far, such that $f^{state}(\sigma_p) = s$, the prediction value is $\mathcal{P}(A(s))$. It is worth to notice that the prediction is calculated using merely control-flow (i.e., transition system) and temporal (i.e., remaining time) information.

The main difference introduced by our approach is the addition of classifiers and regressors, which take advantage of additional attributes, as annotations. Let us give a brief overview of this approach. As in [3], we start with the transition system construction and then we enrich each state with a Naïve Bayes classifier [25] and each transition with a Support Vector Regressor [4,10,31] (ϵ-SVR) trained with historical data considering all attributes. The introduction of these two machine learning models is based on the intuition that in a state s Naïve Bayes estimates the probability of transition from s to $s' \in s\bullet$, while ϵ-SVR predicts the remaining time if the next state will be s'.

Figure 3 proposes an example of state (s_2) annotated with a Naïve Bayes classifier. In this state, the Naïve Bayes classifier gets the probabilities to reach each exiting state: probabilities to reach states s_3, s_4 and s_5 are respectively 0.6, 0.1 and 0.3. Such values are used to weigh the remaining time values obtained from the support vector regressors. In Figure 3 the state s_2 corresponds to the current

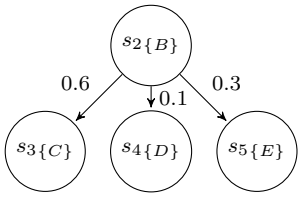

Fig. 3 A state annotated with a Naïve Bayes classifier. The probability of going from state s_2 to any of its exiting states is reported next to the corresponding edge.

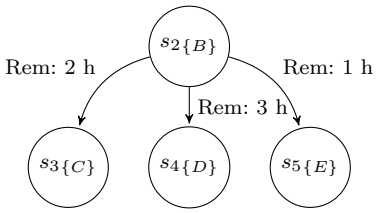

Fig. 4 Example of a Support vector regressor application. The estimated remaining times are suggested by the labels *Rem*.

partial trace σ'. From each outgoing state (i.e., s_3, s_4, s_5) the remaining time is estimated using the SVR associated with the incoming transition (i.e., $s_2 \to s_3$, $s_2 \to s_4$ and $s_2 \to s_5$). These estimations are multiplied by the probability values obtained from the NB classifiers, and finally summed together in order to compute a weighted average over all the possible continuations from s. Figure 4 presents an example of remaining time estimation for each outgoing state. Formally, let $\hat{p}_{s'}$ be the Naïve Bayes estimated probability to reach state $s' \in s\bullet$ from state s, and $\hat{\tau}_{s \to s'}$ the estimated remaining time returned by the ϵ-SVR associated with transition $s \to s'$. Then, given the state s reached after observing a (partial) trace σ, the prediction returned by the annotated transition system is $\sum_{s' \in s\bullet}(\hat{p}_{s'} \cdot \hat{\tau}_{s \to s'})$.

Let us now define the annotations used to decorate the transition system and then the predictor transition system.

Definition 13 (Naïve Bayes Annotation) Let TS be a labeled transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. Let's call k the size of the $\gamma^*(\sigma)$ vector calculated for traces $\sigma \in L$. A *Naïve Bayes Annotation* is a function $NB : S \times \mathbb{R}^k \times S \to [0, 1] \subset \mathbb{R}$, which, given two states $s_i, s_j \in S$ and a data attribute vector $\mathbf{x} \in \mathbb{R}^k$, returns the probability to reach the state s_j starting from s_i through a single transition.

Definition 14 (SVR Annotation) Let TS be a labeled transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. An *SVR Annotation* is a function $R : T \times \mathbb{R}^k \to \mathbb{R}$, such that, given a transition $t \in T$ and a data attribute vector $\mathbf{x} \in \mathbb{R}^k$, it applies Support Vector Regression to produce an estimation of the remaining time.

Definition 15 (Predictor TS) Let $TS = (S, E, T)$ be a labeled transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. A *predictor transition system* is a tuple $PTS = (S, E, T, NB, R)$ where NB, R are respectively a Naïve Bayes and a SVR annotation, based on the event log L and the transition system TS.

Training: In this section, we describe how to construct a predictor transition system. Algorithm 3 shows the construction procedure.

Algorithm 3: Predictor TS construction

Input: L: event log; $TS = (S, E, T)$: labeled TS
Output: T': predictor TS

1 **foreach** $t \in T$ **do**
2 | $svr[t] = \emptyset$ ▷ `training set for` t
3 **end**
4 **foreach** $\sigma_c \in L$ **do**
5 | **for** $i \leftarrow 1$ **to** $|\sigma_c| - 1$ **do**
6 | | $s \leftarrow f^{state}(\sigma_c^i)$
7 | | $s' \leftarrow f^{state}(\sigma_c^{i+1})$
8 | | $e \leftarrow f^{event}(\sigma_c(i+1))$
9 | | $t \leftarrow (s, e, s')$
10 | | $\mathbf{x}, y \leftarrow \gamma^*(\sigma_c^i), rem(\sigma_c, i)$
11 | | $svr[t] \leftarrow svr[t] \cup (\mathbf{x}, y)$
12 | | **if** $|s\bullet| \geq 2$ **then**
13 | | | Update NB for state s with instance $(\mathbf{x}, s')$
14 | | **end**
15 | **end**
16 **end**
17 **foreach** $t \in T$ **do**
18 | Train SVR (R) for transition t with training set $svr[t]$
19 **end**
20 **return** $T' = (S, E, T, NB, R)$

Algorithm 4: Remaining time prediction for a running case

Input: σ_p: (partial) trace; $PTS = (S, E, T, NB, R)$: predictor transition system
Output: P: remaining time prediction

1 $P \leftarrow 0$
2 $s \leftarrow f^{state}(\sigma_p)$
3 $\mathbf{x} \leftarrow \gamma^*(\sigma_p)$
4 **if** $|T_s| \geq 2$ **then**
5 | **foreach** $t = (s, e, s') \in T_s$ **do**
6 | | $P \leftarrow P + NB(s, \mathbf{x}, s') \cdot R(t, \mathbf{x})$
7 | **end**
8 **else**
9 | $s' \leftarrow f^{state}(\sigma_p^{|\sigma_p|-1})$
10 | $t' \leftarrow (s', e, s) \in T_{s'}$
11 | $P \leftarrow R(t', \mathbf{x})$
12 **end**
13 **return** P

The first loop (line 1) initializes all the training sets for the ϵ-SVR model to the empty set, while the second loop (lines 4-16) creates the training sets and updates the NB classifiers. In particular, line 6 constructs the training instances by extracting the additional data from the partial traces and calculating the remaining time. Being possible to build a NB model incrementally, this is done in line 13. Last loop (lines 17-19) trains the ϵ-SVR models using the training sets built previously. The training time of this model can be time consuming because of the fact that, especially with complex processes, many SVR and NB should be trained in addition to the construction of the TS. However, with not very complex processes the training time is reasonable.

Prediction: In this section, we describe how to predict the remaining time for a running case using a predictor transition system constructed with Alg. 3. Algorithm 4 shows the prediction procedure. The algorithm simply applies the formula, seen at the beginning of this section, $\sum_{s' \in s\bullet}(\hat{p}_{s'} \cdot \hat{\tau}_{s \to s'})$. Each $\hat{p}_{s'}$ is the value produced by the application of the NB classifiers and $\hat{\tau}_{s \to s'}$ by the ϵ-SVR. Specifically, let $s = f^{state}(\sigma_p)$ be the current state, then $\hat{p}_{s'} = NB(s, \gamma^*(\sigma), s')$ and $\hat{\tau}_{s \to s'} = R(s \to s', \gamma^*(\sigma))$, for all $s' \in s\bullet$. A core difference w.r.t. [28] is the absence of the expected sojourn time (on the current state): in this revised version this information is implicitly embedded inside the ϵ-SVR and hence can be removed from the formulation. The computational complexity of the algorithm is linear in the average number of outgoing transitions of the states of the transition system. Figure 5 puts together the two representations depicted in Section 4.3: it shows an example of NB and SVR application for a partial trace $\sigma = \langle A, B \rangle$ (see event log in Table 1). It is noteworthy that given a predictor transition system (result of the

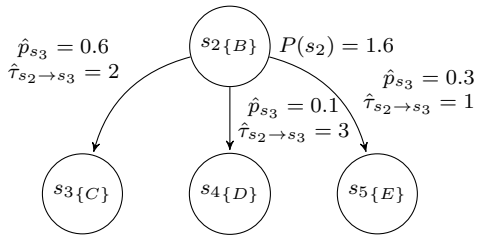

Fig. 5 Example of a prediction calculated by a predictor transition system: $P(s_2) = 0.6 \cdot 2 + 0.1 \cdot 3 + 0.3 \cdot 1 = 1.8$.

learning procedure), the computation of the prediction requires constant number of operations which, in the worst case, corresponds to the size of the largest set $s\bullet$ of the transition system. This property allows the application of the approach in on-line settings [6], where each event is allowed to trigger only a constant number of operations.

4.3.1 Future Path Prediction

The model we used in this last approach can also be exploited in order to predict, for a running case, which is the most likely sequence of activities until the end of the case. Let us take, for example, the situation depicted in Fig. 3 which is a fragment of the TS in Fig. 1: the most likely sequence of states starting from $s_{2\{B\}}$ is $s_{2\{B\}} \to s_{3\{C\}} \to s_{6\{F\}}$, $s_{6\{F\}}$ is an accepting node and so the process instance is complete, with a probability equals to $0.6 \times 1 = 0.6$. Note that the transition between $s_{3\{C\}}$ and $s_{6\{F\}}$ has a probability equals to 1 because is the only way the process can proceed. In general, the sequence can go through many split states in which the transition probabilities are < 1 and finding the full sequence with the highest probability is not a trivial task. We face this problem as a shortest path problem in which the goal is to find a path between two vertices of a graph, such that the sum of the weights of its constituent edges is minimized. Specifically, let us consider the transition system as a directed graph: we would like to find the shortest path between the current node and an accepting node. In order to leverage this idea, we need to define a suitable distance measure (i.e., cost) between nodes.

Given a possible sequence of activities (in a Predictor TS) between the state s_1 to s_n, i.e., $s_1, s_2, \ldots, s_n$, with the corresponding transition probabilities (obtained using the NB annotations), i.e., $p_i \forall s_i \to s_{i+1}, 1 \leq i < n$, then the likelihood of such sequence is defined by $\prod_{1 \leq i < n} p_i$. Since probabilities have to be multiplied to get the likelihood of a sequence we cannot use it directly as edge cost because we need to follow the definition of the shortest path problem (i.e., edge cost have to be summed). However, we can exploit properties of the *logarithm* function to transform probabilities into distance like values, in particular: $\log(pq) = \log(p) + \log(q)$, and since p is a probability: $\forall\ 0 \leq p \leq 1 \in \mathbb{R}, \log(p) \leq 0$, where low values of $log(p)$ mean that the transition probability is low, or, from a graph view point, the distance between the nodes is high. Using this idea, we can use as edge cost the opposite of the logarithm of the transition probability. Note that this logarithm transformation works as we desire: probabilities close to 0 (i.e., rare occurrences) correspond to high costs, while probabilities close to 1 (i.e., very likely) correspond

to almost negligible costs. Using the just mentioned transformation we can construct a graph corresponding to the TS where the shortest path problem can be solved applying a best first search. So, from a computational point of view the prediction of the activity sequence has a cost which is linear in the number of nodes (i.e., states of the TS).

4.4 Application Scenarios

The proposed methods are designed to be used in different operative scenarios. Specifically, we can classify a business process on the basis of its complexity and its dynamicity over time. Anytime the process is stable and it is also not very complex (i.e., structured and with a reasonable number of activities) DATS should be preferred over the other methods. Thanks to the TS and the information on the states/transitions, it is able to take advantage of the peculiarities of each single activity by keeping the learning time (and space) acceptable. In all other cases, i.e., complex and/or very dynamic processes, SVR based approaches should be considered. In particular, SVR+TS is a better choice in cases of dynamic but simple processes because it uses more information and, in general, should be more accurate. In the case of dynamic processes, SVR or SVR+TS can be used depending on how complex the process is. It is a matter of trade off between (expected) accuracy and training time. With particularly complex process SVR is of course the most reasonable choice. Table 2 summarizes the possible scenarios and which method should be preferred. From a computational point of view, all the models

	Static	**Dynamic**
Easy	DATS	SVR+TS
Complex	SVR+TS/SVR	SVR+TS/SVR

Table 2 Possible application scenarios.

once trained can be useful in an almost real time setting because prediction can be calculated in the order of milliseconds. However, the training time could take several hours in particular if the auto-tuning of the parameters is required.

From a scalability point of view, the use of an explicit process model, such as a TS, can be an issue. In particular, by using a TS, the number of states can easily grow and this can cause the following problems: (*i*) for DATS, the number of prediction models that have to be trained could be too high and the training time could be not reasonable; (*ii*) for SVR+TS, the number of features are way greater than the number of training examples which can cause a drop in the performance.

It is always possible to limit the number of states of a TS by using a horizon in the abstraction, but we have to be careful in order to not underfit the real process model. However, we think that in most of the real world business processes it is possible to have a reasonable TS without overgeneralizing the actual model.

5 Experiments

These techniques have been implemented for the ProM framework[3] [34]. To mine the transition systems, we rely on the miner's implementation available inside the framework. Naïve Bayes Classification and the Support Vector Regression are performed using the implementation in the Weka framework [16]. In particular, for SVR we used the SMO (Sequential Minimal Optimization) implementation provided by the framework.

The experiments reported in this section aim to assess how the techniques leveraging on the similarity-based transition system give more accurate predictions for variants of process executions that have not been observed in the training set. Moreover, in the static scenario we want to assess the prediction quality against state-of-the-art methods. The comparison is made with respect to the technique reported in van der Aalst et al. [3] and the LSTM based approach presented in [26] that have been discussed in Section 2. Work [28] shows that no much difference can be appreciated when polynomial or RBF is used as kernel type. Therefore, the experiments only make use of the latter. To measure and compare the accuracy, we used three indicators: the Mean Absolute Percentage Error (MAPE), the Root Mean Square Percentage Error (RMSPE) and, only in the comparison with [26], the Mean Absolute Error (MAE). The first two metrics are the percentage form of the two standard measures, *Mean Absolute Error* and *Root Mean Square Error*. We use their percentage form in order to have values which do not depend on the duration of the process because they are scaled. Let n be the number of samples and let A_i and F_i be respectively the actual value and the predicted value for the i-th sample. MAPE is defined by $100\% \frac{1}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right|$, while RMSEP by $100\% \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\frac{A_i - F_i}{A_i} \right)^2}$. Note that we discard the last event of each trace in order to avoid that $A_i = 0$, which could cause a division by zero. Even though *MAPE* and *RMSPE* are similar, the latter metric tends to punish larger error more than the former. So, with respect to how important is to avoid large errors in the single prediction, one metric can be preferred to the other one.

We performed our experiments on three real case studies. The first case study log[4] concerns the execution of process instances in an information system for the management of road-traffic fines by a local police office of an Italian municipality.

The second case study log [6] concerns the ticketing management process of the help desk of an Italian software company. In particular, this process consists of 14 activities: it starts with the insertion of a new ticket where a seriousness level is applied. Then the ticket is managed by a resource and it is processed. When the problem is resolved it is closed and the process instance ends. In this case the process is not linear as the previous one, but it is well structured. The third log is a reduced version of the dataset from the BPI 2012 challenge (the same dataset as in [26]). It is a real-life log of a Dutch Financial Institute. The event log describes an application process for a personal loan or overdraft within a global financing organization. Table 3 summarizes the logs' details.

[3] Source code available at `https://www.dropbox.com/s/6ps93qmc81clz74/Polato_etal_Prediction_plugin_prom.zip`

[4] The log is a part of the the full log provided by Eindhoven University of Technology [5].

[6] Polato, M.: Ticketing (2017). DOI 10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5b

	#Events	#Traces	$\|\mathcal{A}\|$	μ_D	μ_E	$\|\mathcal{S}\|_{set}$	$\|\mathcal{S}\|_{bag}$	$\|\mathcal{S}\|_{seq}$
Road fines	36716	7314	6	32.6 mths	5.02	8	8	8
Help desk	21348	4580	14	40.9 days	4.66	101	514	703
BPI 12	72413	9658	6	11.4 days	7.50	27*	111*	154*

Table 3 Logs' information: total number of events, total number of traces, number of activities ($|\mathcal{A}|$), average case duration (μ_D), average number of event per case (μ_E) and finally the size of the TS ($|\mathcal{S}|$) with the set, bag and sequence abstraction. (*) means that the horizon has been set to 5.

5.1 Comparison on static processes

5.1.1 Road Fines Log

We extracted from the information systems an event log that refers to executions that end with sending for credit collection. With this set of experiments we want to show that all the methods presented here perform well under the assumption that the training contains all the possible process' behaviours (the same assumption is done by van der Aalst et al. in [3]). The experiments were performed using 5-fold cross validation. The test has been performed over all possible prefix sub-traces of the test log. The SVR hyper-parameters have been tuned automatically using a grid search strategy. In particular we sought for the best combination of C, as penalty in the optimization problem, and γ for the RBF kernel. The transition system has been mined using different abstractions, namely set, multi-set and sequence with no limit. Since, in the event log, for 99% of the traces every activity was performed at most once, and the process is almost linear, the multi-set and the sequence abstraction was not considered to perform experiments.

Table 4 reports the results of the experiments. The acronyms used in the tables mean: VDA is the approach presented in [3], DATS is the data-aware transition system, SVR is the approach which uses a simple support vector regression machine, and SVR+TS is the method which incorporates contextual information (via TS) in the training instances.

	MAPE	RMSPE
VDA*	5.89±0.14%	8.80±0.91%
DATS	2.82±0.10%	**6.06±1.01%**
SVR	2.83±0.07%	6.72±1.40%
SVR+TS	**2.77±0.09%**	6.58±1.41%

Table 4 Road Fines log experiment results using 5-fold cross validation: (*) means that the approach is the baseline.

Results show that every proposed approach outperforms the baseline with similar results. However, it is worth to notice that the approaches based on the transition system, i.e., DATS and SVR+TS, achieve best performances on RMSPE and MAPE respectively. From these results, we can argue that the improvements with respect to the baseline are due to the introduction of the additional data in our models. In this log, we can also notice that the effect of considering the

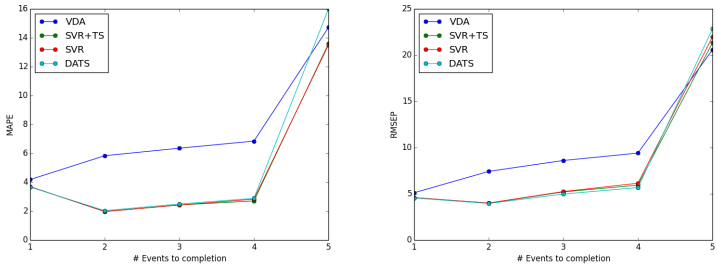

Fig. 6 Error (MAPE on the left, RMSPE on the right) with respect to the number of events to the completion for the Road Fines log.

workflow (i.e., the transition system) is cramped in comparison to the data perspective. Since this log has only two variants, the importance of the process model is limited, however, SVR+TS achieves better results than the simple SVR which does not use any information about the process model. Figure 6 shows how the error is distributed with respect to the number of events that the trace will take to finish. As expected the error increases with the increasing of the number of future events. However, it is worth to notice that the error for the proposed methods with only one event to the completion is higher than with two. This can be due to the fact that in the TS the last non final state is the only one with two possible continuations. Moreover, since SVR has the same behaviour, but does not depend on the TS, we believe that data could be misleading to understand which last activity will be taken and this causes an increase of the error.

5.1.2 Help Desk Log

As in the previous case study, the experiments were performed using 5-fold cross validation and the SVR's hyper-parameters have been tuned automatically using a grid search strategy. Since the process is more complex, we performed two kinds of experiments: *(i)* we compared the performance of our approaches against the baseline assuming that in the training phase all the possible behaviours of the process are present at least once; *(ii)* we removed some variants from the training set in order to have completely new activity sequences in the test set, so the assumption is not valid anymore (see Section 5.2). The second type of experiments aims to show that the approaches based on single SVR performed well even with noisy instances or with new events. Table 5 shows the results. The test has been performed over all possible prefix sub-traces of the test log. We tested all the three trace abstractions, i.e., set, multi-set and sequence. As for the previous case study, the best performing methods are DATS and SVR+TS, however the lowest MAPE and RMSPE are achieved by DATS using the set abstraction. Since the SVR approach did not achieved good results (it is in line with VDA), the transition system information used by SVR+TS played a decisive role. In fact, even though VDA is the worst performing method, it is relatively closer than the previous case study. The improvements with respect to the baseline are on average 14% for the MAPE and 4% for RMSPE. Figure 7 shows the error of the methods with respect to the number of events the instance will take to terminate. In this

MAPE	Sequence	Multi-set	Set
VDA*	29.94±0.59%	29.58±0.35%	29.94±0.36%
DATS	26.96±0.7%	26.54±0.51%	**25.66±0.34%**
SVR	28.82±0.57%	28.03±1.1%	29.12±0.29%
SVR+TS	**25.80±0.51%**	**25.84±0.55%**	25.96±0.26%
RMSPE	**Sequence**	**Multi-set**	**Set**
VDA*	45.10±1.3%	43.32±0.25%	43.44±0.43%
DATS	43.76±0.91%	**42.16±0.61%**	**41.16±0.16%**
SVR	47.78±0.83%	42.39±1.61%	47.46±3.1%
SVR+TS	**41.90±0.98%**	42.70±1.01%	42.76±1.41%

Table 5 Help Desk log experiment MAPE and RMSPE results using 5-fold cross validation: (*) means that the approach is the baseline. The underlined results are the overall best.

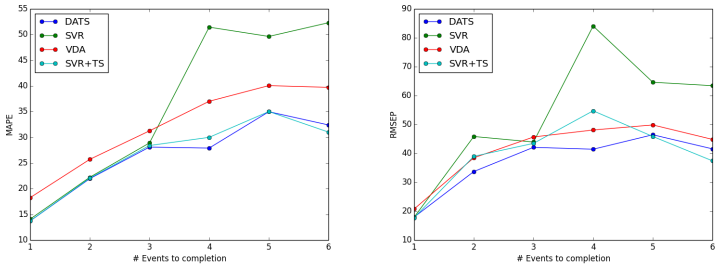

Fig. 7 Error (MAPE on the left, RMSPE on the right) with respect to the number of events to the completion for the Help Desk log using the set abstraction.

case the error's behaviour is as we expected. We can also notice that SVR has a very high error with the number of future events greater than 3. This means that the TS information are helpful, in fact VDA has better performance even without using any additional data. From a workflow point of view, this case study has a more complex structure and we can see how the workflow information, via the TS, had different impact on the results. In this experiment the simple SVR failed in comparison with SVR+TS and the DATS methods, emphasizing the importance of the information brought by the TS.

5.1.3 Comparison with LSTM based method

In this section we show the comparison between our proposals and [3] against the LSTM based approach presented in [26] in the static process scenario (see Section 4.4). For [26] we used the implementation provided by the authors.

In these experiments the setting is different from the previous ones, in particular we perform a single fold, because the deep network would have required too much time for being trained and tested. Since processes are considered stable, we did not include the simple SVR since SVR+TS in this scenario can be considered as a kind of generalization of SVR. Differently from [26], we consider all prefix lengths (as done in the other experiments), including partial traces with a single event. We temporally divided the log as follows: the first 2/3 of the traces as training (and validation) data and the remaining 1/3 as test set. We tested different

	Help Desk	Road Fines	BPI 12
VDA	8.67 (bag)	64.71 (set)	8.45 (set)
DATS	8.21 (bag)	**54.17** (bag)	8.12 (bag*)
SVR+TS	8.29 (seq)	55.11 (bag)	**8.03** (seq*)
LSTM	**7.17**	120.58	9.74

Table 6 MAE performances: errors are reported in days. Inside the parenthesis are highlighted the abstractions that perform better. (*) means that the horizons have been limited to 5 for computational reasons. Bold faced results are the overall best for the dataset.

network configurations, validating the number of LSTM neurons in each layer in the set $\{100, 150, 200\}$ and also the number of layers in the range [1,6] (shared layers are not included). We fixed a single shared layer as suggested in [26]. The validation set is used for selecting the best hyper-parameters and network architecture. The evaluation is done in terms of *Mean Absolute Error* (MAE), that is an error measure among continuous variables. This is the same metric which is optimized by the LSTM network over single events. Let A_i the desired output, and F_i be the output predicted, for some data samples of dimension n, than we can define the MAE as: $MAE = \sum_{i=1}^{n} |A_i - F_i|/n$. The MAE has an intuitive interpretation as the average absolute difference between the prediction and the expected output. The obtained results are reported in Table 6. From the table we can notice that the LSTM based approach achieves good performance in the Help Desk log while in the other two datasets our proposals have better performances. In particular, we can notice that on the more complex log (it has many repeated activities), i.e., BPI 12, the best performing method is SVR+TS as we expected (see Section 4.4). However, also DATS performs quite well with the BPI 12 log. As underlined previously, our approach can suffer from processes with many activities, in fact with BPI 12 we had to limit the abstractions' horizon in order to be able to compute the prediction. Another observation is that the approach in [26] has a significant drawback regarding the parameter selection. A reasonable way to select the parameters of the network is to look at the error on the validation set. However, since the function that the method optimizes is not the overall MAE (indeed, the network predicts the time until the next event instead of the whole remaining time) we empirically found that the configuration with lower validation loss is generally not the best performing one on the test set. The results reported in Table 6 are based on the best performing parameters on the validation test. For some datasets this may not be a problem. For example, in the Help Desk dataset the difference between the selected configuration and the best performing one on the test set is small, namely < 0.1 days. However, in other cases this difference may be huge. This is the case for the Road Fines dataset, where the selected configuration has an error in MAE that is almost the double w.r.t. the best one. We would like to underline the fact that our results are consistent with the ones obtained (on the same logs) by Tax et al. in [26].

5.2 Comparison on dynamic processes

In order to assess our approaches on dynamic processes we used a modified version of the Help Desk log, in which we removed, from the training set, some process'

	Variants		Activity	
	MAPE	**RMSPE**	**MAPE**	**RMSPE**
VDA*	41.26±1.11%	67.96±2.56%	41.28±0.27%	69.02±0.64%
DATS	40.74±1.4%	67.50±2.21%	40.74±0.27%	69.02±0.42%
SVR	41.48±1.2%	69.08±2.12%	41.00±0.25%	69.50±0.68%
SVR+TS	**33.72±0.9%**	**54.44±2.6%**	**33.92±0.39%**	**55.84±0.77%**

Table 7 Help Desk log experiment (without some process variants) results using 5-fold cross validation: (*) means that the approach is the baseline.

variants or a subset of the activities. Table 7 shows the results with 5-fold cross validation on the log without some variants using the set abstraction. We use this abstraction because it is the most general one and in this way there are more chances that states with high similarity to the target one exists. In particular, column "Variants" shows results using training instances without half of the variants present in the starting event log. Column "Activity" instead, shows results removing from the training set all the traces with a specific activity, i.e., "Waiting", which were present in almost 25% of the instances. Then the test phase uses the remaining part of the log, so inside the test there are completely new process behaviours which cause problems to VDA and DATS approaches. Results show that in both cases, SVR+TS outperforms all the other approaches with a MAPE around 34% and a RMSPE of 55%. The introduction of the similarity mechanism in SVR+TS makes this approach less sensible to the noise or change in the workflow, because it is able to mitigate the lack of the correct state using information from correlated ones. It is worth highlighting, that both VDA and DATS would not return any prediction in case of unseen activity' sequences. However, in order to be able to compare the approaches, we implemented a safety mechanism: if a trace does not map into a valid state of the transition system, the last event is removed and the mapping is redone. This process is repeated until obtaining a prefix that can be mapped or, viceversa, is empty (and is discarded), and in this case the average remaining time of the entire training is returned. This explains why SVR has not better performance than VDA and DATS. These experiments show the effectiveness of the SVR+TS approach and the importance of the similarity-based transition system.

5.3 Next Activities Prediction

Using the Help Desk event log we also tested the future path prediction (FPP) method described in Section 4.3.1. We evaluated our method against a random predictor which chooses randomly the next activity according to the possible continuation seen in the event log. If an activity is also a possible termination, the method randomly decides whether to stop or not. In these experiments we want to show that our model can be used also to predict future activities, however, since DATS was not developed for this precise task (but for predicting the remaining time), there could be existing methods, e.g., [7, 11, 32], which perform better than ours. As for the previous experiments, we used 5-fold cross validation. To evaluate the methods, which means assessing how much the predicted path respects the actual one, we used two metrics: the Damerau-Levenshtein similarity (DAM)[8] and the common prefix (PRE). PRE simply counts the length of the common pre-

#	abst.	FPP DAM	FPP PRE	RANDOM DAM	RANDOM PRE
1	sequence	0.9629±0.006	0.9456±0.007	0.5030±0.025	0.2375±0.023
	multi-set	0.9553±0.006	0.9306±0.009	0.5992±0.005	0.4328±0.016
	set	0.9463±0.012	0.9489±0.005	0.2896±0.017	0.2492±0.024
2	sequence	0.8621±0.031	0.7096±0.059	0.6288±0.008	0.3359±0.008
	multi-set	0.8577±0.039	0.7003±0.071	0.5742±0.011	0.3010±0.012
	set	0.8527±0.007	0.7380±0.012	0.4074±0.006	0.2361±0.012
3	sequence	0.8211±0.014	0.5691±0.023	0.5758±0.003	0.1001±0.009
	multi-set	0.8161±0.025	0.5632±0.031	0.5334±0.007	0.0831±0.008
	set	0.8084±0.008	0.6024±0.014	0.3892±0.006	0.0790±0.006
4	sequence	0.6641±0.027	0.2141±0.032	0.5182±0.016	0.0555±0.009
	multi-set	0.6628±0.029	0.2251±0.038	0.4809±0.012	0.0532±0.005
	set	0.6390±0.004	0.2131±0.011	0.3747±0.007	0.0479±0.004
5	sequence	0.5543±0.011	0.1495±0.005	0.4664±0.010	0.0566±0.006
	multi-set	0.5523±0.019	0.1752±0.056	0.4296±0.014	0.0485±0.006
	set	0.5181±0.008	0.1463±0.006	0.3379±0.015	0.0445±0.010
$\mathbb{E}_{\#}$	sequence	0.8205±0.017	0.6265±0.032	0.4382±0.004	0.1415±0.005
	multi-set	0.8140±0.027	0.6118±0.045	0.4143±0.003	0.1477±0.003
	set	0.8088±0.005	0.6484±0.004	0.2548±0.001	0.0936±0.002

Table 8 Activity sequence prediction results obtained with a transition system created using different types of abstractions (*abst.*).

fix between the prediction and the actual trace. The DAM metric gives an overall similarity between the predicted sequence and the actual one, while the PRE metric focuses only on the very next activities.Table 8 shows the results. Each row in the tables represents the similarity considering sequences of exactly n activities where n is indicated in the # column. Rows with the $\mathbb{E}_{\#}$ symbol are the average similarities considering every sequence length. Readers can notice that with every abstraction the proposed method outperforms the random one. In particular, FPP is able to identify the next activity almost 94% of the times and the similarity of the next two is almost 0.86 on average with an hit rate of 71% on average. We achieve good results, with respect to Damerau-Levenshtein similarity, even with 3, 4 and 5 activity sequences, and this could mean that the algorithm finds many right future activities but in the wrong order. We can also notice how the differences in the prediction accuracy are evident with the increasing of the number of future activities. This fact is due to the stockpile of the uncertainty of each step in the future, which makes the prediction of the FPP closer to the random one. However, on average, our approach obtained for DAM and PRE 0.81 and 0.63 respectively, which are far better than the 0.36 and 0.09 obtained by the random method.

6 Conclusions

In this work we presented new methods that can be employed to tackle the problem of predicting the time-to-completion of running business process instances. The contributions we presented in this work can be summarized as the following: (*i*) we proposed three new prediction methods, which take advantage of the additional data present in the event log; (*ii*) we leveraged on solid and well-studied machine learning techniques in order to build models able to manage the additional information; (*iii*) we constructed our approach in order to deal with unexpected behaviours or noisy data, by looking at the closeness between the new trace and

the most similar process flows already observed. The proposed set of methods aims to face different scenarios and to overcome the limitations of the state-of-the-art approaches. Moreover, we distinguished the application of the prediction problem into two main scenarios: (1) the process is stable and consequently the event log used to train the model contains all the possible process behaviours; (2) the process is dynamic (i.e., might contain drifts) and, consequently, the event log used for training does not contain all the possible process behaviours, e.g., seasonability of the process. Experiments in [28] and those reported in Section 5 have shown how the DATS approach has state-of-the-art performance in the first scenario. Assuming that the training event log contains all the possible process' behaviours, it is possible to take advantage of the static nature of the process and rely on the transition system structure. A central role in this method is played by the Naïve Bayes classifiers, which are also involved in the prediction of future sequence of activities in Section 4.3.1. These two tools together can be very useful for business managers because, in addition to the remaining time estimation, they have also some hints about the sequence of activities that the instance is going to take. Using these information business managers can act preventively and they can try to avoid uncomfortable situations. However, we also obtained good results with the single SVR-based methods which are well suited for the second scenario in which not all the process' behaviours are present in the training phase. Here, the SVR+TS method is not affected by the lack of information in the training set because is able to generalize the workflow thanks to the similarity-based transition system and the nature of the single-SVR itself. Experimental results point out that methods which are strongly dependent on the TS structure have problems with new process' variants, while SVR method with the similarity-based TS, outperforms all the other approaches. As future work, we plan to improve the parameters calibration of our approaches, in order to improve the overall results. We would like to investigate whether taking into account only work-hours in the prediction is valuable or not. Moreover, we would like to deploy our approaches on real scenarios, in order to stress the whole approach under production-level constraints.

Acknowledgment The work reported in this paper is supported by the Eurostars-Eureka project PROMPT (E!6696).

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes, 1st edn. Springer (2011)
2. van der Aalst, W.M.P., Rubin, V., Verbeek, E., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software & Systems Modeling **9**(1), 87–111 (2008)
3. van der Aalst, W.M.P., Schonenberg, H., Song, M.: Time prediction based on process mining. Information Systems **36**(2), 450–475 (2011)
4. Basak, D., Pal, S., Patranabis, D.C.: Support Vector Regression. Neural Information Processing - Letters and Reviews **10**(10), 203–224 (2007)
5. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. John Wiley & Sons (2006)
6. Burattin, A.: Process Mining Techniques in Business Environments. Springer International Publishing (2015)
7. Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D.: Completion time and next activity prediction of processes using sequential pattern mining. In: DS 2014, pp. 49–61 (2014)

8. Damerau, F.: A Technique for Computer Detection and Correction of Spelling Errors. Communications of the ACM (CACM) **7**(3), 171–176 (1964)
9. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. IJCAI'15, pp. 3460–3468. AAAI Press (2015)
10. Drucker, H., Burges, C., Kaufman, L., Smola, A.J., Vapnik, V.: Support Vector Regression Machines. Neural Information Processing Systems **1**, 155–161 (1996)
11. Evermann, J., Rehse, J.R., Fettke, P.: A Deep Learning Approach for Predicting Process Behaviour at Runtime, pp. 327–338. Springer International Publishing (2017)
12. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: OTM 2012, vol. 7565, pp. 287–304 (2012)
13. Folino, F., Guarascio, M., Pontieri, L.: Discovering High-Level Performance Models for Ticket Resolution Processes. In: OTM 2013, vol. 8185, pp. 275–282 (2013)
14. Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. In: IEEE Transactions on Services Computing, vol. PP (2017)
15. Ghattas, J., Soffer, P., Peleg, M.: Improving business process decision making based on past experience. Decision Support Systems **59**, 93–107 (2014)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software. ACM SIGKDD Explorations Newsletter **11**(1), 10–18 (2009)
17. Hall, R.W.: Queueing methods for services and manufacturing (1990)
18. IEEE Task Force on Process Mining: Process mining manifesto. In: F. Daniel, K. Barkaoui, S. Dustdar (eds.) Business Process Management Workshops, *Lecture Notes in Business Information Processing*, vol. 99, pp. 169–194. Springer Berlin Heidelberg (2012)
19. 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 (2013)
20. Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., Leymann, F.: Runtime prediction of service level agreement violations for composite services. In: International Workshops, ICSOC/ServiceWave, pp. 176–186. Springer (2009)
21. Leoni, M.D., Aalst, W.M.P.V.D., Dees, M.: A General Framework for Correlating Business Process Characteristics. In: Business Process Management, pp. 250–266 (2014)
22. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: BPM (2015)
23. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive Monitoring of Business Processes, pp. 457–472. Springer International Publishing, Cham (2014)
24. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval, 1st edn. Cambrige University Press (2008)
25. Mitchell, T.M.: Machine Learning, 1st edn. McGraw-Hill (1997)
26. N. Tax I. Verenich, M.L.R., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: CAiSE (2017)
27. Pandey, S., Nepal, S., Chen, S.: A test-bed for the evaluation of business process prediction techniques. In: 7th ICCC: Networking, Applications and Worksharing (CollaborateCom), pp. 382–391 (2011)
28. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-Aware Remaining Time Prediction of Business Process Instances. In: IJCNN (WCCI) (2014)
29. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Information Systems **54**, 1 – 14 (2015)
30. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Information Systems **53**, 278 – 295 (2015)
31. Smola, A.J., Schölkopf, B.: A Tutorial on Support Vector Regression. Statistics and Computing **14**(3), 199–222 (2004)
32. van der Spoel, S., van Keulen, M., Amrit, C.: Process Prediction in Noisy Data Sets: A Case Study in a Dutch Hospital, pp. 60–83. Springer Berlin Heidelberg (2013)
33. Teinemaa, I., Dumas, M., Maggi, F.M., Di Francescomarino, C.: Predictive business process monitoring with structured and unstructured data. In: M. La Rosa, P. Loos, O. Pastor (eds.) BPM 2016, pp. 401–417. Springer International Publishing (2016)
34. Verbeek, E., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: ProM 6 : The Process Mining Toolkit. In: BPM 2010 Demos, pp. 34–39. Springer (2010)
35. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Francescomarino, C.D.: Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In: 11th International Workshop on BPI 2015, pp. 218–229. Springer (2016)