DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
4

ILYA VERENICH

Explainable Predictive Monitoring of Temporal Measures of Business Processes

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

4

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

4

ILYA VERENICH

Explainable Predictive Monitoring of Temporal Measures of Business Processes

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on December 28, 2018 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors

Prof.	Marlon Dumas University of Tartu, Estonia
Assoc. Prof.	Fabrizio Maria Maggi University of Tartu, Estonia
Prof.	Marcello La Rosa University of Melbourne, Australia
Prof.	Arthur ter Hofstede Queensland University of Technology, Australia

Opponents

Prof. Dr.	Peter Fettke German Research Center for Artificial Intelligence (DFKI) Germany
Prof.	Alfredo Cuzzocrea, PhD University of Trieste Italy

The public defense will take place on February 11, 2019 at 10:15 in J.Liivi 2-405.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

European Union
European Regional
Development Fund

Investing
in your future

Eesti Teadusagentuur
Estonian Research Council

ISSN 2613-5906
ISBN 978-9949-77-962-8 (print)
ISBN 978-9949-77-963-5 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

ABSTRACT

Business process monitoring is a central component in any Business Process Management (BPM) initiative. Process monitoring enables analysts and managers to understand which processes are under-performing and to intervene accordingly. Traditional process monitoring techniques are designed to provide a picture of the current performance of a process. As such, they allow us to identify degradations events in the performance of a process, such as an increase in the number of deadline violations. However, these techniques only identify degradation events after the fact, and hence they do not allow us to act in a preventive manner.

Advances in machine learning combined with the availability of process execution data have led to the emergence of *predictive process monitoring techniques*. These techniques allow us to predict future states of the process, thus enabling process workers and operational managers to intervene, in real-time, in order to prevent or otherwise mitigate performance issues or compliance violations.

In the past years, various approaches have been proposed to address typical predictive process monitoring questions, such as whether or not a running process instance will meet its performance targets, or when will it finish. This doctoral thesis starts with a systematic literature review and taxonomy of methods for predictive monitoring of temporal measures of business processes. Based on this review, it identifies gaps in the existing body of methods. One of the key gaps identified is the lack of support for *explainability* in existing approaches. Yet in practice, explainability is a critical property of predictive methods. It is not enough to accurately predict that a running process instance will end up in an undesired outcome. It is also important for users to understand why this prediction is made and what can be done to prevent this undesired outcome.

To address this gap, this thesis proposes methods to build predictive models to monitor temporal measures in an explainable manner. The contribution of the thesis is the design and evaluation of explainable predictive process monitoring techniques for temporal performance measures based on a principle known as "Transparent Box Design". Specifically, the proposed methods decompose a prediction into elementary components.

In one embodiment of this general idea, we apply deep neural network models that have been shown to achieve high levels of accuracy in the context of predictive process monitoring. In order to make these models explainable, we use an instance of multi-task learning where several related predictive monitoring tasks are integrated into a single system which is trained jointly. In a second embodiment, we propose an explainable predictive process monitoring method by extracting a BPMN process model from the event log, predicting the temporal performance at the level of activities, and then aggregating these predictions at the level of the whole process via flow analysis techniques. Both of these embodiments provide local interpretations of the predictions made for each process instance.

In the field of machine learning, it has been observed that there is a fundamen-

tal trade-off between explainability and accuracy. To evaluate this trade-off, we perform a comparative evaluation of the proposed explainable techniques against each other and against various state-of-the-art baselines, including black-box techniques that do not produce explainable predictions. The evaluation is performed on 17 real-life event logs exhibiting different characteristics and originating from different domains.

The research contributions of the thesis have been consolidated into an open-source toolset for predictive business process monitoring, namely Nirdizati. It is a Web-based predictive business process monitoring engine that can be used to train predictive models using the methods described in this thesis, as well as third-party methods, and then to make predictions at runtime, for ongoing process instances. Nirdizati has been validated in collaboration with potential users, providing important feedback as to how this technology is likely to be used in an enterprise setting.

Finally, we present an application of the proposed explainable predictive monitoring methods to improve the efficiency of business processes. Namely, we demonstrate how these methods can be applied to reduce overprocessing waste by ordering certain activities at runtime based on their expected outcome and effort determined via predictive models.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AUC	Area under ROC curve
BPI	Business process intelligence
BPIC	Business process intelligence challenge
BPM	Business process management
BPMN	Business process model and notation
CSV	Comma-separated values file
CV	Coefficient of variation
FA	Flow analysis
JSON	JavaScript object notation
KPI	Key performance indicator
LSTM	Long short-term memory unit
MAE	Mean absolute error
MTL	Multi-task learning
RNN	Recurrent neural network
ROC	Receiver operating characteristic curve
RPST	Refined process structure tree
RQ	Research question
SESE	Single-Entry-Single-Exit
SLA	Service level agreement
SLR	Systematic literature review
SPN	Stochastic Petri net
STL	Single-task learning
SVM	Support vector machine
TS	Transition system
URL	Uniform Resource Locator
XES	IEEE Standard for extensible event stream
XML	Extensible markup language

1. INTRODUCTION

1.1. Problem Area

Business Process Management (BPM) is “a body of methods, techniques and tools to discover, analyze, redesign, execute and monitor business processes” [32] . In this context, a business process is viewed as a collection of inter-connected events, activities and decision points that involve a number of human actors, software systems and physical and digital objects, and that collectively lead to an outcome that adds value to the involved actors.

BPM activities can be organized in a cycle consisting of the following stages (Figure 1) [32]:

- *Process identification.* Processes relevant to the problem being addressed are identified, delimited and related to each other. The outcome of this phase is a process architecture that presents an overview of the identified processes, along with their relations.
- *Process discovery.* The present state of a process is described, in the form of as-is process models.
- *Process analysis.* Issues related to the as-is process are identified and, if possible, quantified using performance metrics.
- *Process redesign.* Options for changes to the process that would help tackle the previously found issues are identified. The output is a to-be process model.
- *Process implementation.* The changes required to migrate from the as-is process to the to-be process are organized and implemented.
- *Process monitoring.* Once the process has been redesigned, data related to the process execution are collected and analyzed to assess the process performance with respect to its performance criteria.

Business processes are typically supported by enterprise systems that record data about each individual execution of a process, also called a process instance or a *case*. These data can be extracted as *event logs* in order to perform various forms of business process analytics.

This research concerns the process monitoring phase of the BPM lifecycle. Herein, data related to process execution are collected and analyzed in order to assess the process performance with respect to a given set of performance criteria, such as process duration or cost [135].

Traditional approaches to process monitoring are based on “post-mortem” (offline) analysis of process execution. This range of techniques is usually referred to as *process intelligence*. They take as input a database of completed process instances that relate to a specific timeframe (e.g. last six months) and output process performance insights, such as identified bottlenecks or historical case duration.

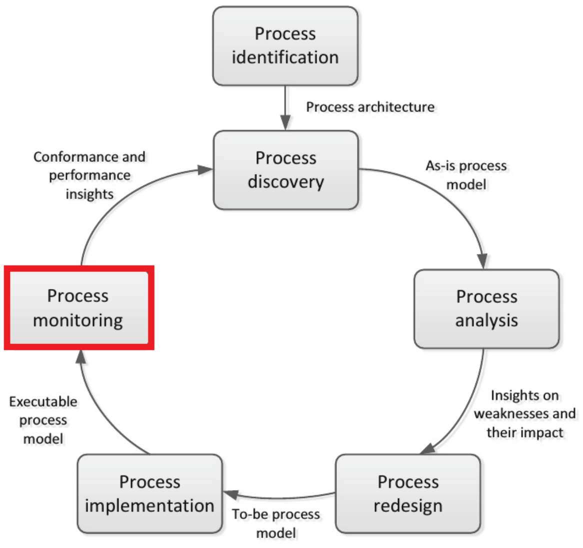
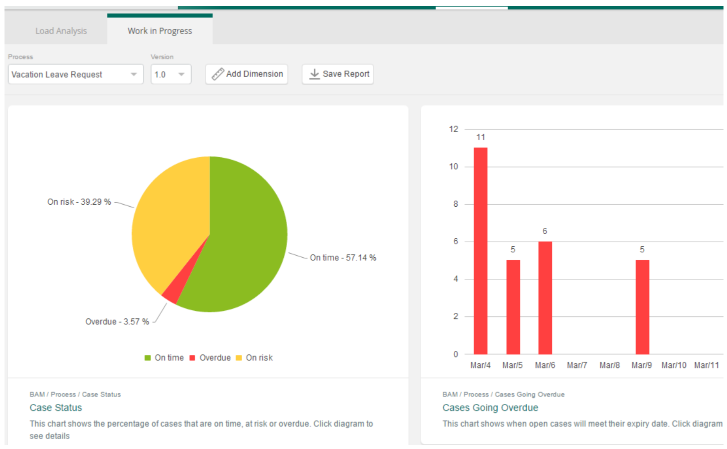


Figure 1: BPM lifecycle model [32].

An example of a process analytics tool is the open-source framework *ProM*[1] that employs a plugin architecture to extend its core functionality. Commercial alternatives for process analytics include Celonis[2], Minit[3], myInvenio[4] and Fluxicon Disco[5].

Another approach to process monitoring includes observations of process work by process analysts at runtime, i.e. *online*. These techniques known as *business activity monitoring* take as input an event stream, i.e. prefixes of ongoing process cases, and output a real-time overview of the process performance, such as the current process load or problematic cases and current bottlenecks. Tools with business activity monitoring capabilities usually provide their output in the form of reports and dashboards. Figure 2 provides an example of a work-in-progress report as produced by the Bizagi Business Activity Monitoring tool [6]. The pie-chart on the left shows the percentage distribution of cases based on whether they are running on time, at risk, or overdue. The bar chart shows the target resolution date of ongoing cases. The horizontal axis depicts the next eight days, and the vertical axis shows the number of cases that are due to expire on each day. For example, there are 11 cases expiring in 1 day and 6 cases expiring in 3 days.

[1]http://www.promtools.org/
[2]http://www.celonis.com/en/
[3]http://www.minit.io/
[4]http://www.my-invenio.com/
[5]http://fluxicon.com/disco/
[6]http://www.bizagi.com/



Process: Vacation Leave Request
Version: 1.0

BAM / Process / Case Status
Case Status
This chart shows the percentage of cases that are on time, at risk or overdue. Click diagram to see details

BAM / Process / Cases Going Overdue
Cases Going Overdue
This chart shows when open cases will meet their expiry date. Click diagram

Figure 2: An example of a real-time work in progress report produced by Bizagi

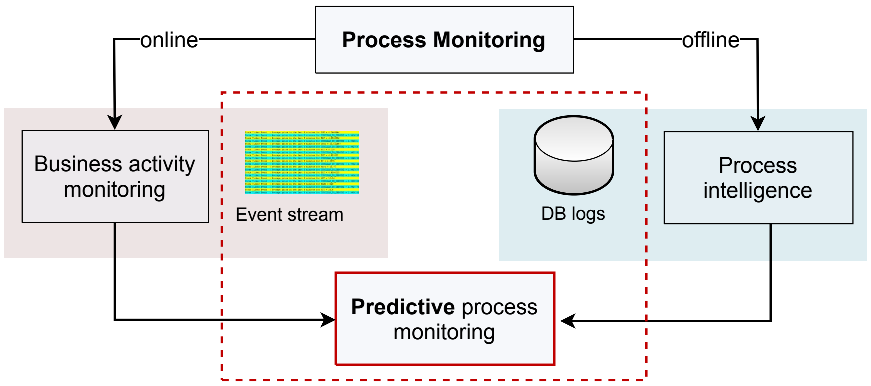


Figure 3: Process monitoring methods [135]

1.2. Problem Statement

Although process intelligence and business activity monitoring techniques are able to provide estimations of process performance, they are in fact reactive, in the sense that they detect process issues only once they have happened, regardless of whether they use historical data or not. *Predictive* process monitoring aims to overcome the limitations of traditional “empirical” monitoring practices by using data produced during process execution to continuously monitor processes performance [138]. In other words, predictive process monitoring builds on top of both process intelligence and business activity monitoring since it taps into past data to allow process workers to steer the process execution by taking preemptive actions to achieve performance objectives (Figure 3).

Prediction models form the core of every predictive process monitoring system. They are built for a specific prediction goal based on an event log of completed cases. At runtime, these models are applied to prefixes of ongoing cases in order to make predictions about future case performance. If the predicted outcome deviates from the normal process behavior, a system raises an alert to the process workers and possibly makes recommendations to them as to the impact of a certain next action on future performance. Therefore, problems are anticipated and can be *proactively* managed.

For example, in a freight transportation process, a prediction goal can be the occurrence of a delay in delivery time. In this case, the outcome will be whether the delay is going to occur (or the probability of delay occurrence). If a delay is predicted, faster means of transport or alternative transport routes could be scheduled proactively and before the delay actually occurs [77].

Predictive process monitoring takes a set of historical process execution cases (i.e. completed cases) and an ongoing process case as input and predicts desired properties of the ongoing case. In other words, predictive process monitoring aims at predicting the value of a given function $F(p,T)$ over a set of completed cases, where p is the case prefix of an ongoing case and T is the set of completed cases. An ongoing case represented as a sequence of so far completed activities is often referred to as a *trace*.

The problem of predictive process monitoring can be unfolded as shown in Figure 4. A *Prediction point* is a point in time where the prediction takes place. A *Predicted point* is a point in time in the future where the performance measure has the predicted value. A prediction is thus based on the knowledge of the predictor on the history of the process execution to the prediction point and the future to the predicted point. The former is warranted by the predictor's *memory* and the latter is based on the predictor's *forecast* (i.e. predicting the future based on trend and seasonal pattern analysis). Finally, the prediction is performed based on a *reasoning method*.

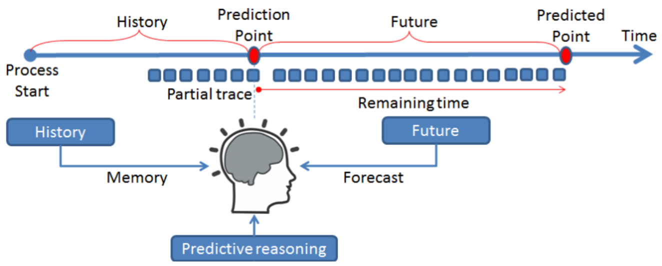


Figure 4: Overview of predictive process monitoring.

Since in real-life business processes the amount of uncertainty increases over time (*cone of uncertainty* [112]), the prediction task becomes more difficult and generally less accurate. As such, predictions are typically made up to a specific point of time in the future, i.e. the time horizon h. The choice of h depends on how fast the process evolves and on the prediction goal.

1.2.1. Explainable Predictions

Modern machine learning algorithms are capable of building more accurate models but they become increasingly complex and are harder for users to interpret [16]. As a result of this inverse relation between model explainability and accuracy, in many real-life problems, black-box machine learning models, such as neural networks, are less preferable since users cannot see the reasoning behind the predictions. This motivates the need to improve the explainability of complex models [59]. Most existing approaches focus on explaining how the model made certain predictions. For instance, Ribeiro et al. [98] approximate predictions of black-box classifiers locally with an explainable model, while Zhou et al. [147] provide a global approximation of neural networks with symbolic rules. However, these approaches generally focus on post-hoc explainability where a trained model is given and the goal is to explain its predictions. By contrast, our objective is to incorporate explainability directly into the structure of the model.

In this thesis, we apply a principle known as "Transparent Box Design" [44]. Specifically, our approach is grounded in the field of transparent box design for *local* interpretability, meaning that we provide explanations for individual predictions, rather than globally for all predictions. To this end, we decompose the predictions made for a given instance (case) into components. These components are the tasks and the conditional flows. For a given instance, we make a prediction that is composed from the predictions from each of these components. For example, the remaining time of the case is explained in terms of the predicted execution times for each task. In this way, users will know which elements contributed to the prediction, i.e. where the time is expected to be spent. As such, we propose a transparent box design that provides a local explanation of the predictions made for each process instance.

1.2.2. Process Performance Measures

A performance measure is a function that maps an activity instance or a process instance (case) to a numerical value for the purpose of reflecting the goodness of its performance. Performance measures can be classified according to multiple dimensions including time, cost, quality, and flexibility [32].

In this thesis, we focus on predicting temporal measures, i.e. performance measures that deal with time. Temporal performance measures include cycle time (also known as throughput time, lead time, or case duration), processing time and waiting time. These measures can be defined at the level of process instances or at the level of individual activities.

At runtime, it makes sense to talk about *remaining* time, which is defined for running cases as opposed to completed cases. As such, one can define remaining cycle time, remaining processing time and remaining waiting time. For example, remaining cycle time reflects the difference between the cycle time of the ongoing (incomplete) case, and its currently elapsed time.

Temporal performance measures defined for cases can be lifted to the level of processes via an aggregation function (e.g. average). For example, we can refer to the average cycle time of a process as being equal to the average of the cycle times of a set of completed cases of the process observed over a given period of time.

1.2.3. Research Questions

The objective of this thesis is to design and evaluate methods for predictive process monitoring, with a particular focus on the explainability of predictions. Additionally, we aim to investigate the trade-off between the explainability and accuracy of predictions. Another focal point is providing a toolset for the replication of the results and the application of the proposed methods at runtime, in order to make predictions for ongoing business process instances. Specifically, the thesis pursues the following research questions:

RQ1 How to make predictive process monitoring techniques for temporal measures more explainable to business process stakeholders?

RQ2 What is the impact of explainability on prediction accuracy in the context of predictive process monitoring techniques for temporal measures?

RQ3 How to embed existing techniques for predictive process monitoring into dashboard-based monitoring systems?

As a prerequisite to answering these research questions, we provide a comprehensive survey and taxonomy of predictive process monitoring methods proposed in related work, and identify the gap in existing methods. This leads to a further research question:

RQ0 What methods exist for predictive monitoring of remaining time of business processes?

1.2.4. Solution Criteria

The predictive monitoring methods that will be designed as a result of this research will be evaluated using the following criteria:

- Explainability: the solution should be able to explain predictions about ongoing cases by decomposing them into elementary components, at the level of activities.
- Accuracy: the solution should be able to make predictions about a given outcome with relatively high levels of accuracy (e.g. above 80%).
- Earliness: the solution should be able to make predictions as early as possible, i.e. as soon as there is enough information in an ongoing case to make a prediction with sufficient confidence.
- Low runtime overhead: the solution should be able to make predictions almost instantaneously (i.e. within 1 second), with limited computing resources, over real-scale scenarios.

- Generality: the proposed solution should be able to deal with logs with different characteristics and originating from different domains.

1.3. Research Benefits and Innovation

The importance of this research is emphasized by the need of many organizations to manage business processes, as shown by Gartner's 2018 Process Mining Market Analysis [73]. Based on its extensive interaction with end-user clients, academic researchers and major process mining vendors, Gartner identified key process mining capabilities. The results highlight the importance of predictive and prescriptive analytics as well as real-time dashboards with support for KPIs that are continuously monitored. These insights could be used to support process workers and operational managers in taking data-aware decisions as business processes unfold. Ultimately, predictive process monitoring enables evidence-based business process management wherein every business decision is made with data, backed by data and continuously put into question based on data.

1.4. Research Approach

The purpose of this project is to devise and operationalize a holistic approach for predictive monitoring of business processes. In line with the defined research questions, this approach will allow for prediction of process case evolution until its completion and prediction of specific characteristics of the case.

The research approach of this project follows the Design Science methodology. Hevner et al. [46] recommend several guidelines for design science research:

Design as an artifact. As an outcome of the project, we produce viable artifacts in the form of models for predicting various properties of running business processes, e.g. whether the process case will meet a certain performance objective and whether the case will contain a re-work loop. Aside from predictions, the models will potentially be able to explain the causes of various case properties.

Problem relevance. The term *relevance* is used to address a research that introduces new knowledge on a topic, proposing a novel and better approach to a problem, or a solution to an unsolved problem. This research is relevant because both practitioners and academics have recognized the importance of predictive monitoring as part of the BPM lifecycle, and academics have been tackling this problem in various contexts.

Design evaluation. A research contribution requires a rigorous evaluation of the artifacts enabling assessment of their utility. Throughout this research, we evaluate the results using a range of real-life datasets exhibiting different characteristics (e.g. derived from different domains, having different attributes), to ensure the generalizability of our contribution.

Research rigor. The term *rigor* is used to identify a scientific approach achieved by applying existing foundations and well-established evaluation methods. The rigor of the approach is ensured by conducting an extensive literature review, using formal methods and state-of-the-art machine learning and process mining algorithms, as well as by constructing a comprehensive evaluation benchmark, using well-defined selection and assessment criteria.

Communication of research. The results of the research are communicated to the scientific community via research papers published in international conferences and journals and presentations at conferences, workshops and other venues.

An underlying assumption of this research is that it is possible to extract from data already stored in event logs non-trivial patterns that can assist in prediction of future process behavior. Hence, all our devised techniques are based on the analysis of event log data. Some techniques make use of only minimal information, such as event classes and their timestamps, while others take into account all available attributes from the executed events, either in raw or aggregated form.

1.5. Thesis Outline

This thesis is organized as follows. Chapter 2 provides definitions, principles and basic concepts from process mining and machine learning areas that are referenced throughout the thesis. Additionally, we provide the relevant background on process models and temporal process performance measures.

Chapter 3 provides a systematic literature review and taxonomy of techniques proposed for predictive business process monitoring of temporal properties, and identifies the gaps in the existing body of methods.

Chapter 4 introduces deep learning models, specifically recurrent neural networks (RNN) and long short-term memory networks (LSTM) for the prediction of temporal measures. Deep learning models are *black-box models*, focusing on accurately describing the sequences, rather than explainability of the results. In this chapter, we aim to make these models explainable by applying them in a multi-task setting where several predictive process monitoring tasks are integrated into a single system which is trained *jointly*.

Chapter 5 proposes a process model driven method for the explainable predictive process monitoring of continuous performance measures, such as the remaining time, based on a BPMN process model automatically discovered from the event log. Unlike in the approach proposed in Chapter 4, here we additionally model decision gateways.

In Chapter 6, we present a comprehensive evaluation of our proposed explainable techniques against each other and various state-of-the-art baselines identified in Chapter 3. Namely, we demonstrate the impact of explainability on prediction accuracy in the context of early prediction of temporal performance measures.

Chapter 7 presents a prototype of an integrated software platform for predictive business process monitoring, namely Nirdizati, that consolidates the research contributions of this thesis, and illustrates a concrete application of this platform in practice.

In Chapter 8, we discuss a possible application of the process model driven technique presented in Chapter 5 to improve the efficiency of business processes. Namely, we illustrate how overprocessing waste can be reduced by ordering certain activities at runtime based on their expected outcome and effort determined via predictive models.

Finally, Chapter 9 concludes this thesis by providing a summary of our contributions and discussing possible avenues for future work.

2. BACKGROUND

Predictive process monitoring is a multi-disciplinary area that draws concepts from process mining on the one side, and machine learning on the other side. In this section, we introduce concepts from the aforementioned areas that will be referred to in later chapters. In particular, Section 2.1 introduces basic process mining concepts and notations, Section 2.2 provides the relevant background on process models, Section 2.3 introduces temporal process performance measures. Finally, Section 2.4 introduces the concept of machine learning, briefly discusses three machine learning algorithms used in this thesis – decision tree, gradient boosting and support vector machine – as well as common hyperparameter tuning strategies for these algorithms.

2.1. Process Mining

Business processes are generally supported by information systems that record data about each individual execution of a process (also called a *case*). Process mining [129] is a research area within business process management that is concerned with deriving useful insights from process execution data (called event logs). Process mining techniques are able to support various stages of business process management tasks, such as process discovery, analysis, redesign, implementation and monitoring. In this section, we introduce the key process mining concepts.

Each case consists of a number of *events* representing the execution of activities in a process. Each event has a range of attributes of which three are mandatory, namely (i) *case identifier* specifying which case generated this event, (ii) the *event class* (or *activity name*) indicating which activity the event refers to and (iii) the *timestamp* indicating when the event occurred[2]. An event may carry additional attributes in its payload. For example, in a patient treatment process in a hospital, the name of a responsible nurse may be recorded as an attribute of an event referring to activity "Perform blood test". These attributes are referred to as *event attributes*, as opposed to *case attributes* that belong to the case and are therefore shared by all events relating to that case. For example, in a patient treatment process, the age and gender of a patient can be treated as a case attribute. In other words, case attributes are static, i.e. their values do not change throughout the lifetime of a case, as opposed to attributes in the event payload, which are dynamic as they change from an event to the other.

Formally, an event record is defined as follows:

Definition 1 (Event)**.** *An event is a tuple $(a, c, t, (d_1, v_1), \dots, (d_m, v_m))$ where a is the activity name, c is the case identifier, t is the timestamp and $(d_1, v_1) \dots, (d_m, v_m)$*

[2]Hereinafter, we refer to the event *completion* timestamp unless otherwise noted.

(where $m \geq 0$) are event attribute names and the corresponding values assumed by them.

Let $\mathscr{E}$ be the event universe, i.e., the set of all possible event identifiers, and $\mathscr{T}$ the time domain. Then there is a function $\pi_{\mathscr{T}} \in \mathscr{E} \to \mathscr{T}$ that assigns timestamps to events.

The sequence of events generated by a given case forms a *trace*. Formally,

Definition 2 (Trace)**.** *A trace is a non-empty sequence $\sigma = \langle e_1, \dots, e_n \rangle$ of events such that $\forall i \in [1..n], e_i \in \mathscr{E}$ and $\forall i, j \in [1..n]\ e_i.c = e_j.c$. In other words, all events in the trace refer to the same case.*

A set of *completed traces* (i.e. traces recording the execution of completed cases) comprises an *event log*.

Definition 3 (Event log)**.** *An event log L is a set of completed traces, i.e., $L = \{\sigma_i : \sigma_i \in \mathscr{S}, 1 \leq i \leq K\}$, where $\mathscr{S}$ is the universe of all possible traces and K is the number of traces in the event log.*

An IEEE standard for representing event logs, called XES (eXtensible Event Stream), has been introduced in [1]. The standard defines the XML format for organizing the structure of traces, events and attributes in event logs. It also introduces some extensions that define some attributes with pre-defined meaning such as: (i) *"concept:name"*, which stores the name of event/trace; (ii) *"org:resource"*, which stores the name/identifier of the resource that triggered the event (e.g., a person name); (iii) *"org:group"*, which stores the group name of the resource that triggered the event.

As a running example, let us consider an extract of an event log originating from an insurance claims handling process (Table 1). The activity name of the first event in case 1 is A, it was completed on *1/1/2017* at *9:13AM*. The additional event attributes show that the cost of the activity was 15 units and the activity was performed by *John*. These two are event attributes. The events in each case also carry two case attributes: the age of the applicant and the channel through which the application has been submitted. The latter attributes have the same value for all events of a case.

Event and case attributes can be categorized into at least two data types – numeric (quantitative) and categorical (qualitative) [117]. With respect to the running example, numeric attributes are *Age* and *Cost*, while categorical attributes are *Channel, Activity* and *Resource*. Each data type requires different preprocessing to be used in a predictive model. Specifically, numeric attributes are typically represented as such, while for categorical attributes, one-hot encoding is typically applied.

One-hot encoding represents each value of the categorical attribute with a binary vector with the i-th component set to one, and the rest set to zero. In other words, it provides a 1-to-N mapping where N is the number of possible distinct values (*levels*), or cardinality of an attribute [79]. As an example, let us consider a categorical attribute A with cardinality N. We take an arbitrary but consistent

Table 1: Extract of an event log.

Case	Case attributes		Event attributes			
ID	Channel	Age	Activity	Timestamp	Resource	Cost
1	Email	37	A	1/1/2017 9:13:00	John	15
1	Email	37	B	1/1/2017 9:14:20	Mark	25
1	Email	37	D	1/1/2017 9:16:00	Mary	10
1	Email	37	C	1/1/2017 9:18:00	Mark	10
1	Email	37	F	1/1/2017 9:18:05	Kate	20
1	Email	37	G	1/1/2017 9:18:50	John	20
1	Email	37	H	1/1/2017 9:19:00	Kate	15
2	Email	52	A	2/1/2017 16:55:00	John	25
2	Email	52	D	2/1/2017 17:00:00	Mary	25
2	Email	52	B	3/1/2017 9:00:00	Mark	10
2	Email	52	C	3/1/2017 9:01:00	Mark	10
2	Email	52	F	3/1/2017 9:01:50	Kate	15

ordering over the set of levels of A, and use $index \in A \to \{1,\dots,N\}$ to indicate the position of a given attribute value a in it. The one-hot encoding assigns the value 1 to feature number $index(a)$ and a value of 0 to the other features.

As we aim to make predictions for traces of incomplete cases, rather than for traces of completed cases, we define a function that returns the first k events of a trace of a (completed) case.

Definition 4 (Prefix function)**.** *Given a trace $\sigma = \langle e_1,\dots,e_n\rangle$ and a positive integer $k \le n$, $hd^k(\sigma) = \langle e_1,\dots,e_k\rangle$.*

For example, for a sequence $\sigma_1 = \langle a,b,c,d,e\rangle$, $hd^2(\sigma_1) = \langle a,b\rangle$.

The application of a prefix function will result in a *prefix log*, where each possible prefix of an event log becomes a trace.

Definition 5 (Prefix log)**.** *Given an event log L, its prefix log L^* is the event log that contains all prefixes of L, i.e., $L^* = \{hd^k(\sigma) : \sigma \in L, 1 \le k \le |\sigma|\}$.*

For example, a complete trace consisting of three events would correspond to three traces in the prefix log – the partial trace after executing the first, the second and the third event. For event logs where case length is very heterogeneous, instead of including every prefix of a trace, it is common to include only prefixes with specific length in the prefix log during the training procedure. This can be done by retaining all prefixes of up to a certain length [66, 121]. An alternative approach is to only include prefixes $hd^k(\sigma)$ such that $\left(\left|hd^k(\sigma)\right| - 1\right) \bmod g = 0$ where g is a gap size. This approach has been explored in [31].

2.2. Process Models

Modern BPM heavily relies on *business process models*. Process models provide a visual representation of the underlying business process. A process model consists of a set of activities and their structuring using directed control flow edges and gateway nodes that implement process routing decisions [92]. Formally,

Definition 6 (Process model)**.** *A process model is a tuple* $(N, E, type)$*, where* $N = N_A \cup N_G$ *is a set of nodes (*N_A *is a nonempty set of activities and* N_G *is a set of gateways; the sets are disjoint),* $E \subseteq N \times N$ *is a set of directed edges between nodes defining control flow, type is a function* $N_G \rightarrow \{AND, XOR, OR\}$ *is a function that assigns a control flow construct to each gateway.*

Ter Hofstede et al. [124] identify three core purposes of a model:

- Providing insight: Process models can be used to provide a clear overview of aspects related to the process to different stakeholders involved in the process.
- Analysis: Process models can be analyzed, depending on the type of model, to check the performance of the system or to detect errors and inconsistencies in the process.
- Process enactment: Process models can be used to execute business processes.

Many modeling notations have been proposed to represent process models, including Event-driven Process Chains (EPC), UML Activity Diagrams and Petri nets. Nevertheless, the definition above is rather generic and abstracts from the specific notation.

One of the most common process modeling notations is the Business Process Model and Notation (BPMN) developed by the Object Management Group (OMG). Figure 5 shows a subset of the core elements of BPMN. The *start* and *end* events represent the initiation and termination of an instance of a process respectively. The tasks denote activities to be performed. The flow represents the order among the events, gateways and tasks. Finally, *gateways* are control flow elements and they can represent either the splitting or merging of paths. In the case of *exclusive gateways* (also known as XOR-gateways), a split has more than one outgoing flow, but only one of them can be activated, according to a pre-defined condition. Its counterpart, the join exclusive gateway, merges the incoming alternative flows. Conversely, a *parallel gateway* (also known as AND-gateways) denotes the parallel activation of all the outgoing branches; whereas, the merging counterpart denotes the synchronization of the multiple incoming paths [3].

A process model can be decomposed into process fragments. A process fragment is a connected part of a process model. For the purpose of this study, we require a fragment to have a single entry and a single exit point. Such fragments are referred to as *SESE fragments*. From a modeling perspective, SESE fragments are very handy: structurally every SESE fragment can be replaced with one aggregating activity. A model that can be decomposed into SESE fragments, without impacting the behavior it describes, is called a block-structured, or a *structured* model.

A process model can be provided by the process stakeholders or can be automatically discovered from the corresponding event log via a process discovery

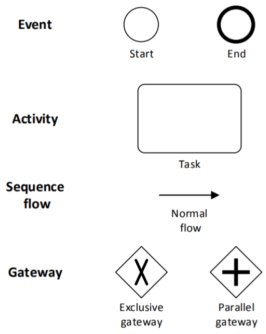


Figure 5: Core BPMN elements [3].

algorithm [128].

Definition 7 (Process discovery algorithm)**.** *Let L be an event log as specified in Definition 3. A process discovery algorithm is a function that maps L onto a process model such that the model is "representative" for the behavior seen in the event log.*

A wide range of process automated process discovery algorithms have been proposed in the literature [5], with Petri nets and BPMN being the most common output model formats.

2.3. Temporal Process Performance Measures

Temporal performance measures include cycle time, processing time and waiting time.

Cycle time is one of the most important measures of performance of a business process [63]. It is also known as lead time or throughput time [97]. Cycle time of a case is the time that it takes to complete an individual case from start to finish. In other words, it is the time that it takes for a case to go from the entry point to exit point of a process [63]. For example, the cycle time of an insurance claim handling process may be the time that elapses from the moment a customer submits the claim until the claim is resolved and the file is closed.

Cycle time can be decomposed into two components. First, processing time (also called service time) includes the time that resources, such as process participants or software applications invoked by the process, spend on actually handling the case [32]. Processing time typically refers to value-adding activities. By contrast, waiting time is associated with non-value-adding activities. It is the time that a case spends in idle mode. Waiting time includes queueing time that happens when no resources are available to handle the case – and other waiting time, for example because synchronization must take place with another process, with other activities, or because an input is expected from a customer or from another external party [32]. In many processes, the waiting time makes up a considerable proportion of the overall cycle time. This situation may, for example, happen

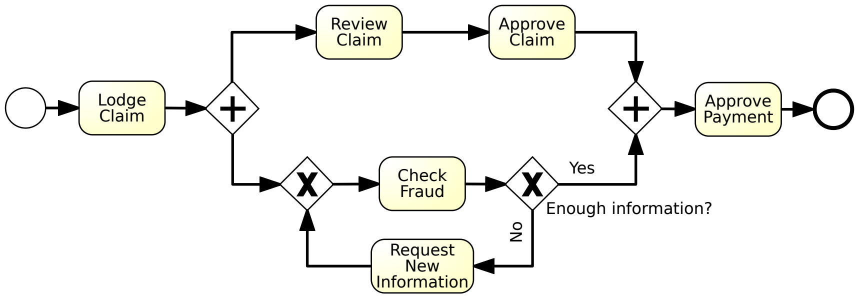


Figure 6: Process model of a claim handling process

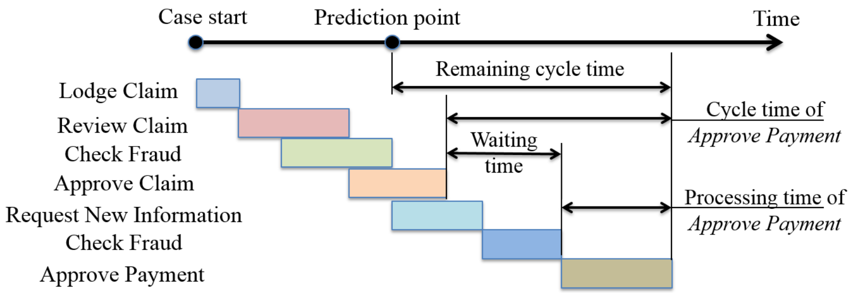


Figure 7: Cycle time components

when the work is performed in batches. In a process related to the approval of purchase requisitions at a company, the supervisor responsible for such approvals in a business unit may choose to batch all applications and check them only once at the start or the end of a working day [32].

Cycle time can also be defined at the level of specific activities. Formally,

Definition 8 (Cycle time of an activity). *A* cycle time *of an activity i is the time it takes between the moment the activity is ready to be executed and the moment it completes. By "ready to be executed" we mean that all activities upon which the activity in question depends have completed. Formally, cycle time is the difference between the timestamp of the activity and the timestamp of the previous activity. i.e.* $\pi_{\mathcal{T}}(\sigma(i)) - \pi_{\mathcal{T}}(\sigma(i-1))$ *for* $1 \leq i \leq |\sigma|$. *Here,* $\pi_{\mathcal{T}}(\sigma(0))$ *denotes the start time of the case.*

As for the process instance level, the cycle time of an activity includes the processing time of the activity, as well as all waiting time prior to the execution of the activity. To illustrate this concept, we refer to the process model of a claim handling process (Figure 6). Activity *Approve Payment* is ready to be executed when *Approve Claim* has completed; however, if it cannot start until *Check Fraud* has completed (Figure 7). Thus, the activity will be in the idle mode until then.

In the context of ongoing cases, the above measures can be extended to refer to their partial, or *remaining*, equivalents. For example, remaining cycle time reflects the difference between the cycle time of the ongoing (incomplete) case, and the current cycle time. For the process instance in Figure 7, if the prediction is made after the first execution of *Check Fraud*, the remaining cycle time will be the time from that moment until the case completes. Analogously, one can define remaining processing time and remaining waiting time.

2.4. Machine Learning

Machine learning is a research area of computer science concerned with the discovery of models, patterns, and other regularities in data [82]. Closely related to machine learning is data mining. Data mining is the "core stage of the *knowledge discovery process* that is aimed at the extraction of interesting – non-trivial, implicit, previously unknown and potentially useful – information from data in large databases" [35]. Data mining techniques focus more on exploratory data analysis, i.e. discovering unknown properties in the data, and are often used as a preprocessing step in machine learning to improve model accuracy.

2.4.1. Overview

A machine learning system is characterized by a learning algorithm and training data. The algorithm defines a process of learning from information extracted, usually as *features vectors*, from the training data. In this work, we will deal with *supervised* learning, meaning training data is represented as n labeled samples:

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) : n \in \mathbb{N}\}, \tag{2.1}$$

where $\mathbf{x}_i \in \mathscr{X}$ are m-dimensional feature vectors ($m \in \mathbb{N}$) and $y_i \in \mathscr{Y}$ are the corresponding labels, i.e. values of the target variable.

Feature vectors extracted from the labeled training data are used to fit a predictive model that assigns labels to new data given labeled training data while minimizing error and model complexity. In other words, a model generalizes the pattern, providing a mapping $f : \mathscr{X} \to \mathscr{Y}$. The labels can be either continuous, e.g. cycle time of activity, or discrete, e.g. loan grade. In the former case, the model is referred to as regression; while in the latter case we are talking about a classification model.

From a probabilistic perspective, the machine learning objective is to infer a conditional distribution $P(\mathscr{Y}|\mathscr{X})$. A standard approach to tackle this problem is to represent the conditional distribution with a parametric model, and then to obtain the parameters using a training set containing $\{\mathbf{x}_n, y_n\}$ pairs of input feature vectors with corresponding target output vectors. The resulting conditional distribution can be used to make predictions of y for new values of $\mathbf{x}$. This is called a *discriminative* approach since the conditional distribution discriminates between the different values of y [64].

Another approach is to calculate the joint distribution $P(\mathscr{X}, \mathscr{Y})$, expressed as a parametric model, and then apply it to find the conditional distribution $P(\mathscr{Y}|\mathscr{X})$ to make predictions of y for new values of $\mathbf{x}$. This is commonly known as a

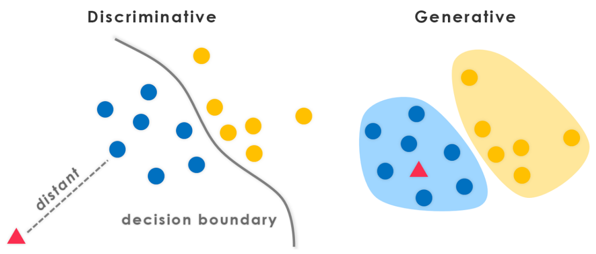


Figure 8: Discriminative and generative models [85].

generative approach since by sampling from the joint distribution one can generate synthetic examples of the feature vector **x** [64].

To sum up, *discriminative* approaches try to define a (hard or soft) decision boundary that divides the feature space into areas containing feature vectors belonging to the same class (see Figure 8). In contrast, *generative* approaches first model the probability distributions for each class and then label a new instance as a member of a class whose model is most likely to have generated the instance [64].

In cases when there is a sufficient amount of labeled data, discriminative models are usually applied since they exhibit very high generalization performance. Specifically, Jordan and Ng [85] showed that logistic regression, which is a purely discriminative model, performs better than its generative counterpart, naive Bayes classifier, given a sufficiently large number of training samples. This results from the fact that even though discriminative models have lower asymptotic error, generative models converge to its (higher) error more quickly.

Although the data collection is often easy, the labeling process can be quite expensive. Accordingly, there is an increasing interest in generative methods, as they can utilize unlabeled training samples, as well as the labeled ones.

The complementary properties of generative and discriminative models have motivated the development of ensemble methods that combine characteristics of purely discriminative or purely generative models, so that the resulting model has an increased performance [64]. In particular, many research proposals have tackled "discriminatively training" generative models [14]. For example, Woodland and Povey [143] used discriminatively trained hidden Markov models to develop a speech recognition framework.

Traditional machine learning models can be extended to predict more complex, structured objects, rather than scalar discrete or continuous values. This is known as structured prediction [6]. In structured prediction, outputs are either structured objects, such as parse trees of natural language sentences and alignments between sentences in different languages, or multiple response variables that are interdependent [6]. The latter is commonly referred to as collective classification. The most common models and algorithms for making structured predictions are Bayesian networks and random fields, structured support vector machines, inductive logic programming, case-based reasoning, Markov logic networks and constrained conditional models [115].

An important issue to be considered when choosing a machine learning model is explainability of the generated models. For example, the black-box patterns produced by neural networks are effectively incomprehensible for the user as they do not provide a structural description. In contrast, structural patterns are comprehensible as their construction reveals the structure of the problem, e.g., association rules [43]. In our research we will prefer models that are easier to interpret, ceteris paribus.

In this research, we will primarily use discriminative models, as case labeling (cycle times, remaining processing times, etc.) is performed automatically by processing event timestamps; thus, we have a sufficient amount of labeled data. However, we will also compare our results with generative models, as well as combinations of discriminative and generative models.

2.4.2. Learning Algorithms

Predictive process monitoring methods have employed a variety of classification and regression algorithms, with the most popular choice being decision trees (e.g. [29, 38]). Although quite simple, decision trees have an advantage in terms of computation performance and explainability of the results. Other choices include random forest [106, 137], support vector regression [91] and extreme gradient boosting [106]. In this section, we describe machine learning algorithms used in this thesis.

Decision trees. Decision trees are hierarchical structures where each internal node has a condition which is evaluated when an example is being classified [17]. Based on the outcome, the example is sent either to the left or to the right child of the node. The process is repeated until a leaf node is reached, when a prediction is made about the example's class. Thus, decision making is organized hierarchically.

The general idea behind decision tree is as follows. A tree learns from data by partitioning the training set into subsets based on the values of the features. At each step the algorithm creates a new node and chooses a suitable condition for splitting the data at that node. This is done in a way that maximizes the *purity* (or, equivalently, minimizes *impurity*) of the node's output after evaluating the condition. Purity indicates the quality of the previous split and can be measured in various ways. Purity is maximal at a node where all input examples have the exact same label.

In case of binary classification, the *Gini index* is commonly used for measuring impurity, which is defined as:

$$G = 2f_1 f_2 = f_1(1-f_1) + f_2(1-f_2) = 1 - (f_1^2 + f_2^2) \tag{2.2}$$

Here, f_1 and f_2 are fractions of input examples with class labels 0 and 1 that satisfy $f_1 + f_2 = 1$. Impurity is the highest when both fractions are 0.5, i.e. the number of correctly and incorrectly labeled examples is the same. The tree is created by choosing such feature and condition that have the smallest Gini index after making the split.

Decision trees are prone to overfitting [102]. Overfitting is the condition of a grown decision tree that is very consistent with the training data and yet not learned underlying patterns. The tree has, in this case, maximized purity on the training set, while an unseen example with a few variations will be incorrectly classified. The standard technique to deal with overfitting is with decision tree *pruning*. Decision tree pruning seeks to reduce the redundancy of the decision nodes. It traverses the tree to find nodes which do not increase purity and shortens the tree by removing such nodes. However, decision tree pruning equivalently reduces the classification accuracy of the tree. There is no classical decision tree approach to increase both classification and generalization accuracy [49].

Ensemble learners. Ensemble methods train multiple predictors for the same task and combine them together for the final prediction. Figure 9 illustrates the general idea behind ensemble learning. Multiple base learners, or "weak" learners, f_k are fitted on the training data or its subset. Their predictions are then combined into one ensemble learner f_Σ by applying some aggregation function $g(\cdot)$. Decision trees are often used as base learners.

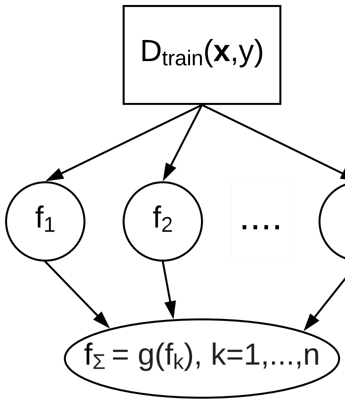


Figure 9: Ensemble learning architecture.

In this thesis, as the primary prediction algorithm, we apply extreme gradient boosting (XGBoost) [23] which is one of the latest implementations of Gradient Boosting Machines (GBM). It has been successfully utilized across various domains [39, 83, 125] as well as in machine learning competitions such as Kaggle[1]. Olson et al. [87] performed a thorough analysis of 13 state-of-the-art, commonly used machine learning algorithms on a set of 165 publicly available classification problems and found that gradient tree boosting achieves the best accuracy for the highest number of problems.

In gradient boosting, predictions from base learners are combined by following a gradient learning strategy [45]. At the beginning of the calibration process, a

[1] `https://www.kaggle.com`

base learner is fit to the whole space of data, and then, a second learner is fit to the residuals of the first one. This process of fitting a model to the residuals of the previous one goes on until some stopping criterion is reached. Finally, the output of the GBM is a kind of weighted mean of the individual predictions of each base learner. Regression trees are typically selected as base learners [126].

The main challenge of boosting learning in general, and of GBMs in particular, is the risk of overfitting, as a consequence of the additive calibration process. Typically, this issue is faced by including different regularization criteria in order to control the complexity of the algorithm. However, the computational complexity of these type of mechanisms is rather high [126]. In this context, the main objective of XGBoost is to control the complexity of the algorithm without excessively increasing the computational cost. To this end, XGBoost aims to minimize the following *Loss+Penalty* objective function:

$$R(\Theta) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \tag{2.3}$$

where Θ includes the model parameters to be learned during training, l is the loss function, which measures the cost between ground truth y_i and prediction $\hat{y}_i$ for the i-th leaf, and $\Omega(f_k)$ is the k-th regularization term. The loss function for the predictive term can be specified by the user, and it is always truncated up to the second term of its Taylor series expansion for computational reasons. The regularization term is obtained with an analytic expression based on the number of leaves of the tree and the scores of each leaf. The key point of the calibration process of XGBoost is that both terms are ultimately rearranged in the following expression [126]:

$$R(\Theta) = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T, \tag{2.4}$$

where G and H are obtained from the Taylor series expansion of the loss function, λ is the L2 regularization parameter and T is the number of leaves. This analytic expression of the objective function allows a rapid scan from left to right of the potential divisions of the tree, but always taking into account the complexity.

Support vector machines. The support vector machine (SVM) classifier has been originally proposed by Vapnik [134] and has become one of the most widely used machine learning techniques. The idea of SVM classification is to find for given classification task a linear separation boundary $\mathbf{w}^T\mathbf{x} + b = 0$ that correctly classifies training samples. Assuming that such a boundary exists, we search for maximal margin separating hyperplane, for which the distance to the closest training sample element is maximal.

Definition 9 (Margin)**.** *The margin m_i of a training sample $\{(\mathbf{x}_i, y_i)\}$ with respect to the separating hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ is the distance from the point $\mathbf{x}_i$ to the* *hyperplane:*

$$m_i = \frac{\|\mathbf{w}^T\mathbf{x}_i + b\|}{\|\mathbf{w}\|}$$

The margin m of the separating hyperplane with respect to the whole training set $\mathscr{X}$ is the smallest margin of an instance in the training set:

$$m = \min_i m_i$$

Finally, maximal margin separating hyperplane for a training set $\mathscr{X}$ is the separating hyperplane having the maximal margin with respect to the training set. Among all existing hyperplanes the maximal margin separating hyperplane is considered to be optimal. The equivalent formalization of this problem of optimization is SVM classifier with linear hard margin, which, according to Vapnik's original formulation, satisfies the following conditions [105]:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq +1, & \text{if } y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1, & \text{if } y_i = -1 \end{cases} \tag{2.5}$$

which is equivalent to:

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1, i = 1, \ldots, n \tag{2.6}$$

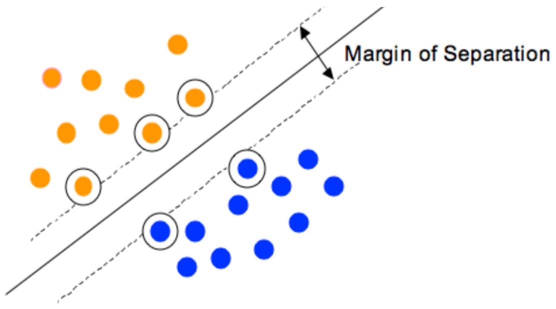


Figure 10: Maximal margin separating hyperplane with support vectors.

The optimization problem is defined as:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}, \tag{2.7}$$

subject to Equation 2.6.

This optimization problem can be solved using the Lagrangian [105]. The solution of this optimization problem is the parameters $(\mathbf{w}_0, b_0)$, which are determined as:

$$\begin{aligned} \mathbf{w}_0 &= \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ b_0 &= \frac{1}{y_k} - \mathbf{w}_0^T\mathbf{x}_k, \end{aligned} \tag{2.8}$$

where α_i are Lagrange multipliers and k is an arbitrary index for which $\alpha_k \neq 0$. The resulting hyperplane is completely defined by the training samples, which are at the minimal distance to this hyperplane. They are showed at Figure 10 in circles. These training samples are called *support vectors*, from which the algorithm inherits its name.

As it is sometimes impossible to find a hyperplane that separates two classes perfectly, there have been proposed various modifications to the hard margin SVM. Using the so-called soft margin hyperplane, it is possible to tune the amount of misclassified training points produced by SVM classifier. The optimization problem of soft margin SVM is then [113]:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi + i \tag{2.9}$$

subject to:

$$y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i, i = 1, \ldots, n \tag{2.10}$$

The variable ξ_i is a slack variable, which allows misclassifications, C is a tuning parameter, which is the penalty parameter of the error term. Thus, C determines the trade-off between two competing goals: maximizing the margin and minimizing the training error. Large values of C force the model to have as few misclassifications as possible, but at the price of large values of w_i. Small C allows the model to have more errors, but it provides smaller coefficient values.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using so-called *kernel trick*, implicitly mapping their inputs into high-dimensional feature spaces [113].

2.4.3. Hyperparameter Optimization

Each prediction technique is characterized by model parameters and by hyperparameters [10]. While model parameters are learned during the training phase so as to fit the data, hyperparameters are set outside the training procedure and are used for controlling how flexible the model is in fitting the data. For instance, the number of base learners is a hyperparameter of the XGBoost algorithm. The impact of hyperparameter values on the accuracy of the predictions can be extremely high. Optimizing their value is therefore important, but, at the same time, optimal values depend on the specific dataset under examination [10].

A traditional way of performing hyperparameter optimization is *grid search*, which is an exhaustive search through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set [82]. However, in a large, multi-dimensional, search space, *random search* is often more effective, given a fixed number of iterations [10]. In random search, hyperparameters are sampled from a pre-defined search space. Another advantage of random search is that it is trivially

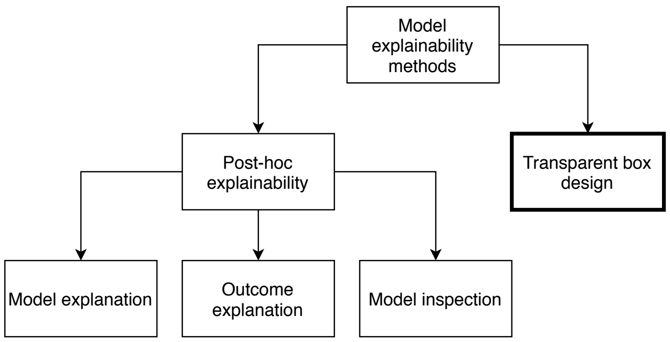


Figure 11: Taxonomy of methods for explaining black box models (adopted from [44]).

parallel [9]. At the same time, both grid and random search memoryless, i.e. they are completely uninformed by past evaluations. As a result, a significant amount of time is spent evaluating inefficient hyperparameters.

This drawback is addressed by sequential model-based global optimization (SMBO), wherein a probabilistic model of the objective function is built. This model provides a mapping from hyperparameter values to the objective evaluated on a validation set and is used to select the next set of hyperparameters to be evaluated. In related work, several SMBO approaches have been recently proposed, such as Bayesian optimization, Sequential Model-based Algorithm Configuration (SMAC) and Tree-structured Parzen Estimator (TPE). Bergstra et al. [10] showed that SMBO approaches find configurations with lower validation set error and with a smaller number of trials.

2.5. Explainability in Machine Learning

A major drawback of black-box machine learning methods in practical deployments is that users have difficulties trusting the prediction when the reasoning behind the prediction is unclear [81, 104]. Indeed, in real-life applications, users do not only need to get predictions, but they also need to be able to act upon these predictions. In turn, to do so, they need to understand the rationale for these predictions. The decision on how to respond to a prediction is largely driven by the user's interpretation of the predictive model's decision logic. If this interpretation matches the intuition of a human decision maker, it might enhance trust in the model [81, 104].

A recent survey by Guidotti et al. [44] provides a comprehensive review of methods for explaining black box models, as well as presents a taxonomy of explainability methods according to the type of problem faced (Figure 11).

The first group of research proposals focus on *explaining* how the model makes certain predictions. To this end, reverse engineering is typically exploited to understand the black box model. These approaches are also known as *post-hoc ex-*

plainability, and they provide interpretations based on natural language explanations, visualizations or by examples. Thus, we can separate two processes – decision making (prediction) and decision explaining. Post-hoc interpretations are also most suitable to provide intuition for complex non-transparent models, such as deep neural networks [104].

Multiple strategies have been proposed for post-hoc explainability of black box models. Firstly, *model explanation* methods seek to provide *globally* explainable models that are able to mimic the behavior of black boxes and that are also understandable by humans. These methods adopt decision trees, sets of rules and linear models as comprehensible global predictors [51]. For example, neural networks can be approximated with decision trees [60] or symbolic rules [147]. Secondly, *outcome explanation* methods provide a *locally explainable* model that is able to explain the prediction of the black box in understandable terms for humans for a specific instance or record. In other words, rather than explaining the entire model, the focus is on explaining individual predictions. An example is the Local Interpretable Model-agnostic Explanations (LIME) approach [98], which uses local approximations based on generated random samples near the sample for which the prediction needs to be explained. In the context of image and text analysis, a common approach to explaining why neural networks predict a certain outcome is based on designing saliency masks that visually highlight the determining aspects of the analyzed record [111]. Finally, *model inspection* methods aim to provide a textual or visual representation for understanding some specific property of a black box model or of its predictions. Common properties of interest include sensitivity to attribute changes, and identification of components of the black box (e.g., neurons in a neural network) responsible for specific decisions. For instance, Samek et al. [103] apply two techniques, namely sensitivity analysis and layer-wise relevance propagation, for explaining the individual predictions of a deep learning model in terms of input variables. In a similar fashion, Evermann et al. [34] made an investigation into understanding the behavior of deep neural networks, using network hallucinations and steady-state analysis. Namely, they feed the network output (prediction) immediately back as new input. The ability of a network to reproduce, on its own, realistic and convincing process traces proves that it has correctly learned the relevant features of the event log and thereby affirms the usefulness of the model for process behavior prediction.

Rather than providing post-hoc explanations for a black-box classifier, the second group of proposals focus on directly designing a *transparent* classifier that solves the same classification problem. This principle is known as "Transparent Box Design" problem [44] and is often solved by means of interpretable predictors based on extracted rule sets [139, 145].

In contrast, in this thesis, we leverage the idea of *decompositional explanation* [67, 68] to enhance understanding of predictors. Specifically, we aim to explain the prediction by decomposing it into its elementary components. Namely, we propose two approaches that use multi-task learning and business process models to decompose the predictions. They are described in Chapter 4 and Chapter 5 respectively.

3. STATE OF THE ART IN THE PREDICTION OF TEMPORAL PROCESS MEASURES

Predictive business process monitoring techniques are concerned with predicting the evolution of running cases of a business process based on models extracted from historical event logs. A wide range of such techniques have been proposed for a variety of prediction tasks: predicting the next activity [7], predicting the future path (continuation) of a running case [91], predicting the remaining cycle time [99], predicting deadline violations [77] and predicting the fulfillment of a property upon completion [71]. An earlier survey of this field by Márquez-Chamorro et al. [74] identified 39 distinct proposals, out of which 15 focus on the prediction of the remaining time and delays in the execution of running business process cases. This chapter aims to answer the research question **RQ0** ("What methods exist for predictive monitoring of remaining time of business processes?") by presenting a detailed and up-to-date systematic literature review and taxonomy of predictive business process monitoring methods, with a focus on the prediction of temporal measures.

This chapter is structured as follows. Section 3.1 outlines the search procedure and selection of relevant studies. Section 3.2 analyzes the selected studies and categorizes them according to several dimensions. Section 3.3 provides a generalized taxonomy of methods for predictive monitoring of temporal measures. Finally, Section 3.4 provides a summary of the chapter.

3.1. Search Methodology

In order to retrieve and select studies for our literature survey, we conducted a *Systematic Literature Review* (SLR) according to the approach described in [57]. We started by specifying the research questions. Next, guided by these goals, we developed relevant search strings for querying a database of academic papers. We applied inclusion and exclusion criteria to the retrieved studies in order to filter out irrelevant ones, and last, we divided all relevant studies into primary and subsumed ones based on their contribution.

3.1.1. Research Questions

The purpose of this survey is to define a taxonomy of methods for predictive monitoring of remaining time of business processes. The decision to focus on remaining time is to have a well-delimited and manageable scope, given the richness of the literature in the broader field of predictive process monitoring, and the fact that other predictive process monitoring tasks might rely on different techniques and evaluation measures.

In line with the selected scope, we break down the research question **RQ0** into the following sub-questions:

RQ0.1 How to classify methods for predictive monitoring of remaining time?

RQ0.2 What type of data has been used to evaluate these methods, and from which application domains?

RQ0.3 What tools are available to support these methods?

RQ0 is the core research question, which aims at identifying existing methods to perform predictive monitoring of remaining time. With RQ0.1, we aim to identify a set of classification criteria on the basis of input data required (e.g. input log) and the underlying predictive algorithms. RQ0.2 explores what tool support the different methods have, while RQ0.3 investigates how the methods have been evaluated and in which application domains.

3.1.2. Study Retrieval

Existing literature in predictive business process monitoring was searched for using Google Scholar, a well-known electronic literature database, as it covers all relevant databases such as ACM Digital Library and IEEE Xplore, and also allows searching within the full text of a paper.

Our search methodology is based on the one proposed in [74], with few variations. Firstly, we collected publications using more specific search phrases, namely "predictive process monitoring", "predictive business process monitoring", "predict (the) remaining time", "remaining time prediction" and "predict (the) remaining * time". The latter is included since some authors refer to the prediction of the remaining *processing* time, while others may call it remaining *execution* time and so on. We retrieved all studies that contained at least one of the above phrases in the title or in the full text of the paper. The search was conducted in March 2018 to include all papers published between 2005 and 2017.

The initial search returned **670** unique results which is about 3 times more than the ones found in [74], owing to the differences in search methodologies (Table 2). Figure 12 shows how the studies are distributed over time. We can see that the interest in the topic of predictive process monitoring grows over time with a sharp increase over the past few years.

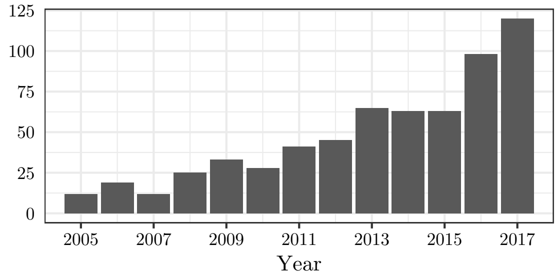


Figure 12: Number of published studies over time.

In order to consider only relevant studies, we designed a range of exclusion criteria to assess the relevance of the studies. First, we excluded those papers

Table 2: Comparison of our search methodology with [74]

	Method in [74]	Our method
Keywords	1. “predictive monitoring” AND “business process” 2. “business process” AND “prediction”	1. “predictive process” monitoring 2. “predictive business process monitoring” 3. “predict (the) remaining time” 4. “remaining time prediction” 5. “predict (the) remaining * time”
Search scope	Title, abstract, keywords	Title, full text
Min number of citations	5 (except 2016 papers)	5 (except 2017 papers)
Years covered	2010-2016	2005-2017
Papers found after filtering	41	53
Snowballing applied	No	Yes, one-hop

not related to the process mining field, written in languages other than English or papers with inaccessible full text. Additionally, to contain the reviewing effort, we have only included papers that have been cited at least five times. An exception has been made for papers published in 2017 – as many of them have not had a chance to accumulate the necessary number of citations, we required only one citation for them. Thus, after the first round of filtering, a total of **53** publications were considered for further evaluation.

Since, different authors might use different terms, not captured by our search phrases, to refer to the prediction target in question, we decided to also include all studies that cite the previously discovered 53 publications (”snowballing”). Applying the same exclusion criteria, we ended up with **83** more studies. Due to the close-knit nature of the process mining community, there is a considerable overlap between these 83 studies and the 53 studies that had been retrieved during the first search stage. Accordingly, a total of **110** publications were finally considered on the scope of our review. All papers retrieved at each search step can be found at `https://goo.gl/kg6xZ1`.

The remaining 110 papers were further assessed with respect to exclusion criteria:

EX1 The study does not actually propose a predictive process monitoring *method*. With this criterion, we excluded position papers, as well as studies that, after a more thorough examination, turned out to be focusing on some research question other than predictive process monitoring. Furthermore, here we excluded survey papers and implementation papers that employ existing predictive methods rather than propose new ones.

EX2 The study does not concern remaining time predictions. Common examples of other prediction targets that are considered irrelevant to this study are failure and error prediction, as well as next activity prediction. At the same time, prediction targets such as case completion time prediction and case duration prediction are inherently related to remaining time and therefore were also considered in our work. Additionally, this criterion does not eliminate studies that address the problem of predicting deadline violations in a boolean manner by setting a threshold on the predicted remaining time rather than by a direct classification.

EX3 The study does not take an *event log* as input. In this way, we exclude methods that do not utilize at least the following essential parts of an event log: the case identifier, the timestamp and the event classes. For instance, we excluded methods that take as input numerical time series without considering the heterogeneity in the control flow. In particular, this is the case in manufacturing processes which are of linear nature (a process chain). The reason for excluding such studies is that the challenges when predicting for a set of cases of heterogeneous length are different from those when predicting for linear processes. While methods designed for heterogeneous processes are usually applicable to those of linear nature, it is not so vice versa. Moreover, the linear nature of a process makes it possible to apply other, more standard methods that may achieve better performance.

Furthermore, we excluded our earlier studies on the application of artificial neural networks [119] and flow analysis techniques [137] for the prediction of temporal measures. These studies will be covered in subsequent chapters of the thesis.

The application of the exclusion criteria resulted in **23** relevant studies which are described in detail in the following section.

3.2. Analysis and Classification of Methods

Driven by the research questions defined in Section 3.1.1, we identified the following dimensions to categorize and describe the relevant studies.

- Type of input data – RQ0.1
- Awareness of the underlying business process – RQ0.1
- Family of algorithms – RQ0.1
- Type of evaluation data (real-life or artificial log) and application domain (e.g., insurance, banking, healthcare) – RQ0.2
- Type of implementation (standalone or plug-in, and tool accessibility) – RQ0.3

This information, summarized in Table 3, allows us to answer the first research question. In the remainder of this section, we proceed with surveying each main study method along the above classification dimensions.

3.2.1. Input Data

As stipulated by *EX3* criterion, all surveyed proposals take as input an event log. Such a log contains at least a case identifier, an activity and a timestamp. In

Table 3: Overview of the 23 relevant studies resulting from the search (ordered by year and author).

Study	**Year**	**Input data**	**Process-aware?**	**Algorithm**	**Domain**	**Implementation**
van Dongen et al. [132]	2008	event log data	No	regression	public administration	ProM 5
van der Aalst et al. [130]	2011	event log	Yes	transition system	public administration	ProM 5
Folino et al. [36]	2012	event log data contextual information	No	clustering	logistics	ProM
Pika et al. [88]	2012	event log data	No	stat analysis	financial	ProM
van der Spoel et al. [131]	2012	event log data	Yes	process graph regression	healthcare	n/a
Bevacqua et al. [11]	2013	event log data	No	clustering regression	logistics	ProM
Bolt and Sepúlveda [12]	2013	event log	Yes	stat analysis	telecom simulated	n/a
Folino et al. [37]	2013	event log data contextual information	No	clustering	customer service	n/a
Pika et al. [89]	2013	event log data	No	stat analysis	insurance	ProM
Rogge-Solti and Weske [99]	2013	event log data	Yes	stoch Petri net	logistics simulated	ProM
Ceci et al. [21]	2014	event log data	Yes	sequence trees regression	customer service workflow management	ProM
Folino et al. [38]	2014	event log data contextual information	No	clustering regression	logistics software development	n/a
de Leoni et al. [28]	2014	event log data contextual information	No	regression	no validation	ProM
Polato et al. [90]	2014	event log data	Yes	transition system regression classification	public administration	ProM
Senderovich et al. [107]	2014	event log	Yes	queueing theory transition system	financial	n/a
Metzger et al. [77]	2015	event log data process model	Yes	neural network constraint satisfaction QoS aggregation	logistics	n/a
Rogge-Solti and Weske [100]	2015	event log data	Yes	stoch Petri net	financial logistics	ProM
Senderovich et al. [108]	2015	event log	Yes	queueing theory transition system	financial telecom	n/a
Cesario et al. [22]	2016	event log data	Yes	clustering regression	logistics	n/a
de Leoni et al. [29]	2016	event log data contextual information	No	regression	no validation	ProM
Polato et al. [91]	2018	event log data	Yes	transition system regression classification	public administration customer service financial	ProM
Senderovich et al. [106]	2017	event log data contextual information	No	regression	healthcare manufacturing	standalone
Navarin et al. [84]	2017	event log	No	neural network	customer service financial	standalone

addition, many techniques leverage case and event attributes to make more accurate predictions. For example, in the pioneering predictive monitoring approach described in [132], the authors predict the remaining processing time of a trace using activity durations, their frequencies and various case attributes, such as the case priority. Many other approaches, e.g. [29], [91], [106] make use not only of case attributes but also of event attributes, while applying one or more kinds of sequence encoding. Furthermore, some approaches, e.g. [36] and [106], exploit contextual information, such as workload indicators, to take into account intercase dependencies due to resource contention and data sharing. Finally, the approach by Metzger et al. [77] also leverages a process model in order to "replay" ongoing process cases on it. Such works treat remaining time as a cumulative indicator composed of cycle times of elementary process components.

3.2.2. Process Awareness

Existing techniques can be categorized according to their process-awareness, i.e. whether or not the methodology exploits an explicit representation of a process model to make predictions. As can be seen from Table 3, nearly half of the techniques are process-aware. Most of them construct a transition system from an event log using set, bag (multiset) or sequence abstractions of observed events. *State transition systems* are based on the idea that the process is composed of a set of consistent states and the movement between them [128]. Thus, a process behavior can be predicted if we know its current and future states.

Bolt and Sepúlveda [12] exploit query catalogs to store the information about the process behavior. Such catalogs are groups of partial traces (annotated with additional information about each partial trace) that have occurred in an event log, and are then used to estimate the remaining time of new executions of the process.

Also *queuing models* can be used for prediction because if a process follows a queuing context and queuing measures (e.g. arrival rate, departure rate) can be accurately estimated and fit the process actual execution, the movement of a queuing item can be reliably predicted. Queueing theory and regression-based techniques are combined for delay prediction in [107, 108].

Furthermore, some process-aware approaches rely on stochastic Petri nets [99, 100] and process models [77].

3.2.3. Family of Algorithms

Non-process aware approaches typically rely on machine learning algorithms to make predictions. In particular, these algorithms take as input labeled training data in the form of feature vectors and the corresponding labels. In case of remaining time predictions, these labels are continuous. As such, various *regression* methods can be utilized, such as regression trees [28, 29] or ensemble of trees, i.e. random forest [131] and XGBoost [106].

An emerging family of algorithms for predictive monitoring are artificial neural networks. They consist of one layer of input units, one layer of output units, and multiple layers in-between which are referred to as hidden units. While traditional machine learning methods heavily depend on the choice of features on which they are applied, neural networks are capable of translating the data into a compact intermediate representation to aid a hand-crafted feature engineering process [8]. A feedforward network has been applied in [77] to predict deadline violations. More sophisticated architectures based on recurrent neural networks were explored in [84].

Other approaches apply trace clustering to group similar traces to fit a predictive model for each cluster. Then for any new running process case, predictions are made by using the predictor of the cluster it is estimated to belong to. Such approach is employed e.g., in [36] and [38].

Another range of proposals utilizes statistical methods without training an explicit machine learning model. For example, Pika et al. [88, 89] make predictions about time-related process risks by identifying and leveraging process risk indicators (e.g., abnormal activity execution time or multiple activity repetition) by applying statistical methods to event logs. The indicators are then combined by means of a prediction function, which allows for highlighting the possibility of transgressing deadlines. Conversely, Bolt and Sepúlveda [12] calculate remaining time based on the average time in the catalog which the partial trace belongs to, without taking into account distributions and confidence intervals.

Rogge-Solti and Weske [99] mine a stochastic Petri net from the event log to predict the remaining time of a case using arbitrary firing delays. The remaining time is evaluated based on the fact that there is an initial time distribution for a case to be executed. As inputs, the method receives the Petri net, the ongoing trace of the process instance up to current time, the current time and the number of simulation iterations. The algorithm returns the average of simulated completion times of each iteration. This approach is extended in [100] to exploit the elapsed time since the last observed event to make more accurate predictions.

Finally, a hybrid approach is proposed by Polato et al. [91] who build a transition system from an event log and enrich it with classification and regression models. Naive Bayes classifiers are used to estimate the probability of transition from one state to the other, while support vector regressors are used to predict the remaining time from the next state.

3.2.4. Evaluation Data and Domains

As reported in Table 3, most of the surveyed methods have been validated on at least one real-life event log. Some studies were additionally validated on simulated (synthetic) logs.

Importantly, many of the real-life logs are publicly available from the *4TU Center for Research Data*.[1] Among the methods that employ real-life logs, we

[1] `https://data.4tu.nl/repository/collection:event_logs_real`

observed a growing trend to use publicly-available logs, as opposed to private logs which hinder the reproducibility of the results due to not being accessible.

Concerning the application domains of the real-life logs, we noticed that most of them pertain to logistics, banking (7 studies each), public administration (5 studies) and customer service (3 studies) processes.

3.2.5. Implementation

Providing publicly available implementation and experiment data greatly facilitates reproducibility of research experiments, as well as to enable future researchers to build on past work. To this end, we found that nearly a half of the methods provide an implementation as a plug-in for the ProM framework.[1] The reason behind the popularity of ProM can be explained by its open-source and portable framework, which allows researchers to easily develop and test new algorithms. Also, ProM provides an Operational Support (OS) environment [141] that allows it to interact with external workflow management systems at runtime. In this way, process mining can also be performed in an online setting. Another two methods have a standalone implementation in Python [2]. Finally, 8 methods do not provide a publicly available implementation.

3.2.6. Predictive Monitoring Workflow

As indicated in Table 3, most predictive monitoring methods make use of machine learning algorithms based on regression, classification or neural networks. Such methods typically proceed in two phases: offline, to train a prediction model based on historical cases, and online, to make predictions on running process cases (Figure 13) [120].

In the offline phase, given an event log, case prefixes are extracted and filtered, e.g. to retain only prefixes up to a certain length, to create a prefix log (cf. Section 2). Next, ”similar” prefixes are grouped into homogeneous *buckets*, e.g. based on process states or similarities among prefixes and prefixes from each bucket are *encoded* into feature vectors. Then feature vectors from each bucket are used to fit a machine learning model.

In the online phase, the actual predictions for running cases are made, by reusing the buckets, encoders and predictive models built in the offline phase. Specifically, given a running case and a set of buckets of historical prefixes, the correct bucket is first determined. Next, this information is used to encode the features of the running case. In the last step, a prediction is extracted from the encoded case using the pre-trained model corresponding to the determined bucket.

Similar observations can be made for non-machine learning-based methods. For example, in [130] first, a transition system is derived and annotated and then

[1] `http://promtools.org`

[2] `https://www.python.org`

the actual predictions are calculated for running cases. In principle, this transition system akin to a predictive model can be mined in advance and used at runtime.

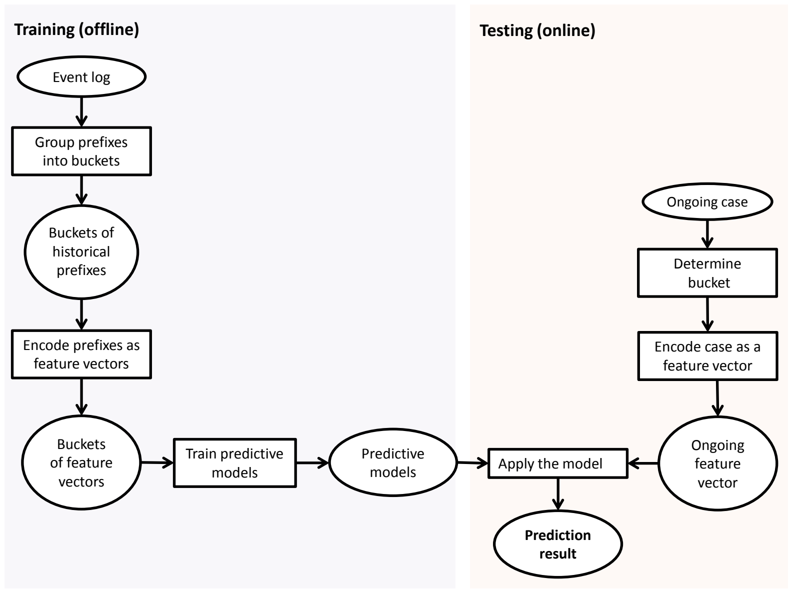


Figure 13: Predictive process monitoring workflow.

3.2.7. Primary and Subsumed (Related) Studies

Among the papers that successfully passed both the inclusion and exclusion criteria, we determined *primary* studies that constitute an original contribution, and *subsumed* studies that are similar to one of the primary studies and do not provide a substantial contribution with respect to it.

Specifically, a study is considered subsumed if:

- there exists a more recent and/or more extensive version of the study from the same authors (e.g. a conference paper is subsumed by an extended journal version), or
- it does not propose a substantial improvement/modification over a method that is documented in an earlier paper by other authors, or
- the main contribution of the paper is a case study or a tool implementation, rather than the predictive process monitoring method itself, and the method is described and/or evaluated more extensively in a more recent study by other authors.

This procedure resulted in **9** primary and **14** subsumed studies, as reported in Table 4.

Table 4: Primary and subsumed studies

Primary study	Subsumed studies
van der Aalst et al. [130]	van Dongen et al. [132], Bolt and Sepúlveda [12]
Folino et al. [36]	Folino et al. [37, 38]
Rogge-Solti and Weske [100]	Rogge-Solti and Weske [99]
Senderovich et al. [108]	Senderovich et al. [107]
Cesario et al. [22]	Bevacqua et al. [11]
de Leoni et al. [29]	Pika et al. [88, 89], de Leoni et al. [28]
Polato et al. [91]	van der Spoel et al. [131], Polato et al. [90], Ceci et al. [21]
Senderovich et al. [106]	van der Spoel et al. [131]
Navarin et al. [84]	Metzger et al. [77]

3.3. Taxonomy of Methods

The methods that resulted from the search procedure are very heterogeneous in terms of the inputs required, the outputs produced and inner details. In this section, we aim to *abstract* the details that are not inherent to the methods and focus on their core differences.

3.3.1. Prefix Bucketing

While some machine learning-based predictive process monitoring approaches train a single predictor on the whole event log, others employ a multiple predictor approach by dividing the prefix traces in the historical log into several *buckets* and fitting a separate predictor for each bucket. To this end, Teinemaa et al. [120] surveyed several bucketing methods out of which three have been utilized in the primary methods:

- No bucketing. All prefix traces are considered to be in the same bucket. As such, a single predictor is fit for all prefixes in the prefix log. This approach has been used in [29], [106] and [84].
- Prefix length bucketing. Each bucket contains the prefixes of a specific length. For example, the n-th bucket contains prefixes where at least n events have been performed. One classifier is built for each possible prefix length. This approach has been used in [66].
- Cluster bucketing. Here, each bucket represents a cluster that results from applying a clustering algorithm on the encoded prefixes. One classifier is trained for each resulting cluster, considering only the historical prefixes that fall into that particular cluster. At runtime, the cluster of the running case is determined based on its similarity to each of the existing clusters and the corresponding classifier is applied. This approach has been used in [36] and [22]
- State bucketing. It is used in process-aware approaches where some kind of process representation, e.g. in the form of a transition system, is derived

and a predictor is trained for each state, or decision point. At runtime, the current state of the running case is determined, and the respective predictor is used to make a prediction for the running case. This approach has been used in [91].

In addition to these “canonical” bucketing methods, various combinations thereof have been explored in related work. For example, in our earlier work [136], we proposed an approach where prefixes are first grouped into buckets according to their prefix length and then clustering is applied for each bucket. This approach has been shown to marginally improve the classification accuracy.

3.3.2. Prefix Encoding

In order to train a machine learning model, all prefixes in a given bucket need to be represented, or *encoded* as fixed-size feature vectors. Case attributes are static, and their number is fixed for a given process. Conversely, with each executed event, more information about the case becomes available. As such, the number of event attributes will increase over time. To address this issue, various sequence encoding techniques were proposed in related work summarized in [66] and refined in [120]. In the primary studies, the following encoding techniques can be found:

- Last state encoding. In this encoding method, only event attributes of the last m events are considered. Therefore, the size of the feature vector is proportional to the number of event attributes and is fixed throughout the execution of a case. $m = 1$ is the most common choice used, e.g. in [91], although in principle higher m values can also be used.
- Aggregation encoding. In contrast to the last state encoding, all events since the beginning of the case are considered. However, to keep the feature vector size constant, various aggregation functions are applied to the values taken by a specific attribute throughout the case lifetime. For numeric attributes, common aggregation functions are minimum, average, maximum and sum of observed values, while for categorical ones *count* is generally used, e.g. the number of times a specific activity has been executed, or the number of activities a specific resource has performed [29].
- Index-based encoding. Here for each position n in a prefix, we concatenate the event e_n occurring in that position and the value of each event attribute in that position $v_n^1, \ldots, v_n^k$, where k is the total number of attributes of an event. This type of encoding is lossless, i.e. it is possible to recover the original prefix based on its feature vector. On the other hand, with longer prefixes, it significantly increases the dimensionality of the feature vectors and hinders the model training process. This approach has been used in [131].
- Tensor encoding. A tensor is a generalization of vectors and matrices to potentially higher dimensions [114]. Unlike conventional machine learning algorithms, tensor-based models do not require input to be encoded in a

two-dimensional $n \times m$ form, where n is the number of training instances and m is the number of features. Conversely, they can take as input a three-dimensional tensor of shape $n \times t \times p$, where t is the number of events and p is the number of event attributes, or features derived from each event. In other words, each prefix is represented as a matrix where rows correspond to events and columns to features for a given event. The data for each event is encoded ”as-is”. Case attributes are encoded as event attributes having the same value throughout the prefix. Hence, the encoding for LSTM is similar to the index-based encoding except for two differences: (i) case attributes are duplicated for each event, (ii) the feature vector is reshaped into a matrix.

To aid the explanation of the encoding types, Tables 5 – 7 provide examples of feature vectors derived from the event log in Table 1. For convenience, we replaced absolute timestamps with relative ones (in seconds), starting from the beginning of a case. Furthermore, for simplicity, we showed only one aggregation function, sum, for numerical attributes. Note that for the index-based encoding, each trace σ in the event log produces only one training sample per bucket, while the other encodings produce as many samples as many prefixes can be derived from the original trace, i.e. up to $|\sigma|$.

Table 5: Feature vectors created from the log in Table 1 using last state encoding.

Channel	Age	Activity_last	Time_last	Resource_last	Cost_last
Email	37	A	0	John	15
Email	37	B	80	Mark	25
Email	37	D	180	Mary	10
Email	37	F	305	Kate	20
Email	37	G	350	John	20
Email	37	H	360	Kate	15
Email	52	A	0	John	25
Email	52	D	300	Mary	25
Email	52	B	57900	Mark	10
Email	52	F	58010	Kate	15

Table 6: Feature vectors created from the log in Table 1 using aggregated encoding.

Channel	Age	Act_A	Act_B	Act_D	Act_F	Act_G	Act_H	Res_John	Res_Mark	Res_Mary	Res_Kate	sum_Time	sum_Cost	…
Email	37	1	0	0	0	0	0	1	0	0	0	0	15	
Email	37	1	1	0	0	0	0	1	1	0	0	80	40	
Email	37	1	1	1	0	0	0	1	1	1	0	180	50	
Email	37	1	1	1	1	0	0	1	1	1	1	305	70	
Email	37	1	1	1	1	1	0	2	1	1	1	350	90	
Email	37	1	1	1	1	1	1	2	1	1	2	360	105	
Email	52	1	0	0	0	0	0	1	0	0	0	0	25	
Email	52	1	0	1	0	0	0	1	0	1	0	300	50	
Email	52	1	1	1	0	0	0	1	1	1	0	57900	60	
Email	52	1	1	1	1	0	0	1	1	1	1	58010	75	

These three “canonical” encodings can serve as a basis for various modifications thereof. For example, de Leoni et al. [29] proposed the possibility of combining last state and aggregation encodings.

While the encoding techniques stipulate how to incorporate *event* attributes in

Table 7: Feature vectors created from the log in Table 1 using index-based encoding, buckets of length $n = 3$.

Channel	Age	Activ_1	Time_1	Res_1	Cost_1	Activ_2	Time_2	Res_2	Cost_2	Activ_3	Time_3	Res_3	Cost_3
Email	37	A	0	John	15	B	80	Mark	25	D	180	Mary	10
Email	52	A	0	John	25	D	300	Mary	25	B	57900	Mark	10

a feature vector, the inclusion of case attributes is rather straightforward, as their number is fixed throughout the case lifetime.

While last state and aggregation encodings can be combined with any of the bucketing methods described in Section 3.3.1, index-based encoding is commonly used with prefix-length bucketing, as the feature vector size depends on the trace length [66]. Nevertheless, two options have been proposed in related work to combine index-based encoding with other bucketing types:

- Fix the maximum prefix length and, for shorter prefixes, impute missing event attribute values with zeros or their historical averages. This approach is often referred to as *padding* in machine learning [109] and has been used in the context of predictive process monitoring in [121] and [84].
- Use the sliding window method to encode only recent (up-to window size w) history of the prefix. This approach has been proposed in [106].

3.3.3. Discussion

Summarizing the above observations, we devised a taxonomy of predictive monitoring techniques (Figure 14). The taxonomy is framed upon a general classification of machine learning approaches into generative and discriminative ones (cf. Section 2.4).

The former correspond to process-aware predictive monitoring techniques, meaning that there is an assumption that an observed sequence is generated by some process that needs to be uncovered via probabilistic reasoning. The process can be represented via a state transition system [130], a Petri net [100] or a queueing model [108].

Conversely, discriminative approaches are non-process-aware techniques that learn a direct mapping from inputs to the output via regression, without providing a model of how input sequences are actually generated. Having analyzed the discriminative studies, we have observed that they tend to mix different encoding types [29] or different bucketing types [36], while some combinations thereof have not been explored.

Finally, a range of newer studies propose hybrid methods that combine generative and discriminative approaches [91]. Such methods can generally be approximated with state bucketing that, for every process state, fits a model to predict the remaining time starting from that state.

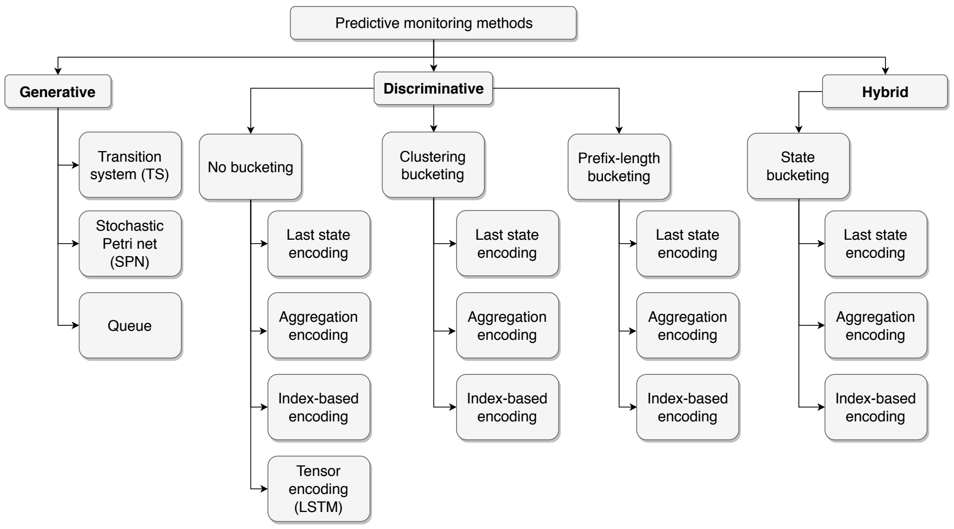


Figure 14: Taxonomy of methods for predictive monitoring of temporal measures.

3.4. Summary

Predictive process monitoring is an actively growing research area, with the number of publications increasing year-by-year. In this chapter, we provided a comprehensive analysis of existing process monitoring techniques to predict the remaining execution time.

Following the literature survey, we identified two primary gaps in the related work. Firstly, the potential of deep learning models, specifically recurrent neural networks, to predict the remaining execution time in the context of business processes has not been fully explored. As such, in this thesis, we aim to design and evaluate various architectures for the remaining time prediction. In the first approach, we predict the remaining time by iteratively predicting the next activity and its timestamp, until the case is predicted to finish. Additionally, we explore LSTM models that are trained to predict the remaining time directly, rather than by predicting the timestamp of each future activity (Chapter 4).

Secondly, we note that most of the proposed methods adopt a black-box approach, insofar as they predict a single scalar value, without seeking to explain this prediction in terms of more elementary components. Yet, quantitative performance measures such as cost or time are aggregations of corresponding performance measures of the activities composing the process. For example, the cycle time of a process instance with sequentially performed activities consists of the sum of the cycle time of the activities performed in that process instance. As such, in this thesis, we propose a process model driven approach to predict performance measures of running process instances. The key idea is to first predict the performance measure at the level of activities, and then to aggregate these predictions at the level of a process instance by means of flow analysis techniques (Chapter 5).

Finally, even though, most of the surveyed techniques have publicly available implementations, they are designed as prototypes to train and evaluate a specific predictive monitoring method, without considering how to apply the proposed methods at runtime. As such, we consolidate the research contributions of this thesis into an integrated platform for predictive business process monitoring that will allow its users not only to train predictive models but also to receive a stream of events from an enterprise information system, make predictions using previously trained models, and finally to present them in a real-time operational dashboard, with multiple visualization options.

4. DEEP LEARNING FOR PREDICTION OF TEMPORAL PROCESS MEASURES

4.1. Introduction

Recurrent neural networks with Long Short-Term Memory (LSTM) architectures [50] have been shown to deliver consistently high accuracy in several sequence modeling application domains, e.g. natural language processing [80] and speech recognition [41].

Business processes, being sequences of activities with associated attributes, can be naturally modeled with recurrent neural networks. Recently, Mehdiyev et al. [75, 76] applied LSTMs to predictive process monitoring, specifically to predict the next activity in a case, using a multi-stage deep learning approach which formulates the next business process event prediction problem as a classification problem and applies deep feed-forward multilayer neural networks. Similarly, Navarin et al. [84] applied LSTM to predict the completion time of running cases using control flow and data attributes extracted from the prefix.

Tax et al. [118] compared the generalizing capabilities of process discovery techniques and black-box sequence models and discovered that LSTM networks more accurately describe previously unseen traces than existing process discovery methods. However, LSTM sequence models are *black-box models*, focusing on accurately describing the sequences, rather than the explainability of the results [118]. In this chapter, we propose a technique to make prediction results from deep learning models, specifically LSTM, explainable to business process stakeholders, thus aiming to answer **RQ1**.

To this end, we use an instance of multi-task learning [25], where several predictive process monitoring tasks are integrated into a single system which is trained *jointly*. Namely, we design and evaluate several LSTM architectures for predicting: (i) the next activity in a running case and its timestamp; (ii) the continuation of a case up to completion; and (iii) the remaining cycle time. The outlined LSTM architectures are empirically compared against tailor-made approaches with respect to their accuracy at different prediction points, using real-life event logs from different domains.

The chapter is structured as follows. Section 4.2 introduces foundational deep learning concepts and notation. Section 4.3 describes a technique to predict the next activity in a case and its timestamp, and compares it against tailor-made baselines. Section 4.4 extends the previous technique to predict the continuation of a running case. Section 4.5 shows how this latter method can be extended to predict the remaining time of a case, while the detailed evaluation for the approach presented here is reported in a dedicated Chapter 6. Finally, Section 4.6 summarizes the chapter.

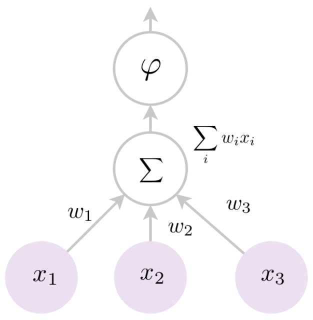


Figure 15: An example of a neural network unit

4.2. Deep learning

4.2.1. Neural Networks

A neural network consists of a set of *units* or nodes. A unit accepts n inputs x_i, computes their weighted sum $\sum_{i=1}^{n} w_i x_i$ and passes it to the *activation function* ϕ (Figure 15). Common activation functions are sigmoid: $sigmoid(x) = \frac{1}{1+exp(-x)}$, hyperbolic tangent: $\phi(x) = \frac{exp(x)-exp(-x)}{exp(x)+exp(-x)}$ and the rectified linear unit (ReLU): $\xi(x) = \max(0,x)$.

Units are connected into a network using edges between them. Specifically, a network consists of *layers* of units, where the first is the *input layer*, last is the *output layer* and there are multiple layers in-between which are referred to as *hidden layers*. The outputs of the input units form the inputs of the units of the first hidden layer, and the outputs of the units of each hidden layer form the input for each subsequent hidden layer. The outputs of the last hidden layer form the input for the output layer.

A network is activated with an input data, and activations are propagated until the output $\hat{y}$ is produced. The network training phase includes iterative updating of the weights w_i that would minimize a loss function $L(\hat{y}, y)$. The weights are typically learned using backpropagation algorithm [101]. Backpropagation applies forward and backward passes, where during the forward pass the weights w_i and the loss function L are calculated and during the backward pass the partial derivatives with respect to each parameter are calculated using the chain rule. The weights are then adjusted through gradient-based optimization.

4.2.2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a special type of neural networks where the connections between neurons form a directed cycle. This allows modeling temporal dynamics.

RNNs can be unfolded, as shown in Figure 16. Each step in the unfolding is referred to as a time step, where x_t is the input at time step t. RNNs can take an

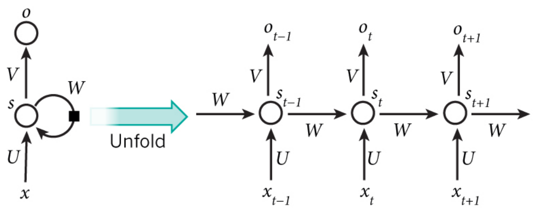


Figure 16: A simple recurrent neural network (taken from [65]).

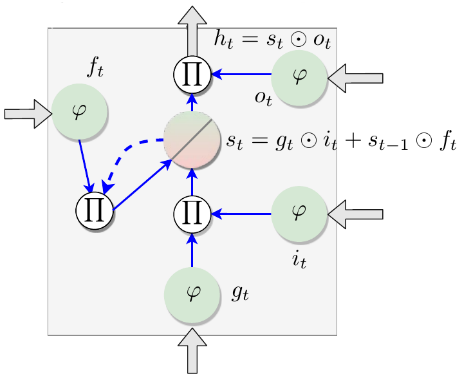


Figure 17: LSTM unit architecture [69].

arbitrary length sequence as input, by providing the RNN a feature representation of one element of the sequence at each time step. s_t is the hidden state at time step t and contains information extracted from all time steps up to t. The hidden state s is updated with information of the new input x_t after each time step: $s_t = f(Ux_t + Ws_{t-1})$, where f is the activation function and U and W are vectors of weights over the new inputs and the hidden state respectively. o_t is the output at step t.

4.2.3. Long Short-Term Memory for Sequence Modeling

The vanishing gradient problem causes RNN to perform poorly on longer temporal dependencies. As a result, the input at the beginning of the sequence may not affect the output at the end.

A Long Short-Term Memory model (LSTM) [50] is a special Recurrent Neural Network architecture that has powerful modeling capabilities for long-term dependencies. In an LSTM unit, the input x_t at time t is fed to the input node g_t. The input node also accepts the information from the hidden layer at the previous step h_{t-1}. Then an activation function is applied to the sum of weighted output (Figure 17).

The main distinction between a regular RNN and a LSTM is that the latter has a more complex memory cell C_t replacing s_t. Where the value of state s_t in a RNN is the result of a function over the weighted average over s_{t-1} and x_t, the LSTM state C_t is accessed, written, and cleared through controlling gates. Namely, an LSTM unit has an input gate i_t, forget gate f_t and output gate o_t. Information on a new input will be accumulated to the memory cell if i_t is activated. Additionally, the past memory cell status C_{t-1} can be "forgotten" if f_t is activated. The information of C_t will be propagated to the output h_t based on the activation of output gate o_t.

Combined, the LSTM model can be described with the following formulas: [69]

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

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

Summing up, an LSTM unit manages the information flow in the forward pass by regulating with the gates how much information to let in and out and how much to forget. In terms of the backward pass it regulates how much error to propagate back [69].

4.3. Next Activity and Timestamp Prediction

In this section we present and evaluate multiple architectures for next event and timestamp prediction using LSTMs.

4.3.1. Approach

We start by predicting the next activity in a case and its timestamp, by learning an activity prediction function f_a^1 and a time prediction function f_t^1. We aim at functions f_a^1 and f_t^1 such that $f_a^1(hd^k(\sigma)) = hd^1(tl^k(\pi_{\mathscr{A}}(\sigma)))$ and $f_t^1(hd^k(\sigma)) = hd^1(tl^k(\pi_{\mathscr{T}}(\sigma)))$ for any prefix length k. We transform each event $e \in hd^k(\sigma)$ into a feature vector and use these vectors as LSTM inputs $x_1, \dots, x_k$. We build the feature vector as follows. We start with $|A|$ features that represent the type of activity of event e in a so-called *one-hot encoding*. We take an arbitrary but consistent ordering over the set of activities A, and use $index \in A \to \{1, \dots, |A|\}$ to indicate the position of an activity in it. The one-hot encoding assigns the value 1 to feature number $index(\pi_{\mathscr{A}}(e))$ and a value of 0 to the other features. We add three time-based features to the one-hot encoding feature vector. The first time-based feature of event $e = \sigma(i)$ is the time between the previous event in the trace and the current event, i.e., $fv_{t1}(e) = \begin{cases} 0 & \text{if } i = 1, \\ \pi_{\mathscr{T}}(e) - \pi_{\mathscr{T}}(\sigma(i-1)) & \text{otherwise.} \end{cases}$. This feature allows the LSTM to learn dependencies between the time differences at different points (indexes) in the process. Many activities can only be performed during office hours; therefore we add a time feature fv_{t2} that contains the time within the day (since midnight) and fv_{t3} that contains the time within the week

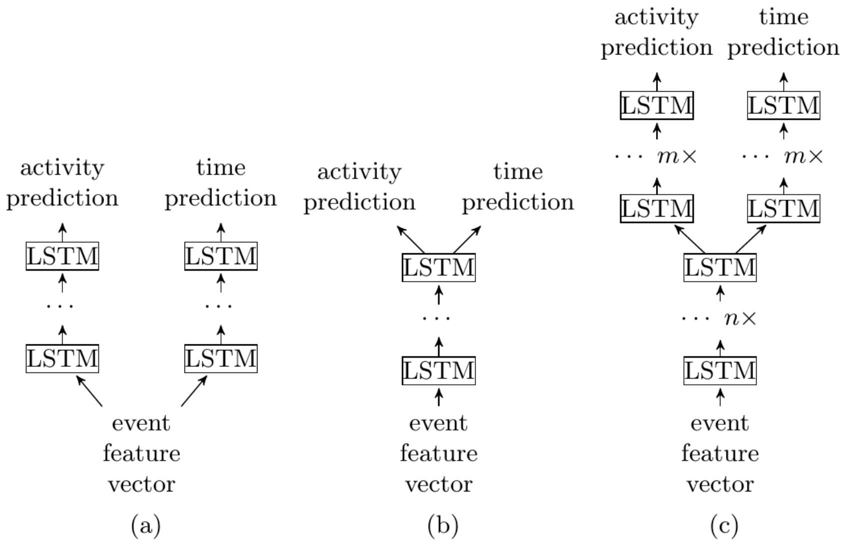


Figure 18: Neural Network architectures with single-task layers *(a)*, with shared multi-tasks layer *(b)*, and with $n+m$ layers of which n are shared *(c)*.

(since midnight on Sunday). fv_{t2} and fv_{t3} are added to learn the LSTM such that if the last event observed occurred at the end of the working day or at the end of the working week, the time until the next event is expected to be longer.

At learning time, we set the target output o_a^k of time step k to the one-hot encoding of the activity of the event one time step later. However, it can be the case that the case ends at time k, in which case there is no new event to predict. Therefore, we add an extra element to the output one-hot-encoding vector, which has value 1 when the case ends after k. We set a second target output o_t^k equal to the fv_{t1} feature of the next time step, i.e. the target is the time difference between the next and the current event. However, knowing the timestamp of the current event, we can calculate the timestamp of the following event. We optimize the weights of the neural network with the Adam learning algorithm [56] such that the cross entropy between the ground truth one-hot encoding of the next event and the predicted one-hot encoding of the next event as well as the mean absolute error (MAE) between the ground truth time until the next event and the predicted time until the next event are minimized.

Modeling the next activity prediction function f_a^1 and time prediction function f_t^1 with LSTMs can be done using several architectures. Firstly, we can train two separate models, one for f_a^1 and one for f_t^1, both using the same input features at each time step, as represented in Figure 18 (a). Secondly, f_a^1 and f_t^1 can be learned jointly in a single LSTM model that generates two outputs, in a multi-task learning setting [20] (Figure 18 (b)). The usage of LSTMs in a multi-task learning setting has shown to improve performance on all individual tasks when jointly learning multiple natural language processing tasks, including part-of-speech tagging, named entity recognition, and sentence classification [25]. A hybrid option between the architecture of Figures 18 (a) and (b) is an architecture of a number of shared LSTM layers for both tasks, followed by a number of layers that specialize in either prediction of the next activity or prediction of the time until the next event, as shown in Figure 18 (c).

It should be noted that activity prediction function f_a^1 outputs the probability distribution of various possible continuations of the partial trace. For evaluation purposes, we will only use the most likely continuation.

We implemented the technique as a set of Python scripts using the recurrent neural network library `Keras` [24]. The experiments were performed on a single NVidia Tesla k80 GPU, on which the experiments took between 15 and 90 seconds per training iteration depending on the neural network architecture. The execution time to make a prediction is in the order of milliseconds.

4.3.2. Experimental setup

In this section we describe and motivate the metrics, datasets, and baseline methods used for evaluation of the predictions of the next activities and of the timestamps of the next events. To the best of our knowledge, there is no existing technique to predict both the next activity and its timestamp. Therefore, we utilize one baseline method for activity prediction and a different one for timestamp prediction.

The remaining cycle time prediction method proposed by van der Aalst et al. [130] can be naturally adjusted to predict the time until the next event. To do so we build a transition system from the event log using either set, bag, or sequence abstraction, as in [130], but instead we annotate the transition system states with the average time until the next event. We will use this approach as a baseline to predict the timestamp of next event.

We evaluate the performance of predicting the next activity and its timestamp on two datasets. We use the chronologically ordered first 2/3 of the traces as training data and evaluate the activity and time predictions on the remaining 1/3 of the traces. We evaluate the next activity and the timestamp prediction on all prefixes $hd^k(\sigma)$ of all trace σ in the set of test traces for $2 \leq k < |\sigma|$. We do not make any predictions for the trace prefix of size one, since for those prefixes there is insufficient data available to base the prediction upon. To measure the error in predicting the time until the next event, we use mean absolute error (MAE).

Helpdesk dataset This log contains events from a ticketing management process of the help desk of an Italian software company[1]. The process consists of 9 activities, and all cases start with the insertion of a new ticket into the ticketing management system. Each case ends when the issue is resolved, and the ticket is closed. This log contains around 3,804 cases and 13,710 events.

BPI'12 subprocess W dataset This event log originates from the Business Pro-

[1]doi:10.17632/39bp3vv62t.1

cess Intelligence Challenge (BPI'12)[1] and contains data from the application procedure for financial products at a large financial institution. This process consists of three subprocesses: one that tracks the state of the application, one that tracks the states of work items associated with the application, and a third one that tracks the state of the offer. In the context of predicting the coming events and their timestamps we are not interested in events that are performed automatically. Thus, we narrow down our evaluation to the work items subprocess, which contains events that are manually executed. Further, we filter the log to retain only events of type *complete*. Two existing techniques [18, 33] for the next activity prediction have been evaluated on this event log with identical preprocessing, enabling comparison. Breuker et al. [18] describe a next activity prediction technique that they call RegPFA, which relies on techniques from the field of grammatical inference. Evermann et al. [33] also use LSTM but do not optimize the network architecture.

4.3.3. Results

Table 8 shows the performance of various LSTM architectures on the helpdesk and the BPI'12 W subprocess logs in terms of MAE on predicted time, and accuracy of predicting the next event. The specific prefix sizes are chosen such that they represent *short*, *medium*, and *long* traces for each log. Thus, as the BPI'12 W log contains longer traces, the prefix sizes evaluated are higher for this log. In the table, *all* reports the average performance on all prefixes, not just the three prefix sizes reported in the three preceding columns. The number of shared layers represents the number of layers that contribute to both time and activity prediction. Rows where the numbers of shared layers are 0 correspond to the architecture of Figure 18 (a), where the prediction of time and activities is performed with separate models. When the number of shared layers is equal to the number of layers, the neural network contains no specialized layers, corresponding to the architecture of Figure 18 (b). Table 8 also shows the results of predicting the time until the end of the next event using the adjusted method from van der Aalst et al. [130] for comparison. All LSTM architectures outperform the baseline approach on all prefixes as well as averaged over all prefixes on both datasets. Further, it can be observed that the performance gain between the best LSTM model and the best baseline model is much larger for the short prefix than for the long prefix. The best performance obtained on next activity prediction over all prefixes was a classification accuracy of 71% on the helpdesk log. On the BPI'12 W log the best accuracy is 76%, which is higher than the 71.9% accuracy on this log reported by Breuker et al. [18] and the 62.3% accuracy reported by Evermann et al. [33]. In fact, the results obtained with LSTM are consistently higher than both approaches. Even though Evermann et al. [33] also rely on LSTM in their approach, there are several differences which are likely to cause the performance gap. First of all, [33] uses a

[1]doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

Layers	Shared	N/l	Helpdesk MAE in days				Helpdesk Accuracy	BPI'12 W MAE in days				BPI'12 W Accuracy
			Prefix 2	4	6	All		Prefix 2	10	20	All	
						LSTM						
4	4	100	3.64	2.79	2.22	3.82	0.7076	1.75	1.49	1.02	1.61	0.7466
4	3	100	3.63	2.78	2.21	3.83	0.7075	1.74	1.47	1.01	1.59	0.7479
4	2	100	3.59	2.82	2.27	3.81	0.7114	1.72	**1.45**	1.00	1.57	0.7497
4	1	100	3.58	2.77	2.24	3.77	0.7074	1.70	1.46	1.01	1.59	0.7522
4	0	100	3.78	2.98	2.41	3.95	0.7072	1.74	1.47	1.05	1.61	0.7515
3	3	100	3.58	2.69	2.22	3.77	0.7116	**1.69**	1.47	1.02	1.58	0.7507
3	2	100	3.59	2.69	2.21	3.80	0.7118	**1.69**	1.47	1.01	1.57	0.7512
3	1	100	3.55	2.78	2.38	3.76	**0.7123**	1.72	1.47	1.04	1.59	0.7525
3	0	100	3.62	2.71	2.23	3.82	0.6924	1.81	1.51	1.07	1.66	0.7506
2	2	100	3.61	2.64	**2.11**	3.81	0.7117	1.72	1.46	1.02	1.58	0.7556
2	1	100	3.57	**2.61**	**2.11**	3.77	0.7119	**1.69**	**1.45**	1.01	**1.56**	**0.7600**
2	0	100	3.66	2.89	2.13	3.86	0.6985	1.74	1.46	0.99	1.60	0.7537
1	1	100	**3.54**	2.71	3.16	**3.75**	0.7072	1.71	1.47	**0.98**	1.57	0.7486
1	0	100	3.55	2.91	2.45	3.87	0.7110	1.72	1.46	1.05	1.59	0.7431
3	1	75	3.73	2.81	2.23	3.89	0.7118	1.73	1.49	1.07	1.62	0.7503
3	1	150	3.78	2.92	2.43	3.97	0.6918	1.81	1.52	1.14	1.71	0.7491
2	1	75	3.73	2.79	2.32	3.90	0.7045	1.72	1.47	1.03	1.59	0.7544
2	1	150	3.62	2.73	2.23	3.83	0.6982	1.74	1.49	1.08	1.65	0.7511
1	1	75	3.74	2.87	2.35	3.87	0.6925	1.75	1.50	1.07	1.64	0.7452
1	1	150	3.73	2.79	2.32	3.92	0.7103	1.72	1.48	1.02	1.60	0.7489
						RNN						
3	1	100	4.21	3.25	3.13	4.04	0.6581					
2	1	100	4.12	3.23	3.05	3.98	0.6624					
1	1	100	4.14	3.28	3.12	4.02	0.6597					
						Time prediction baselines						
Set abstraction [130]			6.15	4.25	4.07	5.83	-	2.71	1.64	1.02	1.97	-
Bag abstraction [130]			6.17	4.11	3.26	5.74	-	2.89	1.71	1.07	1.92	-
Sequence abstraction [130]			6.17	3.53	2.98	5.67	-	2.89	1.69	1.07	1.91	-
						Activity prediction baselines						
Evermann et al. [33]			-	-	-	-	-	-	-	-	-	0.623
Breuker et al. [18]			-	-	-	-	-	-	-	-	-	0.719

Table 8: Experimental results for the Helpdesk and BPI'12 W logs.

technique called *embedding* [80] to create feature descriptions of events instead of the features described above. Embeddings automatically transform each activity into a “useful” large dimensional continuous feature vector. This approach has shown to work really well in the field of natural language processing, where the number of distinct words that can be predicted is very large, but for process mining event logs, where the number of distinct activities in an event log is often in the order of hundreds or much less, no useful feature vector can be learned automatically. Second, [33] uses a two-layer architecture with 500 neurons per layer and does not explore other variants. We found performance to decrease when increasing the number of neurons from 100 to 150, which makes it likely that the performance of a 500-neuron model will decrease due to overfitting. A third and last explanation for the performance difference is the use of multi-task learning, which as we showed, slightly improves prediction performance on the next activity.

Even though the performance differences between our three LSTM architectures are small for both logs, we observe that most best performances (indicated in bold) of the LSTM model in terms of time prediction and next activity prediction are either obtained with the completely shared architecture of Figure 18 (b) or with the hybrid architecture of Figure 18 (c). We experimented with decreasing the number of neurons per layer to 75 and increasing it to 150 for architectures with one shared layer, but found that this results in decreasing performance in both tasks. It is likely that 75 neurons resulted in underfitting models, while 150 neurons resulted in overfitting models. We also experimented with traditional RNNs on one layer architectures and found that they perform significantly worse than LSTMs on both time and activity prediction.

4.4. Future Path Prediction

Using functions f_a^1 and f_t^1 repeatedly allows us to make longer-term predictions that predict further ahead than a single time step. We use $f_a^\perp$ and $f_t^\perp$ to refer to activity and time until next event prediction functions that predict the whole continuation of a running case, and aim at those functions to be such that $f_a^\perp(hd^k(\sigma)) = tl^k(\pi_{\mathscr{A}}(\sigma))$ and $f_t^\perp(hd^k(\sigma)) = tl^k(\pi_{\mathscr{T}}(\sigma))$

4.4.1. Approach

The future path, or suffix can be predicted by iteratively predicting the next activity and the time until the next event, until the next activity prediction function f_a^1 predicts the end of case, which we represent with $\perp$. More formally, we calculate the complete suffix of activities as follows:

$$f_a^\perp(\sigma) = \begin{cases} \sigma & \text{if } f_a^1(\sigma) = \perp \\ f_a^\perp(\sigma \cdot e), \text{with } e \in \mathscr{E}, \pi_{\mathscr{A}}(e) = f_a^1(\sigma) \wedge & \\ \quad \pi_{\mathscr{T}}(e) = (f_t^1(\sigma) + \pi_{\mathscr{T}}(\sigma(|\sigma|))) & \text{otherwise} \end{cases}$$

and we calculate the suffix of times until the next events as follows:

$$f_t^{\perp}(\sigma) = \begin{cases} \sigma, & \text{if } f_t^1(\sigma) = \perp \\ f_t^{\perp}(\sigma \cdot e), \text{with } e \in \mathscr{E}, \pi_{\mathscr{A}}(e) = f_a^1(\sigma) \wedge & \\ \quad \pi_{\mathscr{T}}(e) = (f_t^1(\sigma) + \pi_{\mathscr{T}}(\sigma(|\sigma|))) & \text{otherwise} \end{cases}$$

4.4.2. Experimental Setup

For a given trace prefix $hd^k(\sigma)$ we evaluate the performance of $f_a^{\perp}$ by calculating the distance between the predicted continuation $f_a^{\perp}(hd^k(\sigma))$ and the actual continuation $\pi_{\mathscr{A}}(tl^k(\sigma))$. Many sequence distance metrics exist, with Levenshtein distance being one of the most well-known ones. Levenshtein distance is defined as the minimum number of insertion, deletion, and substitution operations needed to transform one sequence into the other.

Levenshtein distance is not suitable when the business process includes parallel branches. Indeed, when $\langle a,b \rangle$ are the next predicted events, and $\langle b,a \rangle$ are the actual next events, we consider this to be only a minor error since it is often not relevant in which order two parallel activities are executed. However, Levenshtein distance would assign a cost of 2 to this prediction, as transforming the predicted sequence into the ground truth sequence would require one deletion and one insertion operation. An evaluation measure that better reflects the prediction quality of is the Damerau-Levenstein distance [27], which adds a swapping operation to the set of operations used by Levenshtein distance. Damerau-Levenshtein distance would assign a cost of 1 to transform $\langle a,b \rangle$ into $\langle b,a \rangle$. To obtain comparable results for traces of variable length, we normalize the Damerau-Levenshtein distance d by the maximum of the length of the ground truth suffix and the length of the predicted suffix and subtract the normalized Damerau-Levenshtein distance from 1 to obtain Damerau-Levenshtein Similarity (DLS):

$$DLS = 1 - \frac{d(y,\hat{y})}{\max(|y|,|\hat{y}|)}, \tag{4.1}$$

where $y = \pi_{\mathscr{A}}(tl^k(\sigma)), \hat{y} = f_a^{\perp}(hd^k(\sigma))$.

To the best of our knowledge, the most recent method to predict an arbitrary number of events ahead is the one by Polato et al. [91]. The authors first extract a transition system from the log and then learn a machine learning model for each transition system state to predict the next activity. They evaluate on predictions of a fixed number of events ahead, while we are interested in the continuation of the case until its end. We redid the experiments with their ProM plugin to obtain the performance on the predicted full case continuation.

For the LSTM experiments, we use a two-layer architecture with one shared layer and 100 neurons per layer, which showed good performance in terms of next activity prediction and predicting the time until the next event in the previous experiment (Table 8). In addition to the two previously introduced logs, we evaluate

Method	Helpdesk	BPI'12 W	BPI'12 W (no duplicates)	Environmental permit
Polato [91]	0.2516	0.0458	0.0336	0.0260
LSTM	**0.7669**	**0.3533**	**0.3937**	**0.1522**

Table 9: Suffix prediction results in terms of Damerau-Levenshtein similarity.

prediction of the suffix on an additional dataset, described below, which becomes feasible now that we have fixed the LSTM architecture.

Environmental permit dataset This is a log of an environmental permitting process at a Dutch municipality.[1] Each case refers to one permit application. The log contains 937 cases and 38,944 events of 381 event types. Almost every case follows a unique path, making the suffix prediction more challenging.

4.4.3. Results

Table 9 summarizes the results of suffix prediction for each log. As can be seen, LSTM outperforms the baseline [91] on all logs. Even though it improves over the baseline, the performance on the BPI'12 W log is low given that the log only contains 6 activities. After inspection we found that this log contains many sequences of two or more events in a row of the same activity, where occurrences of 8 or more identical events in a row are not uncommon. We found that LSTMs have problems dealing with this log characteristic, causing it to predict overly long sequences of the same activity, resulting in predicted suffixes that are much longer than the ground truth suffixes. Hence, we also evaluated suffix prediction on a modified version of the BPI'12 W log where we removed repeated occurrences of the same event, keeping only the first occurrence. However, we can only notice a mild improvement over the unmodified log.

4.5. Remaining Cycle Time Prediction

The proposed multi-task learning setting can be extended to also predict the remaining cycle time of a case. Indeed, time prediction function $f_t^{\perp}$ predicts the timestamps of all events in a running case that are still to come. Since the last predicted timestamp in a prediction generated by $f_t^{\perp}$ is the timestamp of the end of the case, it is easy to see that $f_t^{\perp}$ can be used for predicting the remaining cycle time of the running case. For a given unfinished case σ, $\hat{\sigma}_t = f_t^{\perp}(\sigma)$ contains the predicted timestamps of the next events, and $\hat{\sigma}_t(|\hat{\sigma}_t|)$ contains the predicted end time of σ, therefore the estimated remaining cycle time can be obtained through $\hat{\sigma}_t(|\hat{\sigma}_t|) - \pi(\sigma(|\sigma|))$.

For consistency of explanations, a dedicated Chapter 6 provides a comprehensive evaluation of our proposed multi-task LSTM and a single-task architecture,

[1] doi:10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

such as the one presented in [84], using a wide range of event logs as well as other baselines, for the problem of remaining cycle time prediction.

4.6. Summary

Deep learning neural networks have been typically applied in a single-task setting, where the prediction target is outputted directly by the output layer of the network. However, these approaches focus on making accurate predictions, rather than trying to explain them.

In this chapter, we apply a multi-task LSTM as a means to decompose the time prediction in terms of its components. Namely, we design a technique to predict the next activity of a running case and its timestamp. We showed that this technique outperforms existing baselines on real-life data sets. Additionally, we found that predicting the next activity and its timestamp via a single model (multi-task learning) yields a higher accuracy than predicting them using separate models. We then showed that this basic technique can be generalized to address two other predictive process monitoring problems: predicting the entire continuation of a running case and predicting the remaining cycle time. In this way, the remaining cycle time can be explained in terms of cycle times of individual activities to be executed in an ongoing case.

Although multi-task learning tends to achieve better accuracy by sharing information across different tasks, it is not trivial to extend it to make use of all available case and event attributes, not only activity names and their timestamps. Indeed, the predictions are performed iteratively – at the i-th step we predict the $(i+1)$-th activity and its timestamp, and the prediction result is then included in the input for the $(i+1)$-th step, and so on. In order to use all data attributes, one would have to predict the value of each such attribute for each step. That would make the architecture overly complex as well as significantly affect the running times. Furthermore, for longer sequences, the propagation of errors for incorrectly predicted attributes will affect the overall time prediction accuracy. To address this limitation, in Chapter 5, we propose an alternative, process model-based, approach for the prediction of temporal process performance measures.

5. PROCESS MODEL DRIVEN PREDICTION OF TEMPORAL PROCESS MEASURES

As illustrated earlier, a number of approaches have been proposed to predict quantitative process performance measures for running instances of a process, including remaining cycle time, cost, or probability of deadline violation. However, these approaches adopt a black-box approach, insofar as they predict a single scalar value without decomposing this prediction into more elementary components. In this chapter, we propose a white-box approach for predictive process monitoring of continuous performance measures, such as the remaining time, based on a BPMN process model automatically discovered from the event log.

The key idea is to first predict the performance measure at the level of activities, and then to aggregate these predictions at the level of a process instance by means of flow analysis techniques. We develop this idea in the context of predicting the remaining cycle time of ongoing process instances. Compared to Chapter 4, here we present an alternative solution to the research question **RQ1**, where explainability is achieved by the mapping prediction components into the process model. The evaluation for the approach presented in this chapter is reported in Chapter 6.

This chapter is structured as follows. Section 5.1 provides an overview of the proposed approach. Section 5.2 describes flow analysis and how to apply it to calculate cycle time of a structured process. The next four sections focus on the key parts of our approach, namely automatic process discovery, replaying traces on a process model, obtaining flow analysis formula, computing the formulas components. Finally, Section 5.7 provides a summary of the chapter.

5.1. Introduction

Flow analysis is a family of techniques that enables estimation of the overall performance of a process given knowledge about the performance of its activities. For example, using flow analysis one can calculate the average cycle time of an entire process if the average cycle time of each activity is known. Flow analysis can also be used to calculate the average cost of a process instance knowing the cost-per-execution of each activity, or calculate the error rate of a process given the error rate of each activity [32]. Since flow analysis is typically applied to structured process models described in the BPMN notation, the estimation can be easily explained in terms of its elementary components.

Our approach exploits historical execution traces in order to discover a structured process model. Once the model has been discovered, we identify its set of activities and decision points and train two families of machine learning models: one to predict the cycle time of each activity, and the other to predict the branching probabilities of each decision point. To speed up the performance at runtime, these steps are performed offline (Figure 19).

At runtime, given an ongoing process instance, we align its partial trace with the discovered process model to determine the current state of the instance. Next, we parse the process model to construct a tree using the algorithm described in [133]. The tree is traversed starting from the state up to the process end and deduce a formula for remaining time using rules described in Section 5.2. The formula includes cycle times of activities and branching probabilities of decision points that are reachable from the current execution state. These components are predicted using previously trained regression and classification models. Finally, we evaluate the formula and obtain the expected value of the remaining cycle time.

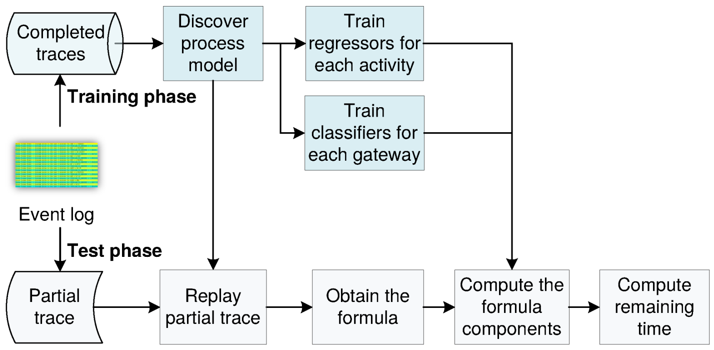


Figure 19: Overview of the proposed approach.

5.2. Overview of Flow Analysis

To understand how flow analysis works, we start with an example of a process with *sequential* SESE fragments of events as in Figure 20a. Each fragment has a cycle time T_i. Since the fragments are performed one after the other, we can intuitively conclude that the cycle time CT of a purely sequential process with N fragments is the sum of the cycle times of each fragment [32]:

$$CT = \sum_{i=1}^{N} T_i \tag{5.1}$$

Let us now consider a process with a decision point between N mutually exclusive fragments, represented by an *XOR gateway* (Figure 20b). In this case, the average cycle time of the process is determined by:

$$CT = \sum_{i=1}^{N} p_i \cdot T_i, \tag{5.2}$$

where p_i denotes the branching probability, i.e. frequency with which a given branch i of a decision gateway is taken.

In case of parallel gateways where activities can be executed concurrently as in Figure 20c, the combined cycle time of multiple fragments is determined by the slowest of the fragments, that is:

$$CT = \max_{i=1...n} T_i \tag{5.3}$$

Another recurrent pattern is the one where a fragment of a process may be repeated multiple times, for instance, because of a failed quality control. This situation is called *rework* and is illustrated in Figure 20d. The fragment is executed once. Next, it may be repeated each time with a probability r referred to as the *rework probability*. Assuming that the probability of reworking for the N-th time is not dependent on the value of N, the number of times that the rework fragment will be executed follows a geometric distribution with the expected value $1/(1-r)$. Thus, the average cycle time of the fragment in this case is:

$$CT = \frac{T}{1-r} \tag{5.4}$$

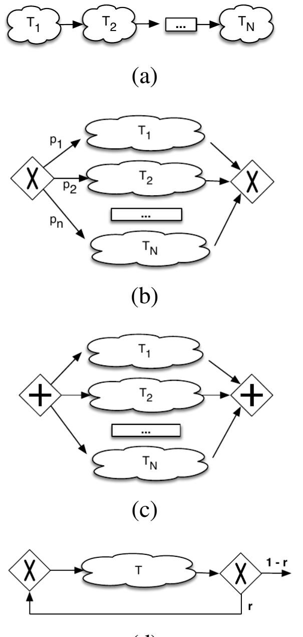


Figure 20: Typical process model patterns: sequential (a), XOR-block (b), AND-block (c) and rework loop (d).

Besides cycle time, flow analysis can also be used to calculate other performance measures. For instance, assuming we know the average cost of each activity, we can calculate the cost of a process more or less in the same way as we calculate cycle time. In particular, the cost of a sequence of activities is the sum of the costs of these activities. The only difference between calculating cycle time and calculating cost relates to the treatment of AND-blocks. The cost of an AND-block such as the one shown in Figure 20c is not the maximum of the cost of the branches of the AND-block. Instead, the cost of such a block is the sum of the costs of the branches. This is because after the AND-gateway is traversed, every branch in the AND join is executed and therefore the costs of these branches add up to one another [32].

In case of structured process models, we can relate each fragment to one of the four described types and use the aforementioned equations to estimate the required performance measure. However, in case of an unstructured process model or if a model contains other modeling constructs besides AND and XOR gateways, the method for calculating performance measures becomes more complicated [144].

Importantly, flow analysis equations were defined when the *average* cycle time of the process is in question. However, the same equations can also be used to predict the *remaining* cycle time of a given ongoing instance σ based on the information available in the prefix $hd^k(\sigma)$. For example, in Equation 5.2, branching probabilities p_i returned by the predictor can serve as confidence values [122]. Thus, among the predicted cycle times T_i for each fragment, the highest weight is given to the one that corresponds to the most likely continuation of σ.

5.3. Discovering Process Models from Event Logs

The proposed approach relies on a process model as input. However, since the model is not always known or might not conform to the real process, generally we need to discover the model from event logs. For that, we use a two-step automated process discovery technique proposed in [4] that has been shown to outperform traditional approaches with respect to a range of accuracy and complexity measures. The technique has been implemented as a standalone tool [1] as well as a ProM plugin, namely *StructuredMiner*.

The technique in [4] pursues a two-phase "discover and structure" approach. In the first phase, a model is discovered from the log using either the Heuristics Miner or Fodina Miner. They have been shown to consistently produce accurate, but potentially unstructured or even unsound models. In the second phase, the discovered model is transformed into a sound and structured model by applying two techniques: a technique to maximally block-structure an acyclic process model and an extended version of a technique for block-structuring flowcharts.

Next, we used a technique called Refined Process Structure Tree (RPST) [133] to parse a process model into a tree that represents a hierarchy of SESE fragments. Each fragment relates to the subgraph induced by a set of edges. A fragment in the tree consists of all fragments at the lower level, but fragments at the same level are disjoint. Each fragment in an RPST belongs to one of four types [93]: a *trivial* fragment includes only a single edge; a *polygon* fragment is a sequence

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

of fragments; a *bond* corresponds to a fragment where all child fragments share a common pair of vertices. Any other fragment is considered a *rigid*.

5.4. Replaying Partial Traces on Process Models

For a given partial trace, to predict its remaining time, we need to determine the current state of the trace relative to the process model. For that, we map, or align, a trace to the process model using the technique described in [2] which is available as a plugin for the open-source process analytics platform Apromore[1].

The technique treats a process model as a graph that is composed of activities as nodes and their order dependencies as arcs. A case replay can be seen as a series of coordinated *moves*, including those over the model activities and gateways and those over the trace events. In that sense, a case replay is also termed an *alignment* of a process model and a trace. Ideally, this alignment should result in as many matches between activity labels on the model and event labels in the trace as possible. However, practically, the replay may choose to skip a number of activities or events in search of more matches in later moves. Moves on the model must observe the semantics of the underlying modeling language which is usually expressed by the notion of *tokens*. For example, for a BPMN model, a move of an incoming token over an XOR split gateway will result in a single token produced on one of the gateway outgoing branches, while a move over an AND split gateway will result in a separate token produced on each of the gateway outgoing branches. The set of tokens located on a process model at a point in time is called a *marking*. On the other hand, a move in the trace is sequential over successive events of the trace ordered by timestamps, one after another. Thus, after every move, either on the model or in the trace, the alignment comes to a *state* consisting of the current marking of the model and the index of the current event in the trace.

In [2], cases are replayed using a heuristics-based backtracking algorithm that searches for the best alignment between the model and a partial trace. The algorithm can be illustrated by a traversal of a process tree starting from the root node, e.g. using depth-first search, where nodes represent partial candidate solution states (Figure 21). Here the state represents the aforementioned alignment state of the case replay. At each node, the algorithm checks whether the alignment state till that node is good enough, based on costs accumulated along the path from the root to the current node. If the alignment is good, it generates a set of child nodes of that node and continues down that path; otherwise, it stops at that node, i.e. it prunes the branch under the node, and backtracks to the parent node to traverse other branches.

[1] `http://apromore.org`

```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for $n_i$ : CHILDREN(n) do EXPLORE($n_i$)
```

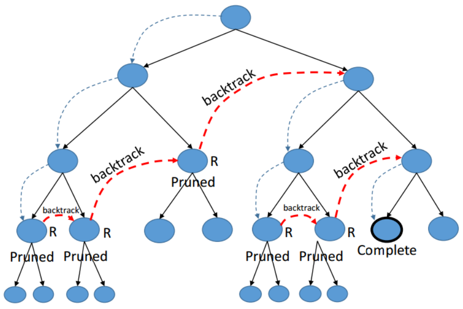


Figure 21: Backtracking algorithm (taken from [2]).

5.5. Obtaining the Flow Analysis Formulas

Having determined the current state of the case execution, we traverse the process model starting from that state until the process completion in order to obtain the flow analysis formulas.

As a running example, let us consider a simple process model in Figure 22. Applying the flow analysis formulas described in Section 5.2, the *average* cycle time of this process can be decomposed as follows:

$$CT = T_A + \max(T_B + T_C, T_D) + T_F + p2\left(T_G + \frac{T_H}{1-r}\right) \tag{5.5}$$

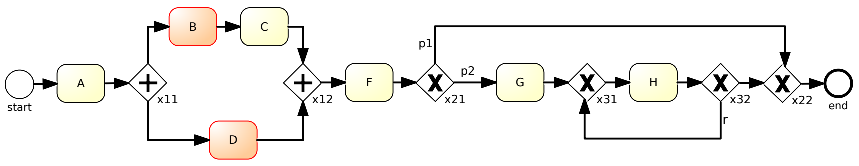


Figure 22: Example process model. Highlighted is the current marking

Note that one of the branches of gateway $X21$ is empty and therefore does not contribute to the cycle time. Therefore, only the branch with the probability $p2$ is included in the equation.

The components of the formula – cycle times of individual activities and branching and rework probabilities – can be estimated as averages of their historical values. However, since we deal with *ongoing* process cases, we can use the information that is already available from the case prefix to *predict* the above components.

Consider, we have a partial trace $hd(\sigma) = \langle A, D, B \rangle$. Replaying this trace on the given model as described in the Section 5.4, we find the current marking to be in the states B and D within the AND-block. Traversing the process model starting from these states until the process end, for the remaining cycle time of $hd(\sigma)$, we obtain the formula:

$$\begin{aligned} CT_{rem} &= 0 + \max(0 + T_C, 0) + T_F + p2\left(T_G + \frac{T_H}{1-r}\right) = \\ &= T_C + T_F + p2\left(T_G + \frac{T_H}{1-r}\right). \end{aligned} \tag{5.6}$$

Since activities A, B and D have already been executed, they do not contribute to the *remaining* cycle time. Thus, they are not a part of the formula. All the other formula terms need to be predicted using the data from $hd(\sigma)$.

Similarly, if a current marking is inside an XOR block, its branching probabilities need not be predicted. Instead, the probability of the branch that has actually been taken is set to 1 while the other probabilities are set to 0.

A more complex situation arises when the current marking is inside the rework loop. In this case, we "unfold" the loop as shown in Figure 23. Specifically, we separate the already executed occurrences of the rework fragment from the potential future occurrences and take the former out of the loop. Let us consider a partial trace $hd(\sigma) = \langle A, D, B, C, F, G, H \rangle$. Since H has occurred once, according to the process model (Figure 22), with a probability r, it may be repeated, otherwise, the rework loop is exited. To signal this choice, we take the first occurrence of H out of the loop and place an XOR gateway after it. One of the branches will contain a rework loop of future events with the same probability r, while the other one will reflect an option to skip the loop altogether. Thus, the cycle time of the whole fragment can be decomposed as follows:

$$CT_H = \mathbf{T_{H'}} + r\frac{T_H}{1-r}, \tag{5.7}$$

where $\mathbf{T_{H'}}$ refers to the cycle time of already executed occurrence(s) of H. It is highlighted in bold font, meaning that we should take the actual cycle time rather than the predicted.

5.6. Computing Cycle Times and Branching Probabilities

We can use the flow analysis formulas produced by the method described in Section 5.5 to compute the remaining cycle time of a case, given: (i) an estimate of the cycle time of each activity reachable from the current execution state; and (ii) an estimate of the branching probability of each flow stemming from a reachable XOR-split (herein called a reachable *conditional flow*). Given an execution state, we can obtain these estimates in several ways including:

1. By using the prediction models produced for each reachable activity and for each reachable conditional flow, taking into account only traces that reach

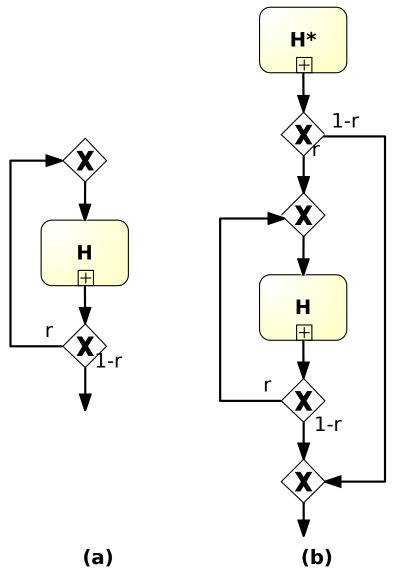


Figure 23: Unfolding the rework loop of F

the current execution state. We herein call this approach *predictive flow analysis*.

2. By computing the mean cycle time of each reachable activity and the traversal frequency of each reachable conditional flow, again based only on the suffixes of traces that reach the execution state in question. We call this approach *mean flow analysis*
3. By combining the above two approaches as follows: If the number of training cases to fit the prediction model for a given activity A is less than a certain threshold N_0, then we find the mean cycle times and branching probabilities as in the mean flow analysis method. Otherwise, we fit the predictors as in predictive flow analysis. This hybrid approach is herein called *adaptive flow analysis*. The value N_0 can be treated as a hyperparameter and is to be determined during the hyperparameter optimization procedure.

The rationale for the adaptive flow analysis is that in a high dimensional feature space, the number of observations to train a model may not be enough, thus the predictions may be unreliable and unstable [96]. This may happen when we predict cycle times of rare activities or branching probabilities of gateways that are rarely taken. Furthermore, if a prefix is too short, there might not be enough information in it to predict cycle times of some activities and gateways' branching probabilities, especially those that are executed near the process end. In such cases, we can then use the mean historical activity cycle times and mean probabilities instead.

For each activity in the process model, to predict its cycle time, we train a regression model, while for predicting branching probabilities we fit classification models for each corresponding XOR gateway. In the latter case, each branch of a gateway is assigned a class starting from 0, and the model makes predictions about the probability of each class. The predictive models are trained for prefixes $hd^k(\sigma)$ of all traces σ in the training set for $2 \leq k < |\sigma|$. We do not train and make predictions after the first event since for those prefixes there is insufficient data available to base the predictions upon.

In order to fit predictive models, we need to encode process execution traces in the form of fixed-length feature vectors. In the related work, there have been proposed two common approaches to achieve this:

1. Training a single predictive model using case attributes as well as aggregated attributes of executed events. This approach has been used in [29, 71, 119, 132], among others. Typical aggregation functions applied over event attributes include *count* (for categorical attributes) and *mean* (for numeric ones).
2. Training multiple predictive models, one for each possible prefix length. In this way, values of each event attribute need not be aggregated across the partial case, therefore, they may be preserved, as well as the order of attributes. In other words, this encoding is lossless as compared to the previous one. This approach has been used in [66, 122, 137], among others.

Although lossless, the latter approach has been shown to be outperformed by a single predictive model in some circumstances [120]. This is due to the fact that combining the knowledge from all events performed so far may provide more signal than using the order of events and their individual attributes. In order to quantify this phenomenon in the context of flow analysis, in our work we consider both approaches.

Feature encoding with a single predictive model

In a *single predictive model* approach, we create feature vectors by extracting the following data from each prefix of the corresponding prefix log:

- Case attributes. These attributes are static in the sense that they do not change as the case progresses. As such, they can simply be appended to feature vectors.
- Aggregated event attributes. As event attributes are dynamic, i.e. each event in a trace has its own attribute values, in order to encode them in a fixed-length vector, we apply several aggregation functions. For numeric attributes, we compute their mean, minimum and maximum values across a partial case, as well as their sum and standard deviation. For categorical attributes, we count how many times a specific level has appeared (e.g. how many times a specific activity has been executed, or how many activities a specific resource has performed).

To fit a predictive model, we append to these feature vectors the value of the target variable y that is to be learned. For example, if we are to predict the cycle time of an activity, we calculate it as the time difference (in seconds) between the completion timestamp of that activity and the completion timestamp of the previous activity. If the activity is never executed in a given case, its cycle time is undefined. Therefore, we exclude such cases from the training data. Conversely, if an activity occurs multiple times in a case, we take its average cycle time.

Similarly, to predict branching probabilities, we assign a class label to each outgoing branch. For example, if we are to predict the branching probabilities for the $X32$ gateway, we can assign class 0 to the branch that leads to rework and class 1 to the other branch. Evidently, the probability of class 0 would be equal to the rework probability r. Thus, for the first case in Table 1 $X32 = 1$, while for the second case $X32$ is undefined since the case does not reach that decision point. Consequently, the second case cannot be used to populate a training set for the prediction of $X32$.

From these examples, it is evident that the dimensionality of feature vectors is constant for a given log and depends on: (i) the number of case attributes, (ii) the number of categorical event attributes and the number of possible values, or levels, of each such attribute, (iii) the number of numeric event attributes and the number of aggregation functions applied for them.

Feature encoding with multiple predictive models

In a *multiple predictive model* approach, we concatenate case attributes and, for each position in a trace, the event occurring in that position and the value of each event attribute in that position. In general, for a case with U case attributes $\{s_1,\dots,s_U\}$ containing k events $\{e_1,\dots,e_k\}$, each of them having an associated payload $\{d_1^1,\dots,d_1^R\},\dots,\ \{d_k^1,\dots,d_k^R\}$ of length R, the resulting feature vector would be:

$$\vec{X} = (s_1,\dots,s_U,e_1,d_1^1,\dots,d_1^R,\dots,e_k,d_k^1,\dots,d_k^R) \tag{5.8}$$

With this encoding, the length of the feature vector increases with each executed event e_k:

$$U + k \cdot R \tag{5.9}$$

Consequently, this approach requires fitting a separate model for each possible length of a test prefix.

As compared to the single model approach where each sample of a prefix log created from the training set becomes a training sample, in the multiple model approach, each training trace produces only one sample. It should be noted that if a case does not reach length k, i.e. it finishes earlier, there are two options to proceed:

- Discard such cases from the training set for prefix lengths k
- Impute missing event attribute values with zeros or their historical averages computed from cases that have at least k events, so that the resulting feature vectors' dimensionality would be determined by Eq. 5.9. This approach is often referred to as *padding* in machine learning [95].

In this work, we will use the former approach, as we are mostly interested in predictions in the *early* stages of a process evolution where many process instances have not finished yet, so there is still sufficient amount of data to train the predictive models.

5.7. Summary

In this chapter, we put forward some potential benefits of a process model-based approach to predicting quantitative process performance measures. Rather than predicting single scalar measures, we demonstrated how these measures can be estimated as aggregations of corresponding performance measures of the activities composing the process. In this way, the predicted measures become more explainable, as they are decomposed into elementary components. Thus, business analysts can pinpoint the bottlenecks in the process execution and provide better recommendations to keep the process compliant with the performance standards.

We implemented and evaluated three approaches – one where the formulas' components are predicted from the trace prefix based on the models trained on historical completed traces, another one that instead uses constant values obtained from the historical averages of similar traces, and finally, a hybrid approach that combines the strengths of the above two approaches. In Chapter 6 we will evaluate these three approaches to predict the remaining cycle time, which is a common process performance measure.

6. EXPERIMENTAL EVALUATION

In the two previous chapters, we have presented two alternative solutions to the research question **RQ1**, one that is based on multi-task deep learning networks, and the other one based on flow analysis. In this chapter, we empirically compare the two approaches with each other and with various baselines proposed in previous work. In particular, we seek to answer the research question **RQ2**, namely "What is the impact of explainability on prediction accuracy in the context of predictive process monitoring techniques for temporal measures?"

This research question can be decomposed into several sub-questions, namely:

RQ2.1. What is the relative performance of different variants of flow analysis-based techniques? Does the adaptive flow analysis approach provide added value over the mean and predictive flow analysis approaches (Chapter 5)?

RQ2.2. What is the relative performance of long short-term memory networks trained in single-task and multi-task learning settings (Chapter 4)?

RQ2.3. Which techniques tend to provide *accurate* predictions in the *early* stages of case evolution? How explainable are they?

This chapter is structured as follows. Section 6.1 provides an overview of datasets used in the evaluation as well as lists their main characteristics. Section 6.2 describes the experimental pipeline, namely data preprocessing and splitting strategies, evaluation metrics employed, baselines to compare against and the hyperparameter optimization procedure. Section 6.3 provides the results of the evaluation and discusses them from various viewpoints. Section 6.4 identifies internal and external threats to validity. Finally, Section 6.5 provides a summary of the chapter.

6.1. Datasets

We conducted the experiments using 17 real-life event logs. To ensure the reproducibility of the experiments, the logs we used are publicly available at the *4TU Center for Research Data*[2] as of May 2018, except for one log, which we obtained from the demonstration version of a software tool.

We excluded from the evaluation those logs that do not pertain to business processes (e.g., JUnit 4.12 Software Event Log and NASA Crew Exploration Vehicle). Such logs are usually not case-based, or they contain only a few cases. Furthermore, we discarded the log that comes from the Environmental permit application process, as it is an earlier version of the BPIC 2015 event log from the same collection. Finally, we discarded the BPIC 2018 event log, as the underlying process exhibits strong seasonal patterns that can be trivially predicted. Specifically, 76% of cases in the log terminate on one of the three days: 2018-01-06, 2017-01-07 and 2016-02-19.

[2] `https://data.4tu.nl/repository/collection:event_logs_real`

Table 10 summarizes the basic characteristics of each resulting dataset, namely the number of complete cases, the ratio of distinct (unique) traces (DTR), the number of event classes, the average number of distinct events per trace (DER), average case length, i.e. the average number of events per case and its coefficient of variation (CV), average case duration (in days) and its CV, and the number of case and event attributes. The datasets possess a very diverse range of characteristics and originate from a variety of domains.

BPIC 2011. This event log originates from the first Business Process Intelligence Challenge [1] held in 2011 and a describes the executions of a healthcare process related to the treatment of patients diagnosed with cancer in a large Dutch academic hospital. Each case refers to the treatment of a different patient. The event log contains domain specific attributes that are both case attributes and event attributes in addition to the standard XES attributes.

BPIC 2012. This event log originates from the 2012 Business Process Intelligence Challenge [2] and contains data from the application procedure for financial products at a large financial institution. The process consists of three subprocesses: one that tracks the state of the application (BPIC'12 A), one that tracks the state of the offer (BPIC'12 O), and a third one that tracks the states of work items associated with the application (BPIC'12 W). For the purpose of this work, we treat these three subprocesses as separate datasets.

BPIC 2015. This dataset assembles event logs from 5 Dutch municipalities, pertaining to the building permit application process. We treat the datasets from each municipality as separate event logs.

BPIC 2017. This log originates from a loan application process in a Dutch financial institution. For each application, the institution can make one or multiple offers, while the customer can accept at most one of them.

Credit Requirement (CR). This log contains information about a credit requirement process in a bank.[3] It contains data about events, time execution, etc. A distinctive feature is that all cases follow the same path, thus the process model is sequential and does not contain gateways.

Helpdesk. This log contains events from a ticketing management process of the help desk of an Italian software company.[4] Each case starts with the insertion of a new ticket into the ticketing management system and ends when the issue is resolved, and the ticket is closed.

Hospital. This log was obtained from the financial modules of the ERP system of a regional hospital.[5] The log contains events that are related to the billing of medical services that have been provided by the hospital. Each trace of the event log records the activities executed to bill a package of medical services that were

[1] doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54

[2] doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[3] doi:10.4121/uuid:453e8ad1-4df0-4511-a916-93f46a37a1b5

[4] doi:10.17632/39bp3vv62t.1

[5] doi:10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741

bundled together. The log is a random sample of process instances that were recorded over three years.

Invoice. This log refers to the invoice approval process and comes as a demonstration log with the Minit process intelligence software.[1]

Production. This log contains data from a manufacturing process [2]. Each trace records information about the activities, workers and/or machines involved in producing an item.

Sepsis. This real-life event log contains events of sepsis cases from a hospital.[3] One case represents the pathway through the hospital.

Traffic fines. The last log describes a road traffic fines management process[4] in an Italian police unit. It contains events related to fine notifications, as well as partial repayments.

Table 10: Statistics of the datasets used in the experiments.

log	# cases	DTR	event classes	DER	mean case length	CV case length	mean case duration	CV case duration	# attributes (case+event)	Domain
bpic2011	1140	0.858	251	0.505	131.342	1.542	387.283	0.875	6+10	HC
bpic2012a	12007	0.001	10	1	4.493	0.404	7.437	1.563	1+4	Fin
bpic2012o	3487	0.002	7	1	4.562	0.126	15.048	0.606	1+4	Fin
bpic2012w	9650	0.235	6	0.532	7.501	0.97	11.401	1.115	1+5	Fin
bpic2015_1	696	0.976	189	0.967	41.343	0.416	96.176	1.298	17+8	PA
bpic2015_2	753	0.999	213	0.969	54.717	0.348	159.812	0.941	17+8	PA
bpic2015_3	1328	0.968	231	0.975	43.289	0.355	62.631	1.555	18+8	PA
bpic2015_4	577	0.998	179	0.97	42	0.346	110.835	0.87	15+8	PA
bpic2015_5	1051	0.997	217	0.972	51.914	0.291	101.102	1.06	18+8	PA
bpic2017	31509	0.207	26	0.878	17.826	0.32	21.851	0.593	3+15	Fin
credit	10035	0	8	1	8	0	0.948	0.899	0+7	Fin
helpdesk	3218	0.002	5	0.957	3.293	0.2	7.284	1.366	7+4	CS
hospital	59228	0	8	0.995	5.588	0.123	165.48	0.671	1+20	HC
invoice	5123	0.002	17	0.979	12.247	0.182	2.159	1.623	5+10	FI
production	225		28	0.454	20.191	1.034	20.467	1.03	2+13	M
sepsis	1035	0.076	6	0.995	5.001	0.288	0.029	1.966	23+10	HC
traffic_fines	150370	0.002	11	0.991	3.734	0.439	341.676	1.016	4+10	PA

Domains: HC – healthcare, Fin – financial, PA – public administration, CS – customer service, M – manufacturing

6.2. Experimental Setup

In this section, we describe our approach to split the event logs into training and test sets along the temporal dimension. Next, we provide a description of our

[1] `https://www.minit.io`

[2] doi:10.4121/uuid:68726926-5ac5-4fab-b873-ee76ea412399

[3] doi:10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

[4] doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

evaluation criteria and the baselines to compare against. Finally, we discuss the hyperparameter optimization procedure employed in our evaluation.

6.2.1. Data Preprocessing

Data preprocessing is a crucial step prior to modeling as it can often have a significant impact on generalization performance of a supervised machine learning algorithm [58].

All the logs have been preprocessed beforehand to ensure the maximum achievable prediction accuracy. Firstly, since the training and validation procedures require having a complete history of each case, we remove incomplete cases, as well as cases that have been recorded not from their beginning. Information about case completion may be recorded in the log explicitly, with a dedicated end event. Otherwise, we need to apply manual rules to filter out incomplete cases. For example, in the Traffic fines log, we consider traces where the last recorded event is *Send Fine* to be pending and therefore incomplete.

Secondly, from the available timestamps we extract additional features, such as the time elapsed since the case has started, time elapsed since the previous event and current day of the week. Intuitively, if the last event observed occurred at the end of the working day or at the end of the working week, the time until the next event is expected to be longer. Furthermore, if both start and complete timestamps are available for a log, we extract event durations.

Thirdly, for the categorical attributes we check the number of distinct values (*levels*). For some attributes, the number of levels can be very large, with some of the levels appearing only in a few cases. Thus, one-hot encoding (Section 2.1) will significantly increase the dimensionality of the data, with many sparse features. In order to avoid that, we mark infrequent attributes that appear in at least ten training samples as *other*. This process is repeated for all categorical attributes except the activity name, where we retain all category levels.

Next, in order to take into account resource contention, we extract the number of open cases at the particular point in time and the number of work items the current resource is also working on. These features are added as event attributes for each event.

Finally, even though event logs are recorded automatically by enterprise systems, there may be some missing data present. However, this usually happens due to the fact that some attributes are not applicable for a particular event. For example, in Traffic fines log, the payment amount is only available for payment events, while for other events it is marked as “missing”. Another reason for missing data is that in some logs data attributes are only recorded if they have changed since the previous event. In such situations, we look for the closest preceding event in the given case where the attribute was available and propagate its value forward.

6.2.2. Data Split

In order to assess the predictive power of a predictor, we need to design experiments in such a way that would prevent overfitting. In other words, we want the accuracy measure to have low variance and low bias. To ensure that, we need to properly separate training data from test data. Two common splitting strategies in machine learning are the train-test split and cross-validation [82].

In order to simulate a real-life situation where prediction models are trained using historical data and applied to running cases, we employ a so-called temporal split to divide the event log into train and test cases. Specifically, all cases in the log are ordered according to their start time and the first 80% are used to fit the models, while the remaining 20% are used to evaluate the prediction accuracy. In other words, the classifier is trained with all cases that started before a given date T_1 which would represent a current point in time in a real-life scenario, and the testing is done only on cases that start afterwards. Technically, cases that start before T_1 and are still running at T_1 should not be included in either set. However, to prevent the exclusion of a significant number of cases, in our experiments, we allow the two sets not to be completely temporally disjoint.

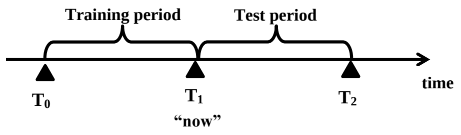


Figure 24: Temporal split of the training and test sets.

6.2.3. Evaluation Metrics

Two measures commonly employed to assess a predictive process monitoring technique are accuracy and earliness [31, 66, 122]. Indeed, in order to be useful, a prediction should be accurate and should be made early on to allow enough time to act upon.

Accuracy. To assess the accuracy of the prediction of continuous variables, well-known error metrics are Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Mean Percentage Error (MAPE) [53], where MAE is defined as the arithmetic mean of the prediction errors, RMSE as the square root of the average of the squared prediction errors, while MAPE measures error as the average of the unsigned percentage error. We observe that the value of remaining time tends to be highly varying across cases of the same process, sometimes with values on different orders of magnitude. RMSE would be very sensitive to such outliers. Furthermore, the remaining time can be very close to zero, especially near the end of the case, thus MAPE would be skewed in such situations. Hence, we use MAE to measure the error in predicting the remaining time. Formally,

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6.1}$$

where $y_i \in \mathscr{Y} = \mathbb{R}$ is the actual value of a function in a given point and $\hat{y}_i \in \mathscr{Y} = \mathbb{R}$ is the predicted value.

Earliness. A common approach to measure the earliness of the predictions is to evaluate the accuracy of the models after each arrived event or at fixed time intervals. Naturally, uncertainty decreases as a case progresses towards its completion. Thus, the earlier we reach the desired level of accuracy, the better the technique is in terms of its earliness.

To measure earliness, we make predictions for prefixes $hd^k(\sigma)$ of traces σ in the test set starting from $k = 1$. However, using all possible values of k is fraught with several issues. Firstly, a large number of prefixes considerably increases the training time of the prediction models. Secondly, for a single model approach, the longer cases tend to produce much more prefixes than shorter ones and, therefore, the prediction model is biased towards the longer cases [120]. Finally, for a multiple model approach, if the distribution of case lengths has a long tail, for very long prefixes, there are not enough traces with that length, and the error measurements become unreliable. Consequently, we use prefixes of up to 20 events only in both the training and the test phase. If a case contains less than 20 events, we use all prefixes, except the last one, as predictions do not make sense when the case has completed. In other words, the input for our experiments is a filtered prefix log $L^* = \{hd^k(\sigma) : \sigma \in L, 1 \leq k \leq min(|\sigma| - 1, 20)\}$.

Inherently, there is often a trade-off between accuracy and earliness. As more events are executed, due to more information becoming available, the prediction accuracy tends to increase, while the earliness declines [106,122]. As a result, we measure the performance of the models w.r.t. each dimension separately.

6.2.4. Baselines

We compare our deep learning and flow analysis-based approaches against several baselines. Firstly, we use all 12 feasible combinations of canonical encoding and bucketing types (Section 3.3) including those that have not been used in any existing approach in the literature. In this way, we will be able to assess the benefits of each combination, while abstracting implementation nuances of each individual primary method.

Secondly, we use a transition system (TS) based method proposed by van der Aalst et al. [130] applying both set, bag and sequence abstractions. In the final evaluation, we report the abstraction that achieved the highest accuracy.

Finally, we use the stochastic Petri-net based approach proposed by Rogge-Solti and Weske [99, 100]. Specifically, we use the method based on the *constrained* Petri net, as it was shown to have the lowest prediction error. However, their original approach makes predictions at fixed time points, regardless of the arriving events. To make the results comparable to our approach, we modify the method to make predictions after each arrived event.

6.2.5. Hyperparameter Optimization

In order to achieve the best performance within each technique, the hyperparameters of the predictors need to be optimized separately for each method and dataset. For that, we further split the training set randomly into 80% training and 20% validation data. The former set is used to fit the predictors using combinations of hyperparameters, while the latter is used to evaluate the performance of each combination. As we use MAE to assess the predictive power each technique, we will also use MAE on the validation data to select the combination of hyperparameters that achieves the best validation performance and retrain a model with these parameters, now using the original training set. For all predictors, we apply random search [9] procedure with 50 iterations. To define the search space, for each hyperparameter, we specify its bounds and a distribution to sample values from.

We perform the experiments using Python libraries Scikit-learn[1] for XGBoost and SVM and Keras[2] with TensorFlow backend for LSTM. These libraries allow for a wide range of learning parameters. Table 11 lists hyperparameters that were optimized during the random search. All other hyperparameters were left to their default values in the corresponding library.

Table 11: Hyperparameters tuned via grid search.

Parameter	Explanation	Search space
	XGBoost	
n_estimators	Number of decision trees ("weak" learners) in the ensemble	$[100, 500]$
learning_rate	Shrinks the contribution of each successive decision tree in the ensemble	$[0.01, 0.10]$
subsample	Fraction of observations to be randomly sampled for each tree.	$[0.5, 0.8]$
colsample_bytree	Fraction of columns (features) to be randomly sampled for each tree.	$[0.5, 0.8]$
max_depth	Maximum tree depth for base learners	$[3, 7]$
mcw	Minimum sum of weights of all observations required in a child	$[1, 3]$
	SVM	
C	Penalty parameter for regularization	$[10^{-5}, 10^{5}]$
gamma	Kernel coefficient gamma for RBF kernel	$[10^{-5}, 10^{5}]$
	LSTM	
units	Number of neurons per hidden layer	$[50, 150]$
n_layers	Number of hidden layers	$\{1, 2, 3\}$
batch	Number of samples to be propagated	$\{8, 16, 32\}$
activation	Activation function to use	$\{ReLU\}$
optimizer	Weight optimizer	$\{RMSprop, Nadam\}$
epochs	Number of training epochs	$\{500\}$

For approaches that involve clustering, we use the k-means clustering algorithm. In order to calculate the Euclidean distance between pairs of prefixes, we map each trace prefix into a feature vector using the aggregation encoding (Section 3.3.2), based on the information from the control-flow of the prefix. This

[1] `http://scikit-learn.org/`

[2] `https://github.com/fchollet/keras/`

approach was also used in [31, 71]. K-means requires one to specify in advance the desired number of clusters k. We searched for the optimal value in the set $k \in \{2, 5, 10\}$. In the case of the index-based bucketing method, an optimal configuration was chosen for each prefix length separately.

For the adaptive flow analysis, we treat the minimum number of training samples N as an additional hyperparameter. The relationship between the size of the training set and the dimensionality of the problem has been studied extensively in the machine learning literature [40,96]. Common rules of thumb are that the number of training samples should be $50+8m$, $10m$ and m^2, where m is the dimensionality. [116] Accordingly, we define a set of thresholds $N_0 \in \{5m, 10m, 20m, m^2\}$. If $N > N_0$, then we fit a predictive model for a given activity or a gateway, otherwise, we use average historical values of cycle times and branching probabilities respectively in the flow analysis formulas. Similarly to the XGBoost hyperparameters, we choose the value of N_0 that minimizes the validation error for a given log.

A similar procedure is performed for methods that do not train a machine learning model. For the method in [130], we vary the type of abstraction – set, bag of sequence – to create a transition system. For the method in [100], we vary the stochastic Petri net properties: (i) the kind of transition distribution – normal, gaussian kernel or histogram and (ii) memory semantics – global preselection or race with memory. We select the parameters that yield the best performance on the validation set and use them for the test set.

6.3. Evaluation Results

In Section 6.3.1 we aim to answer **RQ2.1** while Section 6.3.2 is dedicated to **RQ2.2** and **RQ2.3**.

6.3.1. Evaluation of Flow Analysis-based Techniques

As mentioned in Section 5.2, flow analysis cannot readily deal with unstructured models. In addition, the discovery technique described in Section 5.3 aims to mine maximally structured models, meaning that when the business process is inherently unstructured, the resulting model will not be fully structured. Specifically, the technique sometimes produces models with overlapping loops which our current implementation is unable to deal with. As a result, for flow analysis, we will only use event logs for which we were able to produce structured BPMN models.

In Table 12, we report the prediction accuracy, averaged across all evaluated prefix lengths, together with its standard deviation. The averages are weighted by the relative frequency of prefixes with that prefix, such that longer prefixes get lower weights since not all traces reach that length.

In order to understand why for some of the event logs used mean flow analysis is more accurate than predictive flow analysis, we analyze the performance

Table 12: Weighted average MAE over all prefixes for flow analysis-based techniques.

Method	MAE in days (*mean* ± *std*)			
	bpic2012a	**bpic2012o**	**bpic2012w**	**credit**
predictive FA (single)	**6.677 ± 3.72**	**5.95 ± 2.832**	6.946 ± 1.057	**0.075 ± 0.039**
predictive FA (multiple)	6.838 ± 4.155	6.008 ± 2.643	6.823 ± 0.957	0.087 ± 0.043
mean FA	7.62 ± 3.528	6.243 ± 3.237	**6.197 ± 0.54**	0.339 ± 0.187
adaptive FA	**6.677 ± 3.72**	**5.95 ± 2.832**	6.921 ± 1.057	0.078 ± 0.035
	hospital	**invoice**	**traffic fines**	**helpdesk**
predictive FA (single)	51.689 ± 14.945	**1.169 ± 0.06**	**223.506 ± 74.58**	5.13 ± 2.092
predictive FA (multiple)	72.104 ± 50.779	1.228 ± 0.09	225.467 ± 80.693	**5.046 ± 2.998**
mean FA	**42.58 ± 8.622**	2.012 ± 0.249	229.337 ± 82.628	5.233 ± 2.022
adaptive FA	**42.58 ± 8.622**	1.171 ± 0.061	223.513 ± 78.87	5.233 ± 2.022

Table 13: MAE of cycle time predictions of individual activities and their actual mean cycle times (in days).

Activity	MAE		Mean cycle
	Predictive FA	Mean FA	time
Invoice (Top 6 longest activities)			
Invoice_accounting	**2.61**	3.53	3.11
Check_whether_total_approval	**0.771**	0.962	0.893
Manual_identification_CC	**0.109**	0.132	0.097
Compare_of_sums	**0.081**	0.123	0.060
Get_lowest_approval_level	**0.0117**	0.0271	0.0091
Status_change_to_Accounted	**0.0008**	0.0031	0.0005
Hospital			
FIN	**8.52**	58.77	70.08
BILLED	40.36	**36.58**	36.43
DELETE	18.37	**16.76**	8.20
CHANGE_DIAGN	**3.18**	11.91	7.57
CODE_NOK	**14.58**	22.00	6.27
CODE_OK	2.57	**2.38**	5.14

of these two approaches at the level of individual activities. Specifically, for each activity in the Hospital and the Invoice log, we measure the performance of regressors trained to predict its cycle time and compare it with a constant regressor used in the mean flow analysis. In Table 13, for each activity, we report average MAE of cycle time predictions across all test prefixes. In addition, we report the actual average cycle time values of each activity based on the test set.

As we can see in Table 13, in the Invoice log, prediction-based cycle times are more accurate than those based on historical averages for the six longest activities which make up the largest portion of the remaining cycle time. Hence, the predictive flow analysis approach provides a better estimation of the overall

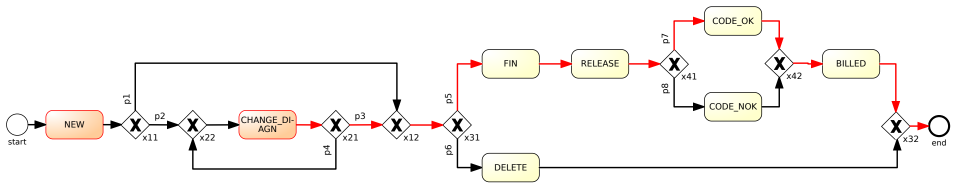


Figure 25: A process model of the Hospital log. Current marking of $hd^2(SWKD)$ and its predicted future path are highlighted.

remaining time than the mean flow analysis approach. In contrast, in the Hospital log, for three of the six longest activities, we get a more accurate estimate using the mean flow analysis. Thus, the performance of the predictive flow analysis approach hinges on the ability to accurately predict cycle times of key activities. In this way, the adaptive approach provides a "golden mean" between the predictive and the mean approaches, while retaining explainability of the predictions.

To illustrate the explainability of the proposed techniques, let us consider a partial trace $hd^2(SWKD) = \langle NEW, CHANGE\_DIAGN \rangle$ of an ongoing case SWKD originating from the Hospital event log. At that stage, the case is predicted to exceed the median case duration for the considered process. Our family of flow analysis-based approaches allows users not only to make predictions but also to explain them. Let us consider the output of the adaptive flow analysis approach trained with a single predictive model. Having replayed the trace on the process model (Figure 25), we obtain the following formula for its remaining time:

$$\begin{aligned} T_{hd^2(SWKD)} &= 0 + p2 * (0 + p4 * (CHANGE\_DIAGN)/(1-p4)) + p5 * \\ &* (FIN + RELEASE + p7 * CODE\_OK + p8 * CODE\_NOK + BILLED) + \\ &+ p6 * DELETE \end{aligned} \tag{6.2}$$

Table 14 lists the predicted values of cycle times and branching probabilities for the trace in question, as well as their historical averages.

Comparing the obtained values with their historical averages (expectations), we notice that the activity *FIN* is going to take 10 days longer while *BILLED* is about to take 20 days longer. Therefore, by pinpointing *FIN* and *BILLED* as potential bottleneck activities, operational managers may gain more leverage over the process execution and be able to steer this towards the target performance values.

Summing up, the experiments suggest that flow analysis-based approaches provide relatively accurate estimations of the remaining cycle time across most logs. Another important observation is that mean flow analysis sometimes outperforms predictive flow analysis. This is due to the lack of data attributes in the event logs that would be able to accurately explain the variation in the cycle times of individual activities and branching probabilities of each conditional flow. Nevertheless, the accuracy of the adaptive approach in most logs corresponds to the

Table 14: Predicted and average values of cycle times and branching probabilities for $hd^2(SWKD)$.

Variable	Predicted	Average
p2	0.9988	0.6268
p4	0.0543	0.0305
p5	0.9609	0.9625
p6	0.0391	0.0375
p7	0.9988	0.9766
p8	0.0012	0.0234
CHANGE_DIAGN	1.08	7.57
FIN	80.17	70.08
RELEASE	1.11	1.45
CODE_OK	4.51	5.14
CODE_NOK	7.81	6.27
BILLED	56.21	36.43
DELETE	13.89	8.20

best accuracy achieved by the predictive and mean methods. In this way, adaptive flow analysis can be regarded as a safeguard against instability in predictions caused by the lack of data (attributes). For the rest of the evaluation, adaptive flow analysis will be used as a representative process model driven technique.

Execution Times. The execution time of the proposed flow analysis approaches is composed of the execution times of the following components: (i) training the predictive models; (ii) replaying the partial traces on the process model (finding an alignment) and deriving the formulas; (iii) applying the models to predict the cycle times and branching probabilities and calculating the overall remaining time. For real-time prediction, it is crucial to output the results faster than the mean case arrival rate. Thus, we also measured the average runtime overhead of our approach. All experiments were conducted on a laptop with a 2.4GHz Intel Core i7 processor and 16GB of RAM.

The training phase includes the time for constructing the prefix log, encoding the prefix traces and fitting the predictive models. It is performed offline and takes between 1 minute (*BPIC'12 O*) and 80 minutes (*Hospital*), depending on the size of the log and the number of models to train, i.e. the number of distinct decision points and activities. Replaying a single test trace takes less than a second. Finally, making the predictions takes between 50 milliseconds and 3 seconds per trace, depending on the length of the trace and the number and complexity of the predictive models. This shows that flow analysis-based prediction approaches perform within reasonable bounds for most online applications.

6.3.2. Evaluation of the Proposed Techniques and State-of-the-art Techniques

Table 15 reports the prediction accuracy, averaged across all evaluated prefix lengths, together with its standard deviation. The averages are weighted by the relative frequency of prefixes with that prefix (i.e. longer prefixes get lower weights since not all traces reach that length). In our experiences, we set an execution cap of 6 hours for each training configuration, so if some method did not finish within that time, the corresponding cell in the table is empty.

Overall, we can see that in 13 out of 17 datasets, LSTM-based networks trained in a single-task learning setting achieve the best accuracy, while multi-task LSTM, flow-analysis and index-based encoding with no bucketing and prefix-length bucketing achieve the best results in one dataset each. On the other side of the spectrum, transition systems [130] and stochastic Petri nets [99] usually provide the least accurate predictions among the surveyed methods.

Figure 26 presents the prediction accuracy in terms of MAE, evaluated over different prefix lengths. To keep the plot readable, we exclude some of the worst performing methods according to Table 15. Each evaluation point includes prefixes of exactly the specified length. In other words, traces that are shorter than the required prefix are left out of the calculation. Therefore, the number of cases used for evaluation is monotonically decreasing when increasing the prefix length.

In most of the datasets, we see that the MAE decreases as cases progress. It is natural that the prediction task becomes trivial when cases are close to completion. However, for some datasets, the predictions become less accurate as the prefix length increases. This phenomenon is caused by the fact that these datasets contain some short traces for which it appears to be easy to predict the outcome. These short traces are not included in the later evaluation points, as they have already finished by that time. Therefore, we are left with longer traces only, which appear to be more challenging for the predictor, hence decreasing the total accuracy on larger prefix lengths. However, different techniques behave differently wrt. earliness. For example, single-task LSTMs generally provide the most accurate predictions early on, but as the case progresses, other techniques may outperform LSTMs.

As a simple bulk measure to compare the performance of the evaluated techniques, we plot their mean rankings achieved across all datasets in Figure 27. Ties were resolved by assigning every tied element to the lowest rank. The rankings illustrate that single-task LSTMs consistently outperform other machine-learning baselines in terms of accuracy (measured by MAE), while stochastic Petri nets and transition systems are usually the least accurate methods.

To complement the above observations, we also compare error values *aggregated* across all datasets. These values need to be normalized, e.g. by the mean case duration, so that they are on a comparable scale. In order to do that, for each log, we divide the average MAE values and their standard deviations across all prefixes reported in Table 15 by the mean case duration for that log reported in Table 10. The results for each technique are illustrated with the boxplots in Figure 28, where each point represents the results for one of the 17 datasets. We can see that LSTM-based techniques have an average error of 40% of the mean case duration across all datasets. In contrast, transition systems on average incur a 59%

Table 15: Weighted average MAE over all prefixes.

Method	MAE in days (*mean* ± *std*)					
	bpic2011	**bpic2012a**	**bpic2012o**	**bpic2012w**	**bpic2015_1**	**bpic2015_2**
TS [130]	236.088 ± 9.98	8.363 ± 4.797	6.766 ± 2.909	7.505 ± 1.036	56.498 ± 8.341	118.293 ± 16.819
LSTM STL [84]	**160.27 ± 24.325**	**3.772 ± 3.075**	6.418 ± 2.768	**6.344 ± 0.994**	39.457 ± 5.708	**61.62 ± 2.061**
LSTM MTL [119]	–	30.06 ± 16.08	26.357 ± 13.41	23.176 ± 8.83	44.342 ± 5.072	104.44 ± 7.74
SPN [100]	–	7.693 ± 1.889	6.489 ± 2.562	8.538 ± 0.772	66.509 ± 17.131	81.114 ± 8.033
FA [137]	–	6.677 ± 3.72	**5.95 ± 2.832**	6.946 ± 1.057	–	–
cluster_agg	211.446 ± 5.941	6.739 ± 4.146	7.656 ± 3.534	7.18 ± 0.953	40.705 ± 1.824	68.185 ± 2.649
cluster_index	225.132 ± 5.212	6.743 ± 4.354	7.439 ± 3.436	7.074 ± 1.254	38.092 ± 2.988	66.957 ± 3.436
cluster_last	216.75 ± 4.338	6.728 ± 4.358	7.435 ± 3.412	7.061 ± 1.019	38.388 ± 3.478	62.781 ± 2.347
prefix_agg	211.401 ± 14.257	6.75 ± 4.452	7.79 ± 3.636	7.26 ± 0.935	46.765 ± 23.581	71.21 ± 8.893
prefix_index	227.288 ± 7.404	6.753 ± 4.45	7.472 ± 3.356	7.155 ± 0.942	37.525 ± 2.746	66.883 ± 3.756
prefix_last	219.781 ± 12.664	6.76 ± 4.429	7.441 ± 3.399	7.139 ± 0.851	37.975 ± 5.903	64.708 ± 5.749
noBucket_agg	200.466 ± 11.786	6.746 ± 3.899	7.744 ± 3.62	7.082 ± 1.02	35.962 ± 3.744	67.914 ± 2.467
noBucket_index	217.139 ± 13.991	6.768 ± 4.249	7.548 ± 3.367	6.982 ± 1.34	**35.451 ± 2.499**	65.505 ± 3.442
noBucket_last	208.711 ± 2.001	6.752 ± 4.15	7.51 ± 3.415	7.021 ± 1.099	37.442 ± 3.607	64.11 ± 2.332
state_agg	271.801 ± 14.676	6.756 ± 4.45	7.656 ± 3.534	7.465 ± 0.622	42.949 ± 2.725	68.768 ± 4.094
state_index	–	6.757 ± 4.453	7.439 ± 3.436	7.51 ± 0.585	–	–
state_last	271.595 ± 14.449	6.746 ± 4.446	7.435 ± 3.412	7.539 ± 0.554	42.946 ± 2.691	68.296 ± 3.762
	bpic2015_3	**bpic2015_4**	**bpic2015_5**	**bpic2017**	**credit**	**helpdesk**
TS [130]	26.412 ± 8.082	61.63 ± 5.413	67.699 ± 7.531	8.278 ± 2.468	0.382 ± 0.194	6.124 ± 2.6
LSTM STL [84]	**19.682 ± 2.646**	48.902 ± 1.527	52.405 ± 3.819	**7.15 ± 2.635**	**0.062 ± 0.021**	**3.458 ± 2.542**
LSTM MTL [119]	20.639 ± 5.116	**45.347 ± 1.615**	89.782 ± 73.728	64.934 ± 38.008	0.321 ± 0.183	4.671 ± 2.001
SPN [100]	26.757 ± 10.378	51.202 ± 5.889	–	10.731 ± 0.369	0.385 ± 0.197	6.646 ± 1.225
FA [137]	–	–	–	–	0.075 ± 0.039	5.13 ± 2.092
cluster_agg	23.087 ± 3.226	51.555 ± 2.363	45.825 ± 3.028	7.479 ± 2.282	0.077 ± 0.036	4.179 ± 3.074
cluster_index	24.497 ± 1.887	56.113 ± 6.411	44.587 ± 4.378	–	0.075 ± 0.035	4.178 ± 3.043
cluster_last	22.544 ± 1.656	51.451 ± 4.189	46.433 ± 4.085	7.457 ± 2.359	0.076 ± 0.035	4.152 ± 3.053
prefix_agg	24.152 ± 2.785	53.568 ± 6.413	46.396 ± 2.466	7.525 ± 2.306	0.075 ± 0.034	4.175 ± 3.045
prefix_index	21.861 ± 3.292	50.452 ± 4.605	**44.29 ± 3.669**	7.421 ± 2.36	0.076 ± 0.035	4.262 ± 3.105
prefix_last	23.574 ± 3.778	53.053 ± 5.665	46.639 ± 3.718	7.482 ± 2.325	0.076 ± 0.034	4.242 ± 3.082
noBucket_agg	24.453 ± 3.577	54.89 ± 1.894	49.203 ± 1.833	7.437 ± 2.381	0.083 ± 0.033	4.252 ± 2.869
noBucket_index	23.025 ± 1.587	52.282 ± 1.182	50.153 ± 1.097	–	0.078 ± 0.034	4.253 ± 2.722
noBucket_last	25.15 ± 1.271	56.818 ± 1.729	49.027 ± 1.954	7.525 ± 2.244	0.082 ± 0.035	4.224 ± 2.814
state_agg	28.427 ± 9.844	49.318 ± 2.699	49.873 ± 2.658	–	0.077 ± 0.036	4.206 ± 3.092
state_index	–	–	–	–	0.079 ± 0.036	4.155 ± 3.023
state_last	27.826 ± 8.28	49.038 ± 2.498	49.556 ± 2.575	7.521 ± 2.341	0.078 ± 0.036	4.111 ± 3.026
	hospital	**invoice**	**production**	**sepsis**	**traffic fines**	
TS [130]	46.491 ± 21.344	1.715 ± 0.891	14.474 ± 7.199	0.019 ± 0.019	190.949 ± 15.447	
LSTM STL [84]	**36.258 ± 23.87**	**0.738 ± 0.266**	**8.76639 ± 1.12300**	**0.009 ± 0.006**	**178.738 ± 89.019**	
LSTM MTL [119]	46.098 ± 16.738	7.54 ± 3.22	125.13 ± 23.484	0.009 ± 0.005	348.236 ± 156.885	
SPN [100]	71.377 ± 29.082	1.646 ± 0.601	29.156 ± 3.006	0.02 ± 0.032	193.807 ± 69.796	
FA [137]	51.689 ± 14.945	1.224 ± 0.51	–	–	223.808 ± 14.859	
cluster_agg	42.934 ± 26.136	1.048 ± 0.355	17.695 ± 4.335	0.011 ± 0.006	210.322 ± 98.516	
cluster_index	–	1.052 ± 0.404	16.055 ± 2.378	0.011 ± 0.006	209.139 ± 98.417	
cluster_last	48.589 ± 26.708	1.085 ± 0.389	18.319 ± 5.903	0.011 ± 0.006	208.599 ± 99.549	
prefix_agg	43.06 ± 25.884	1.03 ± 0.359	17.202 ± 3.934	0.011 ± 0.006	212.614 ± 99.484	
prefix_index	41.698 ± 25.944	1.041 ± 0.365	17.850 ± 3.589	0.011 ± 0.006	209.085 ± 99.708	
prefix_last	48.528 ± 26.714	1.057 ± 0.369	18.239 ± 5.569	0.011 ± 0.006	209.304 ± 102.027	
noBucket_agg	43.483 ± 25	1.186 ± 0.568	17.022 ± 3.572	0.012 ± 0.005	211.017 ± 93.198	
noBucket_index	–	1.053 ± 0.374	17.357 ± 2.437	0.012 ± 0.005	208.879 ± 92.25	
noBucket_last	50.496 ± 23.961	1.144 ± 0.466	17.316 ± 6.033	0.012 ± 0.003	204.758 ± 93.399	
state_agg	43.835 ± 25.984	1.044 ± 0.341	19.538 ± 5.047	0.011 ± 0.006	211.439 ± 98.351	
state_index	41.095 ± 26.499	1.051 ± 0.371	15.623 ± 3.558	0.011 ± 0.006	210.408 ± 99.276	
state_last	48.902 ± 27.001	1.086 ± 0.385	19.154 ± 6.889	0.011 ± 0.006	209.206 ± 100.632	

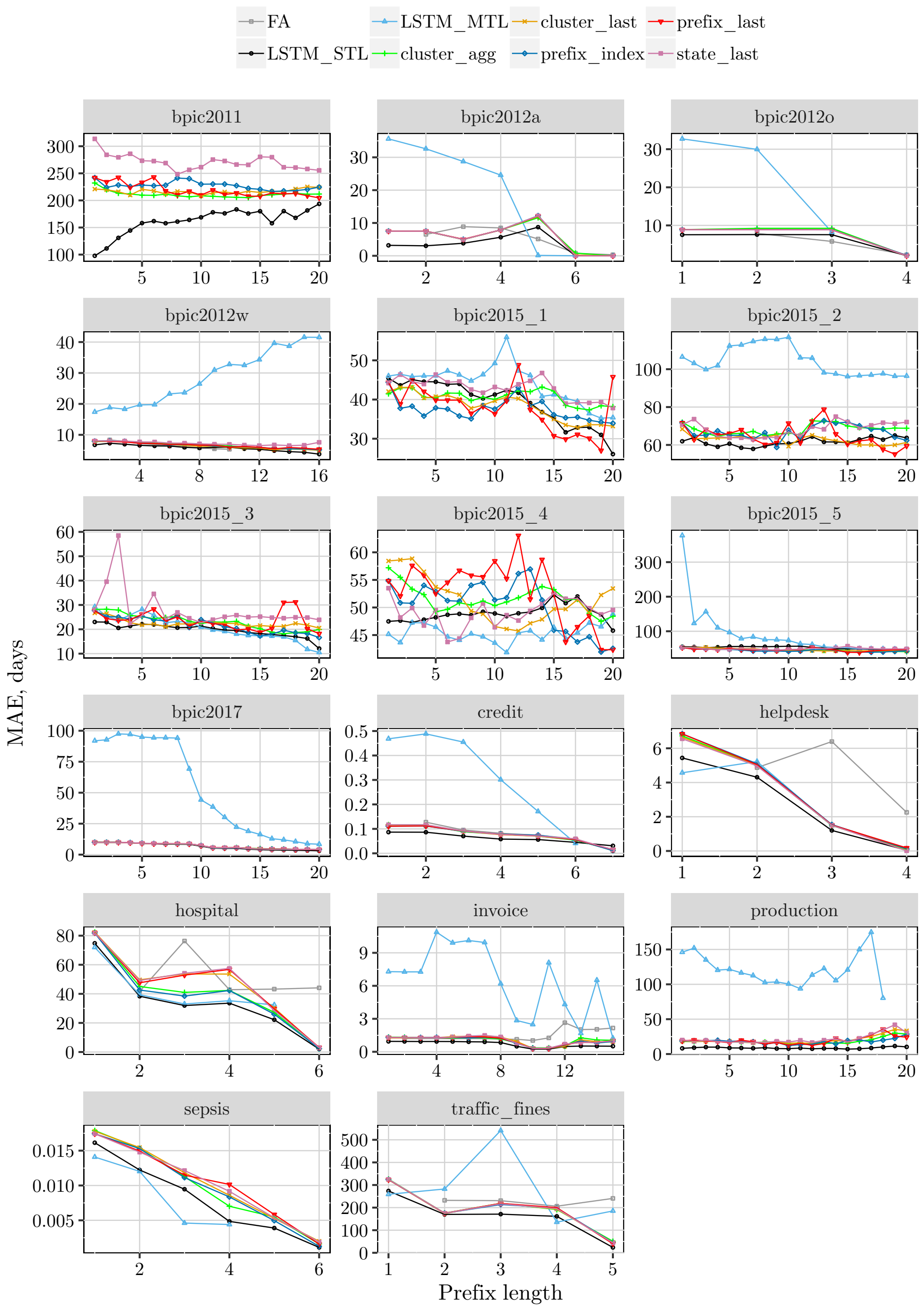


Figure 26: Prediction accuracy (measured in terms of MAE) across different prefix lengths

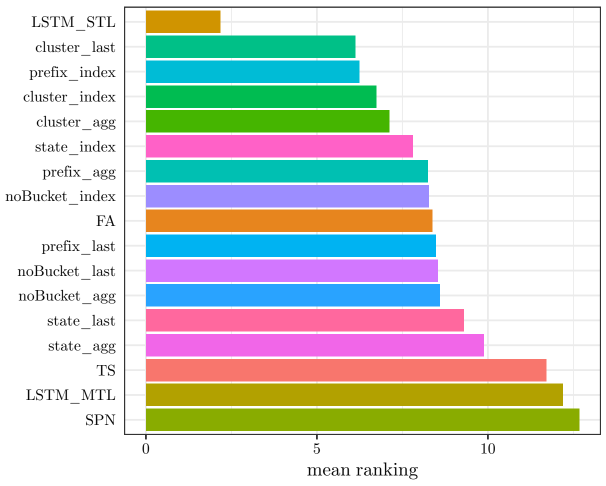


Figure 27: Average ranking of the evaluated methods over all datasets.

error. Importantly, for LSTMs the accuracy varies between 0.07 and 0.56, while index-based encoding with prefix length bucketing, the method that on average achieves the second most accurate results, is more volatile and varies between 0.08 and 0.90 of the mean case duration.

Another important observation is related to the temporal stability of the predictions. In general, methods that provide higher accuracy also have lower volatility of error across case lifetime (Figure 28b). In other words, the difference between successive predictions obtained from these methods is lower, as evidenced by lower standard deviation of the error metrics.

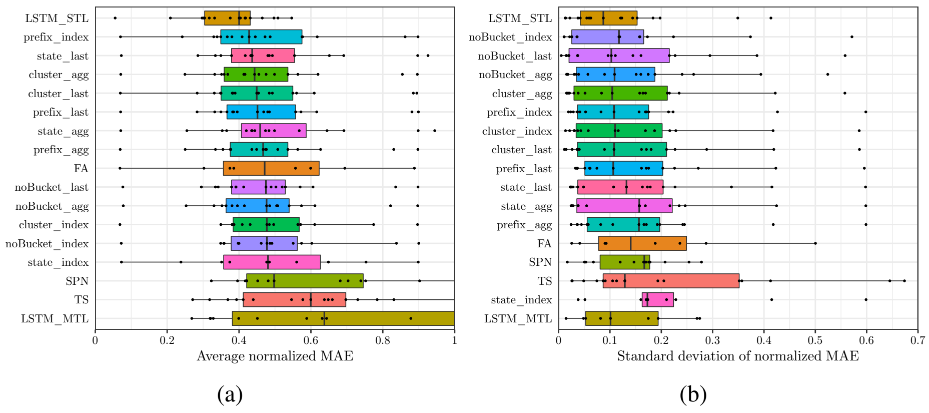


Figure 28: Average normalized MAE values (a) and their standard deviation (b) across case lifetime. XGBoost is used as the main predictor

In order to assess the statistical significance of the observed differences in methods' performance across all datasets, we use the non-parametric Friedman test. The complete set of experiments indicate statistically significant differences according to this test ($p = 4.642 \times 10^{-8}$). Following the procedure suggested in the recent work on evaluating machine learning algorithms [30], in order to find which methods in particular differ from each other, we use the Nemenyi post-hoc test that compares all methods to each other.

Table 16: Post-hoc Nemenyi test of methods' rankings across all datasets.

	cluster_agg	cluster_last	LSTM_MTL	LSTM_STL	noBucket_agg	noBucket_last	prefix_agg	prefix_index	prefix_last	state_last
cluster_last	0.996									
LSTM_MTL	0.169	**0.025**								
LSTM_STL	0.098	0.421	**0**							
noBucket_agg	0.998	0.894	0.740	**0.005**						
noBucket_last	0.995	0.847	0.804	**0.003**	1					
prefix_agg	0.999	0.931	0.669	**0.007**	1	1				
prefix_index	0.999	1	**0.016**	0.515	0.833	0.773	0.883			
prefix_last	1	0.958	0.593	**0.011**	1	1	1	0.923		
state_last	0.958	0.631	0.946	**0.001**	1	1	1	0.535	1	
TS	0.169	**0.025**	1	**0**	0.740	0.804	0.669	**0.016**	0.593	0.946

Table 16 lists p-values of a pairwise post-hoc analysis. Since the test requires complete information for all pairwise comparisons, we included only 10 methods for which we have results on all 17 datasets. For most pairs, the null hypothesis that their performance is similar cannot be rejected. However, the test underlines the impressive performance of LSTM, which significantly outperforms most of the other methods at the $p < 0.05$ level.

While on average most combinations of bucketing and encoding methods provide more or less similar levels of accuracy, we can observe differences for individual datasets. For example, in the hospital dataset, it is clear that clustering with aggregation encoding is better than with clustering with last state encoding. Arguably, aggregating knowledge from all events performed so far provides much more signal than using raw attributes of the latest event.

In order to explain differences in performance of various bucketing and encoding combinations, we try to correlate the characteristics of event logs (Table 10) with the type of bucketing/encoding that achieves the best accuracy on that log. One can notice that if cases in the log are very heterogeneous in terms of case length, i.e. the coefficient of variation of case length is high enough, it is more beneficial to assign all traces to the same bucket. This can be observed in *bpic2012w* and *bpic2011* event logs, where standard deviation of case length is close to or exceeds mean case length. Furthermore, if cases in the log are short (e.g. in *helpdesk*, *traffic_fines*, *bpic2012a*, *bpic2012o*) or very distinctive from each other (e.g. in *bpic2015_2*), last state encoding tends to capture the most signal. Notably, in the aforementioned logs, the index-based encoding, although lossless, is not optimal. This suggests that in these datasets, combining the knowledge from all events performed so far provides more signal for remaining time prediction than the order of events. However, unlike LSTMs, standard classifiers such as XGBoost and SVM, are not able to learn such higher-level features, which is why in some situations even the simple aggregations provide more accurate results than index-based encoding.

So far we reported the results with a single learning algorithm (XGBoost). In order to avoid bias associated with the choice of a particular algorithm, we decided to repeat the experiments replacing XGBoost with SVM where applicable. In the benchmark [87], it was found that SVMs in general provide the third most accurate results, on a set of 165 classification problems, after gradient tree boosting and random forest. We decided not to use random forest, as it essentially belongs to the same family of ensemble algorithms as XGBoost.

The results for each technique are illustrated with the boxplots in Figure 29, where each point represents the results for one of the 17 datasets. One can see that there are no significant differences in the performance as compared to XGBoost (Figure 29). In other words, with SVMs, we obtained qualitatively the same results, relative to the baselines. Thus, in principle, using a different algorithm does not invalidate the results. That said, we acknowledge that the goodness of fit, as in any machine learning problem, depends on the particular classifier/regressor algorithm employed. Hence, it is important to test multiple algorithms for a given dataset and to apply hyperparameter tuning, in order to choose the most adequate algorithm with the best configuration.

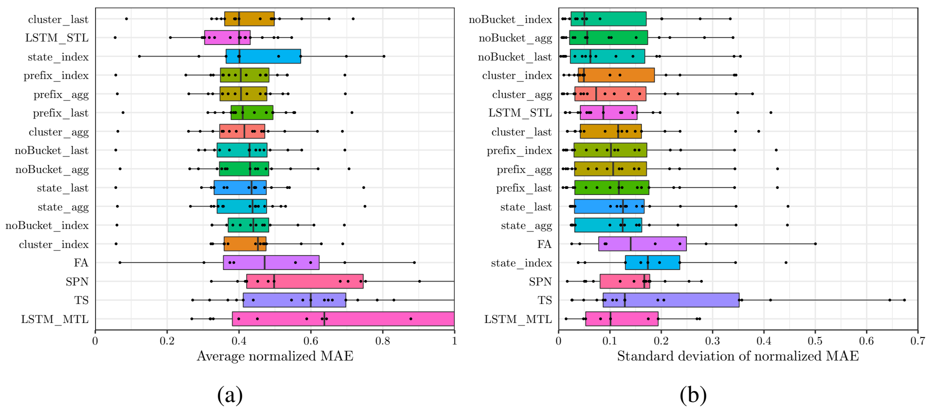


Figure 29: Average normalized MAE values (a) and their standard deviation (b) across case lifetime. SVM is used as the main predictor

6.4. Threats to Validity

One of the threats to the validity [110] of our evaluation is related to the comprehensiveness of the conducted experiments. In particular, only two machine learning algorithm (XGBoost and SVM) and one clustering method (k-means) were tested over all relevant methods. It is possible that there exists, for example, a combination of an untested clustering technique and a predictor that outperforms the settings used in this study. Furthermore, the generalizability of the findings, i.e. external validity, is to some extent limited by the fact that the experiments were performed only on 17 event logs. Although these are all real-life event logs from different application fields that exhibit different characteristics, it may possible that the results would be different using other datasets or different log preprocessing techniques for the same datasets. In order to mitigate these threats, we built an open-source software framework which allows the full replication of the experiments, and made this tool publicly available. Moreover, additional datasets, as well as new sequence classification and encoding methods can be plugged in, so that the framework can be used for future experiments.

Another threat to validity relates to the potential selection bias in the choice of the baselines. To minimize this, we described our literature review procedure (Chapter 3) on a level of detail that is sufficient to replicate the search. However, in time the search and ranking algorithms of the used academic database (Google Scholar) might be updated and return different results. Another potential source of bias is the subjectivity when applying inclusion and exclusion criteria, as well as when determining the primary and subsumed studies. In order to mitigate this issue, all the included papers were collected in a publicly available spreadsheet, together with decisions and reasons about excluding them from the study.

Finally, we explained the observed difference in the performance of the methods in terms of the experiment settings (type of bucketing, type of encoding, predictor, etc.). However, unknown factors may have had an influence on the observed results and therefore put limitations on the internal validity of the study.

6.5. Summary

We conducted a comparative evaluation of the two techniques proposed to make more explainable predictions for temporal process performance measures. These techniques were evaluated with each other and with various state-of-the-art baselines identified in Chapter 3. The results of the evaluation suggest several conclusions.

Firstly, in response to **RQ2.1**, adaptive flow analysis provides a safeguard against instability in predictions caused by the lack of data attributes when predicting the properties of activities and gateways that are rarely taken in the training set. In that regard, it provides a balance between predictive flow analysis and mean flow analysis.

Secondly, referring to **RQ2.2**, we demonstrate that LSTMs trained in a single-task setting typically outperform their multi-task counterparts, both in terms of accuracy and earliness, at the expense of providing non-explainable predictions. This can result from the propagation of errors for incorrectly predicted attributes at earlier stages, for long sequences. Another explanation is that a single-task makes use of all data attributes in the prefix, not only activity names and timestamps.

In the wider context, comparing all the surveyed techniques, we note the inherent trade-off between the prediction accuracy and explainability of the underlying model. Namely, the most accurate predictions are obtained via black-box models, while explainable models, such as those proposed in this research, are generally less accurate. This trade-off should be considered by business process analysis when choosing a suitable prediction method.

7. NIRDIZATI: AN INTEGRATED PREDICTIVE PROCESS MONITORING PLATFORM

In order to demonstrate the practical application of the research conducted in this research project, we have designed and evaluated a prototype system for predictive business process monitoring, namely Nirdizati. Nirdizati implements the techniques for predicting various business process targets, such as the remaining processing time or the next events until case completion, the case outcome, or the violation of compliance rules or internal policies.

This chapter is structured as follows. Section 7.1 provides an overview of the developed prototype. Section 7.2 describes the Apromore platform where our prototype has been integrated to. Section 7.3 describes the predictive model training functionality, while Section 7.4 discusses the runtime component and the dashboard. Section 7.5 provides details on how the designed solution has been validated externally and the lessons learned during the validation. Finally, Section 7.6 provides a summary of the chapter.

7.1. Introduction

A dashboard is a visual display of data used to monitor conditions and/or facilitate understanding [142]. More specifically, a process performance dashboard is a graphical representation of one or more performance measures or other characteristics of a business process [32].

Dumas et al. [32] identify three types of process performance dashboards, depending on their purpose and targeted users, namely *operational*, *tactical*, and *strategical* dashboards. Operational dashboards are aimed at process workers and operational managers and focus on the performance of running or recently completed cases in a way that assists process workers and their managers in planning their short-term work. Figure 2 provides an example of an operational dashboard that displays the number of running cases classified by their types. By contrast, tactical dashboards are designed with the goal to give an overview of the process performance over a relatively long period of time. They can be used to uncover performance variations and long-term bottlenecks, as well as their possible causes. Tactical dashboards are aimed at process owners and functional managers. Finally, strategic dashboards put emphasis on linking process performance measures to strategic objectives and are aimed at executive managers.

This chapter seeks to answer the research question **RQ3**, namely, "How to embed existing techniques for predictive process monitoring into dashboard-based monitoring systems?" To this end, we design a prototype of a predictive monitoring platform that generates predictions about ongoing cases and displays them in an operational dashboard, along with a range of descriptive statistics about completed and ongoing cases. The dashboard offers multiple visualization options,

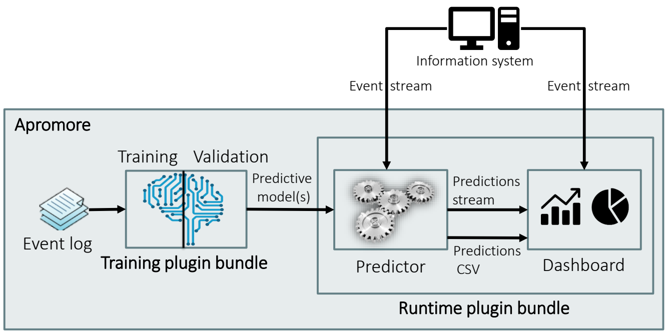


Figure 30: High-level architecture of the predictive monitoring functionality of Apromore.

such as tables, histograms and pie charts. The predictions can also be exported into a text file to be visualized in third-parties business intelligence tools, e.g. for periodic reporting. Based on these predictions, operations managers may identify potential issues early on, and take remedial actions in a timely fashion, e.g. reallocating resources from one case onto another to avoid that the case runs overtime.

Structurally, Nirdizati consists of the two core components, namely *Training* and *Runtime* (Figure 30). Both components have been integrated into the Web-based process analytics platform Apromore[1] as two bundles (i.e. sets) of plugins. The Training plugin bundle takes as input a business process event log stored in the Apromore repository, and produces one or more predictive models, which can then be deployed to the runtime predictive monitoring environment. Once a model is deployed, the Runtime plugin bundle listens to a stream of events coming from an information system supporting the process, or produced by replaying an event log stored in the repository, and creates a stream of predictions. These predictions can then be visualized in a Web dashboard or exported into a text file to be used within third-party business intelligence tools.

7.2. Apromore Platform

Apromore is a Web-based advanced process analytics platform, developed by the business process management (BPM) community under an open-source initiative. Apromore was originally conceived as an advanced process model repository. However, today it offers a wide range of features which go beyond those for managing large process model collections and include a variety of state-of-the-art process mining techniques. These are techniques for the automated discovery of BPMN models, for the conformance checking of BPMN models against event logs, the replaying of event logs on top of BPMN models, the detection and characterization of process drifts from event logs, the visual analysis of process performance, and many others.

[1]`http://apromore.org`

All these features are exposed through a Web portal and organized according to the phases of the BPM lifecycle: discovery, analysis, redesign, implementation and monitoring [32]. These features can also be accessed as external Web services by third-party BPM software environments, such as ProM (for process mining) and WoPeD (for process modeling and verification).

From a technology viewpoint, Apromore relies on four core technologies: Spring, ZK, OSGi and Eclipse Virgo. Spring provides a simplified management of Java-based enterprise applications through the use of Java annotations and XML configurations. ZK is an AJAX framework used for Apromore's main Web interface (the *Portal*). OSGi provides a flexible framework for managing component dependencies through plugin bundles. Finally, Eclipse Virgo is a Web server based on the OSGi component model.

To equip Apromore with predictive business process monitoring capabilities, we have wrapped the two core components of Nirdizati into two OSGi plugin bundles for Apromore: Training and Runtime. Each bundle is a set of OSGi plugins which encapsulate the logic or the user interface (UI) of the various functions offered by Nirdizati. For example, the runtime predictor is a logic plugin, while the runtime dashboard is a portal plugin (UI). These two bundles are accessible from the Monitoring menu of the Apromore Portal (see Figure 31). One can select an event log stored in the repository, and use it to train, tune and test a variety of predictive models, by launching the training plugin bundle. Next, the runtime bundle can be used to stream an event log from the repository, or hook into a live external stream, to generate predictions as process cases unfold.

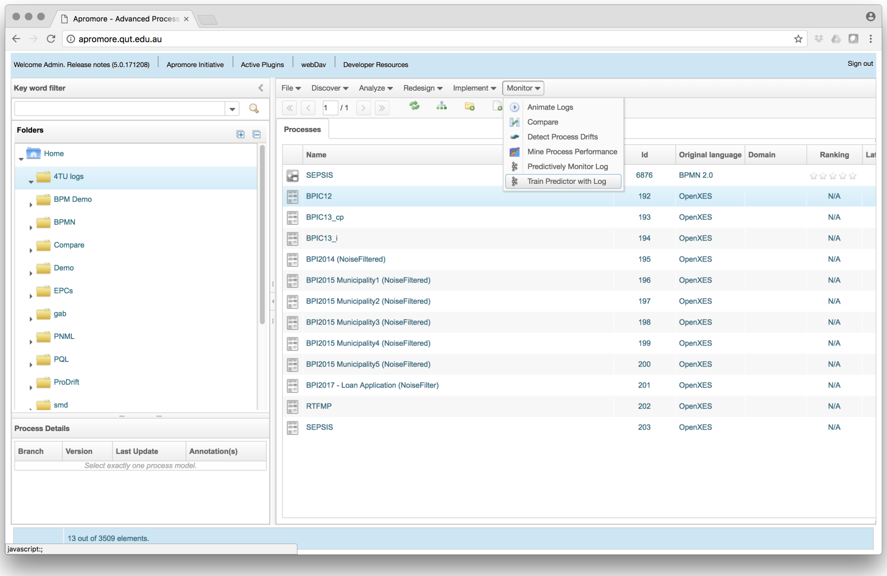
In the next sections, we introduce a working example and use this to describe the functionality of the training and runtime plugins in detail.

7.3. Training Plugin Bundle

The Training plugin bundle provides several algorithms for generating predictive models suitable for different types of predictions. Specifically, it is able to build models for predicting remaining time, the next activity to be performed, whether a case will exceed a specified duration threshold, as well as various static case attributes, for example, the total cost of the order. To this aim, the Training bundle involves two phases: a training and a validation phase. In the former, one or more predictive models are fitted; in the latter, their suitability to the specific dataset is evaluated, so as to support the user in selecting the predictive model that ensures the best results.

The Training bundle is composed of a front-end application (Figure 32), which allows users to select the prediction methods and to assess the goodness-of-fit of the built models, and a back-end application for the actual training and validation. From the data flow perspective, the back-end application performs several tasks shown in Figure 33.

Firstly, when a user uploads their log, the tool extracts and categorizes data



Name	Id	Original language	Domain	Ranking
SEPSIS	6876	BPMN 2.0		
BPIC12	192	OpenXES		N/A
BPIC13_cp	193	OpenXES		N/A
BPIC13_i	194	OpenXES		N/A
BPI2014 (NoiseFiltered)	195	OpenXES		N/A
BPI2015 Municipality1 (NoiseFiltered)	196	OpenXES		N/A
BPI2015 Municipality2 (NoiseFiltered)	197	OpenXES		N/A
BPI2015 Municipality3 (NoiseFiltered)	198	OpenXES		N/A
BPI2015 Municipality4 (NoiseFiltered)	199	OpenXES		N/A
BPI2015 Municipality5 (NoiseFiltered)	200	OpenXES		N/A
BPI2017 - Loan Application (NoiseFilter)	201	OpenXES		N/A
RTFMP	202	OpenXES		N/A
SEPSIS	203	OpenXES		N/A

Process Details

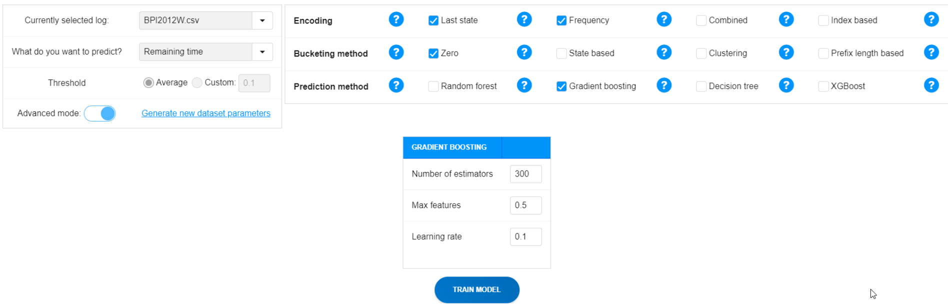
Branch	Version	Last Update	Annotation(s)

Select exactly one process model.

13 out of 3509 elements.

javascript:;

Figure 31: Apromore's Portal with predictive monitoring functionality highlighted.



Currently selected log:	BPI2012W.csv
What do you want to predict?	Remaining time
Threshold	[✓ Average] Custom: 0.1
Advanced mode:	[✓] Generate new dataset parameters

Encoding	[✓ Last state]	[✓ Frequency]	Combined	Index based
Bucketing method	[✓ Zero]	State based	Clustering	Prefix length based
Prediction method	Random forest	[✓ Gradient boosting]	Decision tree	XGBoost

GRADIENT BOOSTING

Number of estimators	300
Max features	0.5
Learning rate	0.1

TRAIN MODEL

Figure 32: Training configuration screen.

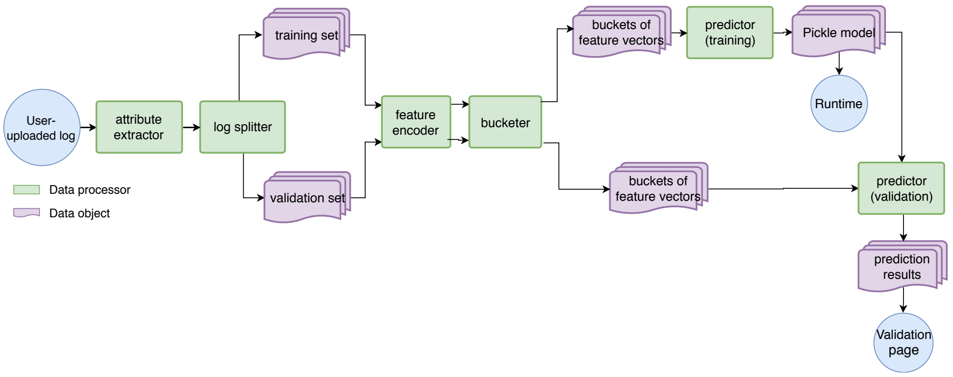


Figure 33: High-level data flow diagram of the Training plugin bundle.

attributes of the log into static case attributes and dynamic event attributes. On the other hand, each attribute needs to be designated as either numeric or categorical. These procedures are performed automatically upon the log uploading, using a set of pre-defined rules. Nevertheless, the user is given an option to override the automatic attribute definitions. Proper attribute categorization ensures best training data quality. The resulting definitions are saved in a configuration file in a JSON format.

Secondly, the log is internally split into training and validation set in a 80-20 proportion. The former is used to train the model, while the latter is used to evaluate the predictive power of the model. Next, all traces of a business process need to be represented as fixed-size feature vectors in order to train a predictive model. To this end, several encoding techniques were proposed in [66] and further refined in [120], out of which we support four, namely last state encoding, frequency (aggregation) encoding, combined encoding and lossless index-based encoding.

While some of existing predictive process monitoring approaches train a single classifier on the whole event log, others employ a multi-classifier approach by dividing the prefix traces in the historical log into several buckets and fitting a separate classifier for each such bucket. At run-time, the most suitable bucket for the ongoing case is determined and the respective classifier is applied to make a prediction. Various bucketing types have been proposed and described in detail in [120]. The Training bundle supports four types of bucketing: no bucketing (i.e. fitting a single classifier), state-based bucketing, clustering-based bucketing and prefix length-based bucketing.

For each bucket of feature vectors, we train a predictive model using one of four supported machine learning techniques: decision tree, random forest, gradient boosting and extreme gradient boosting (XGBoost). For each technique, a user may manually enter the values of the most critical hyperparameters, e.g. the number of base learners (trees) and the learning rate in a gradient boosting model.

In order to accommodate users with varying degrees of expertise in machine learning and predictive process monitoring, the plugin bundle offers two training modes – basic and advanced. By default, the basic mode is activated wherein a user only needs to choose the log and prediction target. If the prediction target is based on the logical rule – whether the case duration will exceed the specified threshold, a user is also invited to key in the threshold value. For all the other settings – bucketing method, encoding method and prediction method and its hyperparameters – the default values which usually achieve the best prediction accuracy will be used. Experienced users may switch the advanced mode toggle and manually choose bucketing, encoding and prediction method settings or any plausible combination thereof. The latter is especially useful when a user wants to train and compare multiple models, e.g. using various sequence encoding methods.

Next, an operating system executes calls to Python command line scripts which are wrapped into training jobs. This is done using Java web application that also provides a basic job scheduler. When multiple training jobs are submitted, the scheduler adds them to the queuing system. The status of the jobs can be verified using the collapsible drawer in the right-hand corner. Upon the training completion, a serialized Python object in the pickle[1] format is produced and persisted in the database. It describes a trained predictive model and includes:

- Configuration parameters of the predictors (whether it is a classifier or a regressor, what learning algorithm it uses)
- Definition of each column of the event log (static or dynamic, numeric or categorical). This information allows the Runtime plugin bundle to construct a feature vector from a given partial trace.
- For each bucket, the trained model, ready to be taken as input by the selected prediction algorithm, e.g. in the case of decision trees, the whole tree representation.
- The bucketing function, which given an input sample, allows us to determine from which bucket a predictor should be taken.

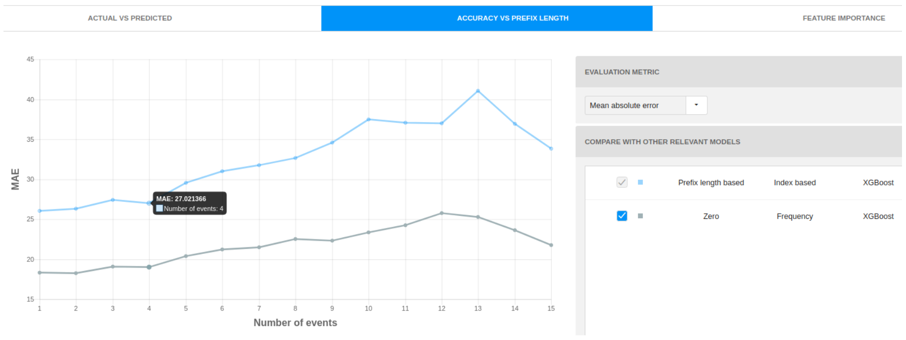
In order to provide users with an estimate of the predictive power of the trained model, we evaluate it on a held-out validation set that is not used in the training phase. By default, a user will see the average accuracy across all partial traces after a certain number of events have completed. This evaluation method was also used in [66] and [120]. For classification tasks, a user can choose which metrics to plot among the accuracy score, F1 score and logarithmic loss. For regression tasks (e.g. remaining time), a user can choose between mean absolute error and root mean square error, either raw or normalized. The accuracy of a particular model can be visually compared with that of other models trained for the same log and the same prediction target (Figure 34). Additionally, we provide a scatter plot of predicted vs. actual values (for regression tasks) or a confusion matrix (for classification tasks) and a bar plot of the relative importance of each feature for the chosen predictor. For a more detailed analysis, we provide an option to download a CSV file with the validation results for each case in the validation set.

7.4. Runtime Plugin Bundle

Once the predictive models have been created, they can be deployed to the Runtime predictive monitoring environment of Apromore, to make predictions on ongoing cases. The Runtime plugin bundle can be used to stream an event log from the repository, or hook into an external stream. In the former case, events in the log are *replayed* according to their timestamps. The replayer component is guided by a playback policy:

- *Real-time* playback: events arrive exactly at intervals specified by timestamps.

[1]`https://docs.python.org/3/library/pickle.html`



EVALUATION METRIC

Mean absolute error

COMPARE WITH OTHER RELEVANT MODELS

✓	🟦	Prefix length based	Index based	XGBoost
✓	■	Zero	Frequency	XGBoost

Figure 34: Model validation page of the Training plugin bundle.

- *Accelerated* playback: the arrival rate is proportionally increased or decreased. For example, if in the original log, the time interval between the two events is 1 hour, it may be useful to scale it to 1 seconds for demonstration purposes.
- *Fixed* playback: events arrive at a constant rate, according to the order defined by the timestamps.

Regardless of the source, the input stream of events is transformed into a stream of predictions which is visualized in a Web-based dashboard. The transformation is implemented using the dataflow pipeline in Figure 35.

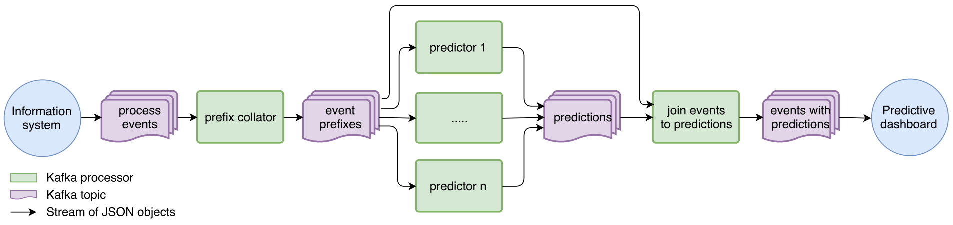
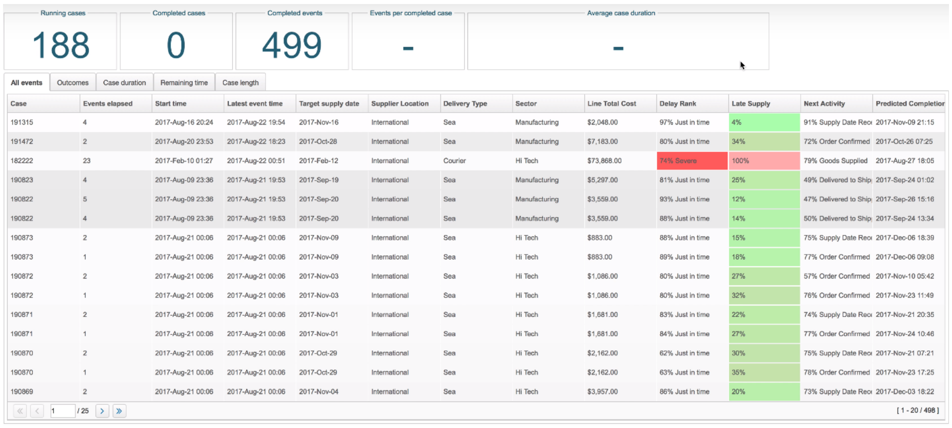


Figure 35: High-level data flow diagram of the Runtime plugin bundle.

The pipeline is built on top of the open-source Apache Kafka stream processing platform.[1] The "predictor" components of the pipeline are the predictive models from the Training plugin bundle. The "topic" components are network-accessible queues of JSON messages with publisher/subscriber support. This allows the computationally intense work of the predictors to be distributed across a cluster of networked computers, providing scalability and fault-tolerance. The "collator" component accumulates the sequence of events-to-date for each case, such that the prediction is a stateless function of the trained predictive model and of the case history. This statelessness is what allows the predictors to be freely duplicated and distributed. The "joiner" component composes the original events with the various predictions, ready for display on the dashboard.

The dashboard provides a list of both currently ongoing cases (colored in gray) as well as completed cases (colored in green), as shown in Figure 36. For each

[1] `https://kafka.apache.org`


Running cases 188 | Completed cases 0 | Completed events 499 | Events per completed case - | Average case duration -

All events | Outcomes | Case duration | Remaining time | Case length

Case	Events elapsed	Start time	Latest event time	Target supply date	Supplier Location	Delivery Type	Sector	Line Total Cost	Delay Rank	Late Supply	Next Activity	Predicted Completion
191315	4	2017-Aug-16 20:24	2017-Aug-22 19:54	2017-Nov-16	International	Sea	Manufacturing	$2,048.00	97% Just in time	4%	91% Supply Date Rec	2017-Nov-09 21:15
191472	2	2017-Aug-20 23:53	2017-Aug-22 18:23	2017-Oct-28	International	Sea	Manufacturing	$7,183.00	80% Just in time	34%	72% Order Confirmed	2017-Oct-26 07:25
182222	23	2017-Feb-10 01:27	2017-Aug-22 00:51	2017-Feb-12	International	Courier	Hi Tech	$73,868.00	74% Severe	100%	79% Goods Supplied	2017-Aug-27 18:05
190823	4	2017-Aug-09 23:36	2017-Aug-21 19:53	2017-Sep-19	International	Sea	Manufacturing	$5,297.00	81% Just in time	25%	49% Delivered to Ship	2017-Sep-24 01:02
190822	5	2017-Aug-09 23:36	2017-Aug-21 19:53	2017-Sep-20	International	Sea	Manufacturing	$3,559.00	93% Just in time	12%	47% Delivered to Ship	2017-Sep-26 15:16
190822	4	2017-Aug-09 23:36	2017-Aug-21 19:53	2017-Sep-20	International	Sea	Manufacturing	$3,559.00	88% Just in time	14%	50% Delivered to Ship	2017-Sep-24 13:34
190873	2	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-09	International	Sea	Hi Tech	$883.00	88% Just in time	15%	75% Supply Date Rec	2017-Dec-06 18:39
190873	1	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-09	International	Sea	Hi Tech	$883.00	89% Just in time	18%	77% Order Confirmed	2017-Dec-06 09:08
190872	2	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-03	International	Sea	Hi Tech	$1,086.00	80% Just in time	27%	57% Order Confirmed	2017-Nov-10 05:42
190872	1	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-03	International	Sea	Hi Tech	$1,086.00	80% Just in time	32%	76% Order Confirmed	2017-Nov-23 11:49
190871	2	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-01	International	Sea	Hi Tech	$1,681.00	83% Just in time	22%	74% Supply Date Rec	2017-Nov-21 20:35
190871	1	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-01	International	Sea	Hi Tech	$1,681.00	84% Just in time	27%	77% Order Confirmed	2017-Nov-24 10:46
190870	2	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Oct-29	International	Sea	Hi Tech	$2,162.00	62% Just in time	30%	75% Supply Date Rec	2017-Nov-21 07:21
190870	1	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Oct-29	International	Sea	Hi Tech	$2,162.00	63% Just in time	35%	78% Order Confirmed	2017-Nov-23 17:25
190869	2	2017-Aug-21 00:06	2017-Aug-21 00:06	2017-Nov-04	International	Sea	Hi Tech	$3,957.00	86% Just in time	20%	73% Supply Date Rec	2017-Dec-03 18:22

1 / 25 [1 - 20 / 498]

Figure 36: Main view of the dashboard in the Runtime plugin bundle.

case, it is also possible to visualize a range of summary statistics including the number of events in the case, its starting time and the time when the latest event in the case has occurred. For the ongoing cases, the Runtime plugin bundle provides the predicted values of the performance measures the user wants to predict. For completed cases, instead, it shows the actual values of the measures. In addition to the table view, the dashboard offers other visualization options, such as pie charts for case outcomes and bar charts for case durations.

Process workers and operational managers can set some process performance targets and subscribe to a stream of warnings and alerts generated whenever these targets are predicted to be violated. Thus, they will be capable of making informed, data-driven decisions to get a better control of the process executions. This is especially beneficial for processes where process participants have more leeway to make corrective actions (for example, in a lead management process).

7.5. Validation

For the designed prototype, we performed an external validation in collaboration with a small-sized IT vendor in Israel. Specifically, we considered a purchase-to-pay process which starts with lodging a purchase order and ends when the requested goods have been supplied. The stakeholders are interested in predicting four variables:

- *Late supply*. It is a boolean variable indicating whether or not the case is closed before the target supply date of that case.
- *Delay Rank* indicating the severity of potential delay, if any.
- *Next activity* indicating which activity will be performed right after the current one.
- *Remaining time* until case completion.

For use with the Training plugin bundle, we extracted an event log of completed purchase orders, while ongoing orders were fed into the Runtime plugin

```
{
  "case_id_col":"Line_ID",
  "activity_col":"Activity_Name",
  "timestamp_col":"time:timestamp",
  "static_cat_cols":["Supplier_ID"],
  "dynamic_cat_cols":["Employee_ID", "weekday"],
  "static_num_cols":["Line_Total_Cost", "Target_Supply_Date_delta"],
  "dynamic_num_cols":["elapsed", "timesincelastevent", "open_cases",
      "workload"],
  "ignore":["Activity_Start_Time", "hour", "Milestone", "Key",
      "Target_Supply_Date", "Part_ID"],
  "future_values":["Late_Supply", "Delay_Rank"]
}
```

Listing 7.1: Example dataset configuration file

bundle to make predictions for them. The log contains a number of case attributes and event attributes ready to be used to train the models. In order to make more accurate predictions, we perform some basic data preprocessing steps described in Section 6.2 before feeding the log into Apromore. Additionally, we define a labeling function that maps the delay for each case (if any) to one of four categories, namely “Just in case”, “Mild”, “Moderate”, “Severe”. The delay is calculated as the time difference between the predicted supply date and the target supply date.

Listing 7.1 illustrates the dataset configuration file, where each attribute in the processed log is mapped to one of the categories that define how to process the attribute.

Individual process events belonging to the same case are collated in prefixes which are stored in the Kafka’s “event prefixes” topic. The header of each prefix is annotated with the identifier of the log it belongs to, the case identifier, the positional event number and the predictor identifier. Listing 7.2 provides an example of a prefix with two events.

Predictions for each of the four target variables are updated after each executed event. To this end, the predictions are stored in the “event with predictions” topic, along with the prefix header (Listing 7.3).

One important observation noted in our validation is that the designed predictive monitoring platform shall be used in conjunction with other business intelligence or monitoring systems. To this end, the platform should offer an API to allow integration with other tools. This task is beyond the scope of the current thesis; however, we provide functionality to export the predictions in a CSV file, for importing into third-party business intelligence tools, e.g. for periodic reporting. In this way, process analysts will be able to build customized dashboards to highlight the required measures. Figure 37 provides an example of such a dashboard built in collaboration with the vendor involved in this validation, using QlikView tool[1]. The dashboard shows the distribution of open cases according to their predicted delay rank, grouped by the delivery type. The second bar chart shows the

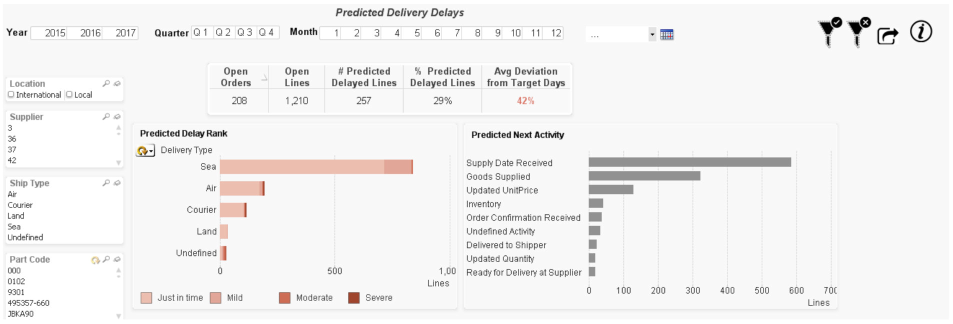
[1]https://www.qlik.com/us/products/qlikview

```json
{
  "log_id": 7,
  "Line_ID": "201608",
  "event_nr": 2,
  "predictor": 7,
  "prefix": [
    {
      "Activity_Name": "Order Opened",
      "Employee_ID": "10932",
      "Line Total Cost": "24000",
      "Supplier ID": "14",
      "time:timestamp": "2017-01-17T19:47:42+11:00",
      "elapsed": "30.0",
      "timesincelastevent": "0",
      "weekday": "3",
      "Milestone": "False",
      "open_cases": "13",
      "workload": "2",
      "Target_Supply_Date_delta": "37.8",
    },
    {
      "Activity_Name": "Order Confirmed",
      "Employee_ID": "10662",
      "Line Total Cost": "24000",
      "Supplier ID": "14",
      "time:timestamp": "2017-01-17T20:07:43+11:00",
      "elapsed": "1231.0",
      "timesincelastevent": "1201",
      "weekday": "3",
      "Milestone": "False",
      "open_cases": "19",
      "workload": "1",
      "Target_Supply_Date_delta": "37.8",
    }
  ]
}
```

Listing 7.2: Example record from the “event prefixes” topic

```json
{
  "log_id": 7,
  "Line_ID": "201608",
  "event_nr": 2,
  "predictor": 7,
  "predictions": {
    "Delay_Rank": {
      "Just in case": 0.867,
      "Mild": 0.104,
      "Moderate": 0.025,
      "Severe": 0.004
    }
  }
}
```

Listing 7.3: Example record from the “event with predictions” topic



Open Orders	Open Lines	# Predicted Delayed Lines	% Predicted Delayed Lines	Avg Deviation from Target Days
208	1,210	257	29%	42%

Figure 37: Custom dashboard in QlikView made from the exported CSV file with predictions.

distribution of open cases according to the predicted next activity to be executed.

Another important lesson learned by testing Nirdizati is that there exist several use cases for predictive process monitoring in practice. One of them is the *real-time dashboard* use case where users can see the current predictions updated continuously. This is especially relevant for very dynamic processes with high throughput. For slower processes, the *regular reports* use case may be sufficient where a user wants to get reports on a regular basis with the current set of predictions. These predictions are possibly filtered so that the report focuses on the cases that are most likely to become deviant and therefore need attention. Finally, the *alarm* use case may be helpful, where users only want to know about a prediction when a case is likely to be deviant. Initially we designed Nirdizati for the first use case, but we have learned the other two use cases also need to be supported.

7.6. Summary

We have developed an integrated open-source process monitoring platform that supports users in selecting and tuning various prediction models, and that enables the continuous prediction of different process performance measures at runtime. In particular, users can upload event logs to train a range of predictive models, and later use the trained models to predict various performance measures of running process cases from a live event stream. Predictions can be presented visually in a dashboard or exported for periodic reporting.

The developed solution was validated with a small-sized IT vendor, providing an important feedback for the future development. Namely, the validation highlighted how the solution is likely to be used in practice and the need for the solution to be integrated with existing enterprise systems.

Video demos of the model training and of the runtime functionality can be found at `http://youtu.be/xOGckUxmrVQ` and at `http://youtu.be/Q4WVebqJzUI` respectively. The source code is available under the LGPL version 3.0 license at `https://github.com/nirdizati`.

8. APPLICATION OF PROCESS MODEL DRIVEN PREDICTION TO OVERPROCESSING REDUCTION

In this chapter, we present an application of previously designed explainable predictive monitoring methods to improve the efficiency of business processes. Namely, we propose an approach to reduce overprocessing waste by ordering certain activities at runtime based on their reject probabilities and processing times estimated via predictive models.

Overprocessing waste occurs in a business process when effort is spent in a way that does not add value to the customer nor to the business. Previous studies have identified a recurrent overprocessing pattern in business processes with so-called "knockout checks", meaning activities that classify a case into "accepted" or "rejected", such that if the case is accepted it proceeds forward, while if rejected, it is canceled, and all work performed in the case is considered unnecessary. Thus, when a knockout check rejects a case, the effort spent in other (previous) checks becomes overprocessing waste. Traditional process redesign methods propose to order knockout checks according to their mean effort and rejection rate. This chapter presents a more fine-grained approach where knockout checks are ordered at runtime based on predictive machine learning models.

The rest of the chapter is organized as follows. Section 8.1 gives a more detailed introduction to the problem of overprocessing. Section 8.2 provides a definition of knockout checks and discusses related work. Section 8.3 presents the proposed knockout check reordering approach. Next, Section 8.4 discusses an empirical evaluation of the proposed approach versus design-time alternatives based on two datasets related to a loan origination process and an environmental permit process. Finally, Section 8.5 provides a summary of the chapter.

8.1. Introduction

Overprocessing is one of seven types of waste in lean manufacturing [47]. In a business process, overprocessing occurs when effort is spent in the performance of activities to an extent that does not add value to the customer nor to the business. Overprocessing waste results for example from unnecessary detail or accuracy in the performance of activities, inappropriate use of tools or methods in a way that leads to excess effort, or unnecessary or excessive verifications [47].

Previous studies in the field of business process optimization have identified a recurrent overprocessing pattern in business processes with so-called "knockout checks" [54, 127]. A knockout check is an activity that classifies a case into "accepted" or "rejected", such that if the case is accepted it proceeds forward, while if rejected, all other checks are considered unnecessary and the case is either terminated or moved to a later stage in the process. When a knockout check rejects a case, the effort spent in previous checks becomes overprocessing waste. This waste pattern is common in application-to-approval processes, where an application goes through a number of checks aimed at classifying it into admissible or not, such as eligibility checks in a University admission process, liability checks in an insurance claims handling process, or credit worthiness checks in a loan origination process. Any of these checks may lead to an application or claim being declared ineligible, effectively making other checks irrelevant for the case in question.

A general strategy to minimize overprocessing due to the execution of unnecessary knockout checks is to first execute the check that is most likely to lead to a negative ("reject") outcome. If the outcome is indeed negative, there is no overprocessing. If on the other hand we execute first the checks that lead to positive outcomes and leave the one that leads to a negative outcome to the end, the overprocessing is maximal – all the checks with positive outcome were unnecessary. On the other hand, it also makes sense to execute the checks that require less effort first, and leave those requiring higher effort last, so that the latter are only executed when they are strictly necessary. These observations lead to a strategy where knockout checks are ordered according to two parameters: their likelihood of leading to a negative outcome and the required effort.

Existing process optimization heuristics [72, 127] typically apply this strategy at design-time. Specifically, checks are ordered at design-time based on their rejection rate and mean effort. This approach achieves some overprocessing reduction but does not take into account the specificities of each case. We propose an approach that further reduces overprocessing by incorporating the above strategy into a predictive process monitoring method. Specifically, the likelihood of each check leading to a positive outcome and the effort required by each check are estimated at runtime based on the available case data and machine learning models built from historical execution data. The checks are then ordered at runtime for the case at hand according to the estimated parameters.

8.2. Background and Related Work

This study is concerned with optimizing the order in which a set of knockout checks are performed in order to minimize overprocessing. The starting point for this optimization is a *knockout section*, defined as a set of *independent binary knockout* checks. By independent we mean that the knockout checks in the section can be performed in any order. By binary we mean that each check classifies the case into two classes, hereby called "accepted" and "rejected". And by knockout we mean that if the check classifies a case as "rejected", the case jumps to a designated point in the process (called an *anchor*) regardless of the outcome of all other checks in the section. An anchor can be any point in the process execution either before or after the knockout section. In this study, we assume that an anchor point is an end event of the process, meaning that a case completes with a negative outcome as soon as one of the checks in the knockout section fails.

For example, a loan application process in a peer-to-peer lending marketplace typically includes several knockout checks. Later in this chapter, we will examine one such process containing three checks: identity check; credit worthiness check; and verification of submitted documents. Any of these checks can lead to rejection of the loan, thus the three checks constitute a knockout section.

The order of execution of checks in a knockout section can impact on overprocessing waste. For example, in the above knockout section, if the identity check is completed first and succeeds and then the credit worthiness check is completed and leads to a rejection, then the identity check constitutes overprocessing, as it did not contribute to the outcome of the case. Had the credit worthiness check been completed first, the identity check would not have been necessary.

Van der Aalst [127] outlines a set of heuristics to resequence the knockout checks according to the average processing time, rejection rate and setup time of each check. One heuristic is to execute the checks in descending order of rejection rate, meaning that the checks that are more likely to reject a case are executed first. A more refined heuristic is one where the checks are executed in descending order of the product of their rejection rate times their required effort. In other words, checks are ordered according to the principle of "least effort to reject" – checks that require less effort and are more likely to reject the case come first. This idea is identified as a redesign best practice by Reijers et al. [72] and called the "knockout principle" by Lohrmann and Reichert [70].

Pourshahid et al. [94] study the impact of applying the knockout principle in a healthcare case study. They find that the knockout pattern in combination with two other process redesign patterns improve some of the process KPIs, such as average approval turnaround time and average cost per application. Niedermann et al. [86] in the context of their study on process optimization patterns introduce the "early knockout" pattern. The idea of this latter pattern is moving the whole knockout section to the earliest possible point.

All of the above optimization approaches re-sequence the knockout checks at design time. In contrast, in this study, we investigate the idea of ordering the checks at runtime based on the characteristics of the current case. Specifically, we seek to exploit knowledge extracted from historical execution traces in order to predict the outcome of the knockout checks and to order them based on these predictions. In this respect, the present work can be seen as an application of *predictive process monitoring*.

Predictive process monitoring is a branch of process mining that seeks to exploit event logs in order to predict how one or multiple ongoing cases of a business process will unfold up to their completion [71]. A predictive monitoring approach relies on machine learning models trained on historical traces in order to make predictions at runtime for ongoing cases. Existing predictive process monitoring approaches can be classified based on the predicted output or on the type of information contained in the execution traces they take as input. In this respect, some approaches focus on the time perspective [130], others on the risk perspective [26]. Some of them take advantage only of a static snapshot of the data manipulated by the traces [71], while in others [66, 122], traces are encoded as complex symbolic sequences, and hence the successive data values taken by each data attribute throughout the execution of a case are taken into account. This study relies on the latter approach. The main difference between the present work and existing predictive monitoring approaches is that the goal is not to predict the outcome of the entire case, but rather to predict the outcome of individual activities in the case in order to re-sequence them.

The idea of using predictive monitoring to alter (or customize) a process at runtime is explored by Zeng et al. [146] in the specific context of an invoice-to-cash process. The authors train a machine learning model with historical payment behavior of customers, with the aim of predicting the outcome of a given invoice. This prediction is then used to customize the payment collection process in order to save time and maximize the chances of successfully cashing in the payment. In comparison, the proposal outlined in this chapter is generally applicable to any knockout section and not tied to a specific application domain.

8.3. Approach

In this section we describe the proposed approach to resequencing knockout checks in order to minimize overprocessing. We first give an overview of the entire solution framework and then focus on the core parts of our approach.

8.3.1. Overview

Given a designated knockout section in a process, the goal of our approach is to determine how the checks in this section should be ordered at runtime in order to reduce overprocessing waste. Accordingly, our approach pre-supposes that any preexisting design-time ordering of the checks be relaxed, so that instead the checks can be ordered by a runtime component.

The runtime component responsible for ordering the checks in a knockout section relies on a predictive monitoring approach outlined in Figure 38. This approach exploits historical execution traces in order to train two machine learning models for each check in the knockout section: one to predict the probability of the check to reject a given case, and the second to predict the *expected processing time* of the check. The former is a classification model while the latter is a regression model.

To train these models, the traces of completed cases are first encoded as feature vectors and fed into conventional machine learning algorithms. The resulting models are then used at runtime by encoding the trace of an ongoing case as a feature vector and giving it as input to the models in order to estimate the *expected processing effort* of each allowed permutation of knockout checks and to select the one with the lowest expected effort. To validate the models, once the case has completed and the actual outcome of the checks is known, we compute the *actual* *processing effort* and compare it with the *minimum processing effort* required to either accept or knock out the case in question. The difference between the actual and the minimum effort is the overprocessing waste.

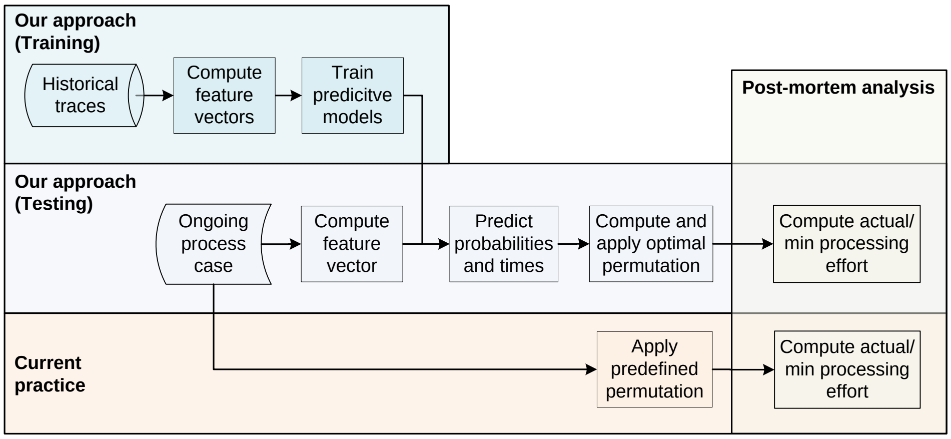


Figure 38: Overview of the proposed approach.

8.3.2. Estimation of Expected Processing Effort

As mentioned in the introduction, overprocessing results from the activities that add no value to the product or service. For example, if knockout activity rejects a case, then the case is typically terminated, and the effort spent on the previous activities becomes overprocessing waste. Consequently, to minimize the overprocessing, we are interested in determining such a permutation σ of activities that the case will be knocked out as early as possible. In the best case, the first executed activity will knock out the case; in the worst case, none of them will knock out the case. Furthermore, among all activities that could knockout the case, the one with lowest effort represents the minimal possible processing effort W_{min} for a particular case to pass the knockout section. If none of the activities knocks out the case, there is no overprocessing.

Since the minimal possible processing effort is constant for a particular process case, minimizing overprocessing of a knockout section is essentially equivalent to minimizing overall processing effort W_σ, which is dependent on the actual number of performed activities M in the knockout section:

$$W_\sigma = \sum_{i=1}^{M} w_i = \sum_{i=1}^{M} T_i R_i, \quad 1 \le M \le N \tag{8.1}$$

where w_i is the effort of an individual activity, T_i is its expected processing time and R_i is the cost of a resource that performs the activity per unit of time, which is assumed constant and known.

At least one activity needs to be performed, and if it gives a negative result, we escape the knockout section. In the extreme case, if all activities are passed normally, we cannot skip any activity; therefore, M varies from 1 to N.

However, the *actual* processing effort can only be known once the case has completed; therefore, we approximate it by estimating the *expected* processing effort $\widehat{W_\sigma}$ of a permutation σ of knockout checks. For that we introduce the notion of reject probability. The reject probability P_i^r of a check is the probability that the given check will yield a negative outcome, i.e. knock out the case. In other words, it is the percentage of cases that do not pass the check successfully.

Let us suppose we have a knockout section with three independent checks. Table 17 lists possible scenarios during the execution of the section depending on the outcome of the checks, as well as the probabilities of these scenarios and the actually spent effort.

Table 17: Possible outcomes of checks during the execution of a knockout section with three activities.

Outcome of checks	**Probability of outcome**	**Actual effort spent**
{failed}	P_1^r	w_1
{passed; failed}	$(1-P_1^r)P_2^r$	w_1+w_2
{passed; passed; failed}	$(1-P_1^r)(1-P_2^r)P_3^r$	$w_1+w_2+w_3$
{passed; passed; passed}	$(1-P_1^r)(1-P_2^r)(1-P_3^r)$	$w_1+w_2+w_3$

Depending on the outcome of the last check, we are either leaving the knockout section proceeding with the case or terminating the case. In either situation, the processing effort would be the same. Thus, joining the last two scenarios, the *expected* effort to execute a knockout section of three checks would be:

$$\widehat{W_\sigma} = w_1P_1^r + (w_1+w_2)(1-P_1^r)P_2^r + (w_1+w_2+w_3)(1-P_1^r)(1-P_2^r) \quad (8.2)$$

Generalizing, the *expected* processing effort of a knockout section with N activities can be computed as follows:

$$\widehat{W_\sigma} = \sum_{i=1}^{N-1}\left(\sum_{j=1}^{i} w_j \cdot P_i^r \prod_{k=1}^{i-1}(1-P_k^r)\right) + \sum_{j=1}^{N} w_j \cdot \prod_{k=1}^{N-1}(1-P_k^r). \quad (8.3)$$

To estimate the expected processing effort, we propose constructing predictive models for reject probabilities P_i^r and processing times T_i (see Section 8.3.4).

Having found the expected processing effort for all possible permutations σ of knockout activities, in our approach we select the one with the lowest expected effort. To validate the results in terms of minimizing overprocessing, we need to compare the *actual* processing effort W_σ taken after following the selected ordering σ with W_{min}.

8.3.3. Feature Encoding

Business process execution traces are naturally modeled as complex symbolic sequences, i.e. sequences of events each carrying data payload consisting of event attributes. However, to make estimations of the reject probabilities and processing times of knockout checks, we first need to encode traces of completed process cases in the form of feature vectors for corresponding predictive models.

As a running example, let us consider the log in Table 18, pertaining to an environmental permit request process. Each case refers to a specific application for the permit and includes activities executed for that application. For example, the first case starts with the activity *T02*. Its data payload *{2015-01-10 9:13:00, R03}* corresponds to the data associated with the *Timestamp* and *Resource* attributes. These attributes are dynamic in the sense that they change for different events. In contrast, attributes like *Channel* and *Department* are the same for all the events in a case, i.e. they are static.

Table 18: Extract of an event log.

Case	Case attributes		Event attributes			
ID	Channel	Department	Activity	Timestamp	Resource	...
1	Email	General	T02	2015-01-10 9:13:00	R03	...
1	Email	General	T06	2015-01-10 9:14:20	R12	...
2	Fax	Customer contact	T02	2015-01-10 9:18:03	R03	...
1	Email	General	T10	2015-01-10 9:13:45	R12	...
2	Fax	Customer contact	T05	2015-01-10 9:13:57	R12	...

To encode traces as feature vectors, we include both static information, coming from the case attributes and dynamic information, contained in the event payload. In general, for a case i with U case attributes $\{s_1,\ldots,s_U\}$ containing M events $\{e_1,\ldots,e_M\}$, each of them having an associated payload $\{d_1^1,\ldots,d_1^R\},\ldots\{d_M^1,\ldots,d_M^R\}$ of length R, the resulting feature vector would be:

$$\vec{X}_i = (s_1,\ldots,s_U,e_1,\ldots,e_M,d_1^1,\ldots,d_1^R,\ldots d_M^1,\ldots,d_M^R) \tag{8.4}$$

As an example, the first case in the log in Table 18 will be encoded as such:

$$\begin{aligned}\vec{X}_1 = (&\text{Email},\text{General},\text{T02},\text{T06},\text{T10},\text{2015-01-10 9:13:00},\text{R03},\\ &\text{2015-01-10 9:14:20},\text{R12},\text{2015-01-10 9:13:45},\text{R12})\end{aligned} \tag{8.5}$$

This kind of encoding, referred to as index-based encoding, is lossless since all data from the original log are retained. It achieves a relatively high accuracy and reliability when making early predictions of the process outcome. [66].

8.3.4. Prediction of Reject Probability and Processing Time

To make online predictions on a running case, we apply pre-built (offline) models using prefixes of historical cases before entering the knockout section. For example, if a knockout section typically starts after the n-th event, as model features we can use case attributes and event attributes of up to $(n-1)$-th event. For predicting reject probabilities of knockout activities, we train classification models, while for predicting processing times we need regression models. To train the models, in addition to historical case prefixes, we need labels associated with the outcome of a check (classification) and its processing time (regression). As a learning algorithm, we primarily use support vector machines (SVM) since they can handle unbalanced data in a robust way [52]. In addition, we fit decision trees and random forest models, for they have been used to address a wide range of predictive process monitoring problems [26,42,66,71].

To assess the predictive power of the classifiers, we use the area under receiver operator characteristic curve (AUC) measure [15]. AUC represents the probability that the binary classifier will score a randomly drawn positive sample higher than a randomly drawn negative sample. A value of AUC equal to 1 indicates a perfect ranking, where any positive sample is ranked higher than any negative sample. A value of AUC equal to 0.5 indicates the worst possible classifier that is not better than random guessing. Finally, a value of AUC equal to 0 indicates a reserved perfect classifier, where all positive samples get the lowest ranks.

As a baseline, instead of predicting the reject probabilities, we use *constant* values for them computed from the percentage of cases that do not pass the particular knockout activity in the log. Similarly, for processing times of activities, we take the average processing time for each activity across all completed cases. This roughly corresponds to the approach presented in [127]. Another, even simpler baseline, assumes executing knockout activities in a random order for each case, regardless of their reject probabilities and processing times.

8.4. Evaluation

We implemented the proposed overprocessing prediction approach as a set of scripts for the statistical software `R`, and applied them to two publicly available real-life logs. Below, we describe the characteristics of the datasets, we report on the accuracy of predictive models trained on these datasets, and we compare our approach against the two baselines discussed above in terms of overprocessing reduction. A package containing the R scripts, the datasets and the evaluation results is available at `http://apromore.org/platform/tools`.

8.4.1. Datasets and Features

We used two datasets derived from real-life event logs. The first log records executions of the loan origination process of *Bondora* [13], an Estonian peer-to-peer lending marketplace; the second one originates from an environmental permit request process carried out by a Dutch municipality, available as part of the *CoSeLoG* project [19]. Table 19 reports the size of these two logs in terms of number of completed cases, and the rejection rate of each check. Each log has three checks, the details of which are provided next.

Bondora dataset The Bondora dataset provides a snapshot of all loan data in the Bondora marketplace that is not covered by data protection laws. This dataset refers to two processes: the loan origination process and the loan repayment pro-

Table 19: Summary of datasets.

Dataset	Completed cases	Knockout checks	
		Name	Rejection rate
Bondora	40,062	*IdCancellation*	0.080
		CreditDecision	0.029
		PostFundingCancellation	0.045
Environmental permit	1,230	*T02*	0.005
		T06	0.013
		T10	0.646

cess. Only the first process features a knockout section, hence we filtered out the data related to the second process. When a customer applies for a loan, they fill in a loan application form providing information such as their personal data, income and liabilities, with supporting documents. The loan origination process starts upon the receipt of the application and involves (among other activities) three checks: the identity check (associated with event *IdCancellation* in the log); the credit worthiness assessment (associated to event *CreditDecision*); and the documentation verification (associated to event *PostFundingCancellation*). A negative outcome of any of these checks leads to rejection of a loan application.

Bondora's clerks perform these checks in various orders based on their experience and intuition of how to minimize work, but none of the checks requires data produced by the others, so they can be reordered. Over time, the checks have been performed in different orders. For example, during a period when listing loans into the marketplace was a priority due to high investor demand, loans were listed before all document verifications had been concluded, which explains why the third check is called *PostFundingCancellation*, even though in many cases this check is performed in parallel with the other checks.

In this log, the knockout section starts immediately after the case is lodged. Thus, the only features we can use to build our predictive models are the case attributes, i.e. the information provided by the borrower at the time of lodging the application. These features can be grouped into three categories. *Demographical* features include age of the loan borrower, their gender, country of residence, language, educational background, employment and marital status. *Financial* features describe the borrower's financial well-being and include information about their income, liabilities, debts, credit history, home ownership, etc. Finally, the third group includes *loan* features, such as amount of the applied loan, and its duration, maximum acceptable interest rate, purpose of the loan and the application type (timed funding or urgent funding). A more detailed description of each attribute is available from the Bondora Web site [13].

It should also be noted that in the Bondora log there is no information about the start time and the end time of each activity. Thus, we can only use it to estimate the reject probabilities, not the processing times.

Environmental permit dataset The second dataset records the execution of the receiving phase of an environmental permit application process in a Dutch municipality [19]. The process discovered from the log has a knockout section (see Figure 39) consisting of three activities: *T02*, to check confirmation of receipt, *T06*, to determine necessity of stop advice, and *T10*, to determine necessity to stop indication. In this scenario, the checks are not completely independent. Specifically, *T10* can only be done after either *T02* or *T06* has been performed – all permutations compatible with this constraint are possible.

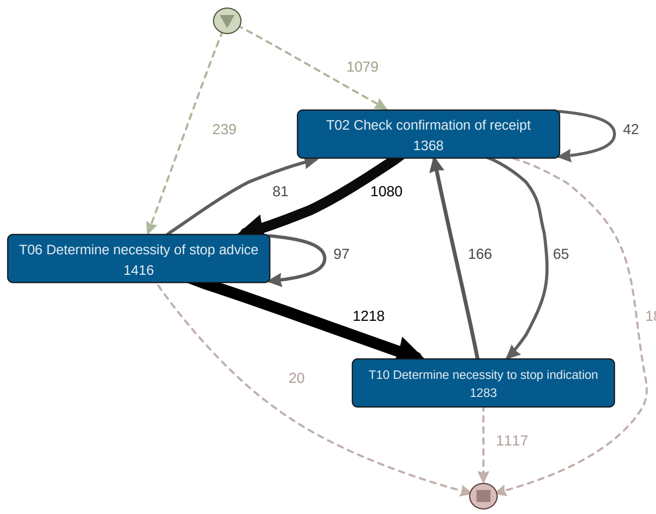

1079
239
T02 Check confirmation of receipt
1368
42
81
1080
T06 Determine necessity of stop advice
1416
97
166
65
181
1218
20
T10 Determine necessity to stop indication
1283
1117

Figure 39: Process map extracted from the environment permit log.

Another special feature of this knockout section is that in a small number of cases some checks are repeated multiple times. If the first check in a case is repeated multiple times, and then the second check is executed (and the first check is not repeated anymore after that), we simply ignore the repetition, meaning that we treat the first check as not having been repeated by discarding all occurrences of this check except the last one. Similarly, we discarded incomplete cases as they did not allow us to assess the existence of overprocessing.

Each case in the log refers to a specific application for an environmental permit. The log contains both case attributes and event payload along with the standard XES attributes. Case attributes include channel by which the case has been lodged, department that is responsible for the case, responsible resource and its group. In addition to the case attributes, the predictive models can utilize attributes of events that precede the knockout section. Generally, there is only one such event, namely *Confirmation of receipt*, that includes attributes about the resource who performed it and its assigned group.

This log contains event completion timestamps but not event start timestamps. So also for this second log we do not have enough information to predict the processing time of each check, and we can only work with reject probabilities.

8.4.2. Prediction Accuracy

We split each dataset into a training set (80% of cases) to train the models, and a test set (20%) to evaluate the predictive power of the models built. As a learning algorithm we applied support vector machine (SVM) classification, trained using the `e1071` package in R. This choice allows us to build a probability model which fits a logistic distribution using maximum likelihood to the decision values of all binary classifiers, and computes the a-posteriori class probabilities for the multi-class problem using quadratic optimization [78]. Therefore, it can output not only the class label, but the probability of each class. The probability of a zero class essentially gives us an estimation of the reject probability.

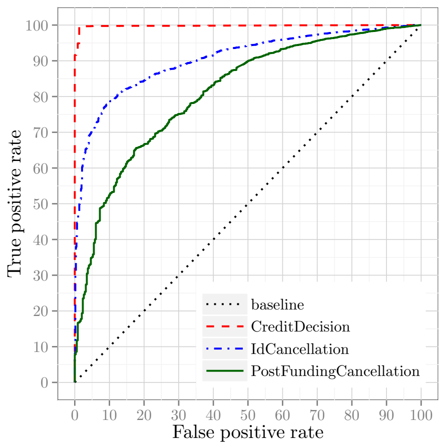
In both datasets the majority of cases pass all the checks successfully, thus the datasets are highly imbalanced with respect to the class labels. A naive algorithm that simply predicts all test examples as positive will have very low error, since the negative examples are so infrequent. One solution to this problem is to use a Poisson regression, which requires forming buckets of observations based on the independent attributes and modeling the aggregate response in these buckets as a Poisson random variable [48]. However, this requires discretization of all continuous independent attributes, which is not desirable in our case. A simpler and more robust solution would be to undersample positive cases. Weiss et al. [140] showed that for binary classification the optimal class proportion in the training set varies by domain and objective, but generally to produce probability estimates, a 50:50 distribution is a good option. Thus, we leave roughly as many positive examples as there are negative ones and discard the rest.

To ensure the consistency of the results we apply five-fold cross-validation. Figure 40 shows the average ROC curves, across all ten runs. the AUC varies from 0.812 (*PostFundingCancellation*) to 0.998 (*CreditDecision*) for the Bondora dataset, and from 0.527 (*T06*) to 0.645 (*T10*) for the Environmental dataset. The lower values in the latter dataset are due to the limited number of features that can be extracted (see Section 8.4.1), as well as by the fact that the dataset has much less completed cases for training (Table 19), which is further exacerbated by having to remove many positive samples after undersampling.

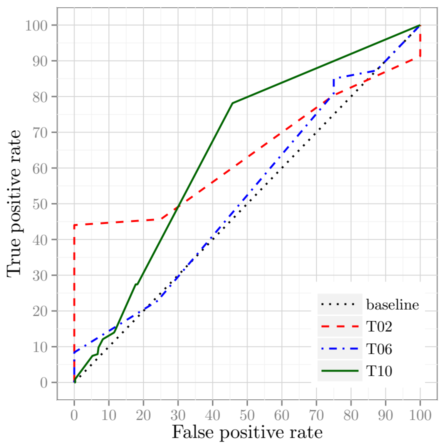
8.4.3. Overprocessing Reduction

As stated in Section 8.3.2, the *actual* processing effort is given by Formula 8.1. However, since the necessary timestamps are absent from our datasets, it is impossible to find the processing times T_i of the activities. Nor do we have data about the resource costs R_i. Therefore, we assume $T_i R_i = 1$ for all activities. Then the actual processing effort simply equals the number of performed activities in the knockout section. It can be shown that in this case the optimal permutation σ that minimizes the expected processing is equivalent to ordering the knockout activities by decreasing reject probabilities.

In Table 20 we report the average number of checks and percentage of over-



(a)



(b)

Figure 40: ROC curves of predictive models for checks in *Bondora* (a) and *Environmental* (b) datasets.

processing of our approach over the ten runs, against the two baselines (constant probabilities for each check and random ordering – see Section 8.3.4). We found that the actual number of performed checks in case of following our suggested ordering is less than the number of checks performed in either baseline. Specifically, for the Bondora dataset we are doing only 1.22% more checks than minimally needed, which represents a 2.62 percentage points (pp) improvement over the baseline with constant probabilities and 4.51 pp improvement over the baseline with random ordering. However, for the environmental permit dataset the advantage of our approach over the constant probabilities baseline is very marginal. This can be explained by the skewed distribution of the knockout frequencies for the three checks in this dataset (the lowest knockout frequency being 0.5% and the highest being 64.6%). Thus, it is clear that the check with the lowest knockout frequency has to be executed at the end. Additionally, as mentioned in the Section 8.4.1, not all checks are independent in the second dataset. Therefore, the solution space for the optimal permutation is rather limited.

Table 20: Average number of performed checks and overprocessing for test cases.

	Average # of checks		**Average overprocessing, %**	
	Bondora	Environmental	Bondora	Environmental
Optimal	21,563	416	0	0
Our approach	21,828	576	1.22	38.49
Constant P_i^r's	22,393	577	3.85	38.89
Random	22,800	657	5.74	58.16

In addition, in Table 21 we report the number of cases with one, two or three knockout checks performed. As shown before, for a dataset with three checks the optimal number of checks is either one (if at least one check yields a negative outcome) or three (if all checks are passed). Therefore, in the cases with two checks, the second one should have been done first. In the Bondora dataset, such suboptimal choices are minimized; for the environmental dataset, again, our approach is just as good as the one that relies on constant probabilities.

Table 21: Distribution of number of checks across the test cases.

Ordering by	**Bondora**			**Environmental**		
	1	2	3	1	2	3
Optimal	1237	0	6775	163	0	83
Our approach	974	261	6777	2	158	86
Constant P_i^r's	642	359	7011	3	155	88
Random	413	410	7189	1	78	167

8.4.4. Execution Times

Our approach involves some runtime overhead to find the optimal permutation as compared to the baseline scenario in which checks are performed in a predefined order. For real-time prediction it is crucial to output the results faster than the mean arrival rate of cases. Thus, we also measured the average runtime overhead of our approach. All experiments were conducted using `R` version 3.2.2 on a laptop with a 2.4 GHz Intel Core i5 quad core processor and 8 Gb of RAM. The runtime overhead generally depends on the length of the process cases and the number of possible permutations of the checks. For the Bondora dataset, it took around 70 seconds to construct the SVM classifiers (offline) for all the checks, using default training parameters. In contrast, for the Environmental dataset with much shorter feature vectors it took less than a second to train the classifier (see Table 22). At runtime, it takes less than 2 milliseconds on average to find the optimal permutation of knockout activities for an ongoing case for both datasets (including preprocessing of the data and application of the classifier). This shows that our approach performs within reasonable bounds for online applications.

Table 22: Execution times of various components of our approach in milliseconds.

Component		**Bondora**		**Environmental**	
		mean	**st dev**	**mean**	**st dev**
Offline, overall	Learn classifier	75,000	9,000	150	20
Online, per case	Preprocess data	0.45	0.03	0.67	0.03
	Apply classifier	1.37	0.15	0.12	0.02
	Find optimal permutation	0.12	0	0.02	0

8.4.5. Threats to Validity

Threats to external validity are the limited number and type of logs we used for the evaluation and the use of a single classifier. While we chose only two datasets from two distinct domains (financial and government), these two datasets represent real-life logs well. They exhibit substantial differences in the number of events, event classes and total number of traces, with one log being relatively large (over 40,000 cases) and the other relatively small (around 1,200 cases).

Both datasets used in this evaluation did not have the required start and end event timestamps to estimate the processing times of the knockout checks. Thus, we assigned a constant time to all checks. The inability to estimate processing time does not invalidate our approach. In fact, our approach would tend to further reduce the amount of overprocessing if processing times were known.

In the Bondora dataset, the three checks have been performed in different orders for different cases. When one of the checks leads to a negative outcome for a given case, the checks that were not yet completed at that stage of the case sometimes remain marked as negative, even if it might be the case that these checks would have led to positive outcomes should they have been completed. This issue may have an effect on the reported results, but we note that it affects both the reported performance of our approach and that of the baselines.

We reported the results with a single classifier (SVM). With decision trees and random forests, we obtained qualitatively the same results, i.e. they all improved over the baselines. However, we decided to only retain SVM because this classifier yielded the highest classification accuracy among all classifiers we tested. However, our approach is independent of the classifier used. Thus, using a different classifier does not in principle invalidate the results. That said, we acknowledge that the goodness of the prediction, as in any classification problem, depends on the particular classifier employed. Hence, it is important to test multiple classifiers for a given dataset, and to apply hyperparameter tuning, in order to choose the most adequate classifier with the best configuration.

8.5. Summary

We have presented an approach to reduce overprocessing by ordering knockout checks at runtime based on their reject probabilities and processing times determined via predictive models. Experimental results show that the proposed runtime ordering approach outperforms a design-time ordering approach when the reject probabilities of the knockout checks are close to each other. In the dataset where one check had a considerably higher rejection rate than the other, the design-time and the runtime ordering approach yielded similar results.

The proposed approach is not without limitations. One limitation of scope is that the approach is applicable when the checks are independent (i.e. can be reordered) and every check is performed once within one execution of the knockout section. In particular, the approach is not applicable when some of the knockout checks can be repeated in case of a negative outcome. This is the case for example in a university admission process, where an eligibility check may initially lead to a rejection, but the applicant can ask the application to be reconsidered (and thus the check to be repeated) after providing clarifications or additional information. In other words, the current approach is applicable when a negative outcome ("reject") is definite and cannot be revoked. Similarly, we assume that a check leading to a positive outcome is definite and cannot be reconsidered. Designing heuristics for cases where the outcomes of checks are revocable is a direction for future work.

Another limitation is that the approach is designed to minimize overprocessing only, without considering other performance dimensions such as cycle time (i.e. mean case duration). If we add cycle time into the equation, it becomes desirable to parallelize the checks rather than sequentializing them. In other words, rather than performing the checks in a knockout section in strict sequence, some or all of checks could be started in parallel, such that whenever the first check fails, the other parallel checks are cancelled. On the one hand this parallelization leads to higher overprocessing effort, since effort is spent in partially completed checks that are later cancelled. On the other hand, it reduces overall cycle time, particularly when some of the checks involve idle time during their execution. For example, in a university admission process when some documents are found to be missing, the checks involving these documents need to be put on hold until the missing documents arrive. If the goal is to minimize both overprocessing and cycle time, this waiting time can be effectively used to perform other checks.

9. CONCLUSION AND FUTURE WORK

9.1. Summary of Contributions

Process monitoring forms an integral part of business process management. It involves activities in which process execution data are collected and analyzed to gauge process performance with respect to a set of performance objectives. Traditionally, process monitoring has been performed at runtime, providing a real-time overview of process performance and identifying performance issues as they arise. Recently, the rapid adoption of enterprise systems with logging capabilities has spawned the active development of data-driven, *predictive* process monitoring that exploits historical process execution data to predict the future course of ongoing instances of a business process. Thus, potentially deviant process behavior can be anticipated and proactively addressed.

To this end, various approaches have been proposed to tackle typical predictive monitoring problems, such as whether an ongoing process instance will fulfill its performance objectives, or when will an instance be completed. However, due to differences in experimental setup, choice of datasets, evaluation measures and baselines, the relative performance of each method remains unclear. As such, the first contribution of this thesis is a comprehensive analysis and classification of state-of-the-art process monitoring techniques to predict the remaining execution time of process instances. The relevant existing studies were identified through a systematic literature review, which retrieved 23 original studies dealing with the problem of remaining time prediction. Out of these, nine were considered to contain a distinct contribution (primary studies). Through further analysis of the primary studies, a taxonomy was proposed based on three main aspects, the type of input data required, process awareness and the family of algorithms employed. We found that most studies employ machine learning algorithms to train predictive models and apply them at runtime. These methods were further broken down into categories depending on how they divide the input traces into homogeneous buckets and how these traces are encoded into feature vectors.

Following the literature survey, we identified that most of the proposed methods adopt a black-box approach, insofar as they predict a single scalar value without decomposing this prediction into more elementary components. As such, in this thesis, we designed and evaluated two explainable predictive monitoring techniques for temporal process performance measures, using a mechanism known as "Transparent Box Design", specifically via decomposition of the prediction into its elementary components. In the first technique, we used deep learning models, specifically RNNs and LSTMs, that had been shown to achieve higher levels of accuracy in predictive monitoring, owing to their ability to learn relevant features from trace prefixes automatically. In order to make these models explainable, we used an instance of multi-task learning where several related predictive monitoring tasks were integrated into a single system which is trained jointly, as a way to decompose the predictions. Namely, our technique provides a prediction of the temporal measure in terms of the predicted temporal measures of each of the activities in the most likely remaining path of the current trace.

The second technique to achieve explainable predictive monitoring is based on a BPMN process model automatically discovered from the event log. In this technique, an ongoing case is replayed on the process model and for each activity and each decision gateway reachable from the current execution state, we predict its performance measure and branching probabilities respectively. These predictions are then aggregated at the level of a process instance by means of flow analysis techniques.

We conducted a comparative evaluation of two proposed explainable approaches with each other and with various state-of-the-art baselines, using a unified experimental set-up and 17 real-life event logs originating from different business domains. The results of the evaluation highlighted the trade-off between the prediction accuracy and explainability of predictions in the context of predictive monitoring of temporal measures. The most accurate predictions were typically obtained via black-box models. Specifically, in 13 out of 17 datasets, LSTM-based models trained in a single-task learning setting achieve the best accuracy. At the same time, explainable models are generally less accurate. This trade-off should be considered by business process analysts when choosing a suitable prediction method.

Next, in order to demonstrate the practical application of the research conducted in this thesis, we designed and evaluated a prototype system for predictive business process monitoring, namely Nirdizati. Nirdizati implements the techniques for predicting future activities, processing and cycle times, and remaining time presented in the thesis. Nirdizati has been validated in a case study involving a small-scale logistics supplier. The results highlighted the merits of the prototype and the techniques it embodies, as well as provided valuable feedback and avenues for future work.

The last but not least contribution of the thesis is the application of process model driven predictive process monitoring techniques to reduce overprocessing. Specifically, we showed how reordering process activities at runtime, based on the predicted branching probabilities and cycle times of individual activities, can reduce the unnecessary process work, and therefore, overall process cost.

9.2. Future Work

This thesis has laid the foundation for the explainable predictive process monitoring. At the same time, our research contributions open up a number of directions for future research.

We have designed and evaluated two techniques for explainable predictive process monitoring of temporal performance measures based on the idea of decomposing a prediction into elementary components. A direction for future work would be to validate how explainable the resulting predictions are for end users – process workers and operational managers. In data mining and machine learning, explainability is defined as the ability to explain or to provide the meaning in understandable terms to a human [44]. The level of explainability of a model may have a significant impact on the usability of a model. Namely, when the reasoning behind the prediction is unclear, users are less likely to trust that model.

A natural extension of the proposed predictive process monitoring techniques is to *recommend* process workers and operational managers the best course of actions in order to steer the process execution towards the declared process performance targets. In other words, not only will the process participants receive the accurate predictions, but they will also be guided as to how to act upon these predictions. These solutions constitute *prescriptive* process monitoring [123]. Providing predictions that are explainable can make process users more willing to let the technology influence decision making and, thus, facilitate the adoption of prescriptive monitoring systems.

The proposed techniques have been validated to predict temporal process performance measures. A direction for future work would be to apply these techniques to predict other quantitative measures, such as cost. One difficulty here is to obtain an event log that contains the necessary measures. Another relevant direction is to apply explainable techniques for the prediction of categorical targets that are not readily decomposable, i.e. in the context of a classification problem.

In this work, we have identified a limitation of the process model driven approach when dealing with unstructured process models. As a result, we were not able to evaluate the approach on some logs. An avenue for future work is to apply process model structuring techniques such as the one proposed in [144] to lift, at least partially, restrictions on the topology of the models.

Next, the proposed process model driven approach relies on the accuracy of the reject probability estimates provided by the classification model. It is known however that the likelihood probabilities produced by classification methods (including random forests) are not always reliable. Methods for estimating the reliability of such likelihood probabilities have been proposed in the machine learning literature [62]. A possible enhancement of the proposed approach would be to integrate heuristics that take into account such reliability estimates.

Another direction is to provide an API to allow integration of our process monitoring platform, Nirdizati, with other tools. It would provide a simple way for process users to export predictions about ongoing cases into their business analytics tools, such as Microsoft Power BI, for the purposes of monitoring and periodic reporting.

Finally, it would be interesting to apply explainable predictive methods to reduce not only overprocessing but other types of waste, such as *defect waste* induced when a defective execution of an activity subsequently leads to part of the process having to be repeated in order to correct the defect (i.e. rework). The idea is that if a defective activity execution can be detected earlier, the effects of this defect can be minimized and corrected more efficiently. Predictive process monitoring can thus help us to detect defects earlier and to trigger corrective actions as soon as possible.

BIBLIOGRAPHY

[1] IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams. *IEEE Std 1849-2016*, pages 1–50, Nov 2016.

[2] Robert Andrews, Suriadi Suriadi, Moe Wynn, Arthur H. M. ter Hofstede, Anastasiia Pika, Hoang Nguyen, and Marcello La Rosa. Comparing static and dynamic aspects of patient flows via process model visualisations. *QUT ePrints 102848*, 2016.

[3] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Diagnosing behavioral differences between business process models: An approach based on event structures. *Information Systems*, 56:304–325, 2016.

[4] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Giorgio Bruno. Automated discovery of structured process models: Discover structured vs. discover and structure. In Isabelle Comyn-Wattiau, Katsumi Tanaka, Il-Yeol Song, Shuichiro Yamamoto, and Motoshi Saeki, editors, *Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*, volume 9974 of *Lecture Notes in Computer Science*, pages 313–329, 2016.

[5] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *CoRR*, abs/1705.02288, 2017.

[6] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, Ben Taskar, and SVN Vishwanathan. *Predicting structured data*. MIT Press, 2007.

[7] Jörg Becker, Dominic Breuker, Patrick Delfmann, and Martin Matzner. Designing and implementing a framework for event-based predictive modelling of business processes. In Fernand Feltz, Bela Mutschler, and Benoît Otjacques, editors, *Enterprise modelling and information systems architectures - EMISA 2014, Luxembourg, September 25-26, 2014*, volume 234 of *LNI*, pages 71–84. GI, 2014.

[8] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.

[9] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011.*

Proceedings of a meeting held 12-14 December 2011, Granada, Spain., pages 2546–2554, 2011.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[11] Antonio Bevacqua, Marco Carnuccio, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. A data-driven prediction framework for analyzing and monitoring business process performances. In Slimane Hammoudi, José Cordeiro, Leszek A. Maciaszek, and Joaquim Filipe, editors, *Enterprise Information Systems - 15th International Conference, ICEIS 2013, Angers, France, July 4-7, 2013, Revised Selected Papers*, volume 190 of *Lecture Notes in Business Information Processing*, pages 100–117. Springer, 2013.

[12] Alfredo Bolt and Marcos Sepúlveda. Process remaining time prediction using query catalogs. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, volume 171 of *Lecture Notes in Business Information Processing*, pages 54–65. Springer, 2013.

[13] Bondora. Loan Dataset. `https://www.bondora.ee/en/invest/statistics/data_export`. Accessed: 2015-10-23.

[14] Guillaume Bouchard and Bill Triggs. The tradeoff between generative and discriminative classifiers. In *16th IASC International Symposium on Computational Statistics (COMPSTAT'04)*, pages 721–728, 2004.

[15] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[16] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[17] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[18] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. *MIS Quarterly*, 40(4):1009–1034, 2016.

[19] J.C.A.M. Buijs. 3TU.DC Dataset: Receipt phase of an environmental permit application process (WABO). `https://data.3tu.nl/repository/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`. Accessed: 2015-10-30.

[20] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[21] Michelangelo Ceci, Pasqua Fabiana Lanotte, Fabio Fumarola, Dario Pietro Cavallo, and Donato Malerba. Completion time and next activity prediction of processes using sequential pattern mining. In Saso Dzeroski, Pance

Panov, Dragi Kocev, and Ljupco Todorovski, editors, *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8777 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014.

[22] Eugenio Cesario, Francesco Folino, Massimo Guarascio, and Luigi Pontieri. A cloud-based prediction framework for analyzing business process performances. In Francesco Buccafurri, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl, editors, *Availability, Reliability, and Security in Information Systems - IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2016, and Workshop on Privacy Aware Machine Learning for Health Data Science, PAML 2016, Salzburg, Austria, August 31 - September 2, 2016, Proceedings*, volume 9817 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2016.

[23] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In Krishnapuram et al. [61], pages 785–794.

[24] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[25] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 160–167. ACM, 2008.

[26] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. A recommendation system for predicting risks across multiple business process instances. *Decision Support Systems*, 69:1–19, 2015.

[27] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[28] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. A general framework for correlating business process characteristics. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, volume 8659 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 2014.

[29] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235–257, 2016.

[30] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[31] Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maggi, and Irene Teinemaa. Clustering-based predictive process monitoring. *CoRR*, abs/1506.01428, 2015.

[32] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[33] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. A deep learning approach for predicting process behaviour at runtime. In Marlon Dumas and Marcelo Fantinato, editors, *Business Process Management Workshops - BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers*, volume 281 of *Lecture Notes in Business Information Processing*, pages 327–338, 2016.

[34] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. Predicting process behaviour using deep learning. *Decision Support Systems*, 100:129–140, 2017.

[35] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[36] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering context-aware models for predicting business process performances. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012, Rome, Italy, September 10-14, 2012. Proceedings, Part I*, volume 7565 of *Lecture Notes in Computer Science*, pages 287–304. Springer, 2012.

[37] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering high-level performance models for ticket resolution processes. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Johann Eder, Zohra Bellahsene, Norbert Ritter, Pieter De Leenheer, and Dejing Dou, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings*, volume 8185 of *Lecture Notes in Computer Science*, pages 275–282. Springer, 2013.

[38] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining predictive process models out of low-level multidimensional logs. In Jarke et al. [55], pages 533–547.

[39] Brady Fowler, Monica Rajendiran, Timothy Schroeder, Nicholas Bergh, Abigail Flower, and Hyojung Kang. Predicting patient revisits at the university of virginia health system emergency department. In *Systems and*

Information Engineering Design Symposium (SIEDS), 2017, pages 253–258. IEEE, 2017.

[40] Keinosuke Fukunaga and Raymond R. Hayes. Effects of sample size in classifier design. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(8):873–885, 1989.

[41] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013.

[42] Daniela Grigori, Fabio Casati, Malú Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.

[43] Christoph Gröger, Florian Niedermann, and Bernhard Mitschang. Data mining-driven manufacturing process optimization. In *Proceedings of the world congress on engineering*, volume 3, pages 4–6, 2012.

[44] Riccardo Guidotti, Anna Monreale, Franco Turini, Dino Pedreschi, and Fosca Giannotti. A survey of methods for explaining black box models. *CoRR*, abs/1802.01933, 2018.

[45] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.

[46] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[47] Ben J. Hicks. Lean information management: Understanding and eliminating waste. *Int J. Information Management*, 27(4):233–249, 2007.

[48] Shawndra Hill, Foster Provost, and Chris Volinsky. Network-Based Marketing: Identifying Likely Adopters via Consumer Networks. *Statistial Science*, 21(2):256–276, 2006.

[49] Tin Kam Ho. Random decision forests. In *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*, pages 278–282. IEEE Computer Society, 1995.

[50] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[51] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[52] Jae Pil Hwang, Seongkeun Park, and Euntai Kim. A new weighted approach to imbalanced data classification problem via support vector ma-

chine with quadratic cost function. *Expert Syst. Appl.*, 38(7):8580–8585, 2011.

[53] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[54] Monique H. Jansen-Vullers, Mariska Netjes, and Hajo A. Reijers. Business process redesign for effective e-commerce. In Marijn Janssen, Henk G. Sol, and René W. Wagenaar, editors, *Proceedings of the 6th International Conference on Electronic Commerce, ICEC 2004, Delft, The Netherlands, October 25-27, 2004*, volume 60 of *ACM International Conference Proceeding Series*, pages 382–391. ACM, 2004.

[55] Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors. *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, volume 8484 of *Lecture Notes in Computer Science*. Springer, 2014.

[56] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2015.

[57] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[58] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas. Data preprocessing for supervised learning. *International Journal of Computer Science*, 1(2):111–117, 2006.

[59] Josua Krause, Adam Perer, and Kenney Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In Jofish Kaye, Allison Druin, Cliff Lampe, Dan Morris, and Juan Pablo Hourcade, editors, *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, pages 5686–5697. ACM, 2016.

[60] R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognition*, 32(12):1999–2009, 1999.

[61] Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 2016.

[62] Meelis Kull and Peter A. Flach. Reliability maps: A tool to enhance probability estimates and improve classification accuracy. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II*,

volume 8725 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2014.

[63] M. Laguna and J. Marklund. *Business Process Modeling, Simulation and Design, Second Edition*. Taylor & Francis, 2013.

[64] Julia Lasserre, Christopher M. Bishop, and J. M. Bernardo. *Generative or Discriminative? Getting the Best of Both Worlds*, volume 8. Oxford University Press, 2007.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[66] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 297–313. Springer, 2015.

[67] Arnon Levy. *Machine-likeness and explanation by decomposition*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2014.

[68] Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018.

[69] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[70] Matthias Lohrmann and Manfred Reichert. Effective application of process improvement patterns to business processes. *Software and System Modeling*, 15(2):353–375, 2016.

[71] Fabrizio Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In Jarke et al. [55], pages 457–472.

[72] Selma Limam Mansar and Hajo A. Reijers. Best practices in business process redesign: validation of a redesign framework. *Computers in Industry*, 56(5):457–471, 2005.

[73] Marc Kerremans. Market Guide for Process Mining, 2018. https://www.gartner.com/doc/3870291/market-guide-process-mining.

[74] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Corts. Predictive monitoring of business processes: a survey. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.

[75] Nijat Mehdiyev, Joerg Evermann, and Peter Fettke. A multi-stage deep learning approach for business process event prediction. In Peri Loucopoulos, Yannis Manolopoulos, Oscar Pastor, Babis Theodoulidis, and Jelena Zdravkovic, editors, *19th IEEE Conference on Business Informatics, CBI*

2017, Thessaloniki, Greece, July 24-27, 2017, Volume 1: Conference Papers, pages 119–128. IEEE Computer Society, 2017.

[76] Nijat Mehdiyev, Joerg Evermann, and Peter Fettke. A novel business process prediction model using a deep learning method. *Business & Information Systems Engineering*, Jul 2018.

[77] Andreas Metzger, Philipp Leitner, Dragan Ivanovic, Eric Schmieders, Rod Franklin, Manuel Carro, Schahram Dustdar, and Klaus Pohl. Comparing and combining predictive business process monitoring techniques. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 45(2):276–290, 2015.

[78] David Meyer, Evgenia Dimitriadou, Kurt Hornik, and Andreas Weingessel. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group, TU Wien*, 2015.

[79] Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explorations*, 3(1):27–32, 2001.

[80] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[81] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269, 2017.

[82] Tom M. Mitchell. *Machine learning*. McGraw-Hill, 1997.

[83] A Möller, V Ruhlmann-Kleider, C Leloup, J Neveu, N Palanque-Delabrouille, J Rich, R Carlberg, C Lidman, and C Pritchet. Photometric classification of type ia supernovae in the supernova legacy survey with supervised learning. *Journal of Cosmology and Astroparticle Physics*, 2016(12):008, 2016.

[84] Nicolò Navarin, Beatrice Vincenzi, Mirko Polato, and Alessandro Sperduti. LSTM networks for data-aware remaining time prediction of business process instances. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*, pages 1–7. IEEE, 2017.

[85] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 841–848. MIT Press, 2001.

[86] Florian Niedermann, Sylvia Radeschütz, and Bernhard Mitschang. Business process optimization using formalized optimization patterns. In Witold Abramowicz, editor, *Business Information Systems - 14th International Conference, BIS 2011, Poznan, Poland, June 15-17, 2011. Proceedings*, volume 87 of *Lecture Notes in Business Information Processing*, pages 123–135. Springer, 2011.

[87] Randal S. Olson, William La Cava, Zairah Mustahsan, Akshay Varik, and Jason H. Moore. Data-driven advice for applying machine learning to bioinformatics problems. *CoRR*, abs/1708.05070, 2017.

[88] Anastasiia Pika, Wil M. P. van der Aalst, Colin J. Fidge, Arthur H. M. ter Hofstede, and Moe Thandar Wynn. Predicting deadline transgressions using event logs. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, volume 132 of *Lecture Notes in Business Information Processing*, pages 211–216. Springer, 2012.

[89] Anastasiia Pika, Wil M. P. van der Aalst, Colin J. Fidge, Arthur H. M. ter Hofstede, and Moe Thandar Wynn. Profiling event logs to configure risk indicators for process delays. In Camille Salinesi, Moira C. Norrie, and Oscar Pastor, editors, *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings*, volume 7908 of *Lecture Notes in Computer Science*, pages 465–481. Springer, 2013.

[90] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Data-aware remaining time prediction of business process instances. In *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 816–823. IEEE, 2014.

[91] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. *Computing*, 100(9):1005–1031, 2018.

[92] Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske. The triconnected abstraction of process models. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, volume 5701 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2009.

[93] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In Mario Bravetti and Tevfik Bultan, editors, *Web Services and Formal Methods - 7th International Workshop, WS-FM 2010, Hoboken, NJ, USA, September 16-17, 2010. Revised Selected Papers*, volume 6551 of *Lecture Notes in Computer Science*, pages 25–41. Springer, 2010.

[94] Alireza Pourshahid, Gunter Mussbacher, Daniel Amyot, and Michael Weiss. An aspect-oriented framework for business process improvement. In Gilbert Babin, Peter G. Kropf, and Michael Weiss, editors, *E-Technologies: Innovation in an Open World, 4th International Conference, MCETECH 2009, Ottawa, Canada, May 4-6, 2009. Proceedings*, volume 26 of *Lecture Notes in Business Information Processing*, pages 290–305. Springer, 2009.

[95] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007.

[96] Sarunas Raudys and Anil K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):252–264, 1991.

[97] Hajo A. Reijers, Irene T. P. Vanderfeesten, and Wil M. P. van der Aalst. The effectiveness of workflow management systems: A longitudinal study. *Int J. Information Management*, 36(1):126–141, 2016.

[98] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" Explaining the predictions of any classifier. In Krishnapuram et al. [61], pages 1135–1144.

[99] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 389–403. Springer, 2013.

[100] Andreas Rogge-Solti and Mathias Weske. Prediction of business process durations using non-markovian stochastic petri nets. *Information Systems*, 54:1–14, 2015.

[101] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[102] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[103] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *CoRR*, abs/1708.08296, 2017.

[104] Johannes Schneider, Joshua Peter Handali, and Jan vom Brocke. Increasing trust in (big) data analytics. In Raimundas Matulevicius and Remco M. Dijkman, editors, *Advanced Information Systems Engineering Workshops - CAiSE 2018 International Workshops, Tallinn, Estonia, June 11-15, 2018, Proceedings*, volume 316 of *Lecture Notes in Business Information Processing*, pages 70–84. Springer, 2018.

[105] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press, 2002.

[106] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maggi. Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings*, volume 10445 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2017.

[107] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Queue mining - predicting delays in service processes. In Jarke et al. [55], pages 42–57.

[108] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. Queue mining for delay prediction in multi-class service processes. *Information Systems*, 53:278–295, 2015.

[109] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 802–810, 2015.

[110] F. Shull, J. Singer, and D.I.K. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer London, 2007.

[111] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[112] David Spiegelhalter, Mike Pearson, and Ian Short. Visualizing uncertainty about the future. *science*, 333(6048):1393–1400, 2011.

[113] I. Steinwart and A. Christmann. *Support Vector Machines*. Information Science and Statistics. Springer New York, 2008.

[114] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 374–383. ACM, 2006.

[115] Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[116] B.G. Tabachnick and L.S. Fidell. *Using Multivariate Statistics*. Number v. 1 in Using Multivariate Statistics. HarperCollins College Publishers, 1996.

[117] P.N. Tan, M. Steinbach, A. Karpatne, and V. Kumar. *Introduction to Data Mining*. What's New in Computer Science Series. Pearson Education, 2013.

[118] Niek Tax, Sebastiaan J. van Zelst, and Irene Teinemaa. An experimental evaluation of the generalizing capabilities of process discovery techniques and black-box sequence models. In Jens Gulden, Iris Reinhartz-Berger, Rainer Schmidt, Sérgio Guerreiro, Wided Guédria, and Palash Bera, editors, *Enterprise, Business-Process and Information Systems Modeling - 19th International Conference, BPMDS 2018, 23rd International Conference, EMMSAD 2018, Held at CAiSE 2018, Tallinn, Estonia, June 11-12, 2018, Proceedings*, volume 318 of *Lecture Notes in Business Information Processing*, pages 165–180. Springer, 2018.

[119] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2017.

[120] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *CoRR*, abs/1707.06766, 2017.

[121] Irene Teinemaa, Marlon Dumas, Anna Leontjeva, and Fabrizio Maggi. Temporal stability in predictive process monitoring. *CoRR*, abs/1712.04165, 2017.

[122] Irene Teinemaa, Marlon Dumas, Fabrizio Maggi, and Chiara Di Francescomarino. Predictive business process monitoring with structured and unstructured data. In Marcello La Rosa, Peter Loos, and Oscar Pastor, editors, *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2016.

[123] Irene Teinemaa, Niek Tax, Massimiliano de Leoni, Marlon Dumas, and Fabrizio Maggi. Alarm-based prescriptive process monitoring. In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Business Process Management Forum - BPM Forum 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings*, volume 329 of *Lecture Notes in Business Information Processing*, pages 91–107. Springer, 2018.

[124] Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, Michael Adams, and Nick Russell, editors. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010.

[125] L. Torlay, Marcela Perrone-Bertolotti, E. Thomas, and Monica Baciu. Machine learning-XGBoost analysis of language networks to classify patients with epilepsy. *Brain Informatics*, 4(3):159–169, 2017.

[126] Ruben Urraca-Valle, Javier Antoñanzas, Fernando Antoñanzas-Torres, and Francisco Javier Martínez de Pisón. Estimation of daily global horizontal irradiation using extreme gradient boosting machines. In Manuel Graña, José Manuel López-Guede, Oier Etxaniz, Álvaro Herrero, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16 - San Sebastián, Spain, October 19th-21st, 2016, Proceedings*, volume 527 of *Advances in Intelligent Systems and Computing*, pages 105–113, 2016.

[127] Wil M. P. van der Aalst. Re-engineering knock-out processes. *Decision Support Systems*, 30(4):451–468, 2001.

[128] Wil M. P. van der Aalst. Process discovery: Capturing the invisible. *IEEE Comp. Int. Mag.*, 5(1):28–41, 2010.

[129] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[130] Wil M. P. van der Aalst, M Helen Schonenberg, and Minseok Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.

[131] Sjoerd van der Spoel, Maurice van Keulen, and Chintan Amrit. Process prediction in noisy data sets: A case study in a Dutch hospital. In Philippe Cudré-Mauroux, Paolo Ceravolo, and Dragan Gasevic, editors, *Data-Driven Process Discovery and Analysis - Second IFIP WG 2.6, 2.12 International Symposium, SIMPDA 2012, Campione d'Italia, Italy, June 18-20, 2012, Revised Selected Papers*, volume 162 of *Lecture Notes in Business Information Processing*, pages 60–83. Springer, 2012.

[132] Boudewijn F. van Dongen, R. A. Crooy, and Wil M. P. van der Aalst. Cycle time prediction: When will this case finally be finished? In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2008.

[133] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.

[134] Vladimir Vapnik. *Statistical learning theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley, 1998.

[135] Ilya Verenich. A general framework for predictive business process monitoring. In Oscar Pastor, Stefanie Rinderle-Ma, Roel Wieringa, Selmin Nurcan, Barbara Pernici, and Hans Weigand, editors, *Proceedings of CAiSE 2016 Doctoral Consortium co-located with 28th International Conference*

on Advanced Information Systems Engineering (CAiSE 2016), Ljubljana, Slovenia, June 13-17, 2016., volume 1603 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[136] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maggi, and Chiara Di Francescomarino. Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In Manfred Reichert and Hajo A. Reijers, editors, *Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers*, volume 256 of *Lecture Notes in Business Information Processing*, pages 218–229. Springer, 2015.

[137] Ilya Verenich, Hoang Nguyen, Marcello La Rosa, and Marlon Dumas. White-box prediction of process performance indicators via flow analysis. In Reda Bendraou, David Raffo, LiGuo Huang, and Fabrizio Maggi, editors, *Proceedings of the 2017 International Conference on Software and System Process, Paris, France, ICSSP 2017, July 5-7, 2017*, pages 85–94. ACM, 2017.

[138] Mark von Rosing, Henrik von Scheel, and August-Wilhelm Scheer, editors. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I.* Morgan Kaufmann/Elsevier, 2015.

[139] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. A bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research*, 18:70:1–70:37, 2017.

[140] Gary M. Weiss and Foster J. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Intell. Res.*, 19:315–354, 2003.

[141] Michael Westergaard and Fabrizio Maggi. Modeling and verification of a protocol for operational support using coloured petri nets. In Lars Michael Kristensen and Laure Petrucci, editors, *Applications and Theory of Petri Nets - 32nd International Conference, PETRI NETS 2011, Newcastle, UK, June 20-24, 2011. Proceedings*, volume 6709 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

[142] S. Wexler, J. Shaffer, and A. Cotgreave. *The Big Book of Dashboards: Visualizing Your Data Using Real-world Business Scenarios.* John Wiley & Sons, Incorporated, 2017.

[143] Philip C. Woodland and Daniel Povey. Large scale discriminative training of hidden markov models for speech recognition. *Computer Speech & Language*, 16(1):25–47, 2002.

[144] Yong Yang, Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, and Liang Zhang. Generalized aggregate quality of service computation for composite services. *Journal of Systems and Software*, 85(8):1818–1830, 2012.

[145] Xiaoxin Yin and Jiawei Han. CPAR: classification based on predictive association rules. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003*, pages 331–335. SIAM, 2003.

[146] Sai Zeng, Prem Melville, Christian A. Lang, Ioana M. Boier-Martin, and Conrad Murphy. Using predictive analysis to improve invoice-to-cash collection. In Ying Li, Bing Liu, and Sunita Sarawagi, editors, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 1043–1050. ACM, 2008.

[147] Zhi-Hua Zhou, Yuan Jiang, and Shifu Chen. Extracting symbolic rules from trained neural network ensembles. *AI Commun.*, 16(1):3–15, 2003.

Appendix A. SOURCE CODE AND SUPPLEMENTARY MATERIALS

To ensure reproducibility of our experiments, as well as to enable future researchers to build upon our work with minimal effort, we release source code and other supplementary materials for each chapter of the thesis into the public domain. The relevant URLs are listed below.

Material	URL
Chapter 3 (SLR)	`https://goo.gl/kg6xZ1`
Chapter 4	`https://verenich.github.io/ProcessSequencePrediction/`
Chapter 5	`https://github.com/verenich/new-flow-analysis/`
Chapter 6	`https://github.com/verenich/time-prediction-benchmark/`
Chapter 7	`https://github.com/nirdizati/nirdizati-training-backend/`
Chapter 8	`https://verenich.github.io/caise2016overprocessing/`

ACKNOWLEDGMENTS

Writing this thesis has been a long journey and the most important phase in my academic career. I would like to thank a number of supportive people who have contributed to the completion of my dissertation in many different ways.

First of all, I would like to express my gratitude to Professor Marlon Dumas, without whom I would not have been able to start nor complete this perilous journey. Marlon was the person who "discovered" me and introduced me to the exciting world of process mining. I appreciate his flexibility, patience, motivation, contributions of time, ideas, immense knowledge and financial support. His precious guidance helped me in all the time of research and writing of this thesis.

Secondly, I am deeply grateful to my co-supervisor Professor Marcello La Rosa for his help at all times. He and Marlon have always been supportive of my research, even when things did not go as expected. I am indebted to him for he has not only guided me throughout my thesis work but has also helped me in securing academic collaborations and post-graduation opportunities.

Thirdly, I really appreciate having worked with Professor Arthur ter Hofstede. He has always been available to discuss and give valuable advice on my research. His academic support and input are greatly appreciated.

I would also like to acknowledge useful feedback from Associate Professor Fabrizio Maria Maggi, Associate Professor Richi Nayak and Dr Sander Leemans. Their comments were very helpful in crafting the final draft of the thesis.

I would like to thank the HPC and Research Support Group, Queensland University of Technology for providing me with computational resources to run all of my research experiments.

I also acknowledge the financial assistance in the form of scholarships from University of Tartu, the Estonian Research Council (grant IUT20-55) and Queensland University of Technology. This has also helped me to attend numerous conferences and summer schools, adding a lot to my experience.

This thesis would not have been possible without the support I received from my colleagues – the members of the BPM research group – who provided a welcoming and challenging environment to work in. I am particularly grateful to Andrii, Stanislav and Simon who helped me transform my ideas into actions by providing assistance with the implementation of the Nirdizati dashboard. I cannot forget to thank all my friends who surrounded me during these years in Tartu, Brisbane and Melbourne.

I would also like to express my gratitude to my alma maters, South Ural State University and University of Tartu, for providing me with an outstanding educational base and an opportunity to learn from the many distinguished teachers.

Finally, I feel a deep sense of gratitude to my wonderful family: my mom, dad, sister as well as grandparents who supported my decision to do my PhD thousands of miles away from home. Thank you for believing and encouraging me all this time!

SUMMARY IN ESTONIAN

Äriprotsesside ajaliste näitajate selgitatav ennustav jälgimine

Äriprosside jälgimine on keskosa igas äriprotsesside juhtimise initsiatiivis. Protsesside jälgimine annab analüütikutele ning juhtidele võimalust tuvastada alajõudustatud protsessid ning reageerida õigeaegselt. Traditsioonilised protsessi jälgimistehnikad on kujundatud nii, et anda ülevaadet protsessi hetkesest jõudlusest. Sellisena nad annavad meile võimalust tuvastada kahanemis-sündmusi tagantjärgi ja selle tõttu nad ei luba meil käituda ennustavalt.

Masinõppe valdkonna edasiareng koos protsesside operatiivsus-admestikke laikättesaadavusega tõi ette ennustatavad protsessijälgimistehnikad. Need meetodid lubavad meile ennustada protsessi tuleviku oleku, ja seega annavad töötajatele ning operatiivjuhtidele võimalust reageerida reaalajas selleks, et vältida või töödelda jõudlusprobleeme ning ühilduvuse rikkeid.

Viimastel aastatel said pakutud erinevaid lahendusi selleks, et lahendada tüüpilisi ennustatava protsessijälgimise küsimusi, näitkes kas antud jooksev protsess saab olla vastavuses oma jõudlus eesmärkidega, või millal protsess lõpetab oma tööd. See doktoritöö algab süstemaatilisest valdkonna ülevaadest ja äriprotsesside ajalise näitajate ennustatava jälgimise meetodite taksonoomiast. Üks põhipuudus, mis saab selle töö raames tuvastatud on olemasolevate meetodite puudulik tugi selgitatavuses. Kuigi praktikas selgitatavus on ennustatavate meetodite kriitiline omadus. Tihtipeale ei piisa täpsest ennustamist, et jooksev protsess saavutab mitterahuldatava tulemust. Kasutaja jaoks on ka tähtis saada aru miks protsessi tulev olek on ennustatud niipidid ning kuidas vältida seda mitterahuldatava tulemust.

Et lahendada seda puudust, antud teesis pakub meetodeid, mis võimaldavad ehitada ennustatavaid mudeleid ajaliste näidete jälgimiseks selgitaval viisil. Töö panus on ajaliste jõudlusenäidete jaoks selgitav-ennustavate protsessijälgimis meetodite kujundus ja nende hinnang, mis on ka tuntud nime „Läbipaistva Kasti Kujundus" (Transparent Box Design) all. Nimelt, lahutavad pakutud meetodid ennustamis ülessannet elementaarseteks komponendideks.

Selle üldise idee integreerides me rakendasime sügav närvvõrgu mudeleid, mis on näidanud kõrgema täpsuse ennustava protsessijälgimise kontekstis. Selleks, et teha neid mudeleid seletavaks me kasutame mitme-ülesanne õpimist (multi-task learning) isendi, kus mitu seotud ennustava jälgimis ülessannet on integreeritud ühte süsteemi, mis on treenitud tervikuna. Järgmises integratsioonis me pakume selgitav-ennustava protsessjälgimis meetodi. Meetod eeldab BPMN protsessi mudeli sündmuste logidest väljavõtmist, ajalise jõudluse ennustamist tegevuste tasemel ja järgnevat ennustuste koondamist terve protsessi tasemel kasutades voolu analüüsi (flow analysis) meetodeid. Need mõlemad integratsioonid pakuvad lokaalse interpretatsiooni ennustustest, mis said tehtud iga protsessi isendi kohta.

Masinõppe valdkonnas sai märgatud fundamentaalne kompromiss selgitavuse ning täpsuse vahel. Selleks et hinnata seda kompromissi me teostame pakutud meetodite võrdleva hindamist omavahel ning vastu erinava kaasaegse lähtejoont, kaasarvatud musta kasti meetodeid, mis ei tooda selgitavaid ennustusi. Hindamine on osutud kasutades 17 päris-elu sündmuste logisid, mis väljendavad erinevaid omadusi ning tulenevad erinevatest valdkondadest.

Töö teadusuuringute panus on koonsolideeritud ühtse platvormi ennustava äriprotsesside jälgimise jaoks, nimelt Nirdizati. Nirdizati on veebipõhine äriprotsesside ennustav-jälgimise mootor, mille abil saab treenida ennustus mudeleid kasutades selles töös kirjeldatud meetodid ning ka kolmanda osapoole meetodeid, ja seejärel ennustada reaalajas jooksva protsessi isendid. Nirdizati sa hinnatud koostöös potentsiaalsete kasutajatega, kes andis väärtusliku tagasisidet selle kohta kuidas see tehnoloogia võiks leida kasutust ettevõtte tasemel.

Lõppkokkuvõtus me esitame pakutud selgitav-ennustava jälgimis meetodite rakendust eesmärgiga parandada äriprotsessi efektiivsust. Nimelt, me demonstreerime kuidas need meetodid saavad olla rakendatud selleks et vähendada ületöötlemis jäätmed järjestades teatud tegevused käitusajal lähtudest nende oletatud tulemusest ja pingutusest, mis on määratud ennustus mudelite poolt.

CURRICULUM VITAE

Personal data

Name:	Ilya Verenich
Data of birth:	11.05.1990
Citizenship:	Russian
Language:	English, Russian

Education

2014–2018	University of Tartu, Faculty of Mathematics and Computer Science, doctoral studies, specialty: computer science.
2012–2014	University of Tartu, Faculty of Mathematics and Computer Science, master studies, specialty: software engineering.
2006–2010	South Ural State University, Faculty of Mathematics and Computer Science, bachelor studies, specialty: electronics engineering.

Employment

2012–2014	STACC OÜ, research engineer

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Ilya Verenich
Sünnikuupäev:	11.05.1990
Kodakondsus:	Vene
Keelteoskus:	inglise, vene

Haridus

2014–2018	Tartu Ülikool, Matemaatika-informaatikateaduskond, doktoriõpe, eriala: informaatika.
2012–2014	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, eriala: software engineering.
2006–2010	Uurali Riiklik Ülikool, Matemaatika- informaatikateaduskond, bakalaureuseõpe, eriala: elektrotehnika.

Teenistuskäik

2012–2014	STACC OÜ, teadur

LIST OF ORIGINAL PUBLICATIONS

Over the course of this research, seven peer-reviewed publications (3 A-rank conference papers, 2 demonstration papers, 1 workshop, 1 doctoral consortium) have been produced and published. In addition, three technical reports have been authored of which two have been submitted to journals and are awaiting the reviewers' feedback. Following is the list of the original contributions in chronological order:

[I] **I. Verenich**, M. Dumas, M. L. Rosa, F. M. Maggi, and C. D. Francescomarino. Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In *Business Process Management Workshops – BPM 2015, 13th International Workshops, Innsbruck, Austria*, pages 218-229, 2015.

[II] **I. Verenich**, M. Dumas, M. L. Rosa, F. M. Maggi, and C. D. Francescomarino. Minimizing overprocessing waste in business processes via predictive activity ordering. In *Advanced Information Systems Engineering – 28th International Conference, CAiSE 2016, Ljubljana, Slovenia. Proceedings*, pages 186-202, 2016.

[III] **I. Verenich**. A general framework for predictive business process monitoring. In *Proceedings of CAiSE 2016 Doctoral Consortium co-located with 28th International Conference on Advanced Information Systems Engineering*, 2016.

[IV] **I. Verenich**, M. Dumas, M. La Rosa, F. M. Maggi, D. Chasovskyi, and A. Rozumnyi. Tell me what's ahead? Predicting remaining activity sequences of business process instances. Technical report. In *QUT ePrints 96732*, 2016.

[V] N. Tax, **I. Verenich**, M. L. Rosa, and M. Dumas. Predictive business process monitoring with LSTM neural networks. In *Advanced Information Systems Engineering – 29th International Conference, CAiSE 2017, Essen, Germany. Proceedings*, pages 477-492, 2017.

[VI] **I. Verenich**, H. Nguyen, M. L. Rosa, and M. Dumas. White-box prediction of process performance indicators via flow analysis. In *Proceedings of the 2017 International Conference on Software and System Process, Paris, France*, pages 85-94, 2017. [**Awarded as the Best Paper**]

[VII] K. Jorbina, A. Rozumnyi, **I. Verenich**, C. D. Francescomarino, M. Dumas, C. Ghidini, F. M. Maggi, M. L. Rosa, and S. Raboczi. Nirdizati: A web-based tool for predictive process monitoring. In *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain*, 2017. [**Awarded as the Best Demo Paper**]

[VIII] **I. Verenich**, M. Dumas, M. L. Rosa, H. Nguyen, and A. ter Hofstede. Predicting process performance: A white-box approach. Technical report. In

QUT ePrints 117379, 2018.

This manuscript is a significantly extended version of [VI], submitted to the special issue of *Journal of Software: Evolution and Process* on December 29, 2017, revised and resubmitted on June 5, 2018 following reviewers' comments.

[IX] **I. Verenich**, S. Mõškovski, S. Raboczi, M. Dumas, M. La Rosa, and F. M. Maggi. Predictive process monitoring in Apromore. In *CAiSE Forum 2018, Tallinn, Estonia. Proceedings, volume 317 of Lecture Notes in Business Information Processing*, pages 244-253, Springer, 2018. [**Awarded as the Best Demo Paper**]

[X] **I. Verenich**, M. Dumas, M. La Rosa, F. Maggi, and I. Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. Technical report. In *arXiv preprint arXiv:1805.02896*, 2018.

Manuscript submitted to *ACM Transactions on Intelligent Systems and Technology* on May 10, 2018.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multiparty computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.