DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
6

IRENE TEINEMAA

Predictive and Prescriptive Monitoring of Business Process Outcomes

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
6

IRENE TEINEMAA

Predictive and Prescriptive Monitoring of Business Process Outcomes

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 March 5, 2019 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

Opponents

Prof. Donato Malerba, PhD
University of Bari
Italy

Prof. Dr. Myra Spiliopoulou
University of Magdeburg
Germany

The public defense will take place on April 26, 2019 at 10:15 a.m. in J. Liivi 2, room 404.

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

ISSN 2613-5906
ISBN 978-9949-03-000-2 (print)
ISBN 978-9949-03-001-9 (pdf)

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

ABSTRACT

Recent years have witnessed a growing adoption of machine learning techniques for business improvement across various fields. Among other emerging applications, organizations are exploiting opportunities to improve the performance of their business processes by using predictive models for runtime monitoring. Such predictive process monitoring techniques take an event log (a set of completed business process execution traces) as input and use machine learning techniques to train predictive models. At runtime, these techniques predict either the next event, the remaining time until the end, or the final outcome of an ongoing case, given its incomplete execution trace consisting of the events performed up to the present moment in the given case. In particular, a family of techniques called outcome-oriented predictive process monitoring focuses on predicting whether a case will end with a desired or an undesired outcome. An outcome-oriented predictive process monitoring system is expected to make accurate predictions in the early execution stages, i.e. given as few events as possible. The user of the system can use the predictions to decide whether or not to intervene, with the purpose of preventing an undesired outcome or mitigating its negative effects. Prescriptive process monitoring systems go beyond purely predictive ones, by not only generating predictions but also advising the user if and how to intervene in a running case in order to optimize a given utility function.

In this context, this thesis addresses the question of how to train, evaluate, and use predictive models for predictive and prescriptive monitoring of business process outcomes. A variety of outcome-oriented predictive process monitoring techniques have been developed in the literature. However, as different authors have used different terminology, experimental settings, datasets, and baselines, there is no clear overview of how these techniques compare to each other. To address this issue, the thesis undertakes an analysis and proposes a taxonomy of methods for training predictive process monitoring models. Moreover, the thesis reports on a comparative experimental evaluation of existing techniques, using a benchmark covering 24 prediction tasks constructed from nine real-life event logs. The results put into question a previous hypothesis that training separate classifiers for each prefix length using a lossless (index-based) feature encoding of a trace is superior to training a single classifier with a lossy (aggregation) encoding.

The analysis of the state of the art unveiled that existing techniques focus on structured data, neglecting the unstructured (textual) data often available in real-life event logs. The thesis addresses this gap by proposing a framework that makes use of text mining techniques to extract features from unstructured data and combines them with features from structured data in order to train more powerful predictive models. An experimental evaluation shows that a simple bag-of-n-grams encoding of textual data often outperforms other text mining techniques in this setting.

The evaluation of predictive process monitoring techniques is traditionally lim-

ited to measuring the accuracy and the earliness of the predictions, ignoring the stability of the sequential predictions generated by a model for increasingly longer prefixes of the same trace. To address this gap, the thesis proposes a notion of temporal stability for predictive process monitoring and evaluates existing techniques with respect to this measure. The results show that LSTM classifiers achieve the highest temporal stability, followed by XGBoost.

Lastly, existing research proposals in the field of predictive process monitoring are either limited to providing the user with predictions without any advice on using these predictions, or they expect the user to specify a decision threshold, whereas the system will trigger an alarm if a prediction score exceeds this threshold. The thesis proposes a framework for alarm-based prescriptive process monitoring that empirically finds the optimal decision threshold based on a cost model that accounts for the cost of an intervention, the cost of the undesired outcome, and the effectiveness of mitigating the undesired outcome if an intervention is made. The experimental evaluation shows that the proposed approach consistently finds thresholds that minimize the overall processing costs.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Meaning
AB	AdaBoost
ACC	Accuracy
AUC	Area under the ROC curve
BoNG	Bag-of-n-grams
BPIC	Business Process Intelligence Challenge
CRM	Customer-relationship management
DFG	Directly-Follows Graph
DR	Debt recovery
ERP	Enterprise resource planning
EX	Exclusion criterion
FN	False negative
FP	False positive
FPR	False Positive Rate
GBM	Generalized boosted regression models
GBT	Gradient boosted trees
HAC	Hierarchical agglomerative clustering
HMM	Hidden Markov Model
idf	Inverse document frequencies
IN	Inclusion criterion
KNN	k-nearest neighbors
LDA	Latent Dirichlet Allocation
logit	Logistic regression
LSTM	Long short-term memory
LtC	Lead-to-cash
LTL	Linear Temporal Logic
MPL	Minimal prediction length
NB	Naive Bayes
PV	Paragraph Vector
RF	Random forest
ROC	Receiver operating characteristic
ROI	Return on investment
RQ	Research question
SGD	Stochastic gradient descent
SLR	Systematic Literature Review
SVM	Support vector machine
tf	Term frequencies
TN	True negative
TP	True positive
TPE	Tree-structured parzen estimator
TPR	True Positive Rate
TS	Temporal stability
XGB/XGBoost	Extreme gradient boosting

LIST OF NOTATIONS

In the following, we list some fundamental notations used throughout the thesis.

Symbol	Meaning
$\mathbf{x}^{(i)} = (x_1^{(i)}, ..., x_p^{(i)})$	a feature vector corresponding to the *i*th instance with *p* features
$\mathbf{x}^{(i)<t>}$	a feature vector corresponding to the *t*th timepoint in the *i*th sequence
$y^{(i)}$	the target variable for the *i*th instance
$\hat{y}^{(i)}$	a prediction (class or score) of the target variable for the *i*th instance
$\hat{y}^{(i)<t>}$	a prediction for the *t*th timepoint in the *i*th instance
e	an event
$\pi_{\mathcal{C}}(e), \pi_{\mathcal{T}}(e), \pi_{\mathcal{A}}(e)$	the case id, timestamp, and the activity name of an event e
$\sigma = \langle e_1, \ldots, e_{\lvert\sigma\rvert} \rangle$	a trace, i.e. a sequence of events
$\lvert\sigma\rvert$	the length of σ
$\sigma(i)$	the *i*th event of trace σ
$hd^k(\sigma)$	a prefix trace containing the first k events of a trace σ
τ	decision threshold

1. INTRODUCTION

Companies and organizations aim to generate value to their customers and stakeholders. This is achieved through *business processes*, i.e. chains of events, activities, and decisions sharing common *business goals*, such as manufacturing of a product or providing a service. For instance, a typical *order-to-cash* process starts when a customer places an order for purchasing a product or a service; encompasses several activities performed by the vendor, such as verifying the purchase order, shipping of the product, composing and sending an invoice to the customer; and concludes with the customer making the payment and receiving the product [25].

1.1. Process mining

Modern organizations use process-aware information systems that record information about the execution of business processes that can be extracted and preprocessed to produce *event logs* [101]. The availability of event logs has lead to a growing interest among organizations to improve their business processes in a data-driven manner. The collection of techniques that aim to extract valuable process-related information from event logs is called *process mining* [102].

An event log (see example in Table 1) consists of a set of *traces*, i.e. sequences of *event records* (*events* for short) that are related to the same *case* (an instance of a business process). For example, a case can refer to all events related to the same purchase order. An event carries information about the execution of a given *activity*[1]. The core elements of every event are the *case id* (e.g. the identifier of the purchase order), the *activity name* (or *event class*, i.e. the type of the executed event), and a *timestamp*. In other words, every event represents the occurrence of an activity at a particular point in time and in the context of a given case. Additionally, an event can contain various other data, e.g. together with a *payment* activity, the *amount of payment* is often recorded. Furthermore, an event often contains information about the *resource*, i.e. the process worker or the software system involved in executing the activity. Such *event attributes* are of dynamic nature, i.e. they can take different values for different events throughout the trace. Conversely, *case attributes* are of static nature, i.e. they belong to the case and are hence shared by all events generated by the same case. Examples of case attributes are the type of the ordered product and the age of the customer. We use the term *control flow* to refer to the case id, the activity name and the timestamp. We use the term *data payload* to refer to the rest of the event and case attributes[2].

[1] We use the term *activity* to refer to all the steps that can happen in a business process, including those that are instantaneous and those with a non-zero duration.

[2] In the XES standard, which is an XML-based standard for event logs, the data payload is referred to as the *optional attributes*, since these attributes are not necessary for discovering a process model from an event log [40].

Table 1: Example of an event log.

Case ID	Event ID	Customer	Product	Timestamp	Activity	Resource	Amount
C1	E01	Kate	P01	2018-07-20 16:13	Create order	Mark	-
C1	E02	Kate	P01	2018-07-20 16:14	Check availability	Mark	-
C1	E03	Kate	P01	2018-07-20 16:16	Create invoice	Mark	100
C1	E04	Kate	P01	2018-07-22 10:45	Receive payment	PM1	100
C1	E05	Kate	P01	2018-07-22 14:10	Ship product	Mark	-
C1	E06	Kate	P01	2018-07-22 17:23	Deliver product	John	-
C2	E07	Tom	P02	2018-07-23 10:05	Create order	Alice	-
C2	E08	Tom	P02	2018-07-23 10:05	Check availability	Alice	-
C2	E09	Tom	P02	2018-07-23 10:07	Create invoice	Alice	200
C2	E10	Tom	P02	2018-07-23 15:32	Cancel order	System	-

More formally, we assume that events are characterized by various *properties* (i.e. event and case attributes).

Definition 1.1.1 (Events, Properties of events)**.** Let $\mathcal{E}$ be the universe of events, i.e. the set of all possible event identifiers. Function $\pi_{\mathcal{P}} : \mathcal{E} \rightarrow \mathcal{P}$ assigns a value of a property $\mathcal{P}$ to an event.

We do not impose a specific set of properties, however, we assume that three of these properties are the case id, the timestamp, and the activity name of an event. Let $\mathcal{C}$ be the domain of case ids, $\mathcal{T}$ the domain of timestamps, and $\mathcal{A}$ the domain of activity names, then there is a function $\pi_{\mathcal{C}} : \mathcal{E} \rightarrow \mathcal{C}$ that assigns a case id to an event, a function $\pi_{\mathcal{T}} : \mathcal{E} \rightarrow \mathcal{T}$ that assigns a timestamp to an event, and a function $\pi_{\mathcal{A}} : \mathcal{E} \rightarrow \mathcal{A}$ that assigns an activity name to an event.

Definition 1.1.2 (Trace)**.** A *trace* is a non-empty sequence $\sigma = \langle e_1, \ldots, e_{|\sigma|} \rangle$ of events such that for $1 \leq i < j \leq |\sigma| : e_i, e_j \in \mathcal{E}; \pi_{\mathcal{C}}(e_i) = \pi_{\mathcal{C}}(e_j) \wedge \pi_{\mathcal{T}}(e_i) \leq \pi_{\mathcal{T}}(e_j)$, where $|\sigma|$ denotes the length of σ. The universe of all possible traces is denoted by $\mathcal{S}_*$.

In other words, all the events refer to the same case, each event appears only once, and time is non-decreasing. If the timestamps of two events are identical, the order between these events is chosen arbitrarily. We use the notation $\sigma(i)$ to refer to the ith element in σ. We say that a trace is *completed* if the corresponding case has finished, i.e. no additional events related to the given case can occur in the future.

Definition 1.1.3 (Completed trace)**.** A *completed trace* is a trace σ such that there exists no event e' which is not an element of σ, but has the same case id as the events in σ, i.e. $\nexists e' : \pi_{\mathcal{C}}(e') = \pi_{\mathcal{C}}(e_i); e', e_i \in \mathcal{E}; e' \notin \sigma; e_i \in \sigma$. The universe of all possible completed traces is denoted by $\mathcal{S}$.

Definition 1.1.4 (Event log)**.** An *event log* L is a set of completed traces, i.e. $L \subseteq \mathcal{S}$.

Process mining encompasses a wide range of techniques that take as input event logs. Examples of process mining techniques include *automated process discovery*, i.e. deriving interpretable models that describe the flow of executing

activities in the process; *conformance checking*, i.e. checking how well the actual execution of the process is aligned with the intended stucture of the process; *performance analysis*, i.e. identifying bottlenecks in the process; and *deviance mining*, i.e. explaining the causes of "deviant" cases in a process, with respect to a given function that classifies cases into "normal" and "deviant".

1.2. Predictive and prescriptive process monitoring

The process mining techniques mentioned in the previous section (automated process discovery, conformance checking, performance analysis, and deviance mining) are *tactical* in nature, i.e. they help the process stakeholders to assess and improve the process over a relatively long period of time. Conversely, *online process monitoring* techniques are designed to aid the process workers on an *operational* level, with the aim of supporting short-term decision making on a day-to-day basis.

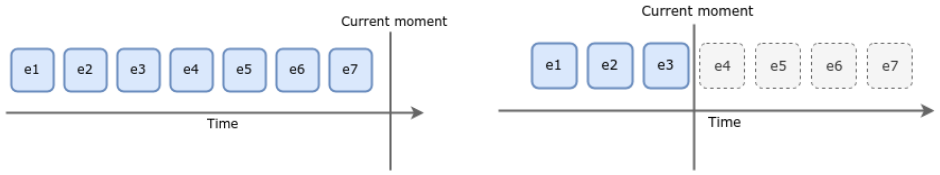
The input to an online process monitor is an *event stream*, i.e. the event records arrive one by one as they are executed. Traditionally, the output of process monitoring methods is in the form of periodically produced reports or dashboards, reporting the (aggregated) *performance* measures of the *running* (i.e. *incomplete*, *ongoing*) instances [12]. Furthermore, *compliance* monitoring aims to check whether an ongoing case is compliant with relevant regulations, constraints, and rules [59].

A family of techniques called *predictive process monitoring* go a step further by trying to *predict* how a running process instance will unfold up to its completion, given only its *prefix*, i.e. an *incomplete* (or *running*) trace containing the sequence of events that are available for a running case at a given point in time (a completed and a running trace are illustrated in Figure 1). In other words, we aim at making predictions for incomplete cases, rather than for completed cases. Therefore, we make use of a *prefix function* which extracts a prefix of a given length from a given trace.

Definition 1.2.1 (Prefix function)**.** Given a trace $\sigma = \langle e_1, \ldots, e_{|\sigma|} \rangle$ and a positive integer $k \leq |\sigma|$, the prefix function $hd^k : \mathcal{S}_* \to \mathcal{S}_*$ returns an (incomplete) trace corresponding to the first k events of σ: $hd^k(\sigma) = \langle e_1, \ldots, e_k \rangle$. For example, $hd^2(\langle a,b,c,d,e \rangle) = \langle a,b \rangle$.

In the context of predictive monitoring, one can think of several different prediction targets that are important from the business perspective, e.g. the *remaining time* until the completion of the case [81], the *next activity* that will be performed in the given case [28, 79, 93], or the final *outcome* of a case [60, 63, 64]. The latter task, called *outcome-oriented predictive process monitoring*, is the core topic of this thesis.

The outcome of a case can be defined in various ways depending on the business goals. The outcome can be thought of as a categorical variable consisting of



(1) A completed trace. All 7 events have been observed.

(2) An incomplete (running) trace. Only 3 events have been observed so far, while the future events remain unknown.

Figure 1: Illustration of a completed and a running trace. In outcome-oriented predictive monitoring, we aim to predict the final outcomes for running traces. However, we can use the historical completed traces in the event log for training the predictive model.

a number of possible values (i.e. *classes*). For example, an order-to-cash process can end with (i) the customer successfully paying for the order and receiving the products, (ii) canceling of the purchase order by the customer, or (iii) canceling of the purchase order by the store. From the business perspective it is usually sufficient to define the outcome as a binary variable with only two possible values reflecting whether the case will finish with a *desired* (*positive*) or an *undesired* (*negative*) result. For instance, a desired outcome in the latter example would be that the customer pays the requested amount and receives the ordered product, while canceling the purchase order (either by the store or by the customer) would be considered an undesired outcome. In the rest of this thesis, we assume that the outcome of a business process is a binary variable.

The outcome can also be defined as meeting a performance target, e.g. delivering the product on time (with respect to a maximum acceptable delivery time) vs. delivering the product late. However, note that outcome-oriented predictive monitoring techniques are orthogonal to those of remaining time prediction techniques, which are widely studied in the literature (see, e.g. [107] for a survey of these methods). In this respect, the problem of outcome-oriented process monitoring is also distinct from survival analysis. In particular, we are interested in predicting *what* the outcome will be, rather than predicting *when* the outcome will be known. Therefore, outcome-oriented process monitoring techniques are not concerned with the timestamps of the events other than to the extent that these may be predictive of the outcome.

The *class label*, expressing the outcome of a (completed) trace, can be determined according to a *labeling function*.

Definition 1.2.2 (Labeling function)**.** A labeling function $out : \mathcal{S} \rightarrow \mathcal{Y}$ is a function that maps a completed trace σ to its class label $out(\sigma) \in \mathcal{Y}$ with $\mathcal{Y}$ being the domain of class labels. For outcome predictions, $\mathcal{Y}$ is a finite set of categorical outcomes; specifically, for binary outcomes $\mathcal{Y} = \{0, 1\}$.

The problem of outcome-oriented predictive process monitoring can be posed as a *classification* task, where the input is a prefix of a sequence of events corresponding to a running case and the goal is to predict the corresponding class label

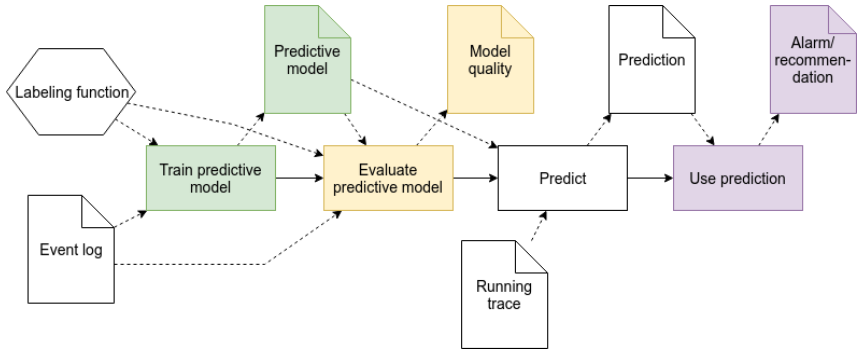


Figure 2: Predictive and prescriptive process monitoring.

(the final outcome). This task is commonly solved by training a *predictive model* (*predictor* for short) using *machine learning* algorithms, based on data from historical traces available in an event log and a predefined labeling function. The model is evaluated on a test set, comprising of a distinct set of historical traces from the event log. If the model quality is acceptable for the process stakeholders, the model is deployed for monitoring the ongoing cases. At runtime, the predictive model is applied to a running trace, producing a prediction about its final outcome. The process worker can use the prediction to decide whether to *intervene* in the running case with the aim of avoiding an undesired outcome. Alternatively, the prediction can be given to a *prescriptive monitoring system* that advises the user if an intervention is necessary and/or which intervention actions to take. This process is illustrated in Figure 2.

1.3. Problem statement

The thesis addresses the question of "How to train, evaluate, and use machine learning models in the context of outcome-oriented predictive and prescriptive business process monitoring?".

The above question has been tackled by several research teams in the past years, resulting in a rich field of outcome-oriented predictive process monitoring methods. Even though these methods serve a common goal, different authors have used different datasets, experimental settings, evaluation measures, and baselines, resulting in a situation with no clear overview of how the different techniques compare to each other methodologically and experimentally. This thesis addresses this gap by: (i) performing a systematic literature review of outcome-oriented predictive process monitoring methods; (ii) providing a taxonomy of the existing methods; (iii) constructing a benchmark of 24 outcome-oriented predictive monitoring tasks based on nine real-life event logs; and (iv) performing a comparative experimental evaluation of the existing methods using this benchmark.

The conducted survey (Chapter 3) reveals further gaps in the existing literature. In particular, the existing approaches focus on:

- training predictive models on structured data, lacking support for unstructured data;
- evaluating the predictive models in terms of accuracy and earliness, lacking attention to stability of the sequential predictions made for a given case;
- generating predictions, without prescribing a particular course of action to prevent negative outcomes and without taking into account the cost and the effect of such actions.

In the following paragraphs, these gaps are described in more detail.

Training predictive models on only structured data. Existing approaches assume that the event records carry only *structured* data payload, i.e. the attributes are assumed to be either of numeric or categorical type. In practice, not all data generated during the execution of a process is structured. For instance, in an order-to-cash process the customer may include a free-text description of special requests. Later, a customer service representative may attach to the case the text of an email exchanged with the customer regarding delivery details, or add a comment to the purchase order following a conversation with the customer. Comments like these ones are even more common in application-to-approval, issue-to-resolution, and claim-to-settlement processes, where the execution of the process involves many unstructured interactions with the customer.

Evaluating the predictive models in terms of accuracy and earliness. Traditionally, methods for outcome-oriented predictive process monitoring aim to make predictions as *accurately* and as *early* (i.e. given only a few event records) as possible. Oftentimes, accuracy is evaluated separately for prefixes of different lengths, allowing one to estimate the expected accuracy of a given prediction, knowing the number of events observed so far. Based on this information, the process worker could decide whether to make a decision now or to postpone until observing another event in the hope of getting a more accurate prediction. However, this evaluation scheme exploits only a limited amount of information that is available at a given time. In particular, at each evaluation point the process worker is expected to decide based on only the latest prediction available for a given case, neglecting the sequential nature of predictive monitoring. Namely, in a setting where the predictive model is applied to a running case successively (after each observed event), a sequence of predictions is produced. Therefore, the process worker could make a more informed decision by using not only the latest prediction, but also the predictions made at earlier stages of the given case. In this context, it becomes relevant to evaluate also the *stability* of the predictions, in order to give the process workers some estimation of how reliable a given prediction is.

Generating predictions without advice on using them. While existing techniques aim to predict, after each event of a case, the probability that the case will end up in an undesired outcome, they do not suggest nor prescribe when and how process workers should intervene in order to decrease the probability of undesired

outcomes. Indeed, existing proposals implicitly assume that the users (analysts, managers, or process workers) are able to manually choose the most suitable accuracy or confidence threshold for their scenario and act upon predictions that reach this threshold. In practice, the optimal threshold depends on many factors, such as the different costs involved in the execution of the process, as well as the scale of the probability scores that the predictive model produces, making it difficult to manually come up with a suitable threshold.

1.4. Contributions and outline

The thesis makes four contributions to the field of predictive and prescriptive process monitoring as described below.

Contribution 1: Comparing and evaluating existing predictive process monitoring methods. We propose a taxonomy of existing methods for training predictive models in the context of outcome-oriented predictive process monitoring (Chapter 3). We perform a comparative experimental evaluation of existing outcome-oriented predictive process monitoring methods. First, we construct a benchmark of 24 predictive monitoring tasks based on 9 real-life event logs. We then evaluate 11 representative methods identified in the literature review, using the benchmark (Chapter 4).

Contribution 2: Training predictive models with structured and unstructured data. We propose a framework that combines text mining techniques to extract features from textual payload, with existing predictive process monitoring techniques for structured data (Chapter 5). We perform a comparative experimental evaluation of several text mining techniques in combination with different predictive process monitoring methods.

Contribution 3: Evaluating the temporal stability of predictive models. We introduce the notion of *temporal stability* of predictions, propose a metric for measuring it, and evaluate existing predictive monitoring techniques with respect to this metric (Chapter 6). Furthermore, we apply a sequential smoothing technique to the series of predictions made for a given case, in order to decrease the volatility of the predictions and produce more stable estimates as compared to using only the latest available predictions.

Contribution 4: Using predictions for prescriptive process monitoring. We propose a framework that extends predictive process monitoring techniques with an *alarm*-generating mechanism that advises the process workers if it is time to act upon the prediction (Chapter 7). The proposed framework is armed with a parameterized cost model that captures, among others, the tradeoff between the cost of an intervention and the cost of an undesired outcome. Based on this cost model and the prediction produced by a predictive model, the alarming mechanism decides whether to raise an alarm or not. If an alarm is raised, a process worker is expected to intervene in the running case with the goal of mitigating (or altogether preventing) an undesired outcome. We propose and empirically evaluate an ap-

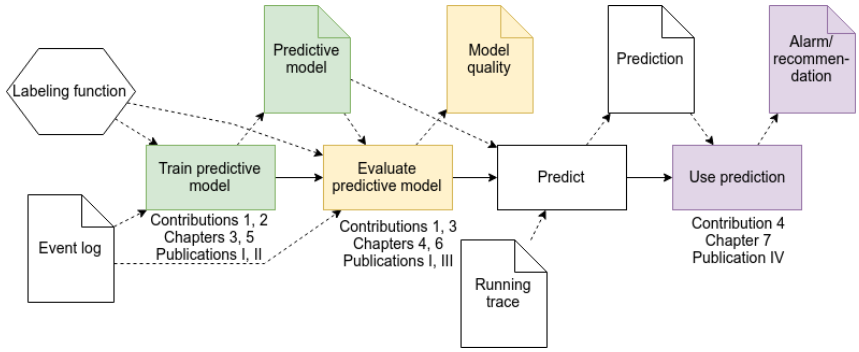


Figure 3: Mapping of the contributions, chapters, and publications.

proach to tune the generation of alarms to minimize the expected cost for a given dataset and set of parameters.

The above contributions have been previously documented in publications I-IV referenced at the end of the thesis (see “List of original publications”). Figure 3 illustrates the mapping between the contributions, the chapters, and the publications.

The rest of the thesis is structured as follows. In Chapter 2 we introduce the relevant concepts and principles from machine learning. Chapter 3 presents the systematic literature review and a taxonomy for existing methods. In Chapter 4 we construct the benchmark and perform an experimental evaluation of the existing methods. Chapter 5 proposes and evaluates a framework for combining structured and unstructured data for predictive process monitoring. Chapter 6 introduces the notion of temporal stability and evaluates the existing methods with respect to the proposed metric. Chapter 7 proposes a prescriptive process monitoring framework for generating alarms based on the output of predictive models. Chapter 8 concludes the thesis and outlines directions for future work.

2. BACKGROUND

In this chapter, we explain the relevant concepts from the machine learning field. We start with describing different types of tasks in machine learning. Then, we introduce evaluation measures and discuss the best practices regarding the experimental settings in classification tasks. We proceed with an overview of classification algorithms that are used later in the thesis. We conclude the chapter with a brief overview of works on early sequence classification.

2.1. Machine learning

Machine learning is a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty [68].

Machine learning tasks can be divided into three categories. Firstly, in *supervised learning* the aim is to learn a function from inputs $\mathbf{x}$ to outputs y, where $\mathbf{x} = (x_1, ..., x_p)$ is a p-dimensional vector of *features* (or *attributes*) and y is the *target* (or *response*) variable. In essence, each feature in $\mathbf{x}$ can be of numeric or categorical type. However, many machine learning algorithms assume that categorical attributes are transformed into numeric values using, for instance, *one-hot encoding*, where each value of a categorical attribute is transformed into a bitvector $(v_1, \cdots, v_m)$, where m is the number of possible *levels* (i.e. unique values) of that attribute, $v_i = 1$ if the given value is equal to the ith level of the attribute, and $v_i = 0$ otherwise. In order to learn the mapping from $\mathbf{x}$ to y, a machine learning algorithm is given as input a *training set* $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, consisting of *N training examples* (or *training instances*) ($\mathbf{x}^{(i)}$, $y^{(i)}$), i.e. feature vectors paired with their corresponding target variables. A supervised learning algorithm learns a *predictive model* that can consequently be used to *predict* (or *estimate*) the value of $\hat{y}$ given a vector $\mathbf{x}$ that was not part of the training set. Supervised learning tasks can be further divided into *classification*, where y is assumed to be a categorical variable (a class label), and *regression*, where y is a real-valued numeric variable.

Conversely from supervised learning, where the algorithm is told which types of patterns to look for (i.e. those that explain the mapping between the input features and the target variable), the task of *unsupervised learning* techniques is to look for any interesting patterns in the data. Namely, these methods take as input a set of feature vectors $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ without any corresponding target variables. A representative of unsupervised learning techniques is *clustering*, where the aim is to divide the inputs $\mathbf{x}$ into (possibly overlapping) groups (*clusters*) $c_1, \ldots, c_k$, where each cluster c_j, $1 \leq j \leq k$ consists of instances $\mathbf{x}$ that are (in some sense) similar to each other. A well-known clustering algorithm is *k-means* where the input space is partitioned into k clusters, each characterized by a *centroid* (a vector of coordinates) $\mu_j, 1 \leq j \leq k$, and each observation is assigned to the cluster j such that μ_j is the closest centroid to the given observation.

Table 2: Confusion matrix.

		Predicted	
		Positive	Negative
Actual	Positive	# true positives (TP)	# false negatives (FN)
	Negative	# false positives (FP)	# true negatives (TN)

The third type of techniques called *reinforcement learning* is concerned with a setting where an *agent* is placed in an *environment* where it must decide which (sequence of) *actions* to take. As a result of an action, the agent arrives to a *state* and observes a *reward*. Reinforcement learning algorithms aim to learn an *optimal policy* for the agent to take in order to maximize the cumulative reward.

Since this thesis is concerned with predicting the outcomes of business processes, which corresponds to a classification task in machine learning, hereinafter, we focus the discussion on classification methods.

2.2. Evaluation measures and experimental settings

After training a classification model, it is important to assess if the obtained model is good for making predictions. For that purpose, different evaluation measures can be used to evaluate the model's performance on a *test set*. In particular, the model is asked to predict the label for each *test example* in the test set and the predictions are compared to the corresponding *ground truth* (i.e. the actual) class labels. In this section we first describe some commonly used evaluation measures for assessing the quality of a classifier. Then, we discuss some best practices related to experimental settings in machine learning. Hereinafter, we focus on binary classification tasks, i.e. $y \in \{0,1\}$, or $y \in \{negative, positive\}$.

2.2.1. Evaluation measures

An intuitive way to get insights about the performance of a classifier is by constructing a *confusion matrix* (see Table 2). Each cell in a confusion matrix refers to the number of test examples that fall into a particular combination of the predicted and the actual outcome. *True positives (TP)* are the test examples where the actual outcome is positive and the model correctly predict the positive class. *True negatives (TN)* are negative test examples that are correctly classified as negative. *False positives (FP)* refer to test examples where the actual outcome is negative, but the model incorrectly predicts the positive class. *False negatives (FN)* are cases where the actual outcome is positive, but the model incorrectly predicts it is negative.

Based on these concepts, several commonly used evaluation measures can be defined, such as:

$$ACC \text{ (Accuracy)} = (TP+TN)/(TP+TN+FP+FN)$$

$$Precision = TP/(TP+FP)$$

$$Recall = TPR \text{ (True Positive Rate)} = TP/(TP+FN)$$

$$F\text{-}score = 2 \cdot Precision \cdot Recall/(Precision+Recall)$$

$$FPR \text{ (False Positive Rate)} = FP/(TN+FP)$$

Probably the most simple and widely used evaluation metric is ACC, measuring the overall proportion of correctly classified instances. However, in case the classes are *imbalanced*, e.g. when there are many more negative instances than positive ones, ACC would give a high score to a classifier that always predicts the negative class, while in reality it would be much more important to correctly classify the rare positive examples. In these cases, it is recommended to use Precision and Recall, where the first measures the proportion of positive predictions that are correct and the latter measures the proportion of all positives that are identified (predicted) by the classifier. Note that a predictor that always predicts the positive class would achieve perfect Recall, but low Precision. In fact, these two measures are often reported together, as they measure different aspects of prediction quality that complement each other. Alternatively, Precision and Recall can be combined into a single metric called F-score, which is the harmonic mean of these two measures. Another pair of metrics that complement each other are Recall (in this context, often called TPR) and FPR, where the latter measures the proportion of all negatives that are incorrectly predicted as positive.

All of these measures assume that the classifier outputs a hard prediction (a binary number) of the class label. However, classifiers often output a real-valued *prediction score* instead, reflecting either the probability or the classifier's confidence that the case will end in one way or the other. A good classifier would give higher scores to cases that will end with a positive outcome, and lower values to those ending with a negative one. In order to use the evaluation measures defined above, a *decision threshold* τ needs to be set on the prediction scores, so that predictions larger than τ would be considered positive predictions and predictions smaller than τ, negative predictions. Often it is assumed that $\tau = 0.5$, but in applications where the costs related to different types of misclassification errors are asymmetric (e.g. when the cost of FP is much higher than the cost of FN), it might be reasonable to increase or decrease τ accordingly. Furthermore, one might decide to adjust the threshold given the fact that prediction scores returned by classifiers are often poorly calibrated, meaning that the scores do not reflect well the actual probabilities of belonging to one class or to the other [69].

Another common technique for evaluating a classifier that outputs real-valued prediction scores is to construct a Receiver Operating Characteristic (ROC) curve, where TPR and FPR over all possible decision thresholds are plotted against each other (see Fig. 4). In other words, each point in the ROC space corresponds to a

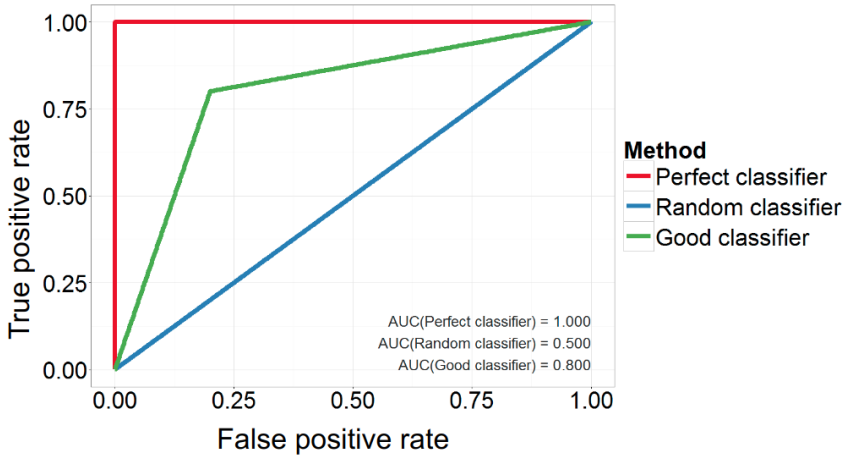


Figure 4: Example ROC curves.

pair of (FPR, TPR) given a specific threshold τ. The area under the ROC curve (AUC) is often used to express the information from a ROC curve as a single performance measure. AUC can also be thought of as the probability that a given classifier will rank a positive case higher than a negative one. A major advantage of the AUC metric over ACC and F-score is that it remains unbiased even in case of a highly imbalanced distribution of class labels [7]. Furthermore, AUC is a threshold-independent measure, as it operates on the ranking of the prediction scores rather than on the binary (predicted) class labels. Note that a random classifier (a diagonal line in the ROC space) would yield $AUC = 0.5$, while a perfect classifier (the line that crosses the coordinates where $FPR = 0$, $TPR = 1$) corresponds to $AUC = 1$.

2.2.2. Model selection and generalization

As mentioned earlier in this chapter, the goal in supervised learning is to train a model that can later be used to predict the class label for unseen examples. Predicting the class label for already seen examples would be trivial, because the model could simply memorize the training data and look up the corresponding class label. Therefore, it is important that the model *generalizes* to instances that were not part of the training set. To evaluate this, the available data is divided into two independent subsets: a training set and a test set. This splitting approach is called the *holdout* method, since the test set is held out from the training process and only used for evaluating the model's generalization performance. The most common way to split the data is via random sampling. Often, *stratification* is used in combination with random sampling, so that the class label proportions observed in the original data set are preserved in the training and test sets.

Classification algorithms typically have several *hyperparameters* which control the complexity of the resulting model and need to be specified manually by

the analyst instead of being learned automatically. On the one hand, a very complex model is able to memorize the whole training set, but is unable to generalize well to the test set, resulting in a situation called *overfitting*. On the other hand, an overly simplistic model is not able to capture the underlying patterns, resulting in an *underfitting* model. Since both over- and underfitting yield low accuracy on the test set, we are typically interested in *tuning* (or *optimizing*) the model's hyperparameters in a way that yields a model with an optimal generalization performance. In order to achieve that, one can train multiple models, each with a specific configuration of hyperparameters, evaluate each model's performance on a test set, and select the configuration with the best performance according to some evaluation measure.

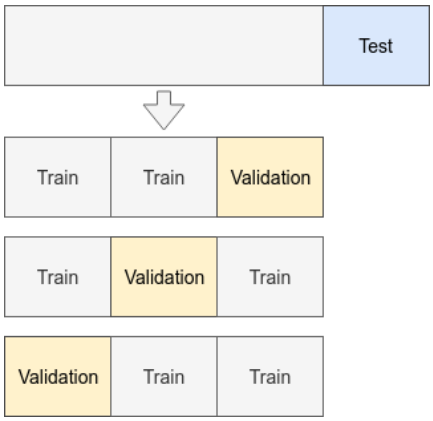
However, it is not a valid approach to use the same test set for performing the model selection and evaluating the generalization performance of the (best) model, since this would result in an overly optimistic estimation of the generalization performance. Therefore, a *three-way holdout* method can be used, splitting the data into three independent subsets: a training, a validation, and a test set. This way, the training and validation sets can be used to test different hyperparameter settings and select the best hyperparameters. Then, the training and validation sets can be concatenated together and the *final model* can be trained on the best parameters using this combined set. Finally, the generalization performance of the final model can be evaluated on the independent test set.

A drawback of the holdout method is that by partitioning the data into independent subsets we reduce the amount of training instances, which can result in a less accurate model. This is especially an issue with small datasets, where the number of training instances could become too low to learn a reasonable model. To alleviate this problem, a method called *k-fold cross-validation* splits the data into k independent chunks and builds k models with the same hyperparameter configurations, so that each model is trained on $k-1$ chunks of the data and tested on the remaining chunk. In other words, each example from the original data is used exactly once for testing and $k-1$ times for training. The performance scores from the k folds are averaged into a single evaluation score.

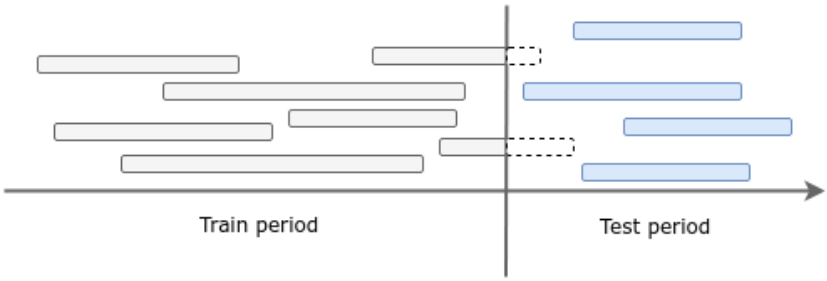
In general, the splitting methods described above make use of random sampling of the data. However, when the data is of temporal nature it is important to ensure that future data is not used for predicting the past. In such cases, a *temporal (holdout) split* can be used, by dividing the data into training and test sets according to the timestamps related to the instances. In particular, the training set would consist of the examples originating from the period up to a given time t and the test set would contain the examples that originate from the period after t. Special care needs to be taken when the data is in the form of sequences, since it is possible that a sequence starts before the splitting timestamp t, but ends after this time. One option in such cases is to discard all the sequences that overlap with both the training and testing periods. However, since this approach wastes (a possibly large) part of the available data, an alternative approach is to discard only

(1) Two-way holdout split.

(2) Three-way holdout split.



(3) Combined holdout split for test set and 3-fold cross-validation for model selection.



(4) Temporal holdout split for sequences.

Figure 5: Illustration of different splitting strategies.

the parts of the sequences that overlap with the test period. The different splitting strategies are illustrated in Figure 5.

A variety or approaches exist for choosing the hyperparameter configurations that are tested during the model selection phase. A common approach is *grid search*, where the analyst is expected to specify for each hyperparameter a set of values and all possible combinations of these values are tested. However, it has been shown that *random search*, where the analyst needs to specify only the ranges and sampling distributions for each hyperparameter rather than a specific set of values, is more efficient than grid search [3]. This is particularly the case in high-dimensional parameter spaces, where grid search tends to allocate too many trials to exploring unimportant dimensions, while random search results in an overall better coverage of the search space. In each iteration of random search, a value is randomly sampled for each hyperparameter from the specified distributions and the resulting configuration of hyperparameters is tested. Both grid search and random search are outperformed by *adaptive* (or *sequential*) hyperparameter optimization techniques, which, similarly to random search, require the

analyst to specify only the ranges and the sampling distributions. Conversely from random search, these techniques perform optimization in a sequential manner, so that in each iteration a configuration of hyperparameters is chosen that appears to be promising given the previously tried settings. A representative of adaptive hyperparameter optimization techniques is Tree-structured Parzen Estimator (TPE) [2].

2.3. Classification algorithms

In this section, we give an overview of the classification algorithms used later in this thesis.

k-nearest neighbors. One of the simplest classification algorithms, the *k*-nearest neighbor (KNN) method makes predictions by retrieving the k training instances that are closest to the input instance according to some similarity measure. The prediction is made by averaging (or taking the mode) of the class labels corresponding to these k training examples. Contrary to the other methods described in this section, KNN is a *lazy learning* (or *instance-based learning*) algorithm, meaning that there is no model training stage. Instead, the algorithm uses the training set only at prediction time. The number of considered neighbors k is a hyperparameter of the method.

Logistic regression. Logistic regression is one of the simplest classification algorithms, which learns to model the prediction target through a linear combination of the input features. In order to make a prediction for $\mathbf{x} = (x_1, ..., x_p)$, the model calculates a weighted sum of the input features $x_1, ..., x_p$, i.e. $\hat{y} = b + \sum_{j=1}^{p} w_j \cdot x_j$, where w_j are the learnable *weight coefficients* corresponding to input features x_j and b is the *bias* (or *intercept*) term. Then, this sum is passed through a *logistic* (or *sigmoid*) function, which ensures that the output is between 0 and 1 as needed for binary classification. The weight coefficients and the bias term are learned by optimizing a *loss function* (or *cost function*), most commonly *logistic loss* (also called *cross-entropy loss*). Often, the *gradient descent* optimization algorithm is used to iteratively update the weights. In order to avoid overfitting, a *regularization* term is often added to the loss function, for instance, as a sum of the absolute values of the weights (L1 regularization) or as the sum of their squares (L2 regularization). Optimizing the weights using an L1 regularization term results in a sparser weight vector (a smaller number of non-zero weights) in comparison to an L2 term. The regularization strength in a logistic regression model can be controlled with a hyperparameter.

Neural network. A neural network consists of one or more *(hidden) layers* of *neurons* (or *units*), where each neuron calculates a weighted sum of its inputs and passes it through an *activation function*, such as the sigmoid function. The resulting *activations* constitute the inputs for the neurons in the next layer. The weights in the neurons are learned through an optimization procedure, commonly using

stochastic gradient descent (SGD). The updates in SGD are calculated based on *batches* of training instances, i.e. subsets of training data of size $b, 1 \leq b < N$, where the subsets are constructed via random sampling (without replacement) over the training set. A complete pass over the training data (consisting of $\lceil N/b \rceil$ batch updates) is called an *epoch*; the SGD procedure is commonly performed over multiple epochs. In order to model sequences and time series data, a special *recurrent* architecture of neural networks can be used, where at each timestep t, the neurons take as input both the feature vector $\mathbf{x}^{(i)<t>}$ and the activation from $t-1$. Such recurrent neural networks are, however, known to be unable to model long-term dependencies in sequential data, caused by the *vanishing gradient problem*. In order to solve this issue, special types of neurons have been developed, such as *long short-term memory* (LSTM) [43] units, which learn to control the flow of information from the recent and the earlier timesteps via *input, output* and *forget gates*. The hyperparameters of neural networks are the number of layers, the number of neurons in each layer, the learning rate (controlling the step size in the SGD updates), the batch size, the number of epochs, and a variety of regularization parameters, e.g. *dropout*, L1 and L2 regularization.

Support vector machine. A support vector machine (SVM) tries to find a *hyperplane* separating the two classes with a *maximum margin*. Specifically, SVMs are commonly trained by optimizing *hinge loss*, which is zero for correctly classified instances and proportional to the distance between the given example and the margin otherwise. By default, SVMs perform linear classification, similarly to logistic regression. However, SVMs can easily be extended to non-linear classification using the *kernel trick*, i.e. mapping the inputs into high-dimensional feature spaces using a *kernel function*. The model complexity in SVM can be controlled with a penalty parameter C, where a low C places more weight on obtaining a larger margin (a smoother decision surface), while a high C aims at classifying all training examples correctly [74]. Other hyperparameters depend on the chosen kernel function; for instance, Radial Basis Function requires setting a coefficient *gamma* for controlling how much influence a single training example has. By default, SVMs return a binary prediction instead of a real-valued prediction score. In order to calculate probabilistic prediction estimates, Platt scaling can be used, which fits a logistic regression model to the SVM outputs [76].

Decision tree. A decision tree (DT) learning algorithm recursively splits the input space, aiming for subsets that have high purity in terms of the class label. The resulting model can be represented as a tree structure, where each internal node represents a splitting condition used to determine the branch where a given instance belongs to. When a leaf is reached, a prediction is made by aggregating the class labels of the training instances that fall into the same leaf. Decision trees are widely used in practice thanks to their simplicity and interpretability. A decision tree can easily be transformed into a set of *decision rules* in "if-else" format. Decision trees are prone to overfitting, which can be mitigated by *pruning* the tree, i.e. by reducing the size of the tree by removing parts that add little to the

predictive power of the model. Hyperparameters such as the maximum tree depth or the minimum number of instances needed to create a new node can be used to limit the complexity of a decision tree.

Random forest. The random forest (RF) [9] algorithm constructs an *ensemble* of decision trees via *bagging*. Namely, for a number of m times, the following procedure is repeated: 1) a subset of training instances is randomly sampled, with replacement, from the complete training set and 2) a decision tree is built on the sampled subset, so that for each split a random sample of features is considered. At prediction time, the input is passed through all of the constructed decision trees and the prediction is made as the average or the mode of the individual classifiers' outputs. The RF algorithm greatly mitigates the problem of overfitting compared to using a single decision tree. The hyperparameters of the method are the number of iterations (i.e. the number of built decision trees) m and the number (or the proportion) of features to consider for each split. Additionally, hyperparameters related to the underlying decision trees can be set, such as the maximum depth of the trees. The size of the sampled subset of training instances is often kept the same as the original dataset size. Note that due to sampling with replacement, these subsets are different from the original dataset because they can contain some training instances multiple times, while some other instances might be missing.

Gradient boosted trees. Similarly to RF, the gradient boosted trees algorithm (GBT) [31] constructs an ensemble of decision trees and the prediction is made as the average of the individual trees' outputs. However, while RF builds decision trees in a parallel manner, GBT constructs the trees sequentially via *boosting*, i.e. the decision tree at step m aims at correcting the mistakes made by the tree from step $m-1$. In each iteration, the *residuals* (the differences between the predicted and the actual labels) are calculated and the next tree is fitted on these residuals, resulting in a boosted version of the previous model. Compared to RF, GBTs are more prone to overfitting and more sensitive to the selection of hyperparameters. Common hyperparameters to tune for GBTs include the number of boosting iterations, the learning rate (controlling the weight of each added tree), the proportion of training instances to sample in each boosting iteration, the proportion of features to sample in each boosting iteration, the maximum depth of the individual decision trees, and the minimum number of instances needed in each node. A well-known library that provides a very efficient implementation of GBT is called XGBoost [15] (stands for "extreme gradient boosting").

2.4. Early sequence classification

With respect to the broader literature on machine learning, we note that predictive process monitoring corresponds to the problem of *early sequence classification* [109]. In other words, given a set of labeled sequences, the goal is to build a model that for a sequence prefix predicts the label this prefix will get when completed.

The works on early sequence classification are generally focused on determining a prefix length that yields a good prediction, also referred to as the *minimal prediction length* (MPL) [111]. The specific criteria for determining the MPL differs in the literature. For instance, Xing et al. [109] introduced the notion of *seriality* in sequence classifiers, referring to the property that for each sequence, there exists a prefix length starting from which the classifier outputs (almost) the same prediction. Another method by Xing et al. [111] finds the earliest timestamp when the nearest neighbor relationships in the training data become stable (i.e. remain the same in the subsequent prefixes). Parrish et al. proposed a method based on the *reliability* of predictions, i.e. the probability that the label assigned to a given prefix is the same as the label assigned to the whole sequence [73]. More recently, Mori et al. [67] designed an approach to make an early prediction when the ratio of accuracy between the prediction made for the prefix and for the full sequence exceeds a predetermined threshold.

While there is substantial literature on the problem of (early) sequence classification for simple symbolic sequences (e.g. sequences of events without payloads), there is a lack of proposals addressing the problem for complex symbolic sequences (i.e. sequences of events with payloads) [83, 110]. The problem of outcome-oriented predictive process monitoring can be seen as an early classification over complex sequences where each element has a timestamp, a discrete attribute referring to an activity, and a payload made of a heterogeneous set of other attributes. One of the few works on early classification on complex sequences is [55], where Lin et al. propose constructing *serial decision trees* and monitor the error rate in leaf nodes in order to determine the MPL.

3. LITERATURE REVIEW

The purpose of the survey conducted in this chapter is to define a taxonomy of methods for training predictive models for outcome-oriented predictive monitoring of business processes and to give a structured overview of existing approaches for evaluating and deploying such models. The decision to focus on outcome prediction 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 rely on entirely different techniques and evaluation measures.

In line with the selected scope, the survey aims at answering the following research questions:

RQ1 (Existing training methods) Given an event log of completed business process execution traces and the final outcome (class) of each trace, what methods exist for training a model that can accurately and efficiently predict the outcome of an incomplete (running) trace, based on a given prefix only?

RQ2 (Taxonomy) How to categorize these training methods in a taxonomy?

RQ3 (Evaluation procedures) What approaches exist for assessing the quality of predictive models built for outcome-oriented predictive process monitoring?

RQ4 (Deployment use cases) What use cases exist for deploying the predictive models built for outcome-oriented predictive process monitoring?

In Section 3.1, we describe our approach to identifying existing methods for outcome-oriented predictive process monitoring. In Section 3.2, we analyze (RQ1) and categorize (RQ2) the existing training methods. As the choice of quality measures for evaluating the models depends on the deployment use case, we first analyze the existing use cases (RQ4) in Section 3.3 and continue with the evaluation procedures (RQ3) in Section 3.4. The chapter is concluded with a discussion on the threats to validity of the conducted survey in Section 3.5.

3.1. Search methodology

In order to retrieve and select studies for the survey, we conducted a *Systematic Literature Review* (SLR) according to the approach described in [48]. We started by developing relevant search strings for querying a database of academic papers (Section 3.1.1). We then applied inclusion and exclusion criteria to the retrieved studies in order to filter out irrelevant ones (Section 3.1.2). Lastly, we divided all relevant studies into primary and subsumed ones based on their contribution (Section 3.1.3).

3.1.1. Study retrieval

First, we selected keywords that are relevant to outcome-oriented predictive process monitoring:

- "(business) process" — a relevant study must take as input an event log of business process execution data;
- "monitoring" — a relevant study should concern runtime monitoring of business processes, i.e. work with incomplete (running) traces;
- "prediction" — a relevant study needs to estimate what will happen in the future, rather than monitor what has already happened.

We deliberately left out "outcome" from the set of keywords. The reason for this is that we presumed that different authors might use different words to refer to this prediction target. Therefore, in order to obtain a more exhaustive set of relevant papers, we decided to filter out studies that focus on other prediction targets (rather than the final outcome) in an a-posteriori filtering phase.

Based on these selected keywords, we constructed three search phrases: "predictive process monitoring", "predictive business process monitoring", and "business process prediction". We applied these search strings to the Google Scholar academic database and retrieved all studies that contained at least one of the phrases in the title, keywords, abstract, or the full text of the paper. We used Google Scholar, a well-known electronic literature database, as it encompasses all relevant databases such as ACM Digital Library and IEEE Xplore, and also allows searching within the full text of a paper.

The search was conducted in August 2017 and returned 93 papers, excluding duplicates.

3.1.2. Study selection

All the retrieved studies were matched against several inclusion and exclusion criteria to further determine their relevance to predictive outcome-oriented process monitoring. In order to be considered relevant, a study must satisfy all of the inclusion criteria and none of the exclusion criteria.

Inclusion criteria. The inclusion criteria are designed for assessing the relevance of studies in a superficial basis. Namely, these criteria are checked without working through the full text of the paper. The following inclusion criteria were applied to the retrieved studies:

IN1 The study is concerned with predictions in the context of business processes (this criterion was assessed by reading title and abstract).

IN2 The study is cited at least five times.

The application of these inclusion criteria to the original set of retrieved papers resulted in 8 relevant studies. We proceeded with one-hop-snowballing, i.e. we retrieved the papers that are related to (cite or are cited by) these 8 studies and applied the same inclusion criteria. This procedure resulted in 545 papers, of which we retained 72 unique papers after applying the inclusion criteria.[1]

[1] All retrieved papers that satisfy the inclusion criteria can be found at `http://bit.ly/2uspLRp`

Exclusion criteria. The list of studies that passed the inclusion criteria were further assessed according to a number of exclusion criteria. Determining if the exclusion criteria are satisfied could require a deeper analysis of the study, e.g. examining the approach and/or results sections of the paper. The applied exclusion criteria are:

EX1 The study does not actually propose a predictive process monitoring *method*.

EX2 The study does not concern *outcome*-oriented prediction.

EX3 The technique proposed in the study is tailored to a specific labeling function.

EX4 The study does not take an *event log* as input.

The EX1 criterion excludes overview papers, as well as studies that, after a more thorough examination, turned out to be not focusing on predictive process monitoring. EX2 excludes studies where the prediction target is something other than the final outcome. Common examples of other prediction targets that are considered irrelevant to this study are remaining time and next activity prediction. Using EX3, we excluded studies that are restricted to specific labeling functions rather than being applicable to any definition of an (categorical) outcome. For example, studies that predict deadline violations by means of setting a threshold on the predicted remaining time, rather than by directly classifying the case as likely to violate the deadline or not. The reason for excluding such studies is that, in essence, they predict a numeric value, and are thus not applicable for predicting an arbitrarily defined case outcome. EX4 concerns studies that propose 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 (event classes). 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 heterogenous lengths are different from those when predicting for linear processes. While methods designed for heterogenous 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.

The application of the exclusion criteria resulted in 16 relevant studies out of the 72 studies selected in the previous step.

3.1.3. Primary and subsumed studies

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

Table 3: Primary and subsumed studies.

Primary study	Subsumed studies
de Leoni et al. [20]	de Leoni et al. [19]
Maggi et al. [60]	
Grigori et al. [37]	Grigori et al. [36], Castellanos et al. [13]
Schwegmann et al. [85]	Schwegmann et al. [86]
Lakshmanan et al. [52]	
Conforti et al. [17]	Conforti et al. [16]
Di Francescomarino et al. [30]	
Leontjeva et al. [54]	
van der Spoel et al. [104]	
Verenich et al. [106]	
Ghattas et al. [34]	

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 11 primary and 5 subsumed studies, listed in Table 3. In the next section, we present the primary studies in detail, and classify them using a taxonomy.

3.2. Analysis and taxonomy of the training methods

In this section, we present a taxonomy to classify the 11 primary studies that we selected through our SLR. Specifically, with this section we aim at answering RQ1 (What training methods exist?) and RQ2 (How to categorize them?) – cf. beginning of Chapter 3. The taxonomy is framed upon a general workflow for predictive process monitoring, which we derived by studying all the surveyed methods. In the following subsections, we first introduce the concepts and the workflow for outcome-oriented predictive process monitoring, proceed with characterizing the existing methods with respect to the different steps observable in the workflow, and conclude with a taxonomy for the studied techniques.

3.2.1. General concepts and workflow

The analysis of the surveyed methods revealed that all of the existing techniques operate on some common concepts, which we define below (cf. Chapter 1 for definitions of event, trace, prefix function, etc.).

Predictions are made using a classifier that takes as input a fixed number of independent variables (features) and learns a function to estimate the dependent variable (class label). This means that in order to use the data in an event log as input to a classifier, each trace in the log must be *encoded* as a feature vector.

Definition 3.2.1 (Sequence/trace encoder)**.** A sequence (or trace) encoder $enc : \mathcal{S}_* \to \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ is a function that takes a (running) trace σ and transforms it into a feature vector in the p-dimensional vector space $\mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ with $\mathcal{X}_j \subseteq \mathbb{R}, 1 \leq j \leq p$ being the domain of the j-th feature.

Despite the fact that the aim of a classifier is to estimate a class label, more commonly the classifiers return real-valued scores.

Definition 3.2.2 (Classifier)**.** A probabilistic binary classifier $cls : \mathcal{X}_1 \times \cdots \times \mathcal{X}_p \to [0,1]$ is a function that takes a p-dimensional feature vector as input and returns a real-valued prediction score estimating the probability of the positive class.

The construction (training) of a classifier for outcome-oriented predictive process monitoring is achieved by applying a classifier learning algorithm over a *prefix log*.

Definition 3.2.3 (Prefix log)**.** Given a log L, a *prefix log* L_* is an event log that contains a (sub)set of prefixes of L.

For instance, a prefix log containing all possible prefixes is constructed as $L_* = \{hd^k(\sigma) : \sigma \in L, 1 \leq k \leq |\sigma|\}$.

The prefixes in a prefix log are divided into *buckets* using a *bucketing function* (or *bucketer*) and separate classifiers are trained for each bucket.

Definition 3.2.4 (Bucketing function)**.** A bucketing function $buc : \mathcal{S}_* \to \mathbb{N}$ is a function that takes a (running) trace σ and assigns it to an integer i, corresponding to bucket $b_i, 1 \leq i \leq B$, where B is the number of buckets. In other words, $b_i = \{\sigma : \sigma \in L_*, buc(\sigma) = i\}$.

Given these concepts, we define a predictive model in the context of outcome-oriented predictive process monitoring as follows.

Definition 3.2.5 (Predictive model)**.** A predictive model is a function $\widehat{out} : \mathcal{S}_* \to [0,1]$ that takes a (running) trace σ, assigns it to a bucket b_i using a bucketer buc, encodes σ as a feature vector using a sequence encoder enc, and returns a prediction score using a classifier cls_i corresponding to bucket b_i, i.e. $\widehat{out}(\sigma) = cls_{buc(\sigma)}(enc(\sigma))$.

We refer to a predictive model where all prefixes are assigned to the same bucket as *single classifier*. Conversely, a predictive model where prefixes are assigned to multiple buckets (so that multiple classifiers are trained) is called a *multiclassifier*.

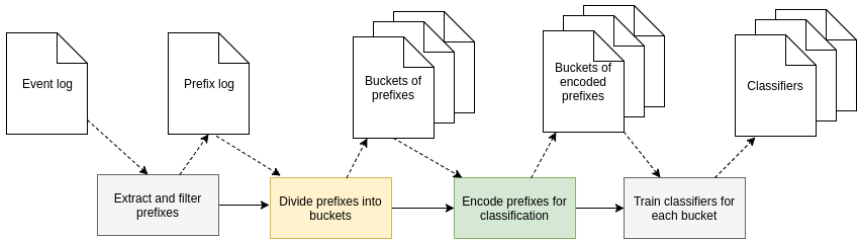


Figure 6: Predictive process monitoring workflow (offline phase).

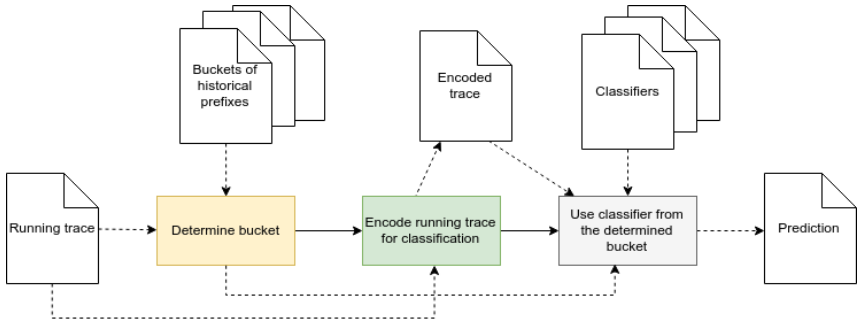


Figure 7: Predictive process monitoring workflow (online phase).

The general workflow for predictive process monitoring can be divided into two phases: the offline phase, to train a predictive model based on historical traces, and the online phase, to make predictions on running process traces. The *offline* phase, shown in Fig. 6, consists of four steps. First, given an event log, case prefixes are extracted and filtered (e.g. to retain only prefixes up to a certain length). Next, the identified prefixes are divided into buckets (e.g. based on process states or similarities among prefixes) and features are encoded from these buckets for classification. Finally, each bucket of encoded prefixes is used to train a classifier.

The *online* phase, shown in Fig. 7, concerns the actual prediction for a running trace, by reusing the elements (buckets, classifiers) built in the offline phase. Specifically, given a running trace and a set of buckets of historical prefixes, the correct bucket is first determined. Next, the running trace is encoded as a feature vector for classification. In the last step, a prediction is extracted from the encoded trace using the classifier corresponding to the determined bucket.

We note that there is an exception among the surveyed methods that does not perfectly fit the presented workflow. Namely, the KNN bucketing approach proposed by Maggi et al. [60] omits the offline phase. Instead, in this approach, the bucket (a set of similar traces from the training set) is determined and a classifier is trained during the online phase, separately for each running case. Note that conversely from the traditional KNN classifier, where a prediction is made directly based on the class labels of the nearest neighbors, in this bucketing approach an additional classifier (e.g. decision tree) is trained on the examples corresponding to the nearest neighbors.

Table 4: Classification of the 11 primary studies according to the four steps of the offline phase.

No.	Primary study	Prefix extraction and filtering	Sequence encoding			Classification algorithm
			Trace bucketing	Control flow	Data	
1	de Leoni et al. [20]	all	Single bucket	agg., last state	agg., last state	DT
2	Maggi et al. [60]	all	KNN-based	agg.	last state	DT
3	Grigori et al. [37]	all	State-based	agg. (boolean)	last state	DT
4	Schwegmann et al. [85]	all	State-based	agg. (boolean)	last state	SVM
5	Lakshmanan et al. [52]	all	State-based	last state	last state	DT
6	Conforti et al. [17]	all	State-based	last state	last state	DT
7	Di Francescomarino et al. [30]	prefix length 1-21, with gap 3, 5, or 10	Clustering-based	agg.	last state	DT, RF
8	Leontjeva et al. [54]	prefix length 2-20	Prefix length	index-based	index-based	DT, RF, GBM, SVM
				index-based	last state	RF
				agg.	-	RF
9	van der Spoel et al. [104]	prefix length 1-30	Prefix length	index-based	-	DT, RF, AB, NB, KNN
10	Verenich et al. [106]	prefix length 2-20	Prefix length + cluster	index-based	index-based	RF
11	Ghattas et al. [34]	all	Domain knowledge	index-based	index-based	DT

Table 4 lists the 11 primary studies identified in our SLR, and shows their characteristics according to the four steps of the offline phase (prefix selection and filtering, trace bucketing, sequence encoding and classification algorithm). In the rest of this section we survey the primary studies based on their characteristics, and use this information to build a taxonomy that allows us to classify the studies.

3.2.2. Prefix extraction and filtering

After analyzing the identified studies, we found that all of them take as input a prefix log (see Def. 3.2.3) to train a classifier. This choice is natural given that at runtime, we need to make predictions for running traces rather than completed ones. Using a prefix log for training ensures that our training data is comparable to the testing data. For example, for a complete trace consisting of a total of 5 events, we could consider up to 4 prefixes: the (incomplete) trace after executing the first event, the (incomplete) trace after executing the first and the second event, and so on.

Using all possible prefixes raises multiple problems. Firstly, the large number of prefixes as compared to the number of traces considerably slows down the training of the prediction models. Secondly, if the length of the original cases is very heterogenous, longer traces produce much more prefixes than shorter ones and, therefore, the prediction model is biased towards the longer cases. Accordingly, it is common to consider prefixes up to a certain number of events only. For example, Di Francescomarino et al. [30] limit the maximum prefix length to 21, while Leontjeva et al. [54] use prefixes of up to 20 events only. In other words, in their training phase, these approaches take as input the length-filtered prefix log $L_*^K = \{hd^k(\sigma) : \sigma \in L, 1 \leq k \leq min(K, |\sigma|)\}$, where $K = 21$ and $K = 20$, respectively.

Di Francescomarino et al. [30] propose a second approach to filter the prefix log using so-called *gaps*. Namely, instead of retaining all prefixes of up to a certain length, they retain prefixes whose length is equal to a base number (e.g. 1) plus a multiple of a gap (e.g. 1, 6, 11, 16, 21 for a gap of 5) . This approach

helps to keep the prefix log sufficiently small for applications where efficiency of the calculations is a major concern. More formally, a gap-filtered prefix log is constructed as $L_*^g = \{hd^{1+i \cdot g}(\sigma) : \sigma \in L, 0 \leq i < |\sigma|/g)\}$, where g denotes the gap length, e.g. $g = 3$, $g = 5$, or $g = 10$.

We observe that length-based or gap-based filtering can be applied to any predictive process monitoring method. In other words, the choice of length or gap filtering is not an inherent property of a method.

3.2.3. Trace bucketing

Most of the existing predictive process monitoring approaches train multiple classifiers rather than a single one (see Table 4). In particular, the prefix traces in the historical log are divided into several buckets and different classifiers are trained for each such bucket. At runtime, the most suitable bucket for the ongoing case is determined and the respective classifier is applied to make a prediction. In the following, we describe the bucketing approaches that have been proposed by existing predictive process monitoring methods.

Single bucket. All prefix traces are considered to be in the same bucket. A single classifier is trained on the whole prefix log and applied directly to the running cases. This approach has been used in the work by de Leoni et al. [20].

KNN-based bucketing. This approach differs from the general workflow in the sense that the offline training phase is skipped and the prefixes are not divided into non-overlapping buckets. Instead, for each running prefix trace, its k nearest neighbors are selected from the historical prefix traces and a classifier is trained (at runtime) on this "bucket" of k neighbors. This means that the number of buckets (and classifiers) is not fixed, but grows with each executed event at runtime. Note that this approach is similar to the KNN classifier introduced in Chapter 2 in the sense that both methods retrieve the k nearest neighbors. However, a KNN classifier predicts the average (or the mode) of the class labels associated with these neighbors, while a KNN bucketer only retrieves the neighbors, so that a classifier (such as DT or RF) can be constructed from these instances. The KNN method for predictive process monitoring was proposed by Maggi et al. [60]. Namely, they calculate the similarities between prefix traces using string-edit distance on the control flow. All instances that exceed a specified similarity threshold are considered as neighbors of the running trace. If the number of neighbors found is less than 30, the top 30 similar neighbors are selected regardless of the similarity threshold.

State-based bucketing. State-based approaches determine the buckets based on elements in a process model, e.g. each task [37, 85] or decision point [17, 52] corresponds to a *state*. A bucket is formed from trace prefixes having the same current (i.e. the most recent) state and a classifier is trained on these prefixes. At runtime, the current state of the running case is determined, and the respective classifier is used to make a prediction for the running case.

Some of the state-based approaches (Grigori et al. [37], Schwegmann et al. [85], and Conforti et al. [17]) assume that the process model is constructed beforehand, i.e. they take as input both an event log and a process model. Note that these methods implicitly or explicitly assume that the event log they take as input perfectly fits the given process model, since for prefixes that do not correspond to any state in the process model it is not possible to determine the bucket. However, when a process is executed by an enterprise system such as an enterprise resource planning (ERP) or customer-relationship management (CRM) system or by a custom-made information system, it is not guaranteed that the process will always abide to the reference process model (there can be deviations) and as such, this assumption is often unrealistic.

Conversely, Lakshmanan et al. [52] derive a process model automatically from a given event log. Specifically, they construct a so-called *activity graph* where there is one node per possible activity (event class) in the log, and there is a directed edge from node a_i to a_j iff a_j has occurred immediately after a_i in at least one trace. This type of graph is also known as the *Directly-Follows Graph* (DFG) of an event log [101]. We observe that the DFG is the state-transition system obtained by mapping each trace prefix in the log to a state corresponding to the last activity appearing in the trace prefix (and hence the state of a running case is fully determined by its last activity). The edges in the DFG are annotated with transition probabilities, where the transition probability from node a_i to a_j captures how often after performing activity a_i, a_j is performed next. We also observe that a DFG annotated with transition probabilities is a first order Markov chain. Also, buckets constructed based on a DFG correspond to the *last state abstraction* of a trace, i.e. the bucket of a prefix is determined based on the activity name of the last event executed in the given prefix. Alternative methods for constructing state abstractions are identified in [103] (e.g. set-based, multiset-based and sequence-based state abstractions), but these have not been used for outcome-oriented predictive process monitoring. Also, these latter state abstractions are likely not to be suitable for most logs since they can generate a very large number of states, which would lead to very large number of buckets, so that most of these buckets would be too small to train a separate classifier.

Clustering-based bucketing. The clustering-based bucketer relaxes the requirement of a direct transition between the buckets of two subsequent prefixes. Conversely, the buckets (clusters) are determined by applying a clustering algorithm on the encoded prefix traces. This results in a number of clusters that do not exhibit any transitional structure. In other words, the buckets of $hd^k(\sigma)$ and $hd^{k+1}(\sigma)$ are determined independently from each other. Both of these prefixes might be assigned to the same cluster or different ones. One classifier is trained per each resulting cluster, considering only the historical prefix traces 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 respective classifier is applied.

A clustering-based approach is proposed by Di Francescomarino et al. [30]. They experiment with two clustering methods, DBScan (with string-edit distance) and model-based clustering (with Euclidean distance on the frequencies of performed activities), while neither achieves constantly superior performance over the other. Another clustering-based method is introduced by Verenich et al. [106]. In their approach, the prefixes are encoded using index-based encoding (see 3.2.4) using both control flow and data payload, and then either hierarchical agglomerative clustering (HAC) or k-medoids clustering is applied. According to their results, k-medoids clustering consistently outperforms HAC.

Prefix length based bucketing. In this approach, each bucket contains only the incomplete traces of a specific length. For example, one bucket contains traces where only the first event has been executed, another bucket contains those where the first and the second event have been executed, and so on. One classifier is built for each possible prefix length. The prefix length based bucketing has been employed by van der Spoel et al. [104] and Leontjeva et al. [54]. Also, Verenich et al. [106] bucket the prefixes according to prefix length before applying a clustering method.

Domain knowledge based bucketing. While the bucketing methods described so far can detect buckets through an automatic procedure, it is possible to define a bucketing function that is based on manually constructed rules. In such an approach, the input from a domain expert is needed. The resulting buckets can, for instance, refer to *context categories* [34].

The aim of this survey and benchmark is to derive general principles by comparing methods that are applicable in arbitrary outcome-based predictive process monitoring scenarios and, thus, the methods that are based on domain knowledge about a particular dataset are left out of scope. For this reason, we do not further consider bucketing approaches based on domain knowledge.

3.2.4. Sequence encoding

In order to train a classifier, all prefix traces in the same bucket need to be represented as fixed length feature vectors. The main challenge here comes from the fact that with each executed event, additional information about the case becomes available, while each trace in a bucket (independent of the number of executed events) should still be represented with the same number of features. This can be achieved by applying a trace abstraction technique [103], for example, considering only the last m events of a trace. However, choosing an appropriate abstraction is a difficult task, where one needs to balance the trade-off between the generality[2] and loss of information. After a trace abstraction is chosen, a set of feature extraction functions may be applied to each event data attribute of the abstracted

[2]Generality in this context means being able to apply the abstraction technique to as many prefix traces as possible; as an example, the last m states abstraction is not meaningful for prefixes that are shorter than m events.

trace. Therefore, a *sequence encoding* method can be thought of as a combination of a trace abstraction technique and a set of feature extraction functions for each data attribute.

In the following paragraphs we describe the sequence encoding methods that have been used in the existing predictive process monitoring approaches. As described in Chapter 1, a trace can contain any number of static case attributes and dynamic event attributes. Both the case and the event attributes can be of numeric, categorical, or textual type. As none of the compared methods deal with textual data, hereinafter we will focus on numeric and categorical attributes only.

Static encoding. The encoding of case attributes is rather straightforward. As they remain the same throughout the whole case, they can simply be added to the feature vector "as is" without any loss of information. In order to represent all the information as a numeric vector, we assume the "as is" representation of a categorical attribute to be one-hot encoding (see Chapter 2). The static encoding was first mentioned explicitly by Leontjeva et al. [54].

Last state encoding. In this encoding method, only the last available snapshot of the data is used. Therefore, the size of the feature vector is proportional to the number of event attributes and is fixed throughout the execution of a case. A drawback of this approach is that it disregards all the information that has happened in the past, using only the very latest data snapshot. To alleviate this problem, this encoding can easily be extended to the last m states, in which case the size of the feature vector increases m times. As the size of the feature vector does not depend on the length of the trace, the last state (or the last m states) encoding can be used with buckets of traces of different lengths.

Using the last state abstraction, only one value (the last snapshot) of each data attribute is available. Therefore, no meaningful aggregation functions can be applied. Similarly to the static encoding, the numeric attributes are added to the feature vector "as is", while one-hot encoding is applied to each categorical attribute.

The last state encoding is the most common encoding technique, having been used in the KNN approach [60], state-based bucketing [17, 37, 52, 85], as well as the clustering-based bucketing approach by Di Francescomarino et al. [30]. Leontjeva et al. [54] combine index-based encoding for control flow with last state encoding for data attributes. Furthermore, de Leoni et al. [20] mention the possibility of using the last and the previous (the last two) states.

Aggregation encoding. The last state encoding has obvious drawbacks in terms of information loss, neglecting all data that have been collected in the earlier stages of the trace. Another approach is to consider all events since the beginning of the case, but ignore the order of the events. This abstraction method paves the way to several aggregation functions that can be applied to the values that an event attribute has taken throughout the case.

In particular, the frequencies of performed activities (control flow) have been used in several existing works [30, 54]. Alternatively, boolean values have been

used to express which activities have occurred in the trace in order to determine the state (bucket) of a prefix [37, 85]. However, when used to generate feature vectors to be given as input to a classifier, frequency-based encoding has been shown to be superior to the boolean encoding [54]. For numeric attributes, de Leoni et al. [20] proposed using general statistics as aggregation functions, such as average, maximum, minimum, and sum.

Index-based encoding. While the aggregation encoding exploits information from all the performed events, it still exhibits information loss by neglecting the order of the events. The idea of index-based encoding is to use all possible information (including the order) in the trace, generating one feature per each event attribute per each executed event (each *index*). This way, a lossless encoding of the trace is achieved, which means that it is possible to completely recover the original trace based on its feature vector. A drawback of index-based encoding is that due to the fact that the length of the feature vector increases with each executed event, this encoding can only be used in homogenous buckets where all traces have the same length.

Index-based encoding was used by van der Spoel et al. [104] to encode control flow and was extended to data attributes by Leontjeva et al. [54]. Additionally, in the latter work the authors combined the index-based encoding with HMM log-likelihood ratios. However, we decided not to experiment with HMMs in this study for mainly two reasons. Firstly, the HMMs did not consistently improve the basic index-based encoding in [54]. Secondly, rather than being an essential part of index-based encoding, HMMs can be thought of as an aggregation function that can be applied to each event attribute, similarly to taking frequencies or numeric averages. Therefore, HMMs are not exclusive to index-based encoding, but could also be used in conjunction with the aggregation encoding. Index-based encoding is also used in the approach of Verenich et al. [106].

Summary. An overview of the encoding methods can be seen in Table 5. Note that the static encoding extracts different type of data from the trace (case attributes) than the other three methods (event attributes). Therefore, for obtaining a complete representation for a trace, it is reasonable to concatenate the static encoding with one (or more) of the other three encodings.

3.2.5. Classification algorithm

The existing predictive process monitoring methods have been experimented with different classification algorithms. The most popular choice has been decision tree (DT), which has been employed in 9 out of 11 primary studies. Another popular method has been random forest (RF), used in 4 studies. Additionally, some works have experimented with SVM, Naive Bayes (NB), k-nearest neighbors (KNN), AdaBoost (AB), and generalized boosted regression models (GBM). Works that experimented with RF among other classification algorithms found that the performance of RF is superior [54, 104] or comparable [30] to other techniques.

Table 5: Encoding methods.

Encoding name	Relevant attributes	Trace abstraction	Feature extraction	
			Numeric	Categorical
Static	Case	Case attributes	as is	one-hot
Last state	Event	Last event	as is	one-hot
Aggregation	Event	All events, unordered (set/bag)	min, max, mean, sum, std	frequencies or occurrences
Index-based	Event	All events, ordered (sequence)	as is for each index	one-hot for each index

3.2.6. Discussion

We have observed that the prefix filtering techniques are not inherent to any given predictive process monitoring method. Instead, these techniques are selected based on performance considerations and can be used in conjunction with any of the predictive process monitoring methods. In a similar vein, the choice of a classification algorithm is a general problem in machine learning and is not specific to business process data. Indeed, all of the authors of the methods reviewed above claim that their method is applicable in conjunction with any classifier. Therefore, we treat the prefix filtering technique and the classification algorithm employed as *orthogonal* aspects to the categorization of predictive process monitoring methods. However, while excluded from the taxonomy, the specific prefix filtering technique and classification algorithm used still play an important role in obtaining good predictions, with their performance being influenced by the particular settings used.

The considered methods also differ in terms of the event log attributes that are used for making predictions. However, it has been shown [54] that including more information (i.e. combining control flow and data payload) can drastically increase the predictive power of the models, so it is preferable to use the largest possible set of evant and case attributes independently of the method.

Based on the above, we conclude that existing outcome-oriented predictive process monitoring methods can be characterized on two grounds:

- how the prefix traces are divided into buckets (trace bucketing)?
- how the (event) attributes are transformed into features (sequence encoding)?

Figure 8 provides a taxonomy of the relevant methods based on these two perspectives. Note that the 11 approaches shown in the taxonomy do not correspond directly to the 11 primary studies. The reason for this is that while the primary approaches tend to mix different encoding schemes, e.g. to use aggregation encoding for control flow and last state encoding for data payload (see Table 4), the taxonomy is constructed in a modular way, so that each encoding method constitutes a separate approach. In other words, the taxonomy, as presented here, assumes

that the same encoding is applied to both control flow and data payload. Still, in practice the different encodings (that are valid for a given bucketing method) can be easily combined. Similarly, the taxonomy does not contain combinations of several bucketing methods. An example of such "double bucket" approaches is the method by Verenich et al. [106], where the prefixes are first divided into buckets based on prefix length and, then, clustering is applied within each bucket. In practice, it is possible to come up with a variety of such constructs, combining the elements presented in the taxonomy. However, note that such double bucket approaches divide the prefixes into many small buckets, which often leads to situations where a classifier receives too little training instances to learn meaningful patterns.

We note that the taxonomy generalizes the state-of-the-art, in the sense that even if a valid pair of bucketing and encoding method has not been used in any existing approach in the literature, it is included in the taxonomy (e.g. the state-based bucketing approach with frequency-based aggregation encoding). We also note that while the taxonomy covers the techniques proposed in the literature, all these techniques rely on applying a propositional classifier on an explicit vectorial representation of the traces. One could envisage alternative approaches that do not require an explicit feature vector as input. For instance, kernel-based SVMs have been used in the related setting of predicting the cycle time of a case [105]. Furthermore, one could envisage the use of data mining techniques to extract additional features from the traces (e.g. latent variables or frequent patterns). Although the taxonomy does not cover this aspect explicitly, applying such techniques is consistent with the taxonomy, since the derived features can be used in combination with any of the sequence encoding and bucketing approaches presented here. While most of the existing works on outcome-oriented predictive monitoring use the event/trace attributes "as-is" without an additional mining step, Leontjeva et al. used Hidden Markov Models for extracting additional features in combination with index-based encoding [54]. Further on, although not yet applied to outcome-oriented predictive process monitoring tasks, different pattern mining techniques could be applied to extract useful patterns from the sequences, occurrences of which could then be used as features in the feature vectors of the traces. Such techniques have been used in the domain of early time series/sequence classification [32, 33, 42, 55, 109] and for predicting numeric measures (e.g. remaining time) for business processes [29].

3.3. Deployment use cases

In this section, we discuss the deployment use cases employed in the existing works on outcome-oriented predictive process monitoring (RQ4). In particular, we have identified three main use cases that have been used or mentioned in the primary studies (see Table 6). While in *continuous* predictive monitoring the system remains purely predictive in nature, in *alarm-based* and *recommendation-*

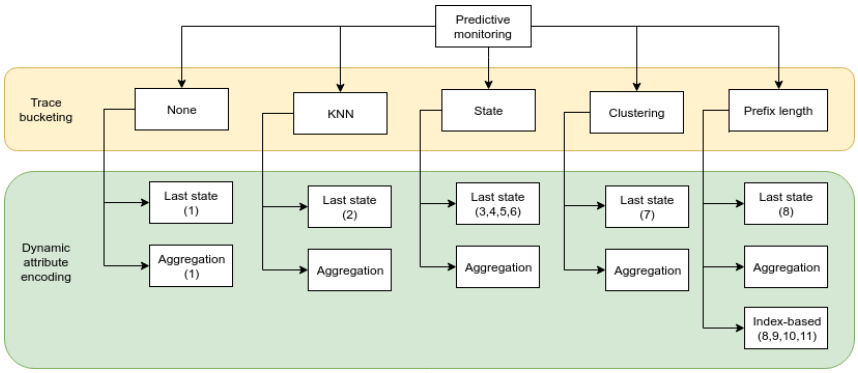


Figure 8: Taxonomy of methods for outcome-oriented predictive business process monitoring. Numbers correspond to the primary studies employing a given method (see Table 4 for mapping between primary studies and their numbers).

Table 6: Deployment use cases in the primary studies.

No.	Study	Use cases
1	de Leoni et al. [20]	continuous, recommendation
2	Maggi et al. [60]	continuous, alarm, recommendation
3	Grigori et al. [37]	continuous, alarm
4	Schwegmann et al. [85]	continuous
5	Lakshmanan et al. [52]	continuous
6	Conforti et al. [17]	recommendation
7	Di Francescomarino et al. [30]	alarm
8	Leontjeva et al. [54]	continuous
9	van der Spoel et al. [104]	continuous
10	Verenich et al. [106]	continuous
11	Ghattas et al. [34]	recommendation

based scenarios the system is expected to estimate *if* the prediction is reliable (or severe) enough for the process worker to act upon it and/or *how* the process worker should intervene. Therefore, we consider the latter two use cases to be instances of *prescriptive* process monitoring.

Continuous predictive monitoring. In this setting, the process worker receives predictions about the case outcome after every event. The predictions can be provided in either binary format (0 for negative class, 1 for positive class) or as a real-valued score, reflecting the probability or confidence towards a certain outcome. The predictive monitoring system does not prescribe if and when the process worker should act upon the prediction and continues to monitor the case until the end.

Alarm-based prescriptive monitoring. In this scenario, the predictive monitoring system only outputs a prediction if the probability or confidence of a cer-

tain outcome is sufficiently high. This corresponds to triggering an alarm, i.e. notifying the process worker that an action is needed in the given process instance. After an alarm is raised, the instance is taken out of monitoring, i.e. an alarm can be raised at most once per case. Alarm-based monitoring can be implemented in a one-sided or a two-sided manner. In the first case, alarms are raised only when an undesired outcome is predicted with sufficient probability, while in the second case alarms are raised for both (sufficiently likely) undesired and desired outcomes. In the existing works, alarm-based settings are usually implemented using a decision threshold on the prediction scores returned by the classifier. In a two-sided setting, often the notion of *confidence* is used, so that an alarm is raised if the prediction score is either sufficiently low or sufficiently high. For instance, given a classifier that outputs scores in the range of 0 to 1, $confidence(0.3) = confidence(0.7) = 0.7$. The threshold in the existing works is chosen manually, either as a fixed value [30] or set as the average *class probability* (i.e. the percentage of examples correctly classified with respect to all the examples following that specific path) over all leaves in a decision tree [60]. In the latter work, the authors also propose an alternative approach where alarms are raised only if the *class support* (i.e. the number of examples in the training set that follow the path from the root to the leaf and that are correctly classified) in the given leaf of the decision tree is higher than the median of class supports over all leaves. In [30], a hybrid approach is used, where the alarm is raised if both the class support and the class probability are higher than the respective thresholds.

Recommendation-based prescriptive monitoring. In this setting, the system is expected to not only raise an alarm, but also to recommend what would be the most appropriate action that the process worker should take in order to prevent or mitigate the undesired outcome. Some of the related works outline an approach that queries the predictive model for predictions about several hypothetical continuations of the process and recommends the action taken in the path with the most desirable predicted outcome [20, 60]. Ghattas et al. [34] derive interpretable decision rules in the form *IF <current state of attributes> THEN <action>*. The approach by Conforti et al. [17] generate recommendations by computing an optimal work-item distribution based on the predictions.

3.4. Evaluation measures and experimental settings

In this section, we discuss the evaluation procedures used in the existing works on outcome-oriented predictive process monitoring (RQ3). We observe that the evaluation procedures in the existing studies differ mainly in terms of four aspects: evaluation measures, evaluation points in the trace, train-test split, and model selection. An overview of the existing studies with respect to these aspects is given in Table 7. In the following subsections, we first discuss evaluation measures and evaluation points (Section 3.4.1) and proceed with log splitting and hyperparameter optimization strategies (Section 3.4.2).

Table 7: Evaluation procedures in the primary studies.

No.	Study	Evaluation measures	Evaluation points	Train/test split	Model selection
1	de Leoni et al. [20]	ACC (train) and F-score (train)	overall	-	-
2	Maggi et al. [60]	TPR, FPR, precision, F-score, ACC	start, 1/4, 1/2, overall	temporal 80-20	-
3	Grigori et al. [37]	ACC (train)	every state	-	-
4	Schwegmann et al. [85]	ACC	every state	CV	CV
5	Lakshmanan et al. [52]	ACC (train)	every state	-	-
6	Conforti et al. [17]	% faulty instances	end of trace	simulation	-
7	Di Francescomarino et al. [30]	F-score, earliness, failure-rate train, processing, prediction times	alarm time	temporal 80-20	-
8	Leontjeva et al. [54]	AUC, earliness, stability of AUC training and prediction times	prefixes 2-20, overall	temporal 80-20	5-fold CV
9	van der Spoel et al. [104]	ACC	prefixes 1-30, overall	random 50-50	-
10	Verenich et al. [106]	AUC	prefixes 2-20, overall	random 80-20	-
11	Ghattas et al. [34]	% excellent, (un)acceptable instances	end of trace	simulation	-

3.4.1. Evaluation measures

In outcome-oriented predictive process monitoring, the quality of the predictions is typically measured with respect to two main dimensions based on the following desiderata: a good prediction should be *accurate* and it should be made in the *early* stages of the process. Furthermore, in order to be applicable in practice a prediction should be produced *efficiently*, i.e. the execution times should be suitable for a given application. In the following paragraphs, we delve deeper into these aspects and map the evaluation measures used in the existing works to these three quality dimensions. Lastly, we briefly describe the evaluation measures used in the recommendation use cases.

Accuracy. Prediction accuracy refers to the correctness of the predictions with respect to the ground truth (i.e. the actual class labels). A prediction that is often inaccurate is a useless prediction, as it cannot be relied on when making decisions. Therefore, accuracy is, in a sense, the most important quality of a prediction.

Two main types of metrics emerge in the existing works for measuring prediction accuracy. In the first approach, predictions are expected to be binary values, i.e. to directly refer to a positive or a negative predicted outcome. In these cases, metrics such as ACC, precision, recall, F-score, or other derivatives from the confusion matrix (see Chapter 2) are used. In the second approach, it is assumed that the classifier returns a real-valued score, reflecting how likely it is that the case will end in one way or the other. Most commonly, AUC is used to measure the prediction accuracy in the latter approach.

In continuous predictive monitoring settings, prediction accuracy is often reported for different evaluation points, e.g. prefix lengths [54, 104, 106], relative prefix lengths [60], or states in a process model [37, 52, 85]. Additionally, in some cases the *overall* accuracy is reported as the mean of the respective accuracy measure over all considered evaluation points. Furthermore, Leontjeva et al. [54] report the standard deviation of the AUC over all prefix lengths, as a measure of *stability of the quality of the results* across different stages in the process.

The evaluation procedure in alarm-based use cases exhibits some differences from the continuous monitoring setting. Firstly, as the output of the system is of

binary nature (alarm or not alarm), the prediction accuracy can only be measured using the first group of metrics, e.g. the F-score. Secondly, the prediction is made at most once per case and, therefore, reporting the quality measures separately for different execution stages is not meaningful. Instead, the evaluation point in each case corresponds to the point in the trace where an alarm was raised. Furthermore, in a two-sided alarm setting, it can happen that the case finishes before the confidence of any prediction reaches the respective threshold and, therefore, no prediction is made for the case. In such cases, Di Francescomarino et al. [30] propose measuring the *failure-rate* as the proportion of cases for which no predictions were made. Note that reaching the end of the case without making a prediction is not considered a failure in one-sided alarm settings, since if the case ends without alarms, it can simply be considered a predicted "desired" outcome.

Earliness. The earlier an accurate (or reliable) prediction is made, the more useful it is in practice, since it leaves more time to act upon the prediction. From the literature, two different approaches emerge for measuring the earliness of the predictions. In continuous predictive monitoring settings, earliness is often measured implicitly by evaluating the accuracy of the models separately for different evaluation points, e.g. for each prefix length [54]. The improvement of prediction accuracy as the prefix length increases serves as a notion of earliness. In particular, the smaller the prefix length is when an acceptable level of accuracy is reached, the better the method in terms of earliness. In these settings, earliness can also be defined explicitly as a metric, i.e. as the smallest prefix length in which the model achieves a specified accuracy threshold [54]. Another approach to measuring earliness corresponds to the alarm-based use case, where a prediction is made at most once per case. Then, earliness is defined as the prefix length when such a prediction is made (i.e. the alarm time) divided by the length of the trace [30].

Efficiency. Efficiency of a predictive monitoring method refers to the ability of the system to produce predictions in reasonable time. In this context, both the offline and the online execution times are measured. In particular, in the offline phase, it should be possible to train the underlying predictive models within a feasible timeframe. In the online phase, the system should be able to output predictions in nearly real time. Di Francescomarino et al. further divide the computation times in the online phase into processing time, i.e. the total time taken to process the entire test dataset (e.g. encoding and assigning prefixes to clusters), and prediction time, i.e. the average time to make a prediction for a given prefix [30].

Quality measures in the recommendation use cases. The works that follow the recommendation-based prescriptive monitoring use case measure the goodness of the recommendation system directly, rather than assessing the quality of the predictions produced in an intermediate step. Conforti et al. [17] measure the proportion of faulty instances in the "as-is" case (no recommender system in place) and in the "to-be" case (when using the recommendation system). Similarly, Ghattas et al. [34] report the proportions of excellent, acceptable, and unacceptable

instances. The goodness of the recommender system is measured by the extent to which it is able to reduce the number of faulty/unacceptable instances and/or increase the number of excellent instances.

3.4.2. Model selection and generalization

As discussed in Chapter 2, it is essential in machine learning to evaluate how well the model generalizes to unseen data. In other words, the performance of the predictive model should be measured on a separate test set that has not been used when training the model. Such evaluation has been performed in 8 out of the 11 primary studies (see Table 7). The most common strategy to split the log into a train and a test log has been a 80-20 temporal split, i.e, the traces are ordered by their start time (the time of the first event); the first 80% are used as the training set and the remaining 20% as the test set [30, 54, 60]. In these works, no correction has been employed for events that belong to training traces but overlap with the time period of the testing traces. Therefore, this setup does not strictly imitate a real-life scenario, since some information from events that happen in "the future" (i.e. during the testing period) is leaked to the training set. Additionally to temporal split, some works have used a random split [104, 106] or cross-validation [85]. Studies that follow the recommendation use case have used a completely different evaluation procedure by simulating new traces according to a process model and evaluating the quality of the recommendations on these unseen but artificial traces [17, 34]. In the rest of the studies, no quantitative evaluation of the predictive models is performed and the discussion remains on the level of training accuracy in selected examples [20, 37, 52].

Hyperparameter optimization has been performed in only 2 existing works, where cross-validation has been used to find the optimal hyperparameters of the models [54, 85].

3.5. Summary

This chapter provides a survey of existing outcome-oriented predictive business process monitoring techniques. The relevant existing studies were identified through a systematic literature review (SLR), which revealed 16 studies dealing with the problem of predicting case outcomes. Out of these, 11 were considered to contain a distinct contribution (primary studies). Through further analysis of the primary studies, a taxonomy was proposed based on two main aspects, the trace bucketing approach and sequence encoding method employed. Combinations of these two aspects led to a total of 11 distinct methods.

One of the threats to the validity of the survey presented in this chapter relates to the potential selection bias in the literature review. To minimize this, we described our systematic literature review procedure 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 alleviate this issue, all the included papers were collected in a publicly available spreadsheet, together with decisions and reasons for excluding them from the study.

In the next chapter, we proceed with an experimental evaluation of the 11 techniques identified in this chapter.

4. BENCHMARK

In Chapter 3, we identified 11 approaches (shown in Figure 8) that have been developed in the context of outcome-oriented predictive process monitoring. However, we observed that different authors have used different datasets, experimental settings, evaluation measures, and baselines to assess their proposals, resulting in poor comparability and an unclear picture of the relative merits and applicability of different methods. This leads us to the following research question:

RQ1 (Benchmark) What is the relative performance of existing methods for outcome-oriented predictive monitoring of business processes?

In this chapter, we aim at answering the posed research question by performing a comparative experimental evaluation of the 11 existing techniques (shown in Figure 8) in a unified setting. To perform our benchmark, we implemented an open-source, tunable and extensible predictive process monitoring framework in Python. All experiments were run using Python 3.5 and the scikit-learn library [74] on a single core of an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz with 64GB of RAM.

In the rest of this chapter, we first introduce the evaluation datasets, then describe the evaluation procedure and conclude with discussing the results of the experiments.

4.1. Datasets

The benchmark is based on 9 real-life event logs that cover 6 different application domains: healthcare (event logs *BPIC2011*, *Sepsis*, and *Hospital billing*), government (*BPIC2015* and *Traffic fines*), manufacturing (*Production*), insurance (*Insurance*), and banking (*BPIC2012* and *BPIC2017*). Eight of these event logs are publicly available and accessible from the 4TU Centre for Research Data[1]. Several of these logs originate from Business Process Intelligence Challenges (BPIC) from different years. The criterion for selecting the public event logs for the evaluation was that the log must contain both case attributes (static) and event attributes (dynamic). Based on this, we discarded the BPIC logs from years 2013-2014. We also discarded the BPIC 2016 dataset because it is a clickstream-dataset of a Web service, rather than an event log of a business process. Even though process mining is also widely used in the insurance domain [90], no event logs from this domain are publicly available. Therefore, we decided to include a private log *Insurance*, which originates from a claims handling process at an Australian insurance company.

In some logs, we applied several labeling functions *out* (see Definition 1.2.2). In other words, the outcome of a case is defined in several ways depending on the goals and needs of the process owner. Each such notion of outcome constitutes a

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

Table 8: LTL Operators Semantics.

operator	*semantics*
$\mathbf{X}\varphi$	φ *has to hold in the next position of a path.*
$\mathbf{G}\varphi$	φ *has to hold always in the subsequent positions of a path.*
$\mathbf{F}\varphi$	φ *has to hold eventually (somewhere) in the subsequent positions of a path.*
$\varphi\ \mathbf{U}\psi$	φ *has to hold in a path at least until* ψ *holds.* ψ *must hold in the current or in a future position.*

separate predictive process monitoring task with a slightly different input dataset. In total, we formulated 24 different outcome prediction tasks based on the 9 original event logs: *bpic2011_1*, *bpic2011_2*, *bpic2011_3*, *bpic2011_4*, *bpic2015_1*, *bpic2015_2*, *bpic2015_3*, *bpic2015_4*, *bpic2015_5*, *production*, *insurance_1*, *insurance_2*, *sepsis_1*, *sepsis_2*, *sepsis_3*, *bpic2012_1*, *bpic2012_2*, *bpic2012_3*, *bpic2017_1*, *bpic2017_2*, *bpic2017_3*, *hospital_1*, *hospital_2*, and *traffic* . In the following paragraphs, we describe the original logs, the applied labeling functions, and the resulting predictive monitoring tasks in more detail.

Event log BPIC2011. This event log contains cases from the Gynaecology department of a Dutch Academic Hospital. Each case assembles the medical history of a given patient, where the applied procedures and treatments are recorded as activities. Similarly to previous work [30, 54], we use four different labeling functions based on LTL rules [77]. Specifically, we define the class label for a case σ according to whether an LTL rule φ is violated or satisfied by each trace σ.

$$out(\sigma) = \begin{cases} 1 & \text{if } \varphi \text{ violated in } \sigma \\ 0 & \text{otherwise} \end{cases}$$

Table 8 introduces the semantics of the LTL operators.

The four LTL rules used to formulate the four prediction tasks on the *BPIC2011* log are as follows:

- *bpic2011_1*: $\varphi = \mathbf{F}(\text{“}tumor\ marker\ CA-19.9\text{”}) \vee \mathbf{F}(\text{“}ca-125\ using\ meia\text{”})$,
- *bpic2011_2*: $\varphi =$ $\mathbf{G}(\text{“}CEA-tumor\ marker\ using\ meia\text{”} \rightarrow \mathbf{F}(\text{“}squamous\ cell\ carcinoma\ using\ eia\text{”}))$,
- *bpic2011_3*: $\varphi =$ $(\neg\text{“}histological\ examination-biopsies\ nno\text{”})\mathbf{U}(\text{“}squamous\ cell\ carcinoma\ using\ eia\text{”})$, and
- *bpic2011_4*: $\varphi = \mathbf{F}(\text{“}histological\ examination-big\ resectiep\text{”})$.

For example, the φ for *bpic2011_1* expresses the rule that at least one of the activities “tumor marker CA-19.9” or “ca-125 using meia” must happen eventually during a case. Evidently, the class label of a case becomes known and irreversible when one of these two events has been executed. In order to avoid bias introduced by this phenomenon during the evaluation phase, all the cases are cut exactly before either of these events happens. Similarly, the cases are cut before the occurrence of “histological examination-biopsies nno” in the *bpic2011_3* dataset and before “histological examination-big resectiep” in *bpic2011_4*. However, no cutting is performed in the *bpic2011_2* dataset, because φ states that a “CEA-tumor marker using meia” event must *always* be followed by a “squamous cell carcinoma using eia” event sometime in the future. Therefore, even

if one occurrence of "CEA-tumor marker using meia" has successfully been followed by a "squamous cell carcinoma using eia" (φ is satisfied), another occurrence of "CEA-tumor marker using meia" will cause φ to be violated again and, thus, the class label is not irreversibly known until the case completes.

Event log BPIC2015. This dataset assembles event logs from 5 Dutch municipalities, pertaining to a building permit application process. We treat the dataset from each municipality as a separate event log and apply a single labeling function to each one. Similarly to *BPIC2011*, the labeling function is based on the satisfaction/violation of an LTL rule φ. The prediction tasks for each of the 5 municipalities are denoted as *bpic2015_i*, where $i = 1 \dots 5$ indicates the number of the municipality. The LTL rule used in the labeling functions is as follows:

- *bpic2015_i*: $\varphi = \mathbf{G}(\text{“}send\ confirmation\ receipt\text{”} \rightarrow \mathbf{F}(\text{“}retrieve\ missing\ data\text{”}))$.

No trace cutting can be performed here, because, similarly to *bpic2011_2*, the final satisfaction/violation of φ is not known until the case completes.

Event log Production. This log contains data from a manufacturing process. Each trace records information about the activities, workers and/or machines involved in producing an item. The labeling (*production*) is based on whether or not the number of rejected work orders is larger than zero.

Event log Insurance. This is the only private log we use in the experiments. It comprises of cases from an Australian insurance claims handling process. We apply two labeling functions:

- *insurance_1*: *out* is based on whether a specific "key" activity is performed during the case or not.
- *insurance_2*: *out* is based on the time taken for handling the case, dividing them into slow and fast cases.

Event log Sepsis. This log records trajectories of patients with symptoms of the life-threatening sepsis condition in a Dutch hospital. Each case logs events since the patient's registration in the emergency room until her discharge from the hospital. Among others, laboratory tests together with their results are recorded as events. Moreover, the reason of the discharge is available in the data in an obfuscated format.

We created three different labelings for this log:

- *sepsis_1*: the patient returns to the emergency room within 28 days from the discharge,
- *sepsis_2*: the patient is (eventually) admitted to intensive care,
- *sepsis_3*: the patient is discharged from the hospital on the basis of something other than *Release A* (i.e. the most common release type).

Event log BPIC2012. This dataset, originally published in relation to the BPIC in 2012, contains the execution history of a loan application process in a Dutch financial institution. Each case in this log records the events related to a particular

loan application. For classification purposes, we defined some labelings based on the final outcome of a case, i.e. whether the application is accepted, rejected, or canceled. Intuitively, this could be thought of as a multi-class classification problem. However, to remain consistent with previous work on outcome-oriented predictive process monitoring, we approach it as three separate binary classification tasks. In the experiments, these tasks are referred to as *bpic2012_1* (accepted vs. not accepted), *bpic2012_2* (rejected vs. not rejected), and *bpic2012_3* (canceled vs. not canceled).

Event log BPIC2017. This event log originates from the same financial institution as the *BPIC2012* one. However, the data collection has been improved, resulting in a richer and cleaner dataset. As in the previous case, the event log records execution traces of a loan application process. Similarly to *BPIC2012*, we define three separate labelings based on the outcome of the application: *bpic2017_1*, *bpic2017_2*, and *bpic2017_3*, referring to binary classification tasks on identifying accepted, rejected, and canceled cases, respectively.

Event log Hospital billing. This dataset comes from an enterprise resource planning (ERP) system of a hospital. Each case is an execution of a billing procedure for medical services. We created two labelings for this log:

- *hospital_1*: the billing package was not (eventually) closed,
- *hospital_2*: the case is reopened.

Event log Traffic fines. This log comes from an Italian local police force. The dataset contains events about notifications sent about a fine, as well as (partial) repayments. Additional information related to the case and to the individual events include, for instance, the reason, the total amount, and the amount of repayments for each fine. We created the labeling (*traffic*) based on whether the fine is repaid in full or is sent for credit collection.

Preprocessing. It has been shown [54] that including more information (i.e. combining control flow and data payload) can drastically increase the predictive power of the models. In order to provide a fair comparison of the different methods, it is preferable to provide the same set of attributes as input to all methods, and preferably the largest possible set of attributes. Accordingly, in the comparative evaluation below, we will encode traces using all the available case and event attributes (covering both control flow and data payload).

While most of the attributes can be readily included in the train and test datasets, timestamps should be preprocessed in order to derive meaningful features. In our experiments, we use the following features extracted from the timestamp: month, weekday, hour, duration from the previous event in the given case, duration from the start of the case, and the position of the event in the case. Additionally, some recent works have shown that adding features constructed from collections of cases (inter-case features) increases the accuracy of the predictive models, particularly when predicting deadline violations [17, 87]. For example, waiting times are highly dependent on the number of ongoing cases of a process

(the so-called "Work-In-Process"). In turn, waiting times may affect the outcome of a case, particularly if the outcome is defined with respect to a deadline or with respect to customer satisfaction. Accordingly, we construct an inter-case feature reflecting the number of cases that are "open" at the time of executing a given event. All the abovementioned features are used as numeric dynamic (event) attributes.

Each of the categorical attributes has a fixed number of possible values (levels). For some attributes, the number of distinct levels can be very large, with some of the levels appearing only in a few cases. In order to avoid exploding the dimensionality of the input dataset, we filter only the category levels that appear in at least 10 examples. The less frequent levels are set to a common level "other". This filtering is applied to each categorical attribute except the event class (activity), where we use all category levels.

Due to the fact that event logs consist of data that are recorded automatically by information systems during the execution of tasks of a process, there is none or very little *missing data* in the traditional sense. However, it is common that different events carry different data payloads, resulting in a situation where some attribute values for a given event can be "missing" due to the fact that they are not applicable for that particular event. This can be caused by mainly two reasons. Firstly, in most event logs, an event records only the values of data attributes that were changed during that particular event. Therefore, in order to determine the value of an attribute at the point where an event occurred, we need to search for the latest event in the trace (or trace prefix) where the value of the attribute in question changed (or the first event if no change point is found). For instance, the name of the resource involved in the execution of an activity in a case is often logged only if the resource has changed since the previous event. In such cases, we search for the closest preceding event in the same case where the resource name was present and use the same value in the feature vector produced for the current event. Secondly, different activities can produce different types of data. For instance, in a loan application process, information about the offer made to the customer becomes available only when an offer is made (before that, no offer nor information about it exists). Similarly, in a fine collection process, the amount of the payment is only available for payment events. These examples constitute a form of *legitimately missing data* [72] or *missing data that is out of scope* [84]. In our experiments, we decided to address such cases by adding an additional feature (for each data attribute) to the dataset, indicating whether the given value is present for a given event or not. The value of the attribute itself was set to 0 if not present.

In event logs where information is available about case completion, we filter out incomplete cases in order to not mislead the classifier. Also, we cut each trace before the occurrence of the event that was used to define the label. For instance, in the *production* log, the traces are cut immediately before the number of rejected work orders becomes larger than zero.

Table 9: Statistics of the datasets used in the experiments.

dataset	# traces	min length	med length	max length	trunc length	# variants (after trunc)	pos class ratio	# event classes	# static attr-s	# dynamic attr-s	# static cat levels	# dynamic cat levels
bpic2011_1	1140	1	25.0	1814	36	815	0.4	193	6	14	961	290
bpic2011_2	1140	1	54.5	1814	40	977	0.78	251	6	14	994	370
bpic2011_3	1121	1	21.0	1368	31	793	0.23	190	6	14	886	283
bpic2011_4	1140	1	44.0	1432	40	977	0.28	231	6	14	993	338
bpic2015_1	696	2	42.0	101	40	677	0.23	380	17	12	19	433
bpic2015_2	753	1	55.0	132	40	752	0.19	396	17	12	7	429
bpic2015_3	1328	3	42.0	124	40	1280	0.2	380	18	12	18	428
bpic2015_4	577	1	42.0	82	40	576	0.16	319	15	12	9	347
bpic2015_5	1051	5	50.0	134	40	1048	0.31	376	18	12	8	420
production	220	1	9.0	78	23	203	0.53	26	3	15	37	79
insurance_1	1065	6	12.0	100	8	785	0.16	9	0	22	0	207
insurance_2	1065	6	12.0	100	13	924	0.26	9	0	22	0	207
sepsis_1	754	5	14.0	185	30	684	0.14	14	24	13	195	38
sepsis_2	782	4	13.0	60	13	656	0.14	15	24	13	200	40
sepsis_3	782	4	13.0	185	22	709	0.14	15	24	13	200	40
bpic2012_1	4685	15	35.0	175	40	3578	0.48	36	1	10	0	99
bpic2012_2	4685	15	35.0	175	40	3578	0.17	36	1	10	0	99
bpic2012_3	4685	15	35.0	175	40	3578	0.35	36	1	10	0	99
bpic2017_1	31413	10	35.0	180	20	2087	0.41	26	3	20	13	194
bpic2017_2	31413	10	35.0	180	20	2087	0.12	26	3	20	13	194
bpic2017_3	31413	10	35.0	180	20	2087	0.47	26	3	20	13	194
traffic	129615	2	4.0	20	10	185	0.46	10	4	14	54	173
hospital_1	77525	2	6.0	217	6	246	0.1	18	1	21	23	1756
hospital_2	77525	2	6.0	217	8	358	0.05	17	1	21	23	1755

General statistics. The resulting 24 datasets exhibit different characteristics which can be seen in Table 9. The smallest log is *production* which contains 220 cases, while the largest one is *traffic* with 129615 cases. The most heterogenous in terms of case lengths are the *bpic2011* labeled datasets, where the longest case consists of 1814 events. On the other hand, the most homogenous is the *traffic* log, where the case length varies from 2 to 20 events. The class labels are the most imbalanced in the *hospital_2* dataset, where only 5% of cases are labeled as *positive* ones (class label = 1). Conversely, in *bpic2012_1*, *bpic2017_3*, and *traffic*, the classes are almost balanced. In terms of event classes, the most homogenous are the *insurance* datasets, with only 9 distinct event classes. The most heterogenous are the *bpic2015* datasets, reaching 396 event classes in *bpic2015_2*. The datasets also differ in terms of the number of static and dynamic attributes. The *insurance* logs contain the largest number of dynamic attributes (22), while the *sepsis* datasets contain the largest number of static attributes (24).

4.2. Experimental setup

In this section, we start with describing the employed evaluation measures. We then proceed with describing our approach to splitting the event logs into train and test datasets and optimizing the hyperparameters of the compared methods.

4.2.1. Research questions and evaluation measures

We choose to adopt the continuous predictive process monitoring use case in the benchmark, since it is applicable for all business processes and, thus, provides more general insights into the performance of the methods. As discussed in Chap-

ter 3, the quality of the predictions in predictive process monitoring is typically measured with respect to three aspects: accuracy, earliness, and efficiency. Following the same criteria, we formulate two subquestions as follows:

RQ1.1 (Accuracy and earliness) How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of accuracy and earliness of the predictions?

RQ1.2 (Efficiency) How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of execution times?

We choose AUC as the measure for prediction accuracy due to its properties of being threshold-independent and remaining unbiased even in case of a highly imbalanced distribution of class labels. Still, relying on a single evaluation criterion may provide a biased viewpoint of the results; therefore, we report the F-scores (on the default threshold of 0.5) additionally to AUC. To measure prediction earliness, we use the implicit approach of evaluating the models on each prefix length. The reason for this choice is that it does not require specifying accuracy (or confidence) thresholds and, therefore, provides a more general view on the results.

When measuring the execution times of the methods, we distinguish the time taken in the offline and the online modes. The *offline time* is the total time needed to construct the classifier from the historic traces available in an event log. Namely, it includes the time for constructing the prefix log, bucketing and encoding the prefix traces, and training the classifier(s). Note that we do not add the time spent on model selection to the offline time measurements. The reason for this is that we consider the extent of hyperparameter optimization to be largely dependent on the requirements and the constraints of a given project. Specifically, if obtaining a final model with minimal amount of time is critical in a project, one can settle for a smaller number of iterations for hyperparameter optimization, while if the accuracy of the final model is of greater importance than the time for obtaining the model itself, more time can be spent on model selection. Still, a rough estimate of the total time needed for optimizing the hyperparameters can be obtained by multiplying the time taken for building the final model by the number of optimization rounds to be performed. In the online phase, it is essential that a prediction is produced almost instantaneously, as the predictions are usually needed in real time. Accordingly, we define the *online time* as the average time for processing one incoming event (incl. bucketing, encoding, and predicting based on this new event).

The execution times are affected by mainly two factors. Firstly, since each prefix of a trace constitutes one example, the lengths of the traces have a direct effect on the number of (training) examples. It is natural that the more examples are used for training, the better the accuracy the predictive monitoring system could yield. At the same time, using more examples increases the execution times of the system. In applications where the efficiency of the predictions is of critical importance, reducing the number of training examples can yield a reasonable tradeoff,

bringing down the execution times to a suitable level, while accepting lower accuracy. One way to reduce the number of examples is by using the gap-based filtering, where a prefix is added to the training set only after each g events in the trace. This leads us to the third subquestion:

RQ1.3 (Gap-based filtering) To what extent does gap-based filtering improve the execution times of the predictions?

The second factor that affects the execution times is the number and the diversity of attributes that need to be processed. In particular, the number of unique values (levels) in the categorical attribute domains has a direct effect on the length of the feature vector constructed for each example, since each level corresponds to a feature in the vector (this holds for the one-hot encoding, as well as when using occurrences or frequencies). The dimensionality of the vector can be controlled by filtering the levels, for instance, by using only the most frequent levels for each categorical attribute. However, such filtering may negatively impact the accuracy of the predictions. In the fourth subquestion, we aim at answering the following:

RQ1.4 (Categorical attribute levels) To what extent does filtering the levels of categorical attributes based on their frequencies improve the execution times of the predictions?

Train-test split. In order to simulate the real-life situation where prediction models are trained using historic data and applied to ongoing cases, we employ a temporal holdout split to divide the event log into train and test cases. Namely, the cases are ordered according to the start time and the first 80% are used for selecting the best model parameters and training the final model, while the remaining 20% are used to evaluate the performance of the final model. Specifically, splitting is done on the level of completed traces, so that different prefixes of the same trace remain in the same chunk (either all in the train set or all in the test set). In other words, the classifier is optimized and trained with all cases that started before a given date, and the testing is done only on cases that start afterwards. Note that, using this approach, some events in the training cases could still overlap with the test period. In order to avoid that, we cut the training cases so that events that overlap with the test period are discarded.

4.2.2. Classifier learning and bucketing parameters

We selected four classification algorithms for the experiments: random forest (RF), gradient boosted trees (XGBoost), logistic regression (logit), and support vector machines (SVM). We chose logistic regression because of its simplicity and wide application in various machine learning applications. SVM and RF have been used in existing outcome-oriented predictive monitoring studies. RF has shown to outperform many other methods (such as decision trees) in both predictive monitoring scenarios [54, 104] and in more general empirical studies [22]. We also included the XGBoost classifier which has recently gained attention and showed promising results when applied to business process data [82, 87]. Further-

Table 10: Hyperparameters and distributions used in optimization via TPE.

Method	Parameter	Distribution	Values
RF	Max features	Uniform	$x \in [0,1]$
XGBoost	Learning rate	Uniform	$x \in [0,1]$
	Subsample	Uniform	$x \in [0.5,1]$
	Max tree depth	Uniform integer	$x \in [4,30]$
	Colsample bytree	Uniform	$x \in [0.5,1]$
	Min child weight	Uniform integer	$x \in [1,6]$
Logit	Inverse of regularization strength (C)	Uniform integer	$2^x, x \in [-15,15]$
SVM	Penalty parameter of the error term (C)	Uniform integer	$2^x, x \in [-15,15]$
	Kernel coefficient (gamma)	Uniform integer	$2^x, x \in [-15,15]$
K-means	Number of clusters	Uniform integer	$x \in [2,50]$
KNN	Number of neighbors	Uniform integer	$x \in [2,50]$

more, a recent empirical study on the performance of classification algorithms across 165 datasets has shown that RF and boosted trees generally outperform other classifier learning techniques [70]. For the clustering-based bucketing approach (cf. Section 3.2.3), we use the k-means clustering algorithm, which is one of the most widely used clustering methods.

The classification algorithms as well as some of the bucketing methods (clustering and KNN) require one to specify a number of parameters. In order to achieve good performance with each of the techniques, we optimize the hyperparameters using the TPE algorithm, separately for each combination of a dataset, a bucketing method, and a sequence encoding method. For each combination of parameter values (i.e. a configuration) we perform 3-fold cross-validation, dividing the traces in the training set randomly into chunks (i.e. so that any two prefixes originating from the same trace remain in the same chunk), and we select the configuration that leads to the highest mean AUC calculated across the three folds. In the case of the prefix length based bucketing method, an optimal configuration is chosen for each prefix length separately (i.e. for each combination of a dataset, a bucketing method, an encoding approach and a prefix length). Table 10 presents the bounds and the sampling distributions for each of the parameters, given as input to the optimizer. In the case of RF and XGBoost, we found via exploratory testing that when increasing the *number of estimators (i.e. trees)* constructured per model, the accuracy converges around 500 estimators. In other words, when increasing the number of estimators further, there is little or no gain in accuracy. Therefore, we use a fixed value of $n\_estimators = 500$ throughout the experiments.

Both k-means and KNN require us to map each trace prefix into a feature vector in order to compute the Euclidean distance between pairs of prefixes. To this end, we apply the aggregation encoding approach, meaning that we map each

trace to a vector that tells us how many times each possible activity appears in the trace. In order to be consistent with the original methods [30, 60], we decided to use only the control flow information for the clustering and the identification of the nearest neighbors.

In the case of the state-based bucketing, we employ an approach similar to Lakshmanan et al. [52], since, conversely from the other state-based approaches, it does not require a process model as additional input. In particular, we use the last-activity encoding to map each trace prefix to a state in a DFG, meaning that one state is defined per possible activity and a trace prefix is mapped to the state corresponding to the last activity in the prefix. The reason for this choice is that this state abstraction leads to reasonably large buckets (the number of buckets produced by this approach is equal to the number of unique activities in the dataset, see Table 9). We also experimented with the multiset state abstraction approach, but it led to too many buckets, some of small size, so that in general there were not enough examples per bucket to train a classifier with sufficient accuracy. Conversely from the original method by Lakshmanan et al. [52], we omit the transition probabilities from the DFG, since we aim at making a prediction for any running case regardless of its frequency. Furthermore, their approach builds one classifier per decision point, i.e. the places in the process model where the execution splits into multiple alternative branches. Given that in our problem setting, we need to be able to make a prediction for a running trace after each event (not just at decision points), we employ a natural extension to their approach by building one classifier for every task in the DFG.

Note that in some cases it can happen that all of the training instances in a bucket belong to the same class. In these cases, no classifier is trained for this bucket and, instead, the test instances falling into this bucket are simply assigned the same class as the training instances (i.e. the assigned prediction score is either 0 or 1).

In case of logit and SVM, the features are standardized by subtracting the mean and scaling to unit variance before being given as input to the classifier.

4.2.3. Filtering and feature encoding parameters

As discussed in Section 3.2.2, training a classifier over the entire prefix log (containing all prefixes of all traces) can be time-consuming. Furthermore, we are only interested in making predictions for earlier events rather than making predictions towards the end of a trace. Additionally, we observe that the distributions of the lengths of the traces can be different within the classes corresponding to different outcomes (see Figures 41-42 in Appendix). When all instances of long prefixes belong to the same class, predicting the outcome for these (or longer) prefixes becomes trivial. Accordingly, during both the training and the evaluating phases, we vary the prefix length from 1 to the point where 90% of the minority class have finished (or until the end of the given trace, if it ends earlier than this point),

as both training and evaluation of the classifier would be unreliable when having very few sequences from either of the classes (for histograms of case lengths in both classes, see Figure 41 and Figure 42 in Appendix). For computational reasons, we set the upper limit of the prefix lengths to 40, except for the *bpic2017* datasets where we further reduced the limit to 20, since the predictions mostly converge by that point. The truncated lengths for each dataset can be seen in Table 9. We argue that setting a limit to the maximum prefix length is a reasonable design choice, as the aim of predictive process monitoring is to predict as early as possible and, therefore, we are more interested in predictions made for shorter prefixes. When answering RQ1.3, we additionally apply the gap-based filtering to the training set with $g \in \{3,5\}$. For instance, in case of $g = 5$, only prefixes of lengths 1, 6, 11, 16, 21, 26, 31, and 36 are included in the training set.

In Section 3.2.4, we noted that the aggregation encoding requires us to specify an aggregation function for each event attribute. For activities and resource attributes we use the count (frequency) aggregation function (i.e. how many times a given activity has been executed, or how many activities a given resource has executed). The same principle is applied to any other event attribute of a categorical type. For each numeric event attribute, we include 6 numeric features in the feature vector: mean, median, maximum, minimum, sum, and standard deviation. Furthermore, to answer RQ1.4, we filter each of the categorical attribute domains by using only the top $\{10,25,50,75,90\}$ percent of the most frequent levels from each attribute.

In order to provide a fair comparison of different encoding schemes, we have decided to evaluate each encoding separately, while the same encoding is applied to both control flow and data payload. An exception is the static encoding, which, conversely from the other encoding methods, extracts features from the case attributes rather than from the event attributes. In the experiments, we include the static encoding in every evaluated method, e.g. the "last state" method in the experiments refers to the static encoding for case attributes concatenated with the last state encoding for event attributes.

4.3. Results: accuracy and earliness

Table 11 reports the overall AUC and F-score for each dataset and method using XGBoost, while Tables 28, 29 and 30 in the Appendix report the same results for RF, logit, and SVM. The overall metric values (AUC or F-score) are obtained by first calculating the scores separately for each prefix length (using only prefixes of a given length) and then by taking the weighted average of the obtained scores, where the weights correspond to the number of prefixes used for the calculation of a given score. This weighting assures that the overall metrics are influenced equally by each prefix in the evaluation set, instead of being biased towards longer prefixes (i.e. where many cases have already finished). The best-performing classifiers are XGBoost, which achieves the highest (or shared highest) AUC in 15 out

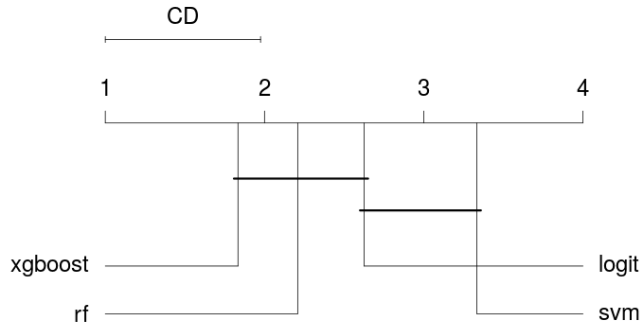


Figure 9: Comparison of all classifiers against each other with the Nemenyi test. The classifiers are compared in terms of the best AUC achieved in each of the 24 datasets. Groups of classifiers that are not significantly different (at $p < .05$) are connected.

of 24 datasets and the highest (or shared highest) F-score in 11 datasets, and RF, which achieves the best (or shared best) AUC in 11 datasets and F-score in 14. Logit achieves the highest (or shared highest) AUC in 7 and F-score in 6 datasets. SVM in general does not reach the same level of accuracy as the other classifiers, the only exceptions being *bpic2012_3*, *traffic*, and *hospital_2* (and only in terms of AUC).

In order to further assess the relative performance of the classifiers, we applied the Nemenyi test (as proposed in [23]) as a means for statistical comparison of classifiers over multiple datasets. In this setting, we compared the best AUC scores obtained by each classifier for a given dataset, i.e. we selected the best combination of bucketing and encoding technique for each dataset and classification algorithm. The resulting critical difference diagram (Figure 9), obtained using a 0.05 significance level, confirms that XGBoost is on average the best performing classifier, achieving an average rank of around 1.8. However, the difference between XGBoost, RF, and logit is not statistically significant (indicated by the horizontal line connecting these three classifiers). On the other hand, SVM performs significantly worse than XGBoost and RF. In the following, we analyze the results obtained by XGBoost in detail.

Concerning the bucketing and encoding methods, we can see in Table 11 that single_agg achieves the best AUC in 10 out of 24 datasets (and the best F-score in 9 datasets), followed by prefix_agg, which is the best in 8 datasets (4 in terms of F-score). They are followed by cluster_agg, state_agg, and prefix_index, which obtain the best AUC in 6, 5, and 4 datasets, respectively. With a few exceptions, which are discussed separately below, the last state encodings in general perform worse than their aggregation encoding counterparts and KNN performs worse than

Table 11: Overall AUC (F-score) for **XGBoost**

	bpic2011_1	**bpic2011_2**	**bpic2011_3**	**bpic2011_4**	**insurance_1**	**insurance_2**
single_laststate	0.85 (0.73)	0.91 (0.82)	0.94 (0.78)	**0.89** (**0.8**)	0.86 (0.36)	0.83 (0.44)
single_agg	0.94 (0.86)	**0.98** (**0.95**)	**0.98** (**0.94**)	0.86 (0.78)	0.9 (0.5)	0.8 (0.51)
knn_laststate	0.87 (0.86)	0.91 (0.93)	0.88 (0.81)	0.71 (0.64)	0.85 (0.49)	0.78 (0.49)
knn_agg	0.87 (0.85)	0.91 (0.93)	0.88 (0.82)	0.72 (0.64)	0.84 (0.52)	0.78 (0.5)
state_laststate	0.87 (0.73)	0.91 (0.84)	0.93 (0.8)	0.87 (0.77)	0.89 (0.55)	**0.84** (0.59)
state_agg	0.94 (0.84)	0.95 (0.91)	0.97 (0.89)	0.85 (0.75)	0.89 (0.59)	0.83 (**0.6**)
cluster_laststate	0.89 (0.74)	0.91 (0.86)	0.97 (0.9)	**0.89** (**0.8**)	0.87 (0.38)	0.81 (0.45)
cluster_agg	**0.95** (0.84)	0.97 (0.94)	0.97 (0.9)	0.84 (0.75)	**0.91** (0.57)	0.8 (0.45)
prefix_index	0.93 (0.79)	0.94 (0.82)	0.97 (0.8)	0.85 (0.74)	0.89 (0.55)	0.8 (0.55)
prefix_laststate	0.89 (0.76)	0.94 (0.86)	0.95 (0.74)	0.88 (0.78)	0.87 (0.42)	0.83 (0.53)
prefix_agg	0.94 (**0.87**)	**0.98** (0.94)	**0.98** (0.85)	0.86 (0.77)	0.9 (**0.6**)	0.83 (**0.6**)
	bpic2015_1	**bpic2015_2**	**bpic2015_3**	**bpic2015_4**	**bpic2015_5**	**production**
single_laststate	0.81 (0.42)	0.83 (0.34)	0.78 (0.45)	0.8 (0.41)	0.83 (0.67)	0.62 (0.57)
single_agg	**0.89** (**0.62**)	**0.92** (**0.75**)	0.9 (**0.75**)	0.85 (0.62)	0.87 (**0.77**)	0.7 (**0.59**)
knn_laststate	0.8 (0.37)	0.87 (0.64)	0.83 (0.6)	0.81 (0.55)	0.86 (0.72)	0.62 (0.56)
knn_agg	0.79 (0.39)	0.87 (0.67)	0.84 (0.61)	0.8 (0.56)	0.86 (0.72)	0.62 (0.55)
state_laststate	0.77 (0.46)	0.85 (0.56)	0.85 (0.56)	0.86 (0.46)	0.85 (0.66)	0.62 (0.51)
state_agg	0.8 (0.54)	0.88 (0.67)	0.87 (0.61)	**0.88** (0.63)	0.87 (0.72)	0.68 (0.56)
cluster_laststate	0.7 (0.39)	0.85 (0.46)	0.86 (0.66)	0.87 (0.61)	0.87 (0.71)	0.68 (0.57)
cluster_agg	0.88 (0.58)	**0.92** (0.73)	0.9 (0.72)	0.87 (**0.64**)	0.88 (0.76)	0.71 (**0.59**)
prefix_index	0.8 (0.46)	0.83 (0.39)	0.88 (0.63)	0.86 (0.5)	0.85 (0.64)	0.68 (0.57)
prefix_laststate	0.75 (0.32)	0.82 (0.28)	0.76 (0.4)	0.82 (0.4)	0.83 (0.62)	0.68 (0.56)
prefix_agg	0.84 (0.6)	0.88 (0.7)	**0.91** (0.71)	0.87 (0.6)	**0.89** (0.75)	**0.73** (0.57)
	sepsis_1	**sepsis_2**	**sepsis_3**	**bpic2012_1**	**bpic2012_2**	**bpic2012_3**
single_laststate	0.4 (**0.08**)	0.84 (**0.48**)	0.65 (0.24)	0.68 (0.61)	0.59 (0.09)	**0.7** (0.38)
single_agg	0.33 (0.0)	**0.85** (0.42)	0.72 (**0.35**)	**0.7** (0.59)	0.57 (0.16)	0.69 (0.36)
knn_laststate	**0.49** (0.07)	0.61 (0.01)	0.61 (0.23)	0.57 (**0.72**)	0.55 (**0.34**)	0.59 (0.45)
knn_agg	0.45 (0.05)	0.68 (0.05)	0.59 (0.15)	0.63 (0.61)	0.59 (0.05)	0.63 (**0.48**)
state_laststate	0.39 (0.0)	0.83 (0.42)	0.71 (0.13)	0.68 (0.62)	0.61 (0.12)	**0.7** (0.35)
state_agg	0.42 (0.0)	0.83 (0.43)	0.71 (0.2)	**0.7** (0.59)	0.6 (0.13)	**0.7** (0.33)
cluster_laststate	0.48 (0.04)	0.81 (0.42)	0.72 (0.11)	0.65 (0.6)	0.59 (0.12)	0.69 (0.3)
cluster_agg	0.46 (0.0)	0.82 (0.44)	0.7 (0.28)	0.67 (0.6)	0.6 (0.17)	**0.7** (0.29)
prefix_index	0.44 (0.07)	0.79 (0.36)	**0.73** (0.15)	0.68 (0.61)	**0.62** (0.13)	0.69 (0.35)
prefix_laststate	0.47 (0.07)	0.82 (0.38)	0.72 (0.06)	0.66 (0.61)	0.59 (0.1)	0.69 (0.35)
prefix_agg	0.48 (**0.08**)	0.8 (0.4)	0.71 (0.21)	0.68 (0.61)	0.59 (0.13)	**0.7** (0.35)
	bpic2017_1	**bpic2017_2**	**bpic2017_3**	**traffic**	**hospital_1**	**hospital_2**
single_laststate	0.81 (0.66)	**0.81** (0.42)	0.79 (0.73)	0.66 (0.67)	0.89 (**0.66**)	0.73 (0.11)
single_agg	**0.84** (0.71)	**0.81** (0.45)	0.79 (**0.76**)	0.66 (0.67)	**0.9** (0.63)	**0.76** (0.08)
knn_laststate	0.76 (0.66)	0.6 (0.04)	0.62 (0.53)	0.63 (0.69)	0.78 (0.37)	0.56 (0.06)
knn_agg	0.74 (0.59)	0.56 (0.0)	0.62 (0.52)	0.59 (**0.7**)	0.75 (0.47)	0.55 (0.01)
state_laststate	0.83 (0.7)	0.79 (0.45)	0.78 (0.72)	0.66 (0.67)	**0.9** (0.65)	0.74 (0.11)
state_agg	0.83 (0.7)	0.79 (**0.46**)	0.79 (0.73)	**0.67** (0.66)	**0.9** (0.65)	0.69 (0.11)
cluster_laststate	**0.84** (0.69)	0.8 (0.39)	0.78 (0.72)	0.66 (0.67)	0.89 (0.64)	0.68 (**0.12**)
cluster_agg	**0.84** (0.7)	0.79 (0.44)	0.79 (0.73)	**0.67** (0.66)	0.88 (0.64)	0.69 (0.09)
prefix_index	0.83 (**0.72**)	0.8 (0.45)	**0.8** (0.73)	**0.67** (0.66)	0.87 (0.64)	0.69 (0.11)
prefix_laststate	0.82 (0.7)	0.76 (0.44)	0.79 (0.72)	0.66 (0.66)	0.86 (0.63)	0.74 (0.08)
prefix_agg	**0.84** (0.71)	0.77 (0.45)	0.79 (0.73)	**0.67** (0.66)	0.87 (0.64)	0.74 (0.1)

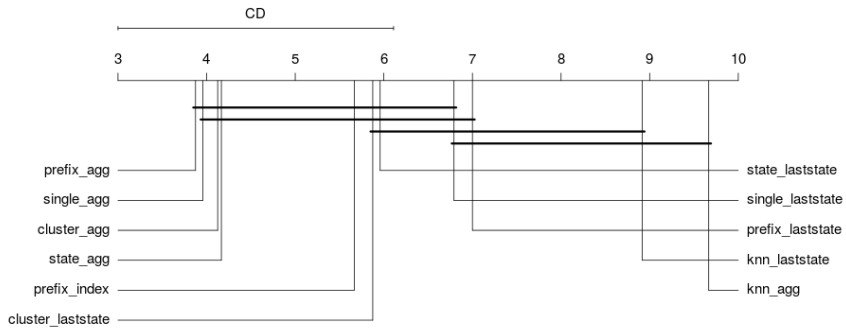


Figure 10: Comparison of the bucketing/encoding combinations with the Nemenyi test. The methods are compared in terms of AUC achieved in each of the 24 datasets using the **XGBoost** classifier. Groups of methods that are not significantly different (at $p < .05$) are connected.

the other bucketing methods.

The critical difference diagram in Figure 10 shows that in terms of the average rank, prefix_agg slightly outperforms single_agg, while both are closely followed by cluster_agg and state_agg. Despite a larger gap in the average ranks, the differences between prefix_agg and prefix_index, cluster_laststate, state_laststate, and single_laststate are not statistically significant either. On the other hand, prefix_laststate, knn_laststate, and knn_agg are found to perform significantly worse than prefix_agg.

Figures 11 and 12 present the prediction accuracy in terms of AUC for the 6 best performing methods (according to Figure 10), evaluated over different prefix lengths[2]. Each evaluation point includes prefix traces of exactly the given length. In other words, traces that are altogether shorter than the required prefix are left out of the calculation. Therefore, the number of cases used for evaluation is monotonically decreasing for increasing prefix lengths. In most of the datasets, we see that starting from a specific prefix length the methods with aggregation encoding achieve perfect prediction accuracy ($AUC = 1$). It is natural that the prediction task becomes trivial when cases are close to completion, especially if the labeling function is related to the control flow or to the data payload present in the event log. However, there are a few exceptions from this rule, namely, in the *bpic2012* and *sepsis* datasets, the results seem to decline on larger prefix sizes. To investigate this phenomenon, we recalculated the AUC scores on the longer traces only, i.e. for traces that have a length larger than or equal to the maximum considered trace length (see Figure 47 in Appendix). This analysis confirmed (with

[2]For a comparison of all the 11 methods, see Figures 45 and 46 in Appendix.

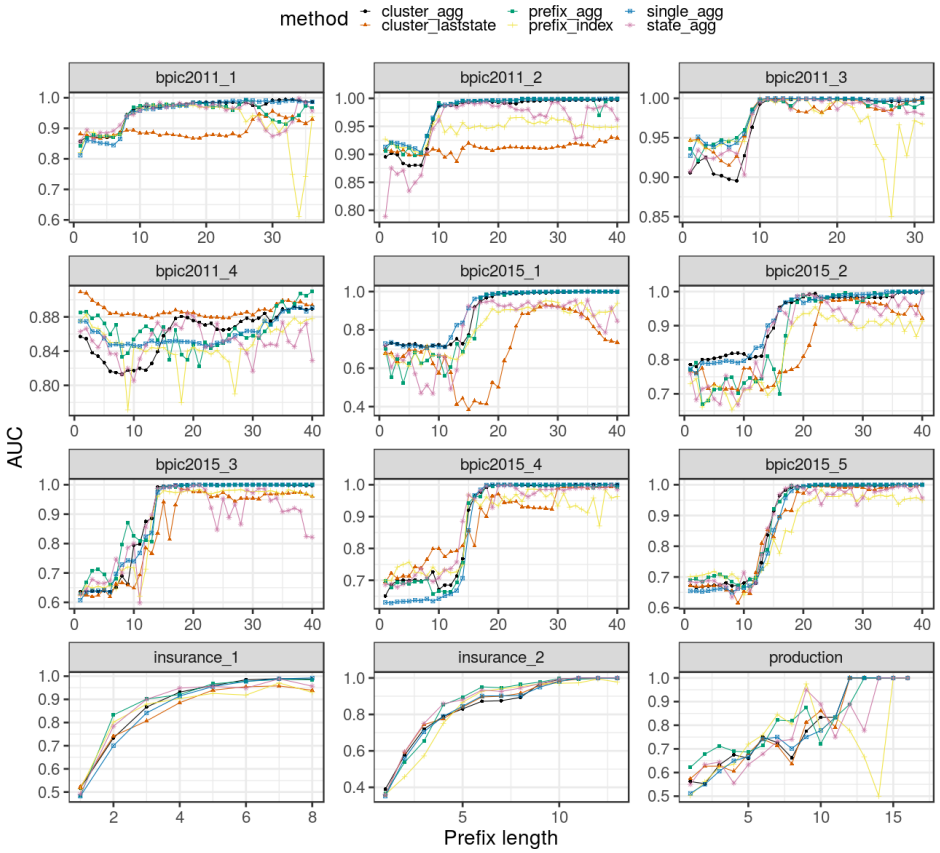


Figure 11: AUC across different prefix lengths using **XGBoost**.

the exception of *sepsis_1*, which we discuss separately later in this section) that the phenomenon is caused by the fact that the 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 classifier, dragging down the total AUC score on larger prefix lengths.

From the results presented above, we see that the choice of the bucketing method seems to have a smaller effect on the results than the sequence encoding. Namely, the best results are usually achieved using the aggregation encoding with either the single bucket, clustering, prefix length based, or state-based bucketing. In general, these methods achieve very comparable results. Still, it appears that in event logs with many trace variants (relative to the total number of traces), such as *insurace*, *production*, *bpic2012*, and *bpic2015_4* (see Table 9) it may be preferred to use a multiclassifier instead of a single bucket. However, this only holds if each bucket receives enough data to learn relevant patterns. For

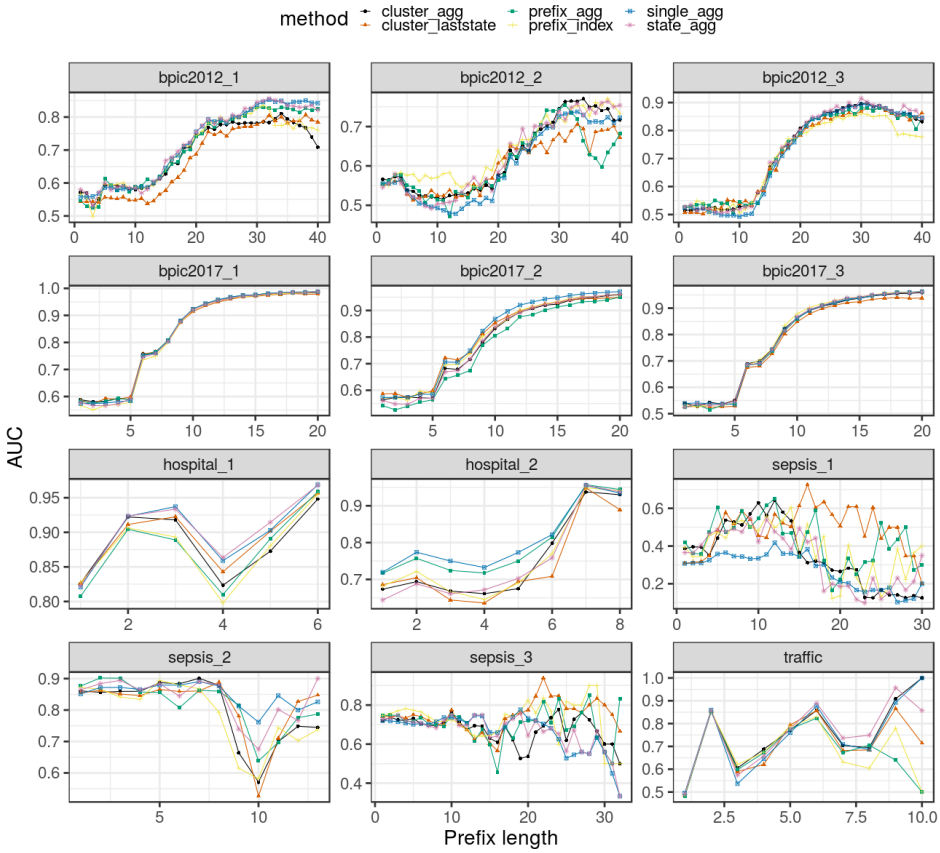
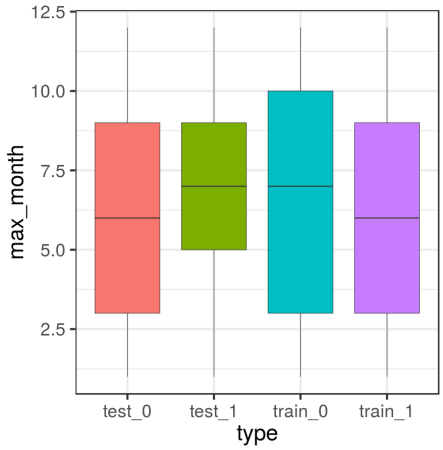


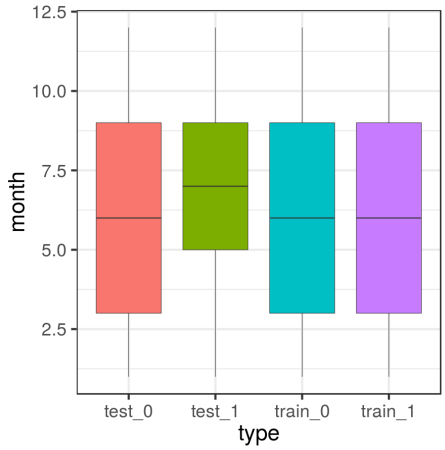
Figure 12: AUC across different prefix lengths using **XGBoost** (continued).

instance, when the number of categorical attribute levels is very high (as in the *bpic2011*, *bpic2015*, and *hospital* datasets), a single classifier is usually able to produce better predictions. Similarly, when the classes are very imbalanced (*hospital*, *bpic2017_2*, *sepsis_2*), it is likely that some buckets receive too limited information about the minority class and, therefore, a single classifier is recommended over a multiclassifier. The effect of having too many buckets can easily be seen in the case of state-based bucketing when the number of different event classes (and therefore, the number of buckets) is very large (see *bpic2011* and *bpic2015* in Figure 11 and the counts of training prefix traces in each bucket in Figures 43-44 in Appendix). As a result, each classifier receives a small number of traces for training, resulting in a very spiky performance across different prefix lengths. The same phenomenon can be seen in the case of prefix_agg, which usually achieves very good performance, but at times can produce unexpectedly inaccurate results (like in the longer prefixes of *bpic2011_1* and *bpic2012_2*). On the other hand, single_agg and cluster_agg in general produce stable and reliable results on all datasets and across all prefix sizes. The optimal number of clusters in case of cluster_agg with XGBoost was often found to be small, i.e. between 2-7 (see Table 26 in Appendix), which explains why these two methods behave similarly. In some cases where the optimized number of clusters was higher, e.g. *bpic2012_1* and *hospital_2*, the accuracy of cluster_agg drops compared to single_agg.

We can see from Figure 11 that in several cases (e.g. *bpic2011*, *bpic2015*, *bpic2012_1*, and *sepsis_2*), all the methods achieve a similar AUC on shorter prefixes, but then quickly grow apart as the size of the prefix increases. In particular, the aggregation encoding seems to be able to carry along relevant information from the earlier prefixes, while the last state encoding entails more limited information that is often insufficient for making accurate predictions. Comparing the overall AUC in Table 11, the last state encodings outperform the other methods only in three datasets. One such exceptional case is *bpic2011_4*, where single_laststate and cluster_laststate considerably outperform their aggregation encoding counterparts. A deeper investigation of this case revealed that this is due to overfitting in the presence of a concept drift in the dataset. In particular, the aggregation encodings yielded a more complex classifier (e.g. the optimized maximum tree depth is 15 in case of single_agg and 6 in case of single_laststate), memorizing the training data completely. However, a concept drift occurs in the relationship between some data attributes and the class labeling, which affects the aggregated features more than the "as-is" features (see Figure 13). Another exception is *sepsis_1*, where the best results are achieved by knn_laststate. In this dataset, all the methods consistently yield an AUC less than 0.5 (i.e. worse than random). Further investigation revealed that this phenomenon is also due to a concept drift in the dataset, which makes it impossible to learn useful patterns in case of a temporal train-test split. For instance, Figure 14 illustrates that in the training set the values of *CRP* are larger in positive instances, while in the test



(1) Attribute = *max_month*. Significant differences in means between train_0–test_0 ($Z = 6.077$, $p < .001$) and train_1–test_1 ($Z = 7.972$, $p < .001$).



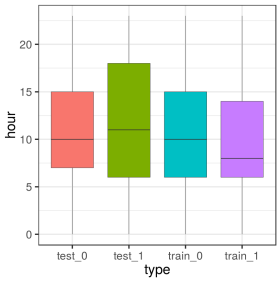
(2) Attribute = *month*. The difference is significant between train_1–test_1 ($Z = 8.754$, $p < .001$), but not between train_0–test_0 ($Z = .028$, $p = .978$).

Figure 13: Concept drift in the *bpic2011_4* log. The distributions of the variables are different across the two classes in the train and the test set. The drift becomes more evident in the *max_month* feature used by the aggregation encoding, while it is not so severe in the original *month* feature used by the last state encoding. Statistical significance of the differences is assessed using Wilcoxon signed-rank test.

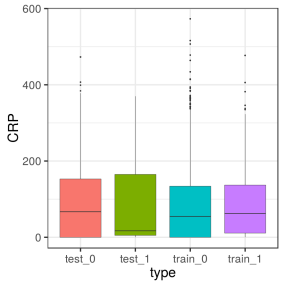
set the *CRP* values are larger in negative instances. The third exceptional dataset is *insurance_2*, where state_laststate slightly outperforms other techniques. We did not find any peculiarities in this dataset that would explain this phenomenon, however, the differences in scores (between, e.g. state_laststate and state_agg) are much smaller compared to the previous two datasets.

These peculiar results on *bpic2011_4* and *sepsis_1* illustrate that in order to obtain reliable predictions from a predictive model, it is essential that the train and test cases follow the same distribution. Therefore, it is important to check for concept drift before training a predictive model and choose the training set so that there is no drift within the training period. Also, one should keep monitoring for drift while the model is deployed and consider retraining the model in case signs of a drift are detected.

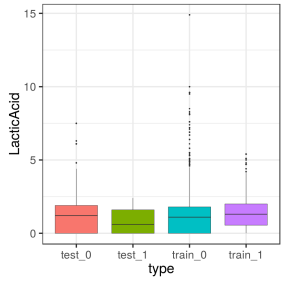
We can also observe that the index-based encoding, although lossless, in general does not outperform the lossy encoding schemes, reaching the highest overall AUC only in 4 datasets: *bpic2012_2*, *bpic2017_3*, *sepsis_3*, and *traffic*. In these logs the number of levels in dynamic categorical attributes is not very high (see Table 9), which helps to keep the size of the feature vector in reasonable bounds. Still, even in these cases the difference in AUC compared to the other methods (such as prefix_agg) is marginal. In fact, in some datasets (e.g. *hospital_2* and



(3) Attribute = *hour*. Significant difference between train_1–test_1 ($Z = 3.651$, $p < .001$), but not between train_0–test_0 ($Z = 1.373$, $p = .17$).



(4) Attribute = *CRP*. Significant differences between train_0–test_0 ($Z = 3.61$, $p < .001$) and train_1–test_1 ($Z = 2.492$, $p = .013$).



(5) Attribute = *LacticAcid*. Significant differences in train_0–test_0 ($Z = 3.064$, $p = .002$) and train_1–test_1 ($Z = 7.337$, $p < .001$).

Figure 14: Concept drift in data attributes in *sepsis_1* log. The distributions of the variables are different across the two classes in the train and the test set. Statistical significance of the differences is assessed using Wilcoxon signed-rank test.

sepsis_2) the index-based encoding performs even worse than the last state encoding. This suggests that in the given datasets, the order of events is not that relevant for determining the final outcome of the case. Instead, combining the knowledge from all events performed so far provides much more signal. Alternatively, it may be that the order of events (i.e. the control flow) does matter in some cases, but the classifiers considered in this study (including XGBoost) are not able to infer high-level control-flow features by themselves, which would explain why we see that even the simple aggregation-based methods outperform the index-based encoding. This phenomenon deserves a separate in-depth study.

These observations, further supported by the fact that KNN does not appear among the top performing methods, lead to the conclusion that it is preferable to build few classifiers (or even just a single one), with a larger number of traces as input. XGBoost seems to be a classifier sophisticated enough to derive the "bucketing patterns" by itself when necessary. Another advantage of the single_agg method over cluster_agg is the simplicity of a single bucket. In fact, no additional preprocessing step for bucketing the prefix traces is needed. On the other hand, clustering (regardless of the clustering algorithm) comes with a set of parameters, such as the number of clusters in k-means, that need to be tuned for optimal performance. Therefore, the time and effort needed from the user of the system for setting up the prediction framework can be considerably higher in case of cluster_agg, which makes single_agg the overall preferred choice in terms of accuracy and earliness. This discussion concludes the answer to RQ1.1 (How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of accuracy and earliness of the predictions?).

4.4. Results: time performance

The time measurements for all of the methods and classifiers, calculated as averages over 5 identical runs using the final (optimal) parameters, are presented in Tables 12 and 13 (XGBoost), Tables 31 and 32 (RF), Tables 33 and 34 (logit), and Tables 35 and 36 (SVM). In the offline phase, the fastest of the four classifiers is logit. The ordering of the others differs between the small (*production*, *bpic2011*, *bpic2015*, *insurance*, and *sepsis*) and the large (*bpic2017*, *traffic*, *hospital*) datasets. In the former group, the second fastest classifier is SVM, usually followed by RF and, then, XGBoost. Conversely, in the larger datasets, XGBoost appears to scale better than the others, while SVM tends to be the slowest of the three. In terms of online prediction time, logit, SVM, and XGBoost yield comparable performance, while RF is usually slower than the others. In the following, we will, again, analyse deeper the results obtained with the XGBoost classifier.

Recall that the KNN method (almost) skips the offline phase, since all the classifiers are built at runtime. The offline time for KNN still includes the time for constructing the prefix log and setting up the matrix of encoded historical prefix traces, which is later used for finding the nearest neighbors for running traces. Therefore, the offline times in case of the KNN approaches are almost negligible. The offline phase for the other methods (i.e. excluding KNN) takes between 3 seconds on the smallest dataset (*production*) to 6 hours and 30 minutes on *hospital_1*. There is no clear winner between the last state encoding and the corresponding aggregation encoding counterparts, which indicates that the time for applying the aggregation functions is small compared to the time taken for training the classifier. In general, the index-based encoding takes the most time in the offline phase.

In terms of bucketing, the fastest approach in the offline phase is usually the state-based bucketing, followed by either the prefix length or the clustering based method, while the slowest is single bucket. This indicates that the time taken to train multiple ("small") classifiers, each trained with only a subset of the original data, is smaller than training a few ("large") classifiers using a larger portion of the data.

In general, all methods are able to process an event in less than 100 milliseconds during the online phase (the times in Tables 12 and 13 are in milliseconds per processed event in a running trace). Exceptions are *hospital_1* and *hospital_2*, where processing an event takes around 0.3–0.4 seconds. The online execution times are very comparable across all the methods, except for KNN and prefix_index. While prefix_index often takes double time with respect to the other methods, the trends for KNN are less straightforward. Namely, in some datasets (*bpic2012*, *sepsis*, *production*, *insurance*, and *traffic*), the KNN approaches take considerably more time than the other techniques, which can be explained by the fact that these approaches train a classifier at runtime. However, somewhat surprisingly, in other datasets (*hospital* and *bpic2011* datasets) the KNN approaches

yield the best execution times even at runtime. A possible explanation for this is that in cases where all the selected nearest neighbors are of the same class, no classifier is trained and the class of the neighbors is immediately returned as the prediction. However, note that the overall AUC in these cases is 7–21 percentage points lower than that of the best method (see Table 11). In the online phase, the overhead of applying the aggregation functions becomes more evident, with the last state encoding almost always outperforming the aggregation encoding methods by a few milliseconds. The fastest method in the online phase tends to be prefix_laststate, which outperforms the others in 17 out of 24 datasets. This method is followed by knn_laststate, state_laststate, and single_laststate.

In terms of online execution times, the trends observed in the case of XGBoost are in line with those of other classifiers. However, there are some differences in the offline phase. Namely, in case of RF, the single classifiers perform relatively better as compared to bucketing methods. Furthermore, the difference between the encoding methods becomes more evident, with the last state encodings usually outperforming their aggregation encoding counterparts. The index-based encoding is still the slowest of the techniques. In case of logit, all the methods achieve comparable offline times, except for the index-based encoding and the clustering based bucketings, which are slower than the others. In case of SVM, the single_laststate method tends to be much slower than other techniques. This discussion concludes the answer to RQ1.2 (How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of execution times?).

4.5. Results: gap-based filtering

In order to investigate the effects of gap-based filtering on the execution times and the accuracy, we selected 4 methods based on their performance in the above subsections: single_agg, single_laststate, prefix_index, and prefix_agg combined with the XGBoost classifier. The first three of these methods were shown to take the most time in the offline phase, i.e. they have the most potential to benefit from a filtering technique. Also, single_agg and prefix_agg achieved the highest overall AUC scores, which makes them the most attractive candidates to apply in practice. Furthermore, we selected 6 datasets which are representative in terms of their sizes (i.e. number of traces), consist of relatively long traces on average, and did not yield a very high accuracy very early in the trace.

Figures 15–17 plot the performance of the methods over different gap sizes, i.e. on the x-axis, $g = 1$ corresponds to no filtering (using prefixes obtained after every event), $g = 3$ to using prefixes obtained after every 3rd event, and $g = 5$ to prefixes after every 5th event. In Figure 15, we can see that using $g = 3$ yields an improvement of about 2-3 times in the offline execution times, while using $g = 5$, the improvement is usually around 3-4 times, as compared to no filtering ($g = 1$). For instance, in case of single_agg on the *bpic2017_2* dataset with $g = 5$, this

Table 12: Execution times for **XGBoost**.

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	418.35 ± 0.56	69 ± 98	581.68 ± 1.09	62 ± 96	217.69 ± 1.38	71 ± 96
single_agg	317.18 ± 0.58	69 ± 99	342.3 ± 2.02	62 ± 97	271.33 ± 0.54	71 ± 97
knn_laststate	5.9 ± 0.31	**44 ± 65**	9.82 ± 0.66	**37 ± 59**	4.14 ± 0.06	**48 ± 72**
knn_agg	6.63 ± 0.12	52 ± 76	9.79 ± 0.44	46 ± 72	4.57 ± 0.06	61 ± 91
state_laststate	142.53 ± 0.31	52 ± 72	181.87 ± 0.92	48 ± 74	86.78 ± 0.52	53 ± 70
state_agg	183.67 ± 0.79	61 ± 84	169.98 ± 0.47	58 ± 90	119.84 ± 0.2	62 ± 82
cluster_laststate	211.88 ± 1.0	66 ± 112	592.92 ± 4.84	67 ± 112	93.83 ± 0.48	57 ± 98
cluster_agg	341.4 ± 1.89	70 ± 113	381.61 ± 1.87	62 ± 111	94.45 ± 0.71	72 ± 111
prefix_index	763.8 ± 20.33	114 ± 68	1405.87 ± 88.64	126 ± 62	428.6 ± 19.49	113 ± 69
prefix_laststate	290.64 ± 1.44	57 ± 86	264.71 ± 4.58	50 ± 81	108.84 ± 0.33	59 ± 84
prefix_agg	172.8 ± 12.28	56 ± 82	274.29 ± 9.08	53 ± 82	125.61 ± 7.42	58 ± 80

	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	418.65 ± 17.42	62 ± 97	263.56 ± 0.38	20 ± 30	134.87 ± 0.59	18 ± 29
single_agg	319.27 ± 10.75	62 ± 98	105.58 ± 0.39	22 ± 34	282.97 ± 8.14	20 ± 32
knn_laststate	8.99 ± 0.23	**42 ± 67**	8.44 ± 0.04	31 ± 51	11.46 ± 0.63	34 ± 59
knn_agg	9.47 ± 0.07	57 ± 86	9.45 ± 0.06	45 ± 72	11.95 ± 0.12	38 ± 66
state_laststate	156.45 ± 1.1	48 ± 73	25.2 ± 0.08	25 ± 44	30.28 ± 0.2	27 ± 45
state_agg	301.91 ± 7.54	58 ± 90	53.39 ± 0.34	29 ± 48	64.72 ± 0.08	31 ± 49
cluster_laststate	274.97 ± 8.03	69 ± 112	46.71 ± 0.58	37 ± 58	52.8 ± 1.6	36 ± 60
cluster_agg	252.81 ± 6.36	69 ± 112	62.25 ± 0.48	29 ± 46	135.33 ± 1.21	28 ± 45
prefix_index	794.59 ± 35.34	111 ± 59	396.03 ± 4.79	51 ± 10	442.17 ± 8.47	48 ± 13
prefix_laststate	344.73 ± 13.5	48 ± 80	62.17 ± 0.26	**7 ± 9**	45.27 ± 0.27	**6 ± 7**
prefix_agg	441.82 ± 1.01	63 ± 98	57.14 ± 0.16	10 ± 10	74.98 ± 1.04	8 ± 8

	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	150.3 ± 0.44	21 ± 33	67.21 ± 0.09	18 ± 27	83.78 ± 0.31	17 ± 26
single_agg	627.3 ± 3.48	21 ± 35	191.22 ± 0.21	19 ± 29	428.23 ± 1.24	17 ± 28
knn_laststate	18.9 ± 0.68	37 ± 61	7.38 ± 0.08	29 ± 50	17.22 ± 0.14	36 ± 60
knn_agg	19.57 ± 0.84	41 ± 69	7.5 ± 0.35	34 ± 58	19.67 ± 0.53	37 ± 87
state_laststate	45.06 ± 0.03	29 ± 49	18.5 ± 0.06	25 ± 42	37.97 ± 0.08	23 ± 38
state_agg	86.47 ± 0.06	33 ± 53	30.76 ± 0.07	29 ± 45	73.2 ± 0.46	26 ± 41
cluster_laststate	102.82 ± 0.26	35 ± 60	34.94 ± 0.19	34 ± 62	72.76 ± 0.26	35 ± 60
cluster_agg	181.85 ± 1.13	28 ± 45	61.06 ± 0.84	26 ± 40	74.79 ± 0.7	20 ± 34
prefix_index	2155.36 ± 80.83	55 ± 15	261.21 ± 0.5	41 ± 8	550.87 ± 4.23	48 ± 12
prefix_laststate	98.37 ± 0.7	**7 ± 9**	37.59 ± 0.06	**6 ± 7**	109.77 ± 0.1	**6 ± 8**
prefix_agg	108.91 ± 2.83	9 ± 9	40.83 ± 0.38	8 ± 8	92.61 ± 0.88	8 ± 9

	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	7.28 ± 0.12	25 ± 21	36.0 ± 0.1	38 ± 31	15.52 ± 0.09	33 ± 30
single_agg	4.3 ± 0.13	28 ± 25	12.81 ± 0.12	40 ± 33	78.17 ± 0.62	34 ± 32
knn_laststate	1.09 ± 0.01	51 ± 49	0.66 ± 0.01	49 ± 44	0.96 ± 0.01	48 ± 36
knn_agg	0.76 ± 0.0	62 ± 60	0.66 ± 0.0	57 ± 53	1.06 ± 0.01	69 ± 50
state_laststate	2.69 ± 0.11	**23 ± 19**	11.45 ± 0.1	**31 ± 22**	15.22 ± 0.06	**26 ± 21**
state_agg	4.37 ± 0.04	30 ± 26	17.67 ± 0.11	40 ± 31	30.62 ± 0.17	35 ± 30
cluster_laststate	6.94 ± 0.06	30 ± 30	32.75 ± 0.11	39 ± 37	42.42 ± 0.15	37 ± 34
cluster_agg	6.35 ± 0.07	35 ± 31	30.06 ± 0.2	52 ± 43	47.2 ± 0.14	45 ± 43
prefix_index	13.81 ± 0.04	49 ± 10	58.79 ± 0.58	90 ± 4	80.56 ± 0.12	91 ± 4
prefix_laststate	5.62 ± 0.35	28 ± 23	9.44 ± 0.03	**31 ± 21**	22.3 ± 0.07	**26 ± 21**
prefix_agg	5.89 ± 0.01	28 ± 23	17.17 ± 0.05	35 ± 24	24.66 ± 0.07	30 ± 24

Table 13: Execution times for **XGBoost** (continued).

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	40.05 ± 0.15	27 ± 31	18.68 ± 0.03	33 ± 33	81.33 ± 0.08	29 ± 31
single_agg	39.65 ± 0.2	29 ± 33	21.86 ± 0.15	36 ± 35	54.24 ± 0.18	31 ± 34
knn_laststate	2.85 ± 0.04	54 ± 58	1.04 ± 0.04	64 ± 62	2.08 ± 0.03	57 ± 60
knn_agg	2.91 ± 0.04	61 ± 66	1.07 ± 0.03	83 ± 78	1.95 ± 0.05	68 ± 70
state_laststate	25.72 ± 0.22	29 ± 33	15.5 ± 0.07	35 ± 36	41.82 ± 0.17	31 ± 33
state_agg	28.85 ± 0.13	32 ± 36	24.5 ± 0.06	39 ± 39	61.92 ± 0.82	34 ± 37
cluster_laststate	27.8 ± 0.1	**26 ± 32**	17.28 ± 0.2	37 ± 36	59.4 ± 0.33	32 ± 35
cluster_agg	21.22 ± 0.14	28 ± 33	19.71 ± 0.07	39 ± 39	64.08 ± 0.39	33 ± 37
prefix_index	93.4 ± 1.62	37 ± 24	23.02 ± 0.16	41 ± 26	43.06 ± 0.29	38 ± 25
prefix_laststate	21.86 ± 0.12	**26 ± 29**	23.41 ± 0.12	**31 ± 29**	28.9 ± 0.08	**27 ± 29**
prefix_agg	26.75 ± 0.15	28 ± 31	20.73 ± 0.05	34 ± 33	28.3 ± 0.19	29 ± 31
	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	402.09 ± 3.02	7 ± 11	181.94 ± 6.17	7 ± 10	454.27 ± 1.31	7 ± 11
single_agg	290.33 ± 0.99	8 ± 12	268.54 ± 2.51	8 ± 12	268.29 ± 1.14	8 ± 12
knn_laststate	29.8 ± 0.58	82 ± 93	28.27 ± 0.26	117 ± 132	34.41 ± 0.03	156 ± 171
knn_agg	29.86 ± 0.59	143 ± 159	30.07 ± 0.59	403 ± 434	29.65 ± 0.55	38 ± 52
state_laststate	234.38 ± 0.68	8 ± 10	251.72 ± 0.8	8 ± 10	132.18 ± 0.61	8 ± 10
state_agg	205.54 ± 4.88	10 ± 12	640.61 ± 8.42	10 ± 12	293.0 ± 3.62	9 ± 12
cluster_laststate	718.57 ± 15.83	9 ± 14	533.58 ± 2.9	9 ± 13	141.21 ± 1.96	10 ± 15
cluster_agg	200.12 ± 0.45	9 ± 13	741.83 ± 19.09	10 ± 16	264.94 ± 6.29	10 ± 16
prefix_index	2637.76 ± 3.59	36 ± 12	5857.46 ± 19.9	36 ± 12	2815.82 ± 14.87	34 ± 11
prefix_laststate	313.24 ± 3.13	**4 ± 3**	253.69 ± 1.02	**4 ± 3**	240.8 ± 1.81	**4 ± 3**
prefix_agg	562.56 ± 7.46	5 ± 5	409.94 ± 4.45	5 ± 5	223.96 ± 10.6	5 ± 5
	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	1845.39 ± 36.22	19 ± 23	2116.09 ± 41.53	18 ± 22	2587.32 ± 50.78	19 ± 23
single_agg	4569.55 ± 89.68	19 ± 24	7042.71 ± 138.21	21 ± 26	2021.16 ± 39.67	20 ± 25
knn_laststate	124.61 ± 2.45	1476 ± 1389	125.47 ± 2.46	1474 ± 1386	125.41 ± 2.46	1477 ± 1390
knn_agg	134.2 ± 2.63	1601 ± 1504	125.39 ± 2.46	1488 ± 1398	125.58 ± 2.46	1480 ± 1393
state_laststate	1568.31 ± 30.78	18 ± 20	2661.92 ± 66.32	19 ± 23	2771.55 ± 54.39	18 ± 20
state_agg	2357.57 ± 46.27	20 ± 22	4387.99 ± 194.51	22 ± 25	3051.07 ± 59.88	20 ± 22
cluster_laststate	780.47 ± 15.32	17 ± 21	2894.35 ± 56.8	19 ± 23	2032.37 ± 39.89	16 ± 20
cluster_agg	2556.04 ± 50.16	16 ± 20	4233.3 ± 83.08	20 ± 24	1800.81 ± 35.34	17 ± 20
prefix_index	19581.63 ± 384.29	72 ± 9	15822.79 ± 310.52	81 ± 7	17384.94 ± 341.18	78 ± 13
prefix_laststate	2745.56 ± 53.88	**14 ± 15**	1863.41 ± 36.57	**17 ± 18**	1660.26 ± 32.58	**15 ± 15**
prefix_agg	4751.78 ± 93.25	18 ± 24	2712.28 ± 53.23	**17 ± 17**	2680.83 ± 52.61	16 ± 17
	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	3169.91 ± 76.52	62 ± 34	23191.09 ± 455.13	380 ± 249	5303.26 ± 104.08	410 ± 270
single_agg	4018.67 ± 82.0	71 ± 40	5999.77 ± 6.31	401 ± 264	8634.78 ± 169.46	426 ± 281
knn_laststate	400.63 ± 7.64	96 ± 60	117.9 ± 2.31	**54 ± 37**	320.52 ± 6.29	**104 ± 90**
knn_agg	444.14 ± 8.93	158 ± 101	213.95 ± 4.2	79 ± 51	315.68 ± 6.2	114 ± 93
state_laststate	1088.97 ± 18.94	74 ± 43	10056.83 ± 197.37	312 ± 257	7321.59 ± 143.69	293 ± 235
state_agg	828.18 ± 14.85	75 ± 41	16417.43 ± 322.19	392 ± 238	16783.7 ± 329.38	363 ± 212
cluster_laststate	2152.68 ± 3.26	64 ± 37	11463.75 ± 224.98	296 ± 270	4704.15 ± 92.32	302 ± 282
cluster_agg	1572.57 ± 3.52	69 ± 40	3297.8 ± 64.72	339 ± 254	9174.11 ± 180.04	353 ± 264
prefix_index	2895.03 ± 56.82	102 ± 13	16114.53 ± 316.25	930 ± 136	21000.14 ± 412.13	960 ± 220
prefix_laststate	2963.04 ± 3.57	**59 ± 32**	6756.85 ± 132.6	380 ± 265	9208.33 ± 180.71	323 ± 219
prefix_agg	1669.98 ± 4.9	62 ± 34	11395.98 ± 223.65	399 ± 241	7993.41 ± 156.87	353 ± 204

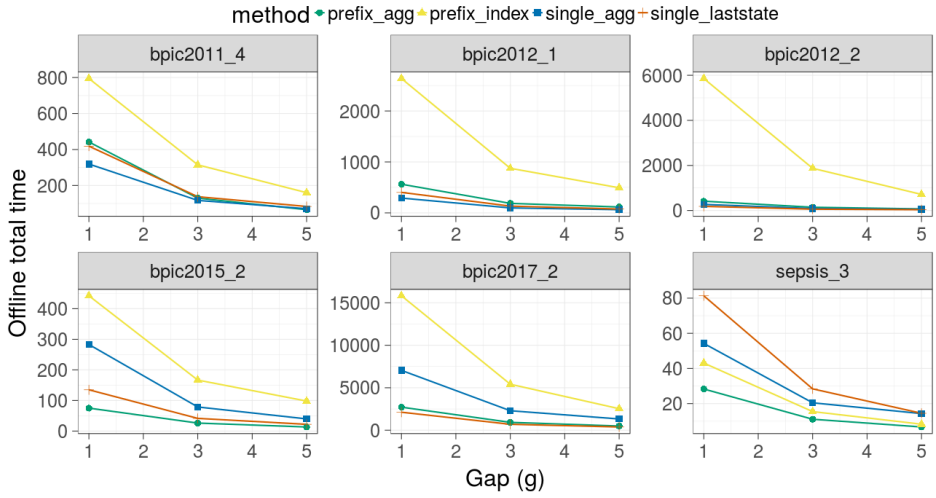


Figure 15: Offline times across different gaps (XGBoost).

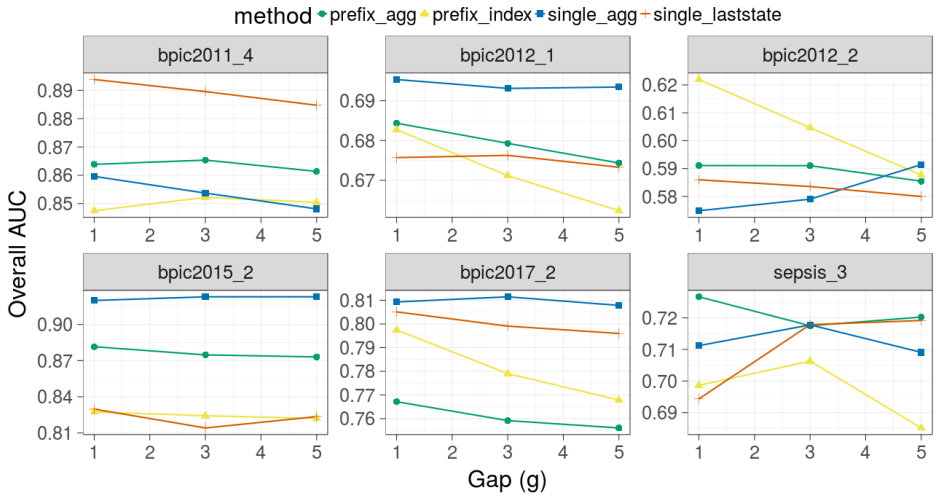


Figure 16: AUC across different gaps (XGBoost).

means that the offline phase takes about 30 minutes instead of 2 hours. At the same time, the overall AUC remains at the same level, sometimes even increasing when a filtering is applied (Figure 16). On the other hand, the gap-based filtering only has a marginal (positive) effect on the online execution times, which usually remain on the same level as without filtering (Figure 17). This concludes the answer to RQ1.3 (To what extent does gap-based filtering improve the execution times of the predictions?).

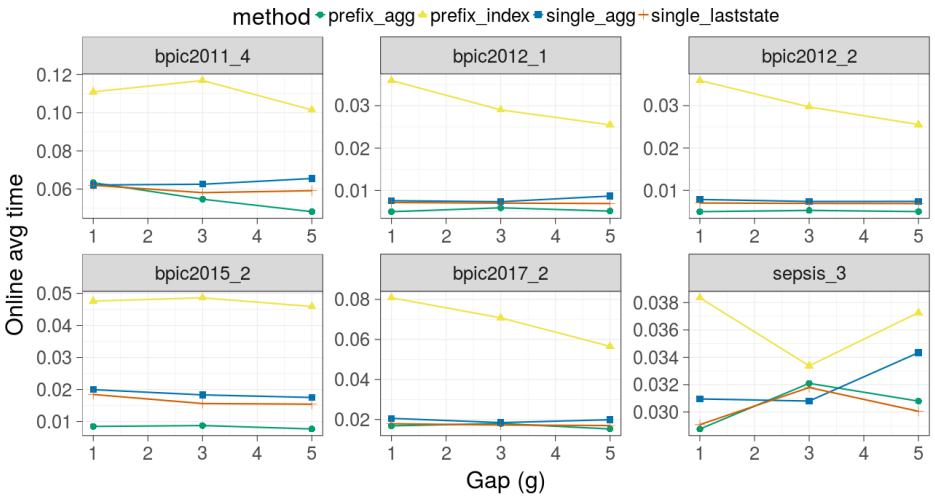


Figure 17: Online times across different gaps (XGBoost).

4.6. Results: categorical domain filtering

To answer RQ1.4 (To what extent does filtering the levels of categorical attributes based on their frequencies improve the execution times of the predictions?), we proceed with the 4 methods as discussed in the previous subsection. To better investigate the effect of filtering the categorical attribute levels, we distinguish between static and dynamic categorical attributes. For investigating the effects of dynamic categorical domain filtering, we selected 9 datasets that contain a considerable number of levels in the dynamic categorical attributes.

Both the offline (Figure 18) and the online (Figure 20) execution times tend to increase linearly when the proportion of levels is increased. As expected, the prefix_index method benefits the most from the filtering, since the size of the feature vector increases more rapidly than in the other methods when more levels are added (the vector contains one feature per level per event). Although the overall AUC is negatively affected by the filtering of levels (see Figure 19), reasonable tradeoffs can still be found. For instance, when using 50% of the levels in case of single_agg on the *hospital_2* dataset, the AUC is almost unaffected, while the training time has decreased by more than 30 minutes and the online execution times have decreased by a half.

We performed similar experiments by filtering the static categorical attribute domains, selecting 6 datasets that contain a considerable number of levels in these attributes. However, the improvement in execution times was marginal compared to those obtained when using dynamic attribute filtering (see Figures 48-50 in Appendix). This is natural, since the static attributes have a smaller effect on the size of the feature vector (each level occurs in the vector only once). This concludes the answer to RQ1.4 (To what extent does filtering levels of categorical attributes improve the execution times?).

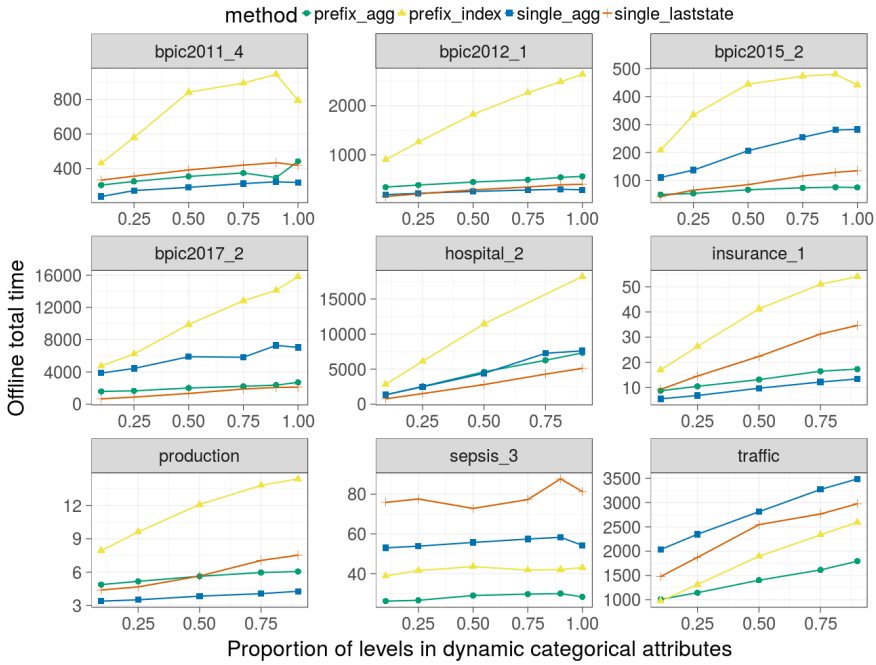


Figure 18: Offline times across different filtering proportions of **dynamic** categorical attribute levels (XGBoost).

4.7. Summary

In this chapter, we performed a comparative evaluation of the 11 existing predictive process monitoring techniques identified in Chapter 3. The benchmark was executed using a unified experimental set-up and 24 predictive monitoring tasks constructed from 9 real-life event logs. To ensure a fair evaluation, all the selected techniques were implemented as a publicly available consolidated framework, which is designed to incorporate additional datasets and methods.

The results of the benchmark show that the most reliable and accurate results (in terms of AUC) are obtained using a lossy (aggregation) encoding of the sequences, e.g. the frequencies of the performed activites rather than the ordered activities. One of the main benefits of this encoding is that it enables to represent all prefix traces, regardless of their length, in the same number of features. This way, a single classifier can be trained over all of the prefix traces, allowing the classifier to derive meaningful patterns by itself. These results put into question the existing opinion in the literature about the superiority of a lossless encoding of the traces (index-based encoding) that requires prefixes to be divided into buckets according to their length, while multiple classifiers are trained on each such subset of prefixes.

A threat to validity of the benchmark is related to the comprehensiveness of the conducted experiments. In particular, only one clustering method was tested,

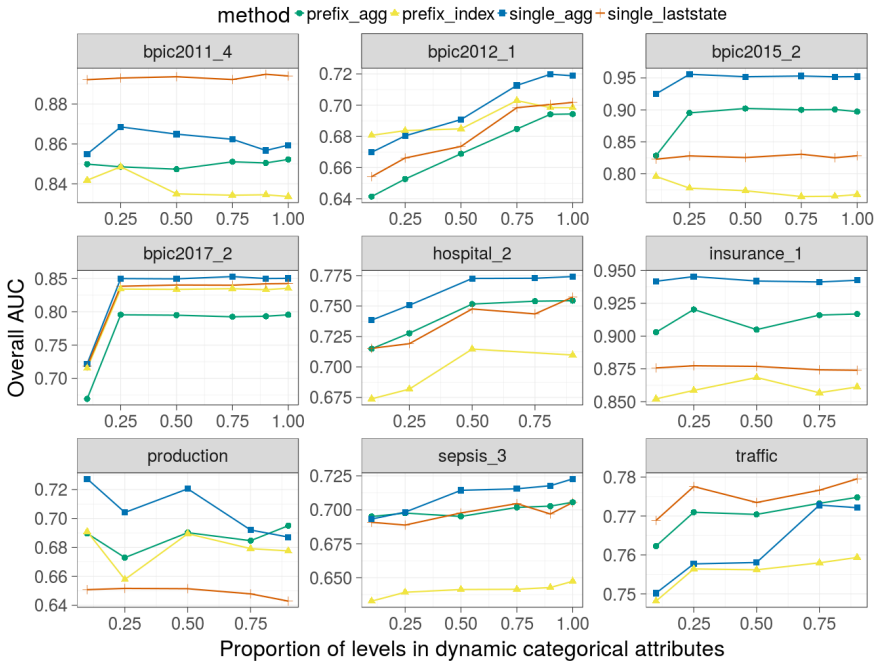


Figure 19: AUC across different filtering proportions of **dynamic** categorical attribute levels (XGBoost).

a single state abstraction was used when building the transition systems for state-based bucketing, and four classification algorithms were applied. It is possible that there exists, for example, a combination of an untested clustering technique and a classifier that outperforms the settings used in this study. Also, although the hyperparameters were optimized using a state-of-the-art hyperparameter optimization technique, it is possible that using more iterations for optimization or a different optimization algorithm, other parameter settings would be found that outperform the settings used in the current evaluation. Furthermore, the generalizability of the findings is to some extent limited by the fact that the experiments were performed on a limited number of prediction tasks (24), constructed from 9 event logs. Although these are all real-life event logs from different application fields that exhibit different characteristics, it may be 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 the framework can be used for future experiments. Also, the preprocessed datasets constructed from the 8 publicly available event logs are included together with the tool implementation in order to enhance the reproducibility of the experiments.

In this chapter, we compared existing predictive process monitoring tech-

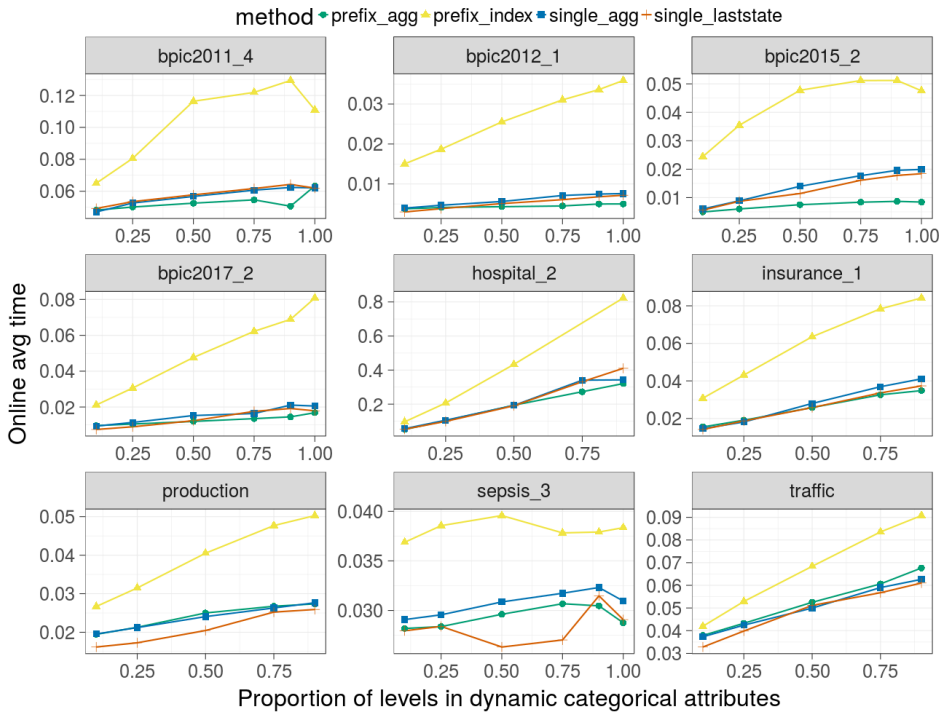


Figure 20: Online times across different filtering proportions of **dynamic** categorical attribute levels (XGBoost).

niques, all of which are limited to using structured data when training the predictive models. In the next chapter, we aim at enchancing the predictive accuracy of these techniques by additionally exploiting unstructured data.

5. PREDICTIVE BUSINESS PROCESS MONITORING WITH STRUCTURED AND UNSTRUCTURED DATA

As observed in Chapter 3, all of the existing outcome-oriented predictive monitoring techniques focus only on structured (i.e. numeric and categorical) data, neglecting the presence of unstructured (textual) data in real-life event logs. Neglecting textual data in predictive models can result in reduced prediction accuracy, since the classifiers are not given all the information that is available about a given case. Therefore, in this chapter we aim to fill this gap by proposing a framework that makes use of text mining techniques in order to combine both structured and unstructured data payload for predictive process monitoring. In this chapter we assume that each trace consists of a sequence of events carrying both structured and unstructured data payload. For example, consider the event log in Table 14 originating from a hypothetical debt recovery process, where each event is associated with structured attributes (Activity, Revenue, Debt sum) and an unstructured one (Comment).

In the following sections, we first explain the relevant concepts and principles from the text mining field (Section 5.1), then introduce the proposed framework (Section 5.2), and, lastly, evaluate different text modeling techniques in combinations with existing predictive process monitoring approaches using the proposed framework (Section 5.3).

5.1. Text mining

While structured data objects (consisting of attributes) can be readily encoded as a feature vector by mapping each attribute to one or more features depending on the type, this approach cannot be applied to textual data. One of the main affordances that text mining techniques provide is the ability to map a document into a feature vector in such a way that the resulting features capture the semantics of the document to an extent that is suitable for training a machine learning model. Here, the term *document* refers to a unit of textual data such as a comment or the text of an email.

Prior to applying a text mining technique, we need to preprocess the input

Table 14: Example event log with structured and unstructured data payload

Case ID	Event ID	Timestamp	Activity	Revenue	Debt sum	Comment
C1	E1	T1	Call	34555	500	Gave extension of 5 days and issued a warning about sending the debt to encashment. 1234567: "Will pay the debt in full tomorrow."
C1	E2	T2	Send letter	34555	500	A warning letter sent on 06/10, 11:10 deadline 13/10.

document. Firstly, the text needs to be *tokenized* — segmented into meaningful pieces. In the simplest approach, text is split into tokens on the white space character. More sophisticated tokenization techniques can be used to obtain multi-word tokens (e.g. "New York") or to separate words such as "it's" into two tokens "it" and "is".

Tokens can also be *normalized* so that tokens with small differences (e.g. "e-mail" and "email") are equated. In addition, inflected forms of words can be grouped together using *stemming* or *lemmatization*. While stemming is usually done by cutting off the endings of the words, lemmatization uses more sophisticated techniques, such as morphological analysis [61]. For instance, lemmatization can group words "good", "better", and "best" under a single *lemma*.

A straightforward way of representing a preprocessed document as a feature vector is by using the words themselves as features and assigning to each word its frequency in the document. For example, the document "The fox jumps over the dog" is represented as {"the":2, "fox":1, "jumps":1, "over":1, "dog":1}. This representation ignores the order of words – a limitation that can be overcome by using sequences of two (*bigrams*), three (*trigrams*), or *n* (*n-grams*) contiguous words instead of or in addition to single words (*unigrams*). The bigrams in the above document are: {"the fox":1, "fox jumps":1, "jumps over":1, "over the":1, "the dog":1}. Features that are constructed based on words that occur in the document are called *terms*, while the corresponding representation is called *bag-of-n-grams*.

Terms that occur frequently in a document collection are not representative of a particular document, yet they receive misleadingly high values in the basic BoNG model. This problem can be addressed by normalizing the term frequencies (tf) with the *inverse document frequencies* (idf) — the number of all documents divided by the number of documents that contain the term, scaled logarithmically. Thus, rare terms receive higher weights, while frequent words (like "with" or "the") receive lower weights. In text classification scenarios, weighing the term frequencies with Naive Bayes log count ratios may improve the accuracy of the predictions [108]. The BoNG model also suffers from high dimensionality, as each document is represented by as many features as the number of terms in the *vocabulary* (the set of all terms in the document collection). A common practice is to apply *feature selection* techniques, such as mutual information or Chi-square test, and retain only the most relevant terms.

Alternative approaches to the BoNG model are *continuous representations* of documents. These techniques represent text with real-valued low-dimensional feature vectors, where each feature is typically a *latent* variable — inferred from the observed variables. One such approach is *topic modeling*, which extracts abstract *topics* from a collection of documents. The most widely used topic modeling technique, Latent Dirichlet Allocation [4], is a generative statistical model, which assumes that each document entails a mixture of topics and each word in the document is drawn from one of the underlying topics.

Continuous representations of words (i.e. *word embeddings*) using neural

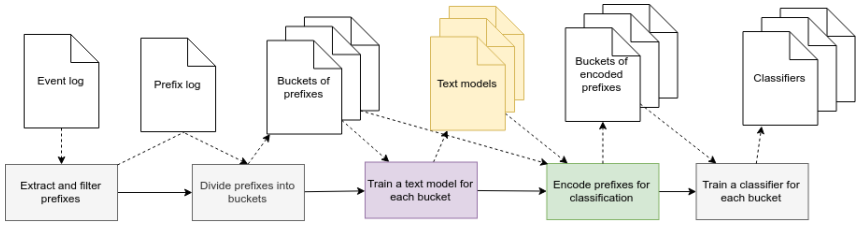


Figure 21: The offline component of the proposed framework.

network-based language models have also shown high performance in natural language processing tasks. These language models are trained to predict a missing word, given its *context* — words in the proximity of the word to be predicted. Techniques have been proposed that extend these approaches from word-level to sentence-, or document-level. For instance, *Paragraph Vector* [53] generates fixed-length feature representations for documents of variable length.

5.2. Predictive process monitoring framework with structured and unstructured data

In the following, we introduce a framework for predictive process monitoring with both structured and unstructured data. We start with an overview of the proposed framework and proceed with describing the techniques from the text mining field that our framework draws upon.

5.2.1. Overview of the framework

The proposed framework embodies two components: the *offline component* takes as input labeled historical traces and trains classifiers which are then used to make predictions about running cases through the *online component*. The offline component (depicted in Figure 21) mainly follows the same worklow as the general predictive process monitoring framework (see Figure 6). The core difference is that additionally to a classifier, a *text model* is trained for each bucket. The purpose of a text model is to transform a variable length chunk of textual data associated to an event into a fixed length numerical feature vector. Differently from the general framework, encoding a prefix now requires dividing the data payload related to each event into a structured and an unstructured part and transforming the latter into a numeric feature vector using the text model (Figure 22). Consequently, a sequence encoding technique (e.g. aggregation, last state, or index-based encoding) can be applied to both types of data payload related to a prefix: the sequence of initially structured payloads and the sequence of transformed unstructured payloads. The online component of the framework remains the same as in the general one (Fig. 7) with the exception of using the updated prefix encoding step.

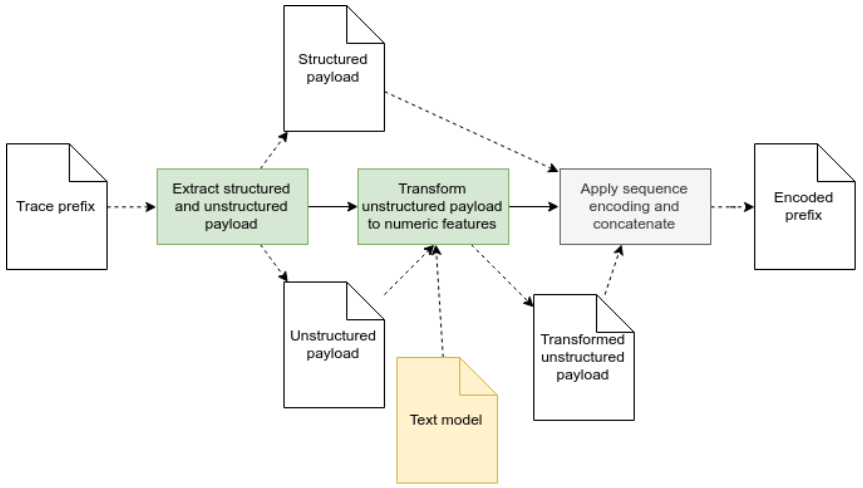


Figure 22: Encoding a prefix carrying both structured and unstructured payload.

5.2.2. Text models

The goal of a text model in the proposed framework is to transform a document (i.e. the textual data associated to a single event) of arbitrary length to a fixed length numeric feature vector. In essence, the text models in the framework can be instantiated with any technique that is able to perform such transformations. In the following, we introduce 4 specific techniques for extracting feature vectors from text and explain how they can be used as instantiations of the text models in our framework. In particular, we employ the bag-of-n-grams (BoNG), bag-of-n-grams weighted with Naive Bayes log count ratios (NB), Latent Dirichlet Allocation topic modeling (LDA), and Paragraph Vector (PV).

BoNG (n, tfidf*):.* In this model, the feature vector is constructed from the frequencies of different n-grams in a given document. The method takes as input two parameters: n, which is the maximum size of the considered n-grams; and tfidf, that is a boolean variable specifying whether the vector of frequencies should be refined with tf-idf weighting. First, the documents from historical prefixes are used to build a vocabulary of n-grams, $V(n)$. Given a vocabulary $V(n)$ of size $|V(n)| = v$, a document j is represented as a vector $\mathbf{d}^{(j)} = (g_{t_1}^{(j)}, g_{t_2}^{(j)}, ..., g_{t_v}^{(j)})$, where:

$$g_{t_i}^{(j)} = \begin{cases} \mathsf{tfidf}(f_{t_i}^{(j)}) & \text{if tfidf} \\ f_{t_i}^{(j)} & \text{otherwise} \end{cases}$$

and $f_{t_i}^{(j)}$ represents the frequency of n-gram t_i in document $\mathbf{d}^{(j)}$.

For instance, consider our running example (Table 14), if $n = 1$, tfidf = false and the vocabulary (after cleaning the text and removing stopwords) $V(1) = \{\mathsf{about}, \mathsf{day}, \mathsf{deadline}, \mathsf{debt}, \mathsf{encashment}, \mathsf{extension}, \mathsf{full}, \mathsf{give}, \mathsf{issue}, \mathsf{letter}, \mathsf{pay}, \mathsf{send}, \mathsf{tomorrow}, \mathsf{warning}\}$, the vector encoding the textual data in the first event would be:

$$\mathbf{d}^{(j)} = (1,1,0,2,1,1,1,1,1,0,1,1,1,1) \tag{5.1}$$

where the word "debt" occurs two times in the document, the word "about" occurs once and the word "deadline" does not occur.

NB (n, α):. In this model, the features are also based on the BoNG vector, but are additionally weighted with Naive Bayes log count ratios [108], $\mathbf{d}^{(j)} = (f_{t_1}^{(j)} \cdot r_1, f_{t_2}^{(j)} \cdot r_2, ..., f_{t_v}^{(j)} \cdot r_v)$. A ratio $r_i, 1 \leq i \leq v$ reflects the polarity of the corresponding term t_i, i.e. how frequently the term occurs in positive/negative cases. The parameter α is a *smoothing parameter* for the weights, while n is, as in BoNG, the maximum size of the n-grams. For instance, if the vocabulary (and hence the term frequency) is the same as the one in (5.1) and the NB vector of polarity ratios is $\mathbf{r} = (0.85, 1.02, 0.76, 0.76, 1.52, 2.03, 1.19, 1.02, 0.45, 0.89, 1.02, 1.4, 1.39, 0.41)$, $\mathbf{d}^{(j)}$ would be:

$$\mathbf{d}^{(j)} = (0.85, 1.02, 0, 1.52, 1.52, 2.03, 1.19, 1.02, 0.45, 0, 1.02, 1.4, 1.39, 0.41) \tag{5.2}$$

LDA (c, tfidf*):.* In this model, a feature vector represents a mixture of *topics* covered by a document. A topic is a latent feature, expressed as a probability distribution over words, where words that are characteristic to a particular topic possess higher values. In turn, each document is represented as a probability vector over topics, $\mathbf{d}^{(j)} = (p_1^{(j)}, p_2^{(j)}, ..., p_c^{(j)})$, where $p_i^{(j)}$ is the probability that document j concerns topic i. The underlying topic model is derived using an inference procedure that takes as input the number of topics c. Also, similarly to BoNG, a boolean parameter tfidf determines whether the tfidf weighting should be performed before applying topic modeling.

For instance, let us imagine that by applying topic modeling on the unstructured data payload in the historical traces, the following three topics have been identified (note that the descriptions of the topics on the right are assigned manually, not as part of the topic modeling procedure):

$$topic1 : (mobile : 0.3, answer : 0.2, switched : 0.15, off : 0.15, ...) \quad (\textit{not accessible by phone})$$
$$topic2 : (send : 0.35, letter : 0.2, warning : 0.1, ...) \quad (\textit{warning letter sent})$$
$$topic3 : (full : 0.4, debt : 0.2, pay : 0.1, ...) \quad (\textit{immediate payment})$$

In this case, the first event in our running example (Table 14) will be represented as a vector of three items, each item corresponding to the probability that the document concerns $topic1$, $topic2$, and $topic3$, respectively. Given that the example document is not very related to $topic1$, is a bit more related to $topic2$, and is closest to the warning letter scenario, the resulting feature vector could be:

$$\mathbf{d}^{(j)} = (0.1, 0.2, 0.7) \tag{5.3}$$

PV (vector_size, window_size):. Using the Paragraph Vector model, not only the terms but also the the context of the word is exploited for the construction of the feature vector. Namely, the method slides a window of size *window_size* over the document, using each of such windows of words as the context. Then, a neural network is trained to predict the context of a word, based on the word itself (or vice versa, depending on the type of the model). The trained word vectors are lifted to the level of documents. As a result, the model is able to provide for each document a vector of features of a fixed length (specified by *vector_size*).

After obtaining the numeric feature vectors from unstructured data, any sequence encoding technique can be applied to these features and the encoded vectors for structured and unstructured data can be concatenated before given as input to the classifier. For instance, given the prefix corresponding to the first event in the example (Table 14), let the one-hot vector of the *Call* activity to be $(1,0)$, then the concatenated feature vector $\mathbf{x}$ using last state encoding and the LDA text model (see 5.3) would be:

$$\mathbf{x} = (1, 0, 34555, 500, 0.1, 0.2, 0.7) \tag{5.4}$$

5.3. Evaluation

In this section we perform an experimental evaluation in order to answer the following research questions:

RQ1 (Accuracy and earliness) Do the features derived from textual data (using different text modeling methods) increase the prediction accuracy and earliness of existing predictive process monitoring techniques?

RQ2 (Efficiency) Is the proposed predictive process monitoring framework computationally efficient?

In the following subsections we describe the approaches, the evaluation datasets, the experimental setup, and the findings.

5.3.1. Approaches

An overview of the approaches can be seen in Table 15. For encoding the structured data payload, we experiment with the last state, the aggregation, and the index-based sequence encodings. The first two are combined with a single bucket approach, while the latter with prefix length based bucketing. For the unstructured data payload, we test all the text modeling methods described in Section 5.2.2, namely, BoNG, NB, LDA, and PV. After obtaining numeric features using these techniques, we apply either the last state or the aggregation encoding. We do not

Table 15: Approaches.

Method name	Bucketing	Enc. (str.)	Enc. (unstr.)	Text model	Classifier
single_agg	single	last	last	BoNG, NB, LDA, PV, none	RF, XGBoost or logit
single_laststate	single	agg	agg	BoNG, NB, LDA, PV, none	RF, XGBoost, logit
prefix_index_laststate	prefix	index	last	BoNG, NB, LDA, PV, none	RF, XGBoost, logit
prefix_index_agg	prefix	index	agg	BoNG, NB, LDA, PV, none	RF, XGBoost, logit

use index-based encoding for textual features, since the text models yield feature vectors that are already high-dimensional and, therefore, growing the size of the vector with each event would be infeasible. Instead, we combine the index-based encoding for structured payload with either the last state or the aggregation encoding for unstructured payload. As a baseline, we implement the approach of using only structured data (neglecting the unstructured data completely). In terms of classifiers, we apply RF and XGBoost, which showed to outperform the other classifiers in the benchmark (see Chapter 4), and logit, as a representative of linear classifiers that are often used in text classification [108].

5.3.2. Datasets

We evaluate the framework on three real-life datasets pertaining to: (i) the debt recovery (DR) process of an Estonian company that provides credit management service for its customers (creditors), (ii) the lead-to-contract (LtC) process of a due diligence service provider in Estonia, and (iii) the issue tracking process in Github projects. Due to privacy constraints, the DR and the LtC event logs are not publicly available. The Github dataset contains data about public repositories, which are made publicly available through APIs. We use a precollected and preprocessed version of the data from [45], which originates from GHTorrent [35].

Event log DR. The DR process starts when the creditor transfers a delinquent debt to the company. This means that the debtor has already *defaulted* — failed to repay the debt to the creditor in due time. Usually, the collection specialist makes a phone call to the debtor. If the phone is not answered, an inquiry/reminder letter is sent. If the phone is answered, the debtor may provide an expected payment date, in which case no additional action is taken during the present week. Alternatively, the specialist and the debtor can agree on a payment schedule that outlines the repayments over a longer time period. If the collection specialist considers the case to be irreparable, she makes a suggestion to the creditor about forwarding the debt to an outside debt collection agency (*send to encashment*) or about sending a warning letter to the debtor on the same matter. The final decision about issuing an encashment warning to the debtor and/or sending the debt to encashment is made by the creditor. If there is no advancement in collecting the debt after 7 days (e.g. the payment was not received on the provided date or the debtor has neither answered the phone nor the reminder letter), the procedure is repeated. It is in the interest of the creditor to discover, as early as possible, cases that will not lead to any payment in a reasonable timeframe. The earlier the debt is recovered, the

more value it entails for the creditor. Moreover, such cases are likely irreparable and could be sent to encashment without further delay. Therefore, our prediction goal is to determine cases where no payment is received within 8 weeks after the beginning of the debt recovery process.

Event log LtC. The lead-to-contract process is logged through a CRM system. The process begins when the sales manager selects companies as "cold leads" and loads them into the CRM system. Based on personal experience, the sales manager selects leads that qualify for an opportunity, or alternatively, makes a phonecall to the company in order to determine qualification. Then, when a case is in the *qualification stage*, a phonecall is initiated with the purpose of scheduling a meeting with the potential customer's representatives. If a meeting is scheduled, the opportunity enters the *presentation stage*. The goal of a sales person is to get the contract signed during the presentation. If she succeeds, the opportunity is marked as *won* and the case terminates. If the offer made during the meeting was acceptable, but the signing of the contract is postponed, the opportunity enters the *contract stage*. If the offer was not accepted during the meeting, an offer is sent via email, and the opportunity moves to the *offer stage*. Any time during the process additional phonecalls can be made, emails exchanged, and follow-up meetings scheduled. When it becomes clear that the company is not interested in collaboration, the opportunity is marked as *lost*. The number of potential customers is very high and it is not feasible for the sales people to deeply explore all of the possible leads. Thus, the process would benefit from a support system that estimates if an opportunity will likely end with a signed contract (opportunity won) or not (opportunity lost). If an opportunity is likely to be lost, the sales person can close it at an early stage (or assign it a lower priority), becoming able to focus on other leads with higher potential. Given this motivation, in the following experiments we aim at predicting, as early as possible, if an opportunity will be lost.

Event log Github. This log, originally constructed by Kikas et al. [45], contains data from the issue tracking system in more than more than 4000 Github projects. This set of projects was obtained from the total pool of over 7 million Github repositories by filtering projects that (i) were created between January 1, 2012 and December 31, 2014, (ii) were not forks of other projects, (iii) had at least 100 opened issues and one closed issue, (iv) had at least five commits to the main repository, and (v) had not shown any activity before the repository creation date. Additionally, projects with very high issue creation acitivity during short periods (2000 issues in a single month or 500 issues in a day), projects shorter than 8 months, and reopened projects were filtered out. The labeling for this dataset is based on the motivation that various stakeholders are interested in estimating the closing time of an issue. For instance, the submitter of the issue is interested in knowing whether it is reasonable to wait for the issue to be fixed or should she switch to using another software. The development team desires to have an estimate of the issue closing time, so that they could better prioritize and plan

Table 16: Data statistics.

dataset	# traces	med length	max (trunc) length	# variants (after trunc)	pos class ratio	# event classes	# dynamic attr-s	# static attr-s	# dynamic cat levels	# lemmas	avg text length	% events with text
DR	14025	1.0	8 (8)	8	0.03	1	73	9	3	10408	11	100
LtC	74739	12.0	431 (33)	21090	0.1	13	0	19	268	67734	251	18
Github	920148	4.0	7 (5)	5	0.69	1	18	14	0	465754	70	55

their tasks. Guided by these motivations, we formulate a classification task on this dataset with the aim to predict at different points in an issue's lifetime, whether or not the issue will close within a year since it was created.

General statistics about the datasets are given in Table 16. We apply the same preprocessing to the structured part of the event logs as described in Section 4.1. We identify 8 static and 69 dynamic features in the debt recovery dataset, and 3 static and 65 dynamic features in the lead-to-contract dataset. The static features are general statistics about the company, for instance, *the size of equity*, *the net profit*, and *field of activity*. The dynamic features in the first dataset are mostly related to the debt, e.g. *the number of days past due*, *the expected repayment amount until the next 7 days*, and *the sum of other debts of the debtor*. In the second dataset, the dynamic features include *activity name*, *resource*, and *expected revenue*. For both datasets, we use dynamic features that reflect the company's (either the debtor's or the potential customer's) risk score, calculated at 6 different months prior to the given event. Moreover, as the first dataset contains a considerable amount of missing values, additional 16 (static) features are added that express whether the value of a particular feature is present or missing. In the Github dataset, we consider each issue as a separate case. This dataset contains unstructured data in both static (the title and the description of the issue) and dynamic (comments about the issue) format. Structured issue payload includes, for instance, the number of people interacting with the issue and the number of times the issue has been mentioned from other issues. We use a preprocessed format of the dataset, where additional features have been extracted, such as the number of comments, the text length of the issue content, and the number of commits made by the people participating in the issue [45].

In the LtC dataset, unstructured data occurs in emails (the subject and the content), as well as in the summaries of phonecalls, which are written in textual format. In the DR dataset, the activity names are not explicitly logged; instead, this information is captured in the debt collector's notes, which are written down in unstructured textual format. These notes constitute a dynamic feature, which may describe the activity taken by the collection specialist, as well as the answer of the debtor and the assessment of the specialist. Similarly, in the Github data the activity names are not present, so that the comment field of an issue becomes essential in carrying the information about the status of an issue. Conversely from the other two logs, the Github dataset additionally entails static textual data, namely, the title and the description (content) of the issue.

Using a richer collection of textual data helps to build potentially more pow-

erful text models. Also, the text models transform each textual field to potentially hundreds or thousands of features, which makes it infeasible to create a separate representation for each of the fields. For these reasons, we have decided to concatenate the textual data originating from different fields within the same event into a single textual attribute.

Prior to building the text models, we performed some preprocessing steps to take into account the specificities of each dataset. In the Github dataset, all source code blocks, tables, links, and other markdown markup has been removed, keeping only the textual content. In the DR and LtC datasets, we generate equivalence classes for different types of numerals by replacing them with a corresponding tag (phone number, date, or other). For instance, in the running example (Table 14), token "1234567" would be replaced by token "phone number", token "06/10" by "date" and token "11:10" by "time". Then, we convert all text into lowercase, remove punctuation, and tokenize the documents using simple white space tokenization. The texts in the DC and the LtC datasets are written in Estonian language, where words can have a lot of possible inflections. Therefore, we lemmatize the texts using the `estnltk` [71] library in Python. The Github dataset is based mainly on English, where the number of possible inflections is smaller, so we apply the Porter stemming algorithm to extract the stem for each word. Table 16 present the statistics of the datasets related to the preprocessed unstructured data.

5.3.3. Experimental setup

In most aspects we follow the same experimental setup as in Chapter 4. Namely, we split the datasets into 80% training (precisely, $L_{train} \cup L_{val}$) and 20% testing traces (L_{test}) using a temporal holdout split; optimize the hyperparameters for each dataset and technique using TPE on the training set; and evaluate the accuracy and earliness by monitoring the AUC across different prefix lengths. In order to measure the efficiency of the framework, we report the offline and online execution times of the final models.

However, contrarily to the benchmark, we omit the 3-fold cross-validation during hyperparameter optimization, since the size of the datasets in combination with the unstructured data makes it infeasible. Instead, in case of DR and LtC we split the training traces randomly into a 80% L_{train} set (64% of original traces) and a 20% L_{val} set (16% of original traces) and select the best model based on AUC on L_{val}. In case of the Github dataset we follow the same procedure, but due to the even larger size of the data we use only 10% of the original traces as the L_{train} set and 5% as the L_{val} set. Note that even such reduction leaves us with about 90k traces for L_{train} and 45k traces for L_{val} sets.

While in the LDA and the PV models, the size of the feature vector is limited by the input arguments (the number of topics and the size of the feature vector, respectively), in case of BoNG and NB a *feature selection* step is required in

Table 17: Hyperparameters of the text models and their sampling distributions used in optimization via TPE.

Method	Parameter	Distribution	Values
BoNG	max n-gram size	Uniform	$x \in \{1,2,3\}$
	tf-idf	Categorical	$x \in [True, False]$
NB	max n-gram size	Uniform	$x \in \{1,2,3\}$
	alpha	Log-uniform	$x \in [0.01, 1]$
LDA	# topics	Log-uniform integer	$x \in [10, 200]$
	tf-idf	Categorical	$x \in [True, False]$
PV	vector size	Log-uniform integer	$x \in [10, 400]$
	window size	Uniform integer	$x \in [1, 13]$

order to keep the number of extracted features in reasonable bounds. In particular, for the basic BoNG model we apply the Chi-square test, while for NB the most discriminative features (i.e. the terms that achieve the top lowest and top highest NB log ratio scores) are selected. In both cases, we fix the number of selected features to 500.

Table 17 reports the hyperparameter values that we consider for optimizing the text models. The hyperparameter spaces for the classifiers (optimized jointly with the text models) remain the same as in Table 10.

Experiments were run using Python 3.5 and the scikit-learn (BoNG and classifiers), gensim (LDA and PV) and estnltk (lemmatization) libraries on a single core of a Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz with 64GB of RAM.

5.3.4. Results

Table 18 shows the overall AUC values (i.e. the weighted sum of AUC over different prefix lengths, where the weights are assigned according to the number of prefixes in that evaluation point) for all datasets, classifiers, bucketing/encoding combinations, and text modeling methods. The best overall AUC for LtC and Github datasets are achieved by XGBoost classifier. In case of Github, the approaches using textual data outperform the baseline (only structured data), while there is little difference between the different bucketing, sequence encoding and text modeling methods. In case of LtC, the best overall AUC is achieved by a single classifier with aggregation encoding and the NB text modeling method, yielding a 4 percentage points increase compared to the baseline. This technique is closely followed by the BoNG method, which yields a 3 percentage points increase. In the DR dataset, RF outperforms XGBoost. The best results in this dataset are achieved using the BoNG text modeling method, while the bucket-

Table 18: Overall AUC.

	RF			XGBoost			logit		
	DR	LtC	Github	DR	LtC	Github	DR	LtC	Github
single_agg no text	0.96	0.56	0.72	0.93	0.6	0.73	0.94	0.6	0.71
single_agg BoNG	**0.98**	0.57	0.73	0.94	0.63	**0.74**	0.93	0.54	0.71
single_agg NB	0.97	0.54	0.72	0.97	**0.64**	**0.74**	0.96	0.59	0.7
single_agg LDA	0.96	0.57	0.72	0.95	0.6	**0.74**	0.96	0.62	0.72
single_agg PV	0.96	0.55	0.73	0.95	0.61	**0.74**	0.94	0.62	0.72
single_laststate no text	0.96	0.56	0.72	0.96	0.61	0.73	0.93	0.48	0.65
single_laststate BoNG	**0.98**	0.57	0.73	0.93	0.6	**0.74**	0.96	0.49	0.65
single_laststate NB	0.97	0.56	0.72	0.97	0.6	0.73	0.97	0.48	0.65
single_laststate LDA	0.97	0.56	0.73	0.96	0.61	**0.74**	0.97	0.5	0.65
single_laststate PV	0.96	0.56	0.73	0.96	0.59	**0.74**	0.94	0.49	0.65
prefix_index_agg BoNG	**0.98**	0.59	0.73	0.96	0.61	**0.74**	0.94	0.54	0.71
prefix_index_agg NB	0.97	0.57	0.72	0.96	0.6	0.73	0.96	0.53	0.71
prefix_index_agg LDA	0.96	0.58	0.73	0.95	0.6	**0.74**	0.96	0.53	0.72
prefix_index_agg PV	0.96	0.58	0.73	0.94	0.6	**0.74**	0.92	0.53	0.72
prefix_index_last BoNG	**0.98**	0.58	0.73	0.97	0.6	**0.74**	0.96	0.54	0.71
prefix_index_last NB	0.97	0.57	0.72	0.96	0.59	0.73	0.96	0.53	0.71
prefix_index_last LDA	0.96	0.57	0.73	0.97	0.6	**0.74**	0.96	0.52	0.72
prefix_index_last PV	0.95	0.58	0.73	0.96	0.6	**0.74**	0.94	0.52	0.71
prefix_index no text	0.97	0.57	0.72	0.95	0.59	0.73	0.94	0.52	0.71

ing/encoding combination does not affect the results much.

Since XGBoost achieved the top performance in two out of three datasets, we will have a closer look at the predictions obtained with this classifier. In Figure 23 the AUC over different prefix lengths are plotted for all datasets and approaches. We can observe that the most consistent improvement for all text modeling methods over the baseline (exploiting structured data only) is achieved with a single classifier with aggregation encoding. In other bucketing/encoding combinations there exist at least a few prefixes where the performance of the methods exploiting textual data drops below that of the baseline. The reason for this lower stability in performance might be that introducing a high number of additional features explodes the dimensionality of the feature vector, so that it becomes more difficult for a classifier to find the relevant patterns. Focusing on the single_agg approach, we can see that NB tends to perform very well on smaller prefix lengths, but is outperformed by others in longer prefixes. Conversely LDA performs better on longer prefixes than on shorter ones. BoNG retains high performance on both short and long prefixes, making it an overall good choice for text modeling. This discussion concludes the answer to RQ1 (Do the features derived from textual data (using different text modeling methods) increase the prediction accuracy and earliness of existing predictive process monitoring techniques?).

Table 19 reports the execution times using the XGBoost classifier, calculated as averages over 5 identical runs using the final (optimal) parameters. The execution times for RF and logit can be found in the Appendix (Tables 37–38). We

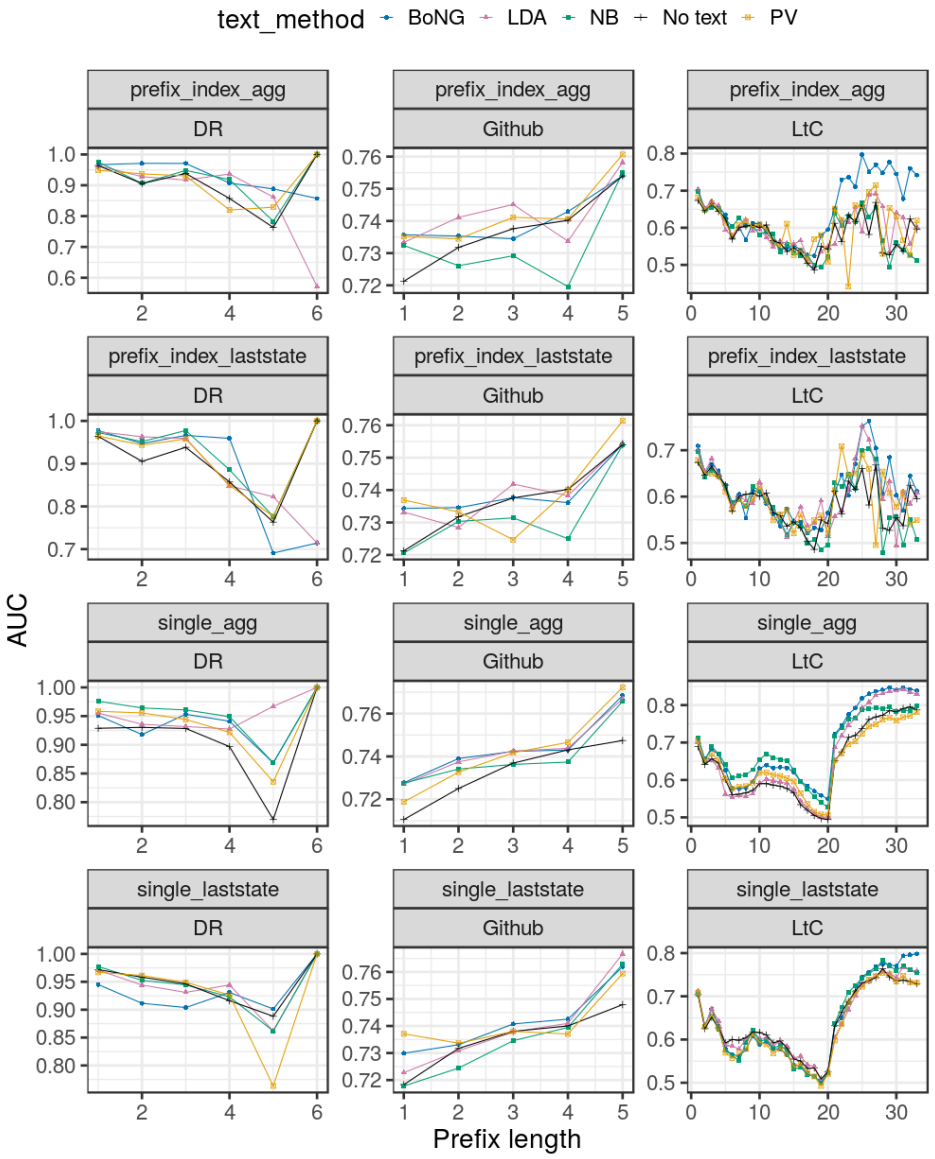


Figure 23: AUC across different prefix lengths using **XGBoost**.

Table 19: Execution times for **XGBoost** with unstructured data.

	DR		LtC		Github	
single_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	**46.15 ± 0.22**	0.04 ± 0.01	8434.35 ± 41.28	**0.02 ± 0.02**	416.56 ± 5.06	0.01 ± 0.0
BoNG	744.57 ± 1.76	0.06 ± 0.02	38362.78 ± 303.36	0.05 ± 0.07	12172.71 ± 1730.25	0.4 ± 0.24
NB	831.38 ± 0.95	0.05 ± 0.02	31420.83 ± 2899.93	0.05 ± 0.06	27300.41 ± 406.14	0.02 ± 0.03
LDA	119.4 ± 0.69	0.04 ± 0.01	43900.04 ± 140.91	0.04 ± 0.05	48613.82 ± 604.37	0.01 ± 0.01
PV	1762.49 ± 15.71	0.06 ± 0.03	17269.98 ± 26.62	0.06 ± 0.5	8980.35 ± 105.81	0.08 ± 0.67
	DR		**LtC**		**Github**	
single_laststate	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	64.3 ± 0.27	**0.02 ± 0.01**	**5144.46 ± 162.66**	**0.02 ± 0.02**	**327.76 ± 10.06**	**0.0 ± 0.0**
BoNG	161.84 ± 0.59	0.09 ± 0.03	16037.87 ± 351.32	0.04 ± 0.05	4144.12 ± 12.22	0.41 ± 0.24
NB	281.16 ± 0.63	0.03 ± 0.01	18881.63 ± 803.84	0.03 ± 0.03	2752.98 ± 20.7	0.01 ± 0.01
LDA	79.02 ± 0.12	**0.02 ± 0.01**	9062.85 ± 337.98	**0.02 ± 0.03**	12466.74 ± 9.19	0.01 ± 0.01
PV	217.19 ± 0.91	0.14 ± 0.45	8297.59 ± 25.33	0.05 ± 0.44	1444.44 ± 18.98	0.03 ± 0.1
	DR		**LtC**		**Github**	
prefix_index_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	48.41 ± 0.1	0.04 ± 0.01	18859.61 ± 1112.83	0.14 ± 0.01	771.33 ± 2.12	0.01 ± 0.0
BoNG	715.8 ± 1.47	0.11 ± 0.02	64155.49 ± 1539.62	0.18 ± 0.03	39749.18 ± 22.28	0.37 ± 0.29
NB	622.41 ± 4.29	0.04 ± 0.01	54377.93 ± 1939.57	0.16 ± 0.01	45020.54 ± 11.73	0.03 ± 0.02
LDA	450.67 ± 1.69	0.03 ± 0.01	158752.32 ± 9170.91	0.15 ± 0.01	110522.33 ± 232.32	0.03 ± 0.01
PV	185.01 ± 10.5	0.27 ± 3.1	51910.47 ± 5968.19	0.21 ± 1.1	14907.05 ± 156.84	0.08 ± 0.71
	DR		**LtC**		**Github**	
prefix_index_last	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	48.41 ± 0.1	0.04 ± 0.01	18859.61 ± 1112.83	0.14 ± 0.01	771.33 ± 2.12	0.01 ± 0.0
BoNG	250.1 ± 1.8	0.05 ± 0.01	26876.55 ± 419.47	0.16 ± 0.01	7493.64 ± 9.59	0.11 ± 0.08
NB	267.65 ± 8.29	0.04 ± 0.02	30886.92 ± 761.83	0.15 ± 0.01	9472.26 ± 6.57	0.02 ± 0.01
LDA	514.37 ± 5.46	0.03 ± 0.03	128553.23 ± 8156.23	0.15 ± 0.01	52794.38 ± 45.14	0.02 ± 0.01
PV	248.66 ± 30.97	0.3 ± 3.51	40380.02 ± 4302.25	0.22 ± 1.33	6340.22 ± 53.46	0.09 ± 0.99

can observe that incorporating textual data increases the offline execution times considerably. This is expected, since additional time is spent on training the text models and the increased size of the feature vector slows down the training of classifiers. Also, it is likely that the larger feature space leads to a configuration of hyperparameters that yields a more complex model, thus, taking more time for training. There is no clear pattern in the offline time measurements of different text modeling techniques, which indicates that the execution times depend heavily on the selected hyperparameter setting. In the online phase, all methods take less than half a millisecond on average to process an event. In general, the online execution times of NB and LDA remain close to the ones where only structured data is used, while BoNG is slightly slower. PV takes, in general, the longest time to process an event, with the standard deviation across different events being even more than a millisecond in some cases (see PV with index-based encoding). This concludes the answer to RQ2 (Is the proposed predictive process monitoring framework computationally efficient?).

5.4. Summary

In this chapter we outlined a framework for predictive process monitoring that combines text mining methods to extract features from textual documents, with existing predictive process monitoring techniques designed for structured data. We studied different combinations of text mining and classification techniques and

evaluated them on three real-life datasets pertaining to a debt recovery process, a lead-to-contract process, and the Github issue resolution process. The evaluation datasets exhibited different characteristics in terms of textual data, from short and homogenous notes to long heterogenous emails and project descriptions.

In the reported evaluation, the proposed approach of including textual data in most cases (though not always) outperforms the baseline of using only structured data. A likely reason for the instability of the results is the increased size of the feature vectors and the complexity of the preditive models. Still, the results show that single classifiers with aggregation encoding consistently outperform the baseline. Also, although logistic regression is a common choice for text classification, RF and XGBoost outperform the former in our techniques. A possible reason for this is that in our case the (sparse) textual features are combined with (denser) structured data. In terms of text modeling techniques, NB often performs best on shorter prefixes, while LDA achieves better performance on longer ones. BoNG is a reliable option that performs well over all prefix lengths.

This chapter (as well as the previous chapter) addressed the task of training more accurate predictive models for predictive business process monitoring. Specifically, we used traditional quality dimensions to evaluate the obtained models, namely, the accuracy, the earliness, and the efficiency of the predictions. In the next chapter we focus more deeply on the phase of evaluating predictive models in the context of predictive process monitoring.

6. TEMPORAL STABILITY IN PREDICTIVE PROCESS MONITORING

As identified in Chapter 3, existing works on outcome-oriented predictive process monitoring measure the goodness of the predictions in terms of accuracy, earliness, and efficiency. Only one of the studies considers the stability of prediction accuracy over different evaluation points, while none of the works have investigated the stability of predictions made for the same case over time. The latter is, however, important when predictive monitoring is applied in practice, i.e. when predictions are provided sequentially for the same case. In those scenarios, predictions that are highly volatile (changing drastically throughout the same case) are undesirable, given that process workers and their supervisors need to make decisions at runtime based on these predictions.

Consider, for instance, a healthcare process where the target is to estimate whether a patient will need intensive or standard care. An accurate prediction could help the patient to receive the suitable treatment in a timely manner, as well as help the hospital to better allocate resources to patients. Suppose that when the patient first arrives at the hospital, the predictive model estimates that she will need intensive care, so she is admitted to the intensive care program. After executing a procedure, the predictor changes the prediction and estimates that standard care is sufficient for the patient, so the patient is brought to standard care. Then, after performing another procedure, the classifier changes the prediction again and recommends transferring the patient back to intensive care. Since the classifier has made two different predictions within the first three events, at least one of them has been accurate and made early in the process. However, switching between standard and intensive care multiple times has caused the hospital to waste resources and brought inconveniences to the patient. This example shows how the practical usefulness of a predictive model is limited if it outputs *unstable* predictions, i.e. if it tends to often change the value of the predictions after seeing new data about the same case. In this example, the treatment of the patient could have been more efficient had the personnel not trusted the intermediate prediction of the predictor and not brought the patient to the standard care. Another example concerns a debt encashment process, where a prediction engine can be used to decide whether the debt should be sent to a credit collection agency or not. In this case, volatile predictions can mislead users of the system to prematurely send the debt to the collection agency, resulting in smaller revenue as compared to waiting some more time for the debt to be repaid. Similarly, in case of fraud detection in a financial institution, unstable predictions may cause the institution to frequently block and unblock the credit of a user, resulting in inconveniences and loss of revenue related to potential transactions that the user was not able to complete.

Note that measuring prediction accuracy and earliness over different prefix lengths would not capture the stability of the predictions over prefixes. For in-

stance, consider two running traces, σ_a and σ_b, one with a desired and the other with an undesired outcome. It is possible that one classifier C_A outputs a constant sequence of prediction scores $(0.6, 0.6, 0.6, 0.6)$ for both σ_a and σ_b over the first 4 prefixes, whereas C_B outputs $(0.4, 0.8, 0.4, 0.8)$ for σ_a and $(0.8, 0.4, 0.8, 0.4)$ for σ_b. In terms of prediction accuracy (e.g. AUC) these classifiers would be equivalent. Also, since the accuracy is the same over all the four prefixes, there is no difference in prediction earliness between the classifiers. However, in terms of prediction stability, the first classifier would be preferred since it would cause the process workers to make fewer switches between the decisions. This example, together with the motivational use cases presented above, illustrates that it is not always sufficient to measure the goodness of the predictions only in terms of accuracy and earliness.

In this chapter, we give an overview of the traditional notions of stability of learning algorithms (Section 6.1), introduce the notion of temporal stability and propose a way to decrease the volatility of sequential predictions via smoothing (Section 6.2), and evaluate the existing techniques for predictive process monitoring with respect to temporal stability (Section 6.3).

6.1. Stability of learning algorithms

Stability of learning algorithms has been a topic of interest for many years. Conventionally, a learning algorithm is considered unstable if small perturbations in the training set can cause significant changes in the predictor [8]. Such instability of single predictors motivated Breiman et al. to introduce *bagging predictors*, showing that the stability and accuracy of a predictor can be increased by aggregating the estimations from multiple versions of the predictor [8]. In this context, increasing stability relates to decreasing the variance between prediction estimates. Bousquet et al. studied the relationship between stability and generalization [6]. Their study is based on *sensitivity analysis*, i.e. how much replacing or deleting a training example affects the prediction loss. They propose three definitions of stability, which are all based on changes in the training set. The reason for this is that they focus on deterministic algorithms, so that all the randomness comes from the sampling on the datasets. Elisseeff et al. extended these notions of stability to non-deterministic algorithms [26] where randomness is present even when the training set remains unchanged. Their stability definitions are supplemented with a randomness parameter. More recently, Liu et al. proposed a metric for measuring stability across several runs of random forest and incorporated it into a framework for selecting the hyperparameters based on a goodness measure combining AUC, stability, and cost [57]. In particular, the question addressed by their notion of stability is the following: if we train multiple classifiers with the same parameter setting but different randomization parameters, would these classifiers agree on the predictions made for the same example or not? Hereinafter, we refer to this notion of stability as the *inter-run stability*.

While existing notions of stability are related to changes made in the training phase (either by changing the training set or by changing the randomness parameter), in this thesis we study the case where both the training dataset and the randomness are fixed, but the input vector changes over time. In particular, we study the *temporal stability* of predictions in the setting where predictions are made successively for different prefixes of the same sequence. In other words, we examine how much increasing the length of the prefix changes the predictions.

Other research proposals from the field of early sequence classification that focus on developing serial classifiers [55, 109] and finding the minimal prediction length (MPL) [111] are closely related to the notion of temporal stability studied in this thesis. In fact, a serial classifier, by definition, must have a prefix length starting from which the predictions made for the same case will not change as the prefix grows further, i.e. the classifier has perfect temporal stability starting from that prefix length. However, instead of determining MPL and making predictions only after the MPL is reached, we are interested in predicting the outcome for every prefix of the sequence. The reason for this is that in a predictive process monitoring setting, it is necessary to give the best estimate of the case outcome even when too few data is available to make a final prediction. In this respect, we aim for temporal stability also on short prefixes, when the prediction might still differ from the one that would be made for the entire sequence.

6.2. Temporal prediction stability

In this section, we start with introducing the notion of prediction scores in outcome-oriented predictive process monitoring. We proceed with defining temporal stability and provide a metric to measure this property. Lastly, we describe our approach for combining prediction scores obtained for prefixes of the same case in order to reduce their volatility.

6.2.1. Prediction scores over time

In predictive process monitoring, the classifier is asked to give an estimation about the case outcome after each performed event. Therefore, the prediction scores estimated after each event of the same case form a time series. As an example, consider the pink time series (Case B) plotted in Figure 24 (left). During the first 5 events, the classifier is unsure about what will be the outcome of this case (the prediction scores for these events are equal to 0.5). Then, the 6th event provides some relevant signal, so that the classifier becomes confident that the case will be positive (the prediction scores for the following events are 0.9). This series is rather stable over time, as the successive prediction scores change only once. An example of a completely stable series of predictions is Case A (the black line), where the prediction scores remain the same for all prefixes. Now consider Cases C and D (green and blue). We can see that the classifier changes the prediction score after almost every event, producing a *volatile* time series for these cases.

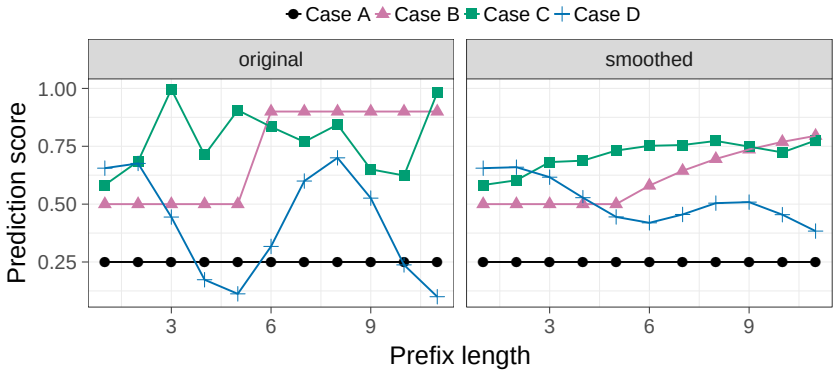


Figure 24: Examples of prediction scores over time: original (left) and smoothed (right).

Such unstable predictions have little practical value, causing users to be cautious about acting upon the prediction and decreasing the overall credibility of the classifier.

6.2.2. Temporal stability

Based on the above rationale, we say that a classifier is *temporally stable* if it (generally) outputs similar predictions to successive prefixes from the same sequence.

Given a threshold on the prediction scores that determines whether the predicted outcome is positive or negative, it would be natural to define temporal instability as the number of times the classifier "flips" its prediction, and to define temporal stability as one minus a normalized measure of instability. The drawback of this approach is that it is dependent on the chosen threshold. Instead, we aim for a more general, threshold-independent measure that would capture the stability of the classifier under any threshold. Accordingly, we propose to measure stability as a function of the magnitude of the changes between successive prediction scores. This latter definition is related to the former: If the difference between successive scores is high, there exist many thresholds that would lead to flips in the predicted outcome. Conversely, if the difference is low, only a low number of thresholds would flip the prediction.

The simplest way to consider the magnitude of the changes would be to measure the (average) absolute difference between successive prediction scores. Note that this metric does not consider the direction of the changes, i.e. a change towards the correct direction (the actual class) affects the measure in the same way as a change towards the wrong direction. As a result, a classifier that consistently improves its prediction is assigned a similar stability score as one that fluctuates around the same score. An alternative would be to consider only the changes that are made to the wrong direction, calculating the (average) absolute difference only over these changes. However, this metric would reflect the *consistency* of the classifier rather than its *stability*. For instance, consider a sequence with the actual

outcome being *positive*, and two classifiers. One of the classifiers outputs a score of 1 at the first event, i.e. it is (correctly) certain that the outcome will be positive, but throughout the case becomes only slightly less certain of it, outputting 0.99 on some events. The other classifier makes a completely wrong estimation at the beginning of the sequence, outputting a score of 0, while throughout the rest of the case, it only improves its estimate (sometimes by large magnitudes), producing scores like 0.1, 0.5, and even 0.95. According to the latter metric, the second classifier, which makes changes in large magnitudes, would be considered more stable than the first classifier, although the first one only changes its prediction by a small amount. In a sense, a measure that considers the direction of the change penalizes classifiers that make the right prediction from the onset, since the only way to maintain their stability throughout the sequence would be to always output exactly the same score. Based on these considerations, we proceed with measuring the average difference between the successive prediction scores without taking into account the direction of the change.

Accordingly, we measure the temporal stability (TS) of a classifier as one minus the average absolute difference between any two successive prediction scores:

$$TS = 1 - \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i - 1}\sum_{t=2}^{T_i}|\hat{y}^{(i)<t>} - \hat{y}^{(i)<t-1>}|, \tag{6.1}$$

where N is the number of cases used for the evaluation, T_i is the total number of events in the i-th case, and $\hat{y}^{(i)<t>}$ is the prediction score of the t-th event of the i-th case. This metric first evaluates the average absolute difference between successive prediction scores within each case in order to eliminate the bias towards long sequences, and then averages over the cases.

6.2.3. Combining prediction scores via smoothing

We can adjust the prediction scores during a post-processing phase to reduce volatility without affecting the pre-trained classifier. Specifically, instead of using explicitly the score that the classifier outputs for a case after observing t events, we combine it with prediction scores made for shorter prefixes of the same case.

To combine predictions, we can use various time series *smoothing* methods, which average out the noise and fluctuations. The simplest way to smooth a time series is via a *moving average*. The smoothed estimate at each event is computed as the average of the last k observations. A different approach, called *single exponential smoothing*, assigns weights that decrease exponentially over time. The smoothed estimate at time t is the combination of the observed value at time t and the smoothed estimate at time $t-1$, using a smoothing parameter α, $0 \leq \alpha \leq 1$: $s^{(i)<t>} = (1-\alpha)\cdot\hat{y}^{(i)<t>} + \alpha \cdot s^{(i)<t-1>}$. Parameter α controls to what extent the previous observations are taken into account. The larger the α, the stronger the smoothing effect. While other smoothing techniques are available, we use the single exponential smoothing because of its simplicity and because it allows us to

directly control the level of smoothing. Also, only techniques that enable sequential smoothing (as opposed to smoothing over the entire sequence) are applicable in our case, as in the predictive process monitoring setting, only the prediction scores made up to a certain point in the sequence are known.

For example, consider the time series plotted in Figure 24 (right). These time series have been derived from the examples in Figure 24 (left) by applying exponential smoothing with $\alpha = 0.8$. We can notice that the fluctuations in Cases C and D have been reduced considerably. However, smoothing can also have a negative effect on the predictions, illustrated by Case B. Namely, changes in the scores do not have an immediate strong effect, as the adjusted score puts some weight on the previous estimates. Therefore, when an event carrying a relevant signal about the case outcome arrives, the smoothed estimate is cautious about trusting it, resulting in a lower accuracy.

6.3. Evaluation

In this section we conduct an empirical evaluation to address the following questions:

RQ1 (Temporal stability) What is the relative performance of different predictive process monitoring methods in terms of temporal stability (in addition to accuracy)?

RQ2 (Inter-run stability) How does maximizing the inter-run stability in combination with prediction accuracy affect the temporal stability?

RQ3 (Smoothing) How does decreasing prediction volatility via exponential smoothing affect the accuracy and the temporal stability?

Below, we describe the employed approaches and evaluation datasets, we explain the experimental setup, and discuss the results.

6.3.1. Approaches

To address RQ1, we choose 7 predictive process monitoring approaches (see Table 20) as basis for the experiments. For structured data, we experiment with two existing sequence encoding techniques, the index-based and the aggregation encoding. For modeling unstructured data we always use BoNG with aggregation encoding, which showed the best overall performance in Chapter 5. Both encodings (index-based and aggregation) are combined with two classification methods, RF and XGBoost. Additionally, we adapt a predictive process monitoring method based on LSTM neural networks [94] to predict the outcome of a case. While the latter method uses only control flow and timestamp information, we employ a straightforward extension of this method by concatenating the (one-hot encoded) feature vectors for each data attribute in an event. For encoding textual data in LSTMs, we consider the 500 most frequent words and concatenate the vector of frequencies of these words in a given event.

Table 20: Approaches.

Approach	Multi/single cls	Encoding	Classifier
RF_agg	single	aggregation	RF
RF_idx_pad	single	index	RF
RF_idx_mul	multi	index	RF
XGB_idx_pad	single	index	XGBoost
XGB_idx_mul	multi	index	XGBoost
XGB_agg	single	aggregation	XGBoost
LSTM	single	index	LSTM

In all of the approaches, each prefix constitutes a separate training instance. For index-based encoding, the fact that different prefixes consist of different numbers of events raises an issue when trying to encode all prefixes with fixed-length vectors. There are two possible solutions to this issue. Firstly, it is possible to fix the maximum prefix length and, for shorter prefixes, pad the data for missing events with zeros. An alternative solution is to build multiple classifiers, one for each prefix length; given a prefix of length l in the testing set, the prediction for this prefix is derived from the classifier constructed based on prefixes (in the training set) of length l. In our experiments, we apply both solutions to the RF and XGBoost based approaches, marked as *RF_idx_pad*/*XGB_idx_pad* and *RF_idx_mul*/*XGB_idx_mul*, respectively. Since the second, multiclassifier solution is not commonly used with LSTMs, in the LSTM-based approach we only apply the padding solution.

Prediction scores returned by classifiers are often poorly calibrated, meaning that the scores do not reflect well the actual probabilities of belonging to one class or to the other [41]. For instance, one classifier may output scores that are always concentrated around 0.5, while another may return scores that are well distributed within the range between 0 and 1. This causes bias when comparing different classifiers in terms of temporal stability. Indeed, the differences between any two prediction scores in the case of the former classifier are very small, making it seem a very stable classifier, while the relative differences within each case might be larger than in the latter classifier. To address this issue, we apply a well-known calibration method, Platt scaling [76], to each of the classifiers before comparison. We choose this technique because it outperforms other methods when data is scarce (e.g. less than 1000 data points available for calibration) [69], which is the case in most of our datasets. Note that, in principle, calibration does not change the order of the prediction scores assigned by the same classifier, so that the AUC of each classifier is not affected by it. However, in order to perform calibration, a separate validation set is needed, so the final models are effectively built using a smaller number of training traces, which might slightly affect the AUC.

To test RQ2 (How does maximizing the inter-run stability in combination with

prediction accuracy affect the temporal stability?), we adapt the approach proposed in [57] to RF and XGBoost hyperparameter optimization. Namely, instead of choosing the optimal parameter setting based on AUC on a single run of classifier training, we perform 5 runs with each setting and choose the one that achieves 1) the best average AUC over all runs, and 2) the best combined AUC and inter-run stability[1] over all runs. For the latter scenario, we give more weight to the inter-run stability, assigning weights 1 and 5 to AUC and stability, respectively.

To decrease prediction volatility (RQ3), we experiment with exponential smoothing, varying the smoothing parameter $\alpha \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$.

6.3.2. Datasets

As evaluation datasets, we use the same event logs and preprocessing as introduced in Chapters 4 and 5.

6.3.3. Experimental setup

In general, we apply the same experimental setup as described in Chapters 4 and 5. In particular, we employ a temporal holdout split into 80% training and 20% testing data. The hyperparameters for RF and XGBoost are optimized using TPE with either 3-fold cross-validation (for event logs containing only structured data payload) or on a single validation set (for event logs containing unstructured data). As LSTM is computationally more demanding than RF and XGBoost, we use a single validation set for optimizing the LSTM hyperparameters on all event logs (the same setup as in Chapter 5). In other words, the only difference in the hyperparameter optimization procedure between RF/XGBoost and LSTM is that for event logs introduced in Section 4.1, in case of LSTM we use a single holdout split rather than 3-fold cross-validation. During training, LSTMs optimize binary cross-entropy, which is why we select the best parameters according to this metric instead of AUC. Table 21 presents the bounds and the sampling distributions for the LSTM hyperparameters, given as input to TPE. The activation function for LSTM is always fixed to sigmoid and the number of epochs to 100. We optimize the weights using Adam, which is a stochastic gradient-based optimization algorithm with adaptive learning rates [46].

After selecting the optimal hyperparameters we use a holdout split to randomly divide the traces in the training set into 80% (i.e. 64% of the total) for building the final model (L_{train}) and 20% (i.e. 16% of the total) for calibration (L_{calib}).

Experiments were implemented in Python 3.5. For RF and XGBoost we used the scikit-learn library on a single core of a Intel(R) Xeon(R) CPU E5-2660 v2 @

[1] *Inter-run stability refers to the MSPD metric introduced in [57]:* $MSPD(f) = 2\mathbb{E}_{x_i}[Var(f(x_i)) - Cov(f_j(x_i), f_k(x_i))]$, *where* $\mathbb{E}_{x_i}$ *is the expectation over all validation data,* f *is a mapping from an example* x_i *to a label* y_i *on a given run,* $Var(f(x_i))$ *is the variance of the predictions of a single data point over the model runs, and* $Cov(f_j(x_i), f_k(x_i))$ *is the covariance of predictions of a single data point over two model runs.*

Table 21: Hyperparameters of LSTM and their sampling distributions used in optimization via TPE.

Classifier	Parameter	Distribution	Values
LSTM	# hidden layers	Categorical	$x \in \{1,2,3\}$
	# units in hidden layer	Log-uniform integer	$x \in [10,150]$
	Initial learning rate	Log-uniform	$x \in [0.000001, 0.0001]$
	Batch size	Categorical	$x \in \{8,16,32,64\}$
	Dropout	Uniform	$x \in [0,0.3]$
	L1 regularizer	Log-uniform	$x \in [0.00001, 0.1]$
	L2 regularizer	Log-uniform	$x \in [0.00001, 0.1]$

2.20GHz with 64GB of RAM. Experiments for LSTM were performed using the Keras[2] with Tensorflow[3] backend.

6.3.4. Results

General comparison. Figures 25–26 show the prediction accuracy (AUC) of the calibrated classifiers across different prefix lengths. One observation is that the multiclassifiers (*RF_idx_mul* and *XGB_idx_mul*) can yield a high accuracy on some prefixes (especially on the shorter ones), but at the same time the results are very volatile, causing the AUC to drop unexpectedly. For instance, see *XGB_idx_mul* with $prefix = 13$ in *production* or $prefix = 6$ in *DR*. On long prefixes, the index-based encoding approaches (both multiclassifiers and single classifiers with padding) tend to perform worse than the other methods. There are some execptions to this rule, e.g. *sepsis_2* and *hospital_1*, where *XGB_idx_pad* performs well over all prefix lengths.

The performance of LSTM differs considerably between datasets. In some datasets (such as *bpic2015*, *bpic2012*, *bpic2017*, and *hospital* variants, as well as *sepsis_2*) LSTM achieves top performance over all prefix lengths. Conversely, in some others (*bpic2011*, *insurance*, *production*, *github*, and *sepsis_3*) LSTM is not competitive with other techniques. In general, LSTM tends to perform better on large datasets, implying that a sufficiently large amount of training data is needed for LSTM to be usable in predictive monitoring scenarios. An interesting observation can be made in the case of *bpic2012_1* and *bpic2012_3*, where the accuracy of LSTM is rather low for shorter prefixes, but after the relevant signal comes in (around prefix length 13), the model is able to make use of it better than the other methods, reaching the highest AUC on long prefixes. Relating this to the characteristics of the datasets (see Table 9 in Chapter 4), the *bpic2012* datasets do not contain any case attributes. This suggests that LSTM is more prone to the "cold start" problem, i.e. when there is few data in the early prefixes, LSTM needs

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

[3] `https://www.tensorflow.org/`

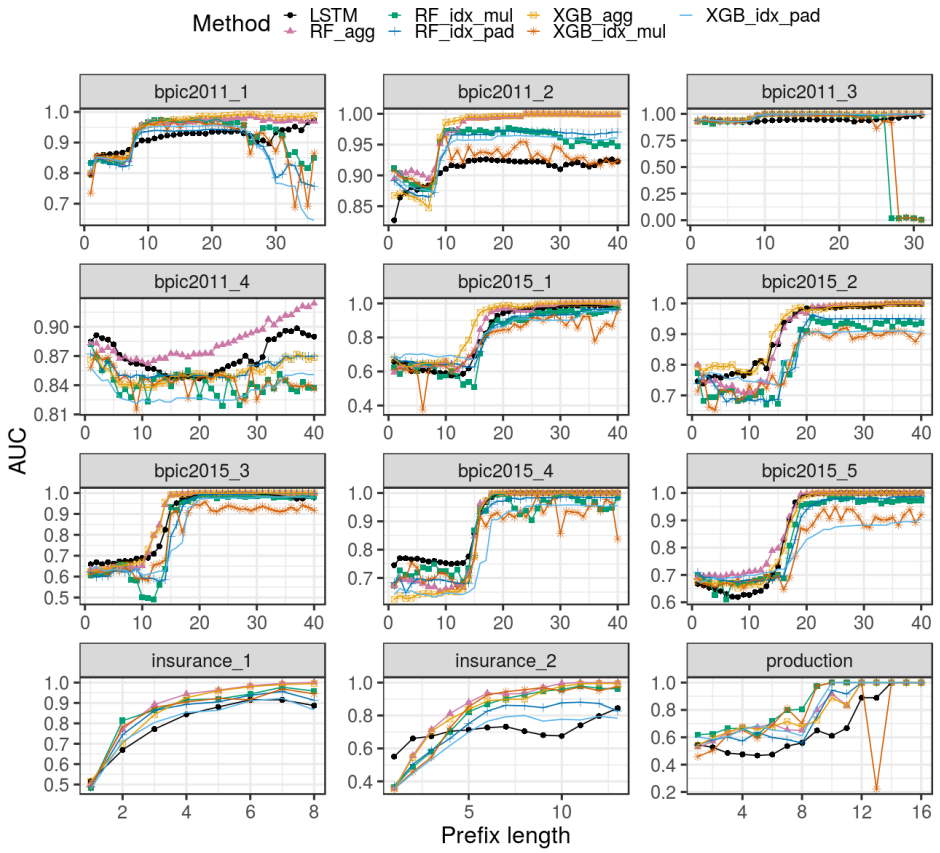


Figure 25: Prediction accuracy.

to wait for more events in order to make a good prediction. A general observation is that even in datasets where LSTM does not achieve the highest AUC, its performance always remains reasonably stable, in the sense that no sudden drops in AUC occur in any prefix length.

The single classifiers with aggregation encoding (*RF_agg* and *XGB_agg*) perform well on both short and long prefixes. Although in some prefix lengths they are outperformed by the index-based encoding methods, they are overall more stable. In particular, these methods are somewhat more volatile than LSTM, but they usually do not undergo strong falls in AUC as the multiclassifiers. For example, see *bpic2015*, *LtC*, and *sepsis_cases_3* where *RF_agg* and *XGB_agg* retain high accuracy on long prefixes, while *RF_idx_mul* and *XGB_idx_mul* become more volatile.

Additionally we assess the effect of calibration on the predictions. Specifically, we measure the Brier score, which is a frequently used and well suited measure for assessing the quality of prediction score in terms of their calibratedness [51]. Table 39 in Appendix shows the overall Brier scores (i.e. measured over all prefix

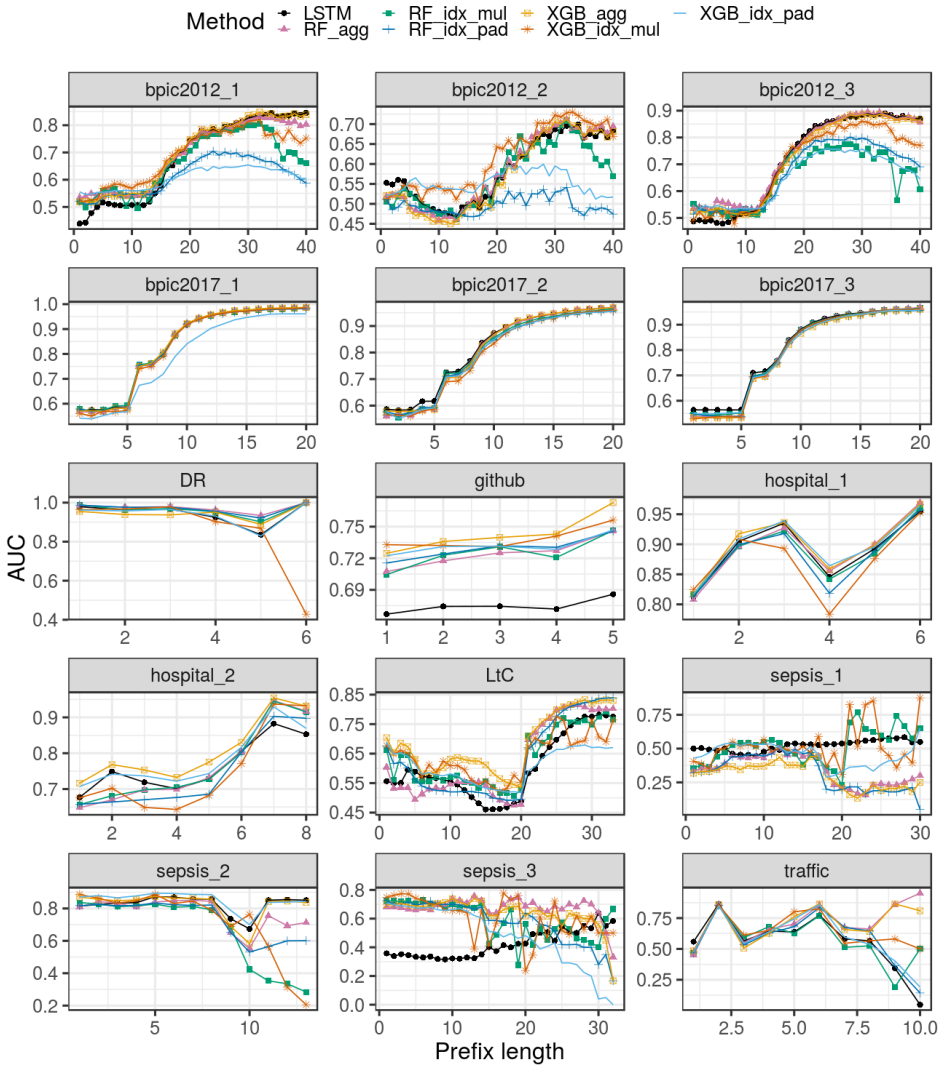


Figure 26: Prediction accuracy (continued).

lengths) for each dataset and method combination. We can observe that single classifiers with aggregation encoding (both RF_agg and XGB_agg) and LSTMs tend to produce better calibrated predictions (i.e. lower Brier scores). LSTMs are initially better calibrated and are not much affected by calibration. In case of RF and XGBoost, the patterns are more mixed: calibration helps to improve the Brier scores in some datasets, while in others can make the predictions even less calibrated compared to the initial ones. The effects of calibration are further explored in Figures 51-52 in Appendix, where the differences in Brier scores between uncalibrated and calibrated classifiers are plotted against the length of the prefixes. In other words, a positive value in these plots indicates that applying Platt scaling resulted in better calibrated classifiers, while a negative value shows applying the calibration technique produces worse scores in terms of calibratedness. We can see that the effects of calibration depend on the prefix length, but the patterns are quite different across the datasets. In some cases, calibration helps considerably on early prefixes (see *bpic2017*, *production*, *hospital_2*), while in other datasets the effects are more positive on longer prefixes (see *github*, *traffic*).

The temporal stability is plotted in Figures 27–28. In 15 out of 27 datasets the highest stability is achieved by LSTM, while *XGB_idx_pad* reaches the highest stability in 9 datasets. In general, RF achieves slightly lower stability than its XGBoost counterparts. The multiclassifier approaches almost always have lower temporal stability than single classifiers, which is not surprising. Namely, as the RF and XGBoost classifiers do not consider the temporal relations between the input features and, instead, assume them to be independent and identically distributed (i.i.d.), the variance between classifiers built for prefixes of length l and $l+1$ can be very high and, thus, the predictions made for two successive prefixes can be completely uncorrelated. This discussion answers RQ1 (What is the relative performance of different predictive process monitoring methods in terms of temporal stability (in addition to accuracy)?).

Increasing inter-run accuracy and stability in model selection. Table 22 presents the overall AUC (the weighted average over all prefix lengths) and the temporal stability for the single classifier with aggregation encoding with RF and XGBoost using three hyperparameter optimization approaches: i) validation based on AUC over a single run with each parameter setting (*RF*, *XGB*); ii) validation based on average AUC over 5 runs with each parameter setting (*RF_5*, *XGB_5*); and iii) validation based on a combined measure of mean AUC and inter-run stability over 5 runs with each parameter setting (*RF_5_S*, *XGB_5_S*).

The results show that selecting the best parameters according to AUC over multiple runs usually (in 19 out of 27 datasets) increases the AUC on the test set as compared to selecting them based on a single run, while the temporal stability is increased in 16 datasets. The trends are more mixed between optimizing only the AUC and optimizing the combined metric (both over multiple runs). The latter approach of optimizing the combined AUC and inter-run stability yields better results for RF, making it the best method in terms of AUC (overall best in

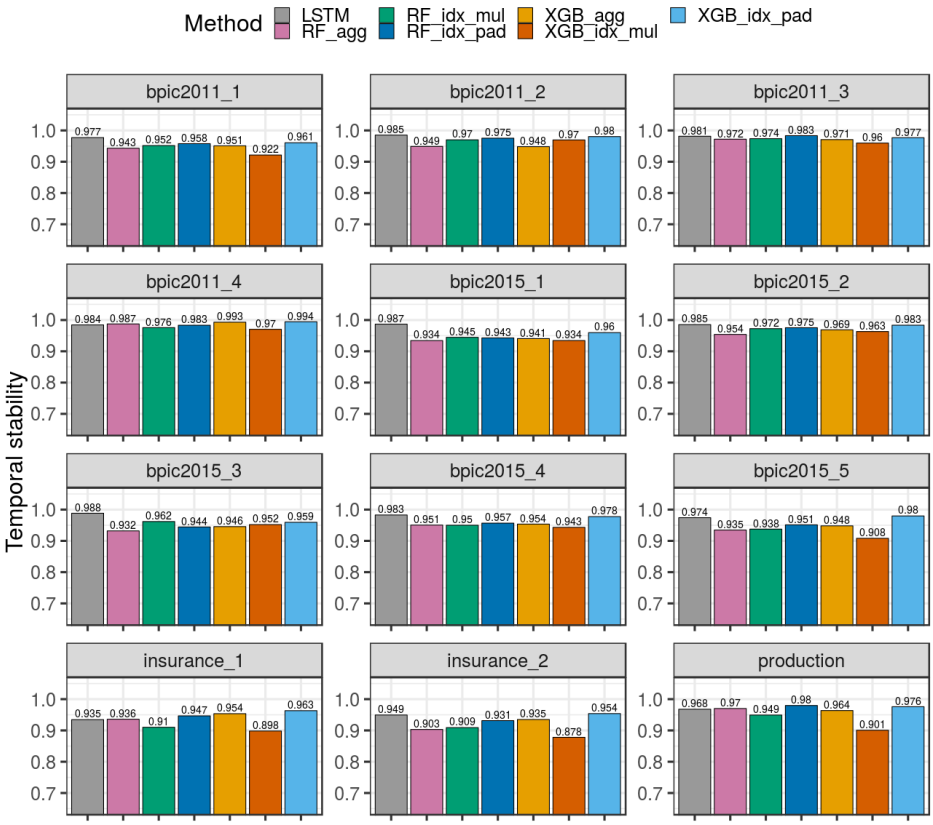


Figure 27: Temporal stability.

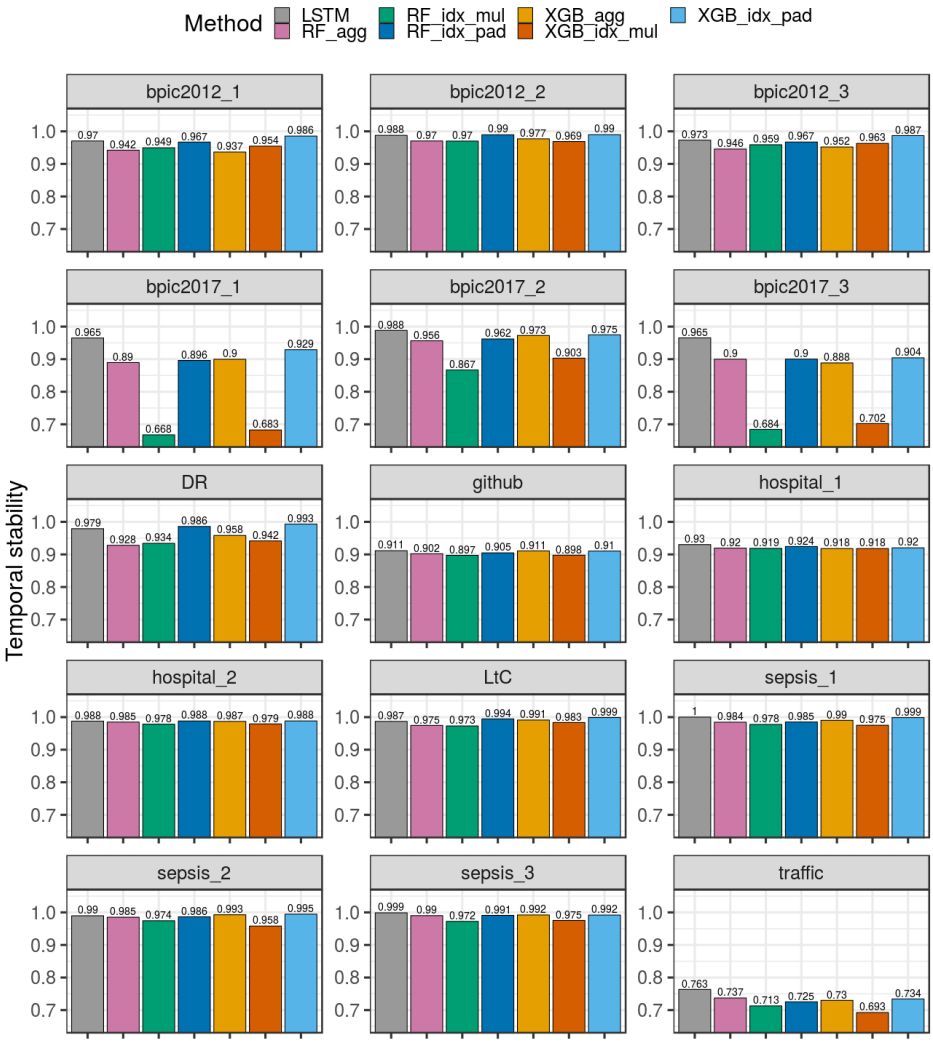


Figure 28: Temporal stability (continued).

Table 22: Effects of maximizing the inter-run stability and accuracy (during hyperparameter optimization) on the temporal stability and accuracy of the final models.

	Prediction accuracy (AUC)						Temporal stability					
Dataset	RF	RF_5	RF_5_S	XGB	XGB_5	XGB_5_S	RF	RF_5	RF_5_S	XGB	XGB_5	XGB_5_S
bpic2011_1	0.937	0.935	0.898	0.944	**0.954**	0.925	0.943	0.947	0.944	0.951	**0.953**	0.948
bpic2011_2	0.972	0.972	**0.973**	0.967	0.964	0.969	0.949	0.956	**0.958**	0.948	0.95	0.942
bpic2011_3	0.979	0.979	0.979	**0.98**	0.979	**0.98**	0.972	**0.975**	0.965	0.971	0.974	0.969
bpic2011_4	0.883	0.871	**0.887**	0.852	0.865	0.884	0.987	0.991	0.987	0.993	**0.996**	0.991
bpic2015_1	0.834	0.827	0.837	**0.859**	0.793	0.84	0.934	0.933	0.934	0.941	**0.943**	0.939
bpic2015_2	0.895	0.898	0.9	0.919	**0.922**	0.869	0.954	0.953	0.954	**0.969**	0.967	0.952
bpic2015_3	0.887	0.886	**0.891**	0.889	0.889	0.886	0.932	0.934	0.932	**0.946**	0.942	0.937
bpic2015_4	0.865	0.862	**0.877**	0.849	0.86	0.876	0.951	0.952	0.949	0.954	0.951	**0.955**
bpic2015_5	**0.881**	0.873	0.874	0.862	0.87	0.863	0.935	0.938	0.937	**0.948**	**0.948**	0.944
production	0.668	0.707	**0.719**	0.674	0.667	0.683	**0.97**	0.954	0.953	0.964	0.962	0.946
insurance_1	**0.879**	0.877	0.875	0.861	0.86	0.87	0.936	0.937	0.929	**0.954**	**0.954**	0.946
insurance_2	0.81	0.828	**0.831**	0.786	0.806	0.808	0.903	0.896	0.893	**0.935**	0.929	0.905
sepsis_1	0.396	0.378	0.388	0.348	0.398	**0.401**	0.984	0.977	0.978	0.99	0.992	**0.999**
sepsis_2	0.779	0.781	0.758	0.824	**0.869**	0.842	0.985	0.987	0.978	0.993	0.992	**0.994**
sepsis_3	0.688	0.691	**0.714**	0.689	0.675	0.708	0.99	0.989	0.992	0.992	0.99	**0.993**
bpic2012_1	0.668	0.667	0.663	0.673	**0.678**	0.676	0.942	0.94	**0.943**	0.937	0.94	0.939
bpic2012_2	0.561	0.565	0.56	0.553	**0.566**	0.561	0.97	0.972	**0.98**	0.977	0.978	0.974
bpic2012_3	0.706	0.708	**0.709**	0.693	0.683	0.665	0.946	0.945	0.945	**0.952**	0.947	0.945
bpic2017_1	0.834	0.835	0.834	0.837	0.843	**0.846**	0.886	0.889	0.892	0.889	0.895	**0.897**
bpic2017_2	0.805	0.808	0.808	0.808	**0.821**	0.817	0.947	0.957	0.957	**0.97**	0.965	0.961
bpic2017_3	0.799	0.801	0.8	0.789	**0.813**	0.808	0.894	**0.896**	0.89	0.881	**0.896**	0.895
traffic	0.646	0.65	**0.661**	0.645	0.641	0.634	0.737	0.74	**0.754**	0.73	0.731	0.726
hospital_1	0.884	0.883	0.884	0.892	**0.897**	0.896	**0.92**	0.919	0.918	0.918	0.919	0.918
hospital_2	0.699	0.701	0.704	**0.758**	0.755	0.752	0.985	0.981	0.979	**0.987**	0.983	0.985
DR	**0.981**	0.978	0.977	0.952	0.96	0.963	0.931	0.911	0.914	0.947	0.952	**0.992**
github	0.724	0.721	0.722	**0.74**	0.734	0.737	0.912	0.906	0.906	**0.915**	0.908	0.909
LtC	0.545	0.564	0.044	**0.63**	0.602	0.599	0.975	0.973	0.956	0.991	**0.995**	0.988

9 datasets), while the effects on temporal stability are inconsistent. For XGBoost, the first approach tends to work better, yielding an increase in AUC in most of the cases (overall best in 8 datasets), while not affecting much the temporal stability.

To answer RQ2 (How does maximizing the inter-run stability in combination with prediction accuracy affect the temporal stability?), we found that validating over multiple runs instead of a single run, in general, results in improvement of AUC, but has little effect on temporal stability. It is also worth noting that the improvements in AUC come at the expense of running 5 times more experiments during the hyperparameter optimization phase.

Decreasing intra-case prediction volatility at prediction time. Figures 29–30 show that decreasing the prediction volatility via exponential smoothing consistently improves the temporal stability. The larger the smoothing parameter α, the larger the increase in temporal stability. The methods that benefit the most from smoothing are multiclassifiers (*RF_idx_mul* and *XGB_idx_mul*). Being initially less stable, smoothing helps these methods to achieve a similar level of temporal stability as the other methods. In some cases, the multiclassifiers even overtake the other methods on large α (see *bpic2011*, *bpic2015*, and *bpic2012* variants).

In Figures 31–32, the overall AUC is plotted against the α parameter. We observe that in most cases smoothing decreases the AUC. The reason for this is that as the smoothed estimate is cautious about the most recent prediction, the

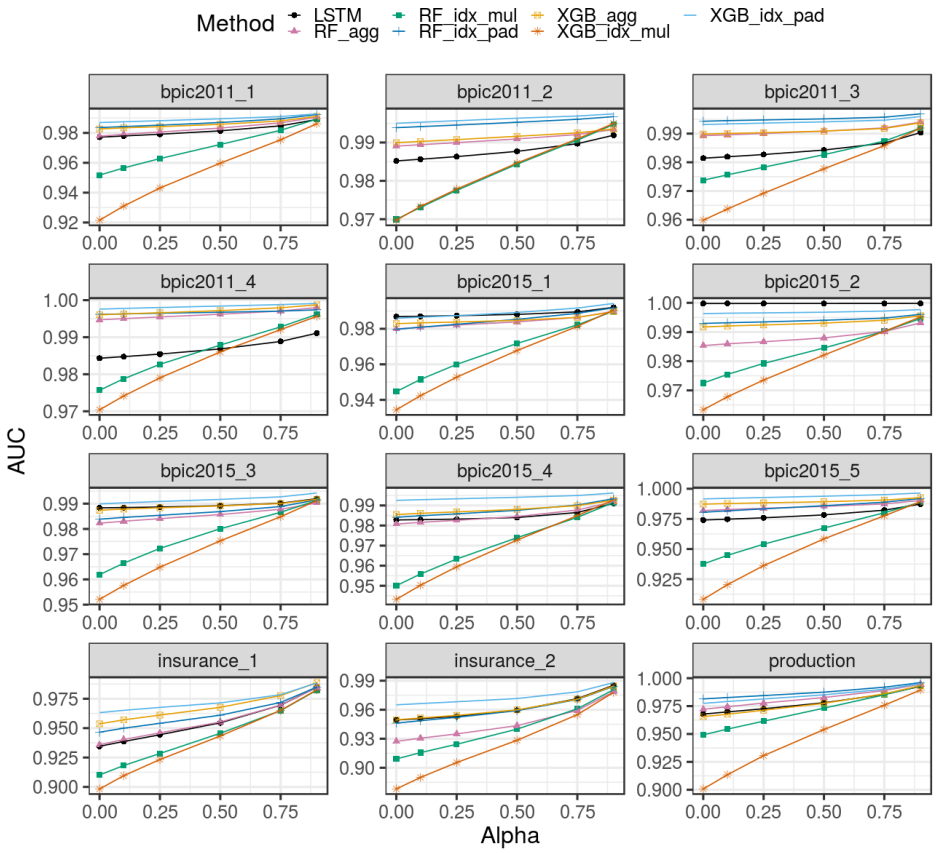


Figure 29: Temporal stability across different levels of smoothing.

true signal in the data occurs after a lag. However, the AUC does not always decrease with smoothing. Especially in smaller logs, the AUC remains almost unchanged by smoothing or even increases (e.g. see *bpic2011_3*, *bpic2015_3*, *sepsis_2*, *sepsis_3*). The methods that benefit the most from smoothing are again the multiclassifiers. While not the most accurate methods before postprocessing, they often overtake the other methods with high levels of smoothing.

To further understand the relationship between AUC and temporal stability, let us look at Figures 33–34, where these two metrics are plotted against each other (each dot corresponds to AUC and temporal stability obtained via smoothing with a particular value of α). We see that *RF_idx_mul* and *XGB_idx_mul* change considerably in the direction from left to right, indicating that they are initially unstable but improve substantially with smoothing. At the same time, their change in the up-down direction is small, meaning that the AUC is not affected much. The least affected by smoothing is the *XGB_idx_pad* method. For instance, in *bpic2015* and *sepsis* variants both the accuracy and the temporal stability remain almost constant. In the top right corner we can usually see *XGB_agg* and *RF_agg*,

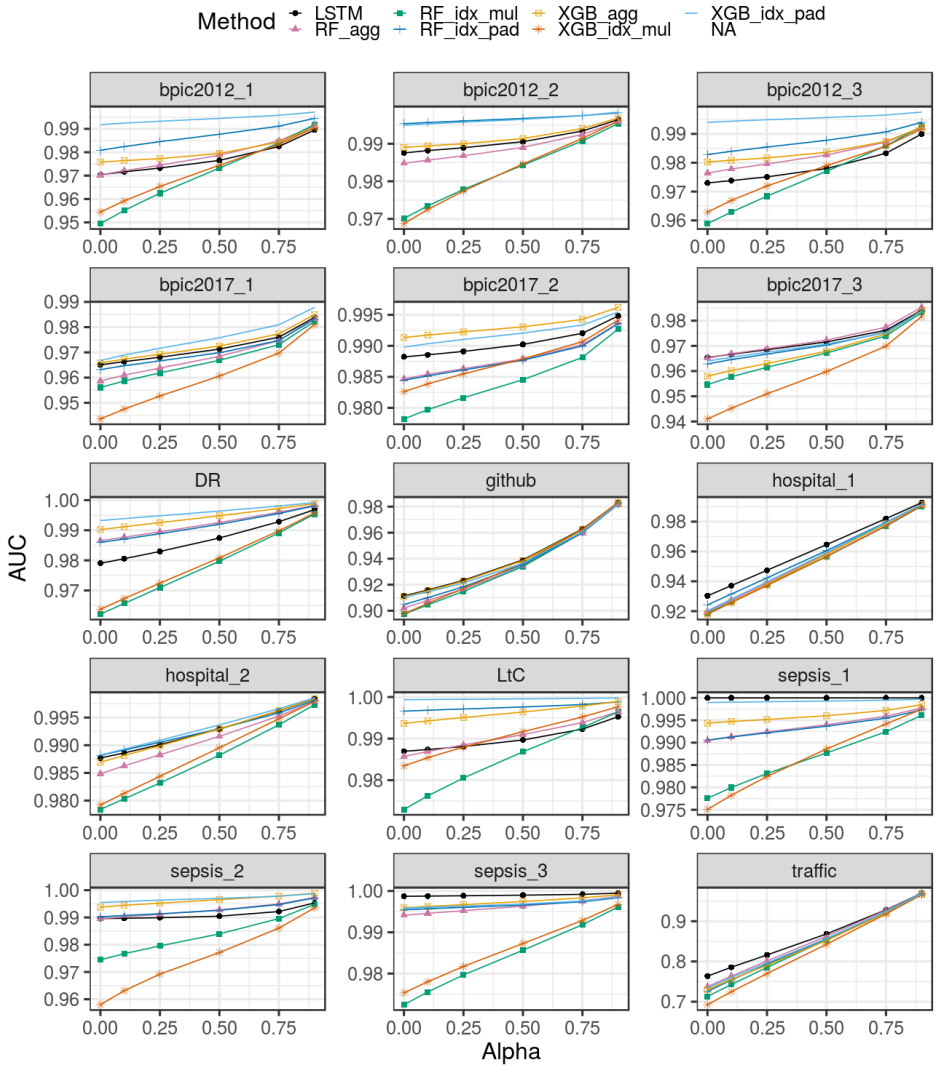


Figure 30: Temporal stability across different levels of smoothing (continued).

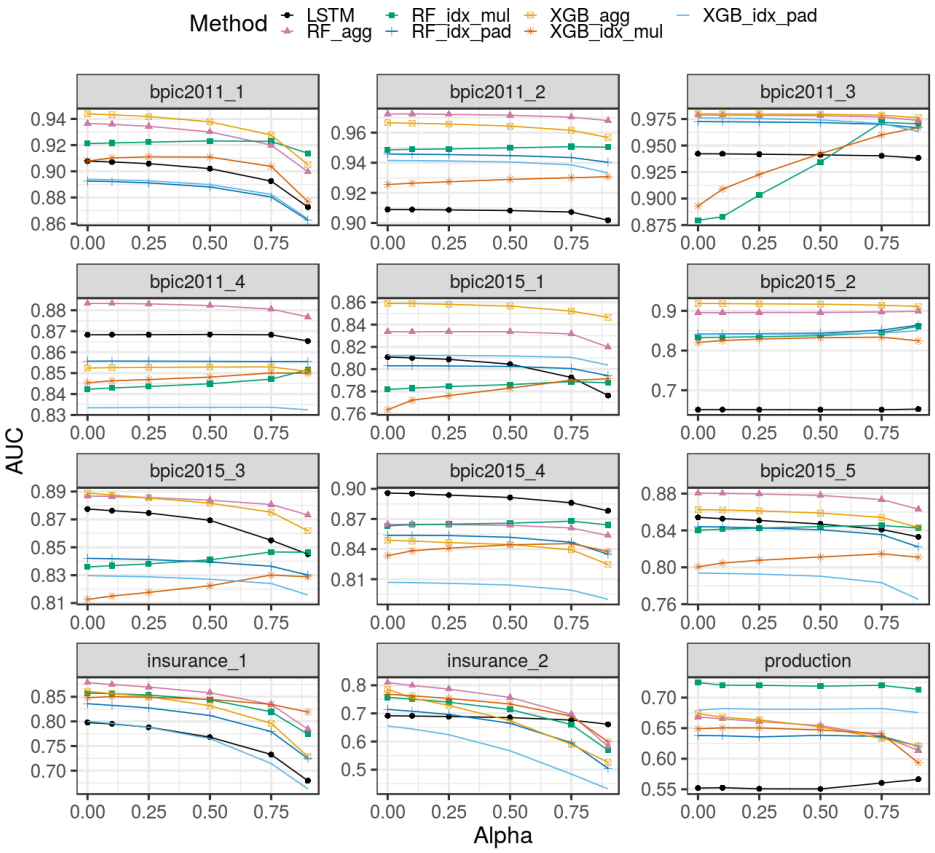


Figure 31: Overall prediction accuracy across different levels of smoothing.

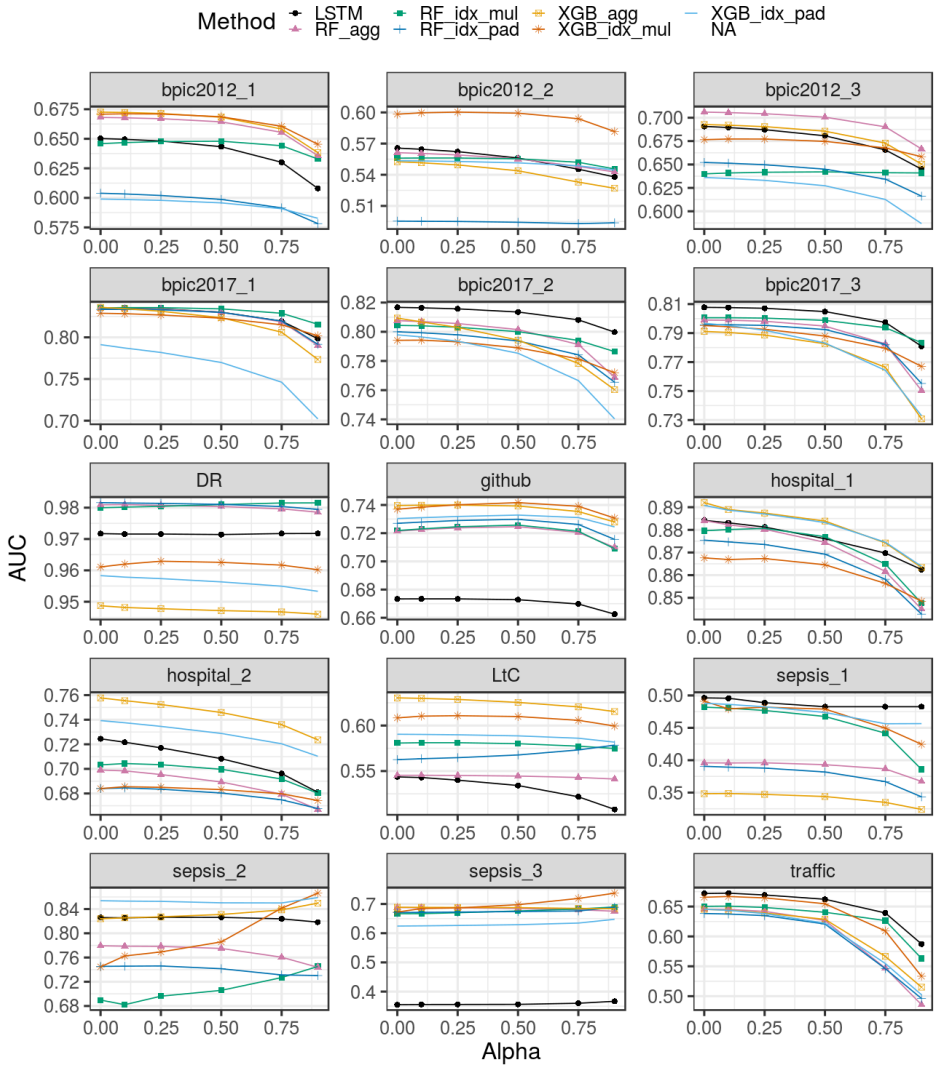


Figure 32: Overall prediction accuracy across different levels of smoothing (continued).

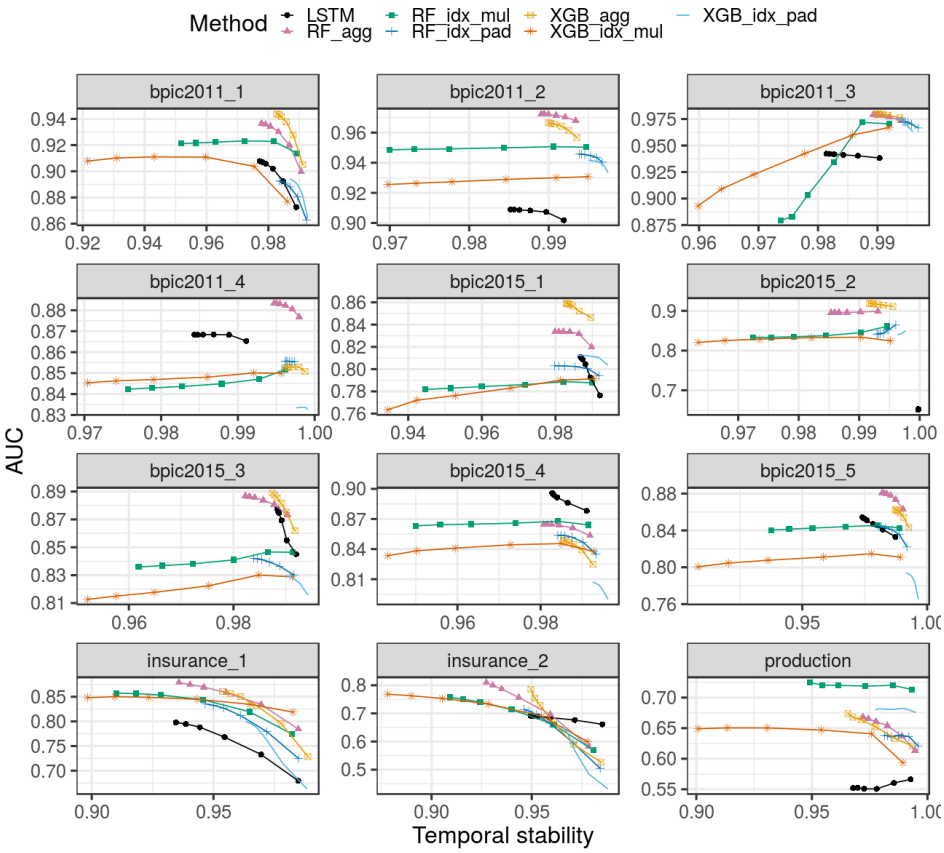


Figure 33: Temporal stability vs. prediction accuracy.

often also LSTM, dominating the other techniques in terms of both accuracy and stability.

To answer RQ3 (How does decreasing prediction volatility via exponential smoothing affect the accuracy and the temporal stability?), exponential smoothing helps to increase the temporal stability, but usually at the expense of lower accuracy. Exceptions are *RF_idx_mul* and *XGB_idx_mul*, where smoothing often increases both temporal stability and AUC. The best tradeoff between temporal stability and accuracy is usually achieved by *XGB_agg* and *RF_agg*.

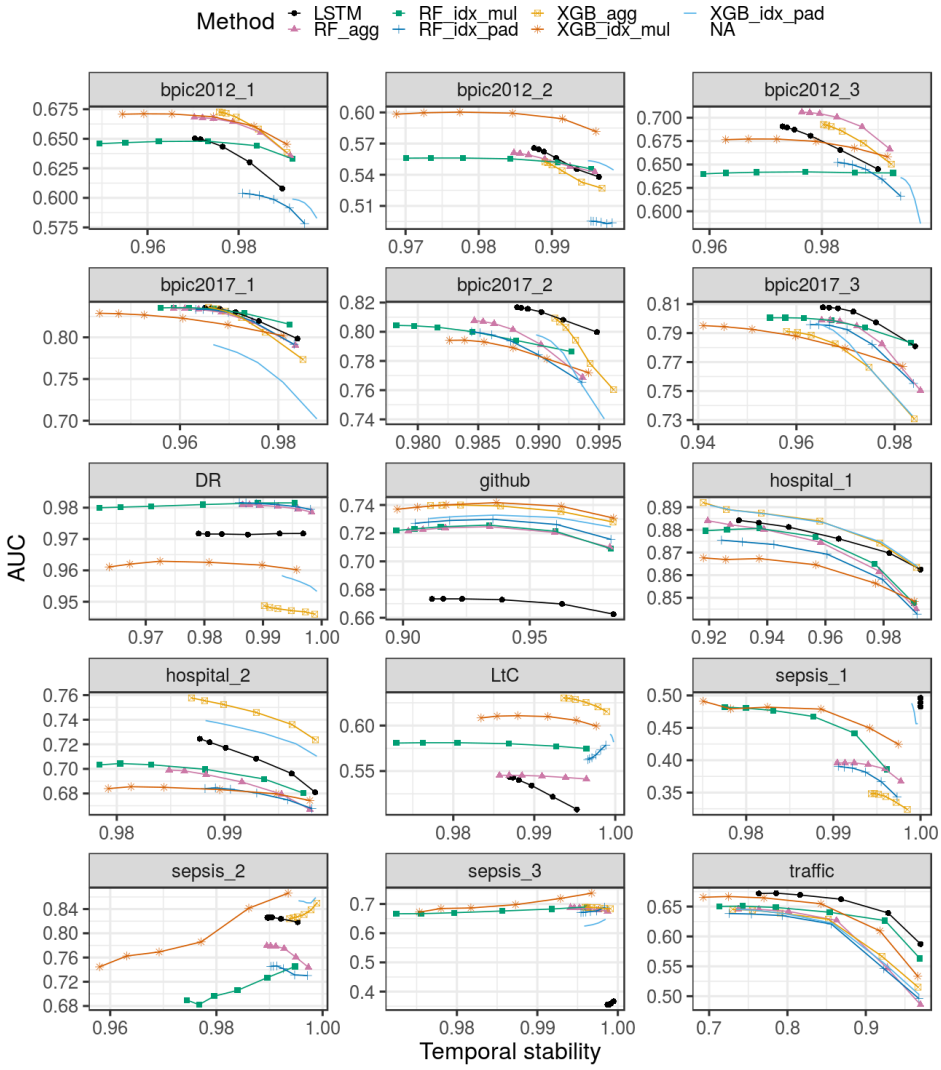


Figure 34: Temporal stability vs. prediction accuracy (continued).

6.4. Summary

In this chapter we introduced the notion of temporal stability for predictive process monitoring. Temporal stability characterizes how much successive prediction scores obtained for the same case (sequence of events) differ from each other. For a temporally stable classifier, such successive prediction scores are similar to each other, resulting in a smooth time series, while in case of an unstable classifier, the resulting time series is volatile. We evaluated the temporal stability of 7 existing predictive process monitoring methods, including single and multiclassifiers using RF, XGBoost, and LSTM. The experiments were done on 27 prediction tasks formulated on 12 real-life datasets. We found that the highest temporal stability was achieved by LSTM, followed by a single classifier approach with XGBoost (using either aggregation or index-based encoding).

We investigated the effects of hyperparameter optimization on temporal stability. We compared the final classifiers constructed after selecting the best parameters based on 1) AUC over a single run for each parameter setting, 2) AUC over 5 runs for each setting, 3) combined AUC and inter-run stability over 5 runs for each setting. The results show that choosing the parameters based on 5 runs can increase both AUC and temporal stability. However, the improvement is small and is subject to the trade-off of 5 times more computations during validation.

Finally, we explored how exponential smoothing affects the AUC and temporal stability. We concluded that smoothing can be a reasonable approach for adjusting the predictions in applications where temporal stability is important at the expense of achieving slightly smaller AUC. Moreover, we observed that the multiclassifiers benefit the most from smoothing, in some cases even increasing both the temporal stability and the AUC at the same time. Therefore, when high temporal stability is required, it may be reasonable to use a multiclassifier approach with smoothing, achieving stable results with little or no loss in accuracy.

This chapter addressed the task of evaluating predictive models in the context of predictive business process monitoring. In the evaluation we assumed a continuous monitoring setting, where predictions are made after each event but no advice is given on using the predictions. In the next chapter we focus on using the predictions returned by a predictive model in order to mitigate or prevent undesired outcomes in business processes.

7. ALARM-BASED PRESCRIPTIVE PROCESS MONITORING

After a predictive process monitoring system has been constructed, it can be used to support decision-making in a real business process setting by either notifying the process workers when they should intervene in a process or recommending intervention actions that would help to prevent or mitigate the undesired outcomes. As seen in Chapter 3, such prescriptive monitoring use cases have received little attention in the literature. In particular, the existing methods for alarm-based prescriptive monitoring are limited to very basic alarming mechanisms, where the user is expected to specify a threshold on the prediction scores. In practice, however, the optimal threshold depends on many factors, such as the different costs involved in the execution of the business process, as well as the scale of the predicition scores that the predictor outputs (i.e. whether the scores are well calibrated or not). Therefore, choosing the threshold manually can be a difficult task for the user. In this chapter, we propose an alarm-based prescriptive process monitoring system that enables building a cost-sensitive alarming mechanism that automatically identifies an optimal threshold on the prediction scores based on different types of costs related to a business process.

In Section 7.1, we give an overview of the works on cost-sensitive learning and prescriptive process monitoring. In Section 7.2, we introduce the alarm-based prescriptive process monitoring framework and outline a procedure for estimating the return on investment (ROI) of such an alarm system. We then propose a technique to empirically find the optimal alarming threshold based on a given cost configuration (Section 7.3) and evaluate the proposed mechanism empirically under different cost settings (Section 7.4).

7.1. Cost-sensitive learning and prescriptive process monitoring

Cost-sensitive learning seeks to find the optimal prediction in the case where different types of misclassifications have different costs and different types of correct classifications have different benefits [27]. There exist different approaches for achieving the cost-sensitivity of a classifier. Firstly, the underlying classifier can be modified to take the costs into account in the learning phase [50, 56]. Other approaches are more general in the sense that they do not require making inherent modifications to the classification algorithm itself. For instance, one can rebalance the proportion of positive and negative examples in the training set (stratification) [27], learn a meta-classifier after relabeling the training examples according to their estimated cost-minimizing class label [24], or use *empirical thresholding* [88] in order to empirically find the optimal threshold on the scores the classifier outputs. Furthermore, direct cost-sensitive decision making can be

used by training estimators for both probabilities of the outcome and for the cost related to a given instance [112].

In these settings, cost can refer to different types of negative consequences related to wrong predictions. Elkan [27] analyzes the basic notion of misclassification cost and defines conditions under which a misclassification cost matrix is reasonable. A broad range of cost variables in the context of inductive context learning is examined in [100]. This latter study introduces, among other types of costs, the notion of cost of intervention, which we include in our proposed cost model. Apart from that, the cost-sensitive learning approaches do not take into account the specific costs that arise in prescriptive process monitoring. In particular, works in the cost-sensitive learning area assume that the prediction must be made at a given timepoint, without the possibility of delaying the decision. In this work we propose a more general formulation of the cost model where the costs may depend also on the time when the decision is made, i.e. it is a possible to delay the decision and wait for more information about the instance to be classified.

To the best of our knowledge, works [18, 66, 91] are the only cost-sensitive early sequence classification methods trying to balance accuracy-related and earliness-related costs. However, these approaches assume that predicting a positive class early has the same effect on the cost function as predicting a negative class early, which is not the case in typical business process monitoring scenarios, where earliness matters only when an undesired outcome is predicted.

As discussed in Chapter 3, alarm-based settings in predictive process monitoring literature rely on fixed-threshold alarming mechanisms where the threshold needs to be provided by the user [30]. The works by Gröger et al. [38] and Krumeich et al. [49] propose architectures for prescriptive process monitoring systems, but neglect the two core elements of a cost-sensitive process monitoring system, i.e. the cost models and the earliness. Metzger et al. [62] study the effect of different prediction score thresholds on the adaptation cost (corresponds to the intervention cost in our framework) and the misclassification penalties. However, their work assumes that the alarms can only be generated when a given state of the process is reached. This assumption restricts their approach to scenarios where: (i) there is a process model that perfectly captures all executions of the process; (ii) raising an alarm early in the case does not yield a higher cost or reward than raising it later. Furthermore, none of these mentioned works propose a way of determining the alarming threshold automatically according to the costs related to the business process.

7.2. Alarm-based prescriptive process monitoring framework

In this section, we introduce a cost model for alarm-based prescriptive process monitoring and illustrate this model using three scenarios (Section 7.2.1). We then formalize the concept of alarm system (Section 7.2.2) and discuss conditions under which an alarm system has a positive return on investment (Section 7.2.3).

7.2.1. Concepts and cost model

An alarm-based prescriptive process monitoring system (*alarm system* for short) is a monitoring system that raises an alarm in relation to a running case of a business process, in order to indicate that the case is likely to lead to an undesired outcome. These alarms are handled by process workers who intervene by performing an action (e.g. calling a customer or blocking a credit card) in order to prevent or mitigate the undesired outcome. These actions may have a cost, which we call *cost of intervention*. Instead, if the case ends in a negative outcome, this leads to a cost called *cost of undesired outcome*.

As an example, consider a municipality that needs to collect city taxes. If the inhabitants do not pay their taxes on time, the municipality may run into cash flow issues. Accordingly, in case of an unpaid tax debt, the municipality may decide to end the tax collection process by outsourcing the debt collection to an external collection agency, for which it has to pay a recovery fee. In this case, sending the debt to a collection agency would be an undesired outcome and the recovery fee would constitute the cost of the undesired outcome. In light of their characteristics and past payment history, certain inhabitants may have a higher risk of missing the payment deadline. Therefore, sending a reminder letter to these high-risk inhabitants before the payment deadline is reached may increase the chances of receiving the payment on time. However, such an intervention comes with costs related to preparing the letter by an employee (proportional to the employee's hourly salary rate) and the postal costs for sending the letter.

In certain scenarios, the cost of an intervention may increase over time, acknowledging the importance of alarming as early as possible. For instance, in a railway maintenance process, if an alarm about a possible railway disruption is raised early, the problem could be solved with regular maintenance procedures. Conversely, if the alarm is raised when the need for maintenance has become urgent, the maintenance provider could be required to allocate more resources in order to solve the problem on time.

When an alarm is raised, there is a certain probability, but no certainty, that the case will reach an undesired outcome if no intervention is made. If the case does not conclude with an undesired outcome even without interventions, doing the intervention causes unnecessary costs (e.g. a company could lose customers and/or opportunities). The cost related to such unnecessary interventions is referred to as *cost of compensation*. For instance, financial institutions may block credit card payments when they suspect that a card was cloned. However, in some cases, it may happen that the suspicion was unfounded and that the payment was legitimate. If these cases become too frequent, the reputation of the financial institution could be hampered.

The purpose of alarming is to avoid an undesired outcome. However, in several scenarios, it is not possible to fully prevent the cost of the undesired outcome, while the intervention could still help to mitigate it. Based on this rationale, we

introduce the concept of *mitigation effectiveness* of an intervention, reflecting the proportion of the cost of an undesired outcome that can be avoided by carrying out the intervention. Oftentimes, the mitigation effectiveness decreases with time, i.e. the earlier the intervention takes place, the higher the proportion of costs that can be avoided. Consider, for instance, the process of paying unemployment benefits by a social security institution. In this case, the aim of an alarm system could be to notify the institution about citizens who might be receiving unentitled benefits. Since the benefits that have already been issued are unlikely to be recollected, the cost of the undesired outcome cannot be avoided completely. Therefore, it is important to raise the alarm as early as possible, in order to effectively mitigate the cost of the undesired outcome.

An alarm system is intended as a system where cases are continuously monitored. However, since continuous monitoring is impractical, we assume that cases are monitored after each executed event and, therefore, alarms can only be raised after an event has occurred. In the remainder, each case is identified by a completed trace σ that is (eventually) recorded in an event log. Definition 7.2.1 formalizes the costs defined above. Since costs may depend on the position in the case in which the alarm is raised and/or on other cases being executed, we define the costs as functions over the number of already executed events and over the entire set of cases under execution.

Definition 7.2.1 (Alarm-based Cost Model)**.** An *alarm-based cost model* is a tuple $(c_{in}, c_{out}, c_{com}, \mathit{eff})$ consisting of:

- a function $c_{in} : \mathbb{N} \times \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}_0^+$ modeling the *cost of intervention*: given a trace σ belonging to an event log L, $c_{in}(k, \sigma, L)$ indicates the cost of an intervention in σ when the intervention takes place after the k-th event;
- a function $c_{out} : \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}_0^+$ modeling the *cost of undesired outcome*;
- a function $c_{com} : \mathcal{S} \times 2^{\mathcal{S}} \to \mathbb{R}_0^+$ modeling the *cost of compensation*;
- a function $\mathit{eff} : \mathbb{N} \times \mathcal{S} \times 2^{\mathcal{S}} \to [0,1]$ modeling the *mitigation effectiveness* of an intervention: given a trace σ belonging to an event log L, $\mathit{eff}(k, \sigma, L)$ indicates the mitigation effectiveness of an intervention in σ when the intervention takes place after the k-th event.

To illustrate the versatility of the above cost model, we discuss three use cases for alarm systems and their corresponding cost model configurations. The first scenario, in Box 1, refers to the provision of unemployment benefits. The cost model for this scenario is based on several discussions with the stakeholders of a real social security institution [21]. The second scenario, in Box 2, refers to the detection of malicious credit card payments in a financial institution. Differently from the previous scenario, in this case, there is a risk of cost of compensation: due to the inconvenience caused by blocking their credit card, customers can switch to competitors. Box 3 refers to the process of predictive maintenance in railway services. This scenario is different from the previous ones because, in this case, the cost of an intervention increases over time.

Box 1 — Scenario "Unemployment Benefits"

In several countries, a social security institution is responsible for the execution of a number of employee-related insurances, such as unemployment benefits. When residents (hereafter customers) become unemployed, they are usually entitled to monthly monetary benefits for a certain period of time. These payments are stopped when the customer reports that he/she has found a new job. Unfortunately, several customers omit to inform the institution about finding a job and, thus, keep receiving benefits they are not entitled to. Those customers are expected to return the amount of benefits that they have received unlawfully. However, in practice, this rarely happens and the overpaid amount is lost to the institution. In light of the above, the social security institution would benefit from an alarm system that would inform about customers who are likely to be receiving unentitled benefits. Let $unt(\sigma)$ denote the amount of unentitled benefits received in a case corresponding to trace σ. Based on discussions with the stakeholders of a real social security institution, we designed the following cost model instantiation for such an alarm system.

Cost of intervention. For the intervention, an employee needs to check if the customer is indeed receiving unentitled benefits and, if so, fill in the forms for stopping the payments. Let S be the employee's average salary rate per time unit; let i_s and i_f denote the positions of the events in σ when the employee started working on the intervention and finished it, respectively. The cost of an intervention can be modeled as: $c_{out}(\sigma, L) = (\pi_{\mathcal{T}}(\sigma(i_f)) - \pi_{\mathcal{T}}(\sigma(i_s))) \cdot S$.

Cost of undesired outcome. The total amount of unentitled benefits that the customer would obtain without stopping the payments, i.e. $c_{out}(\sigma, L) = unt(\sigma)$.

Cost of compensation. The social security institution works in a situation of monopoly, which means that the customer cannot be lost because of moving to a competitor, i.e. there is no cost of compensation: $c_{com}(\sigma, L) = 0$.

Mitigation effectiveness. The proportion of unentitled benefits that will not be paid thanks to the intervention: $eff(k, \sigma, L) = \dfrac{unt(\sigma) - unt(hd^k(\sigma))}{unt(\sigma)}$. Note that this cost function is not employed if there is no undesired outcome (i.e. if $unt(\sigma) = 0$).

Box 2 — Scenario "Financial Institution"

Suppose that the customers of a financial institution use their credit cards to make payments online. Each such transaction is associated with a risk that the transaction is made through a cloned card. In this scenario, an alarm system is intended to determine whether the credit card needs to be blocked due to a high risk of being cloned. However, in case the credit card is not malicious, blocking the card would cause discomfort to the customer who may consequently opt to switch to a different financial institution. Let σ be the trace of credit card transactions for a customer and $value(\sigma)$ the total amount of money related to malicious transactions in σ, the following is a possible cost model instantiation for this scenario.

Cost of intervention. The card is automatically blocked by the system and, therefore, the intervention costs are limited to POST_COST, i.e. to the costs for sending a new credit card to the customer by mail: $c_{in}(k, \sigma, L) = \text{POST\_COST}$.

Cost of undesired outcome. The total amount of money related to malicious transactions that the bank would need to reimburse to the legitimate customer: $c_{out}(\sigma, L) = value(\sigma)$.

Cost of compensation. Denoting the asset value of a customer (consisting of the amount of the investment portfolio, the account balance, etc.) with $asset(\sigma)$ and supposing that a fraction p (i.e. $p \in [0,1]$) of the customers would switch to a different institution, the cost of compensation can be estimated as the value of the lost asset (the customer), multiplied by p: $c_{com} = p \cdot asset(\sigma)$.

Mitigation effectiveness. The proportion of the total amount of money related to malicious transactions that does not need to be reimbursed by blocking the credit card after that k events have been executed: $eff(k, \sigma, L) = \dfrac{value(\sigma) - value(hd^k(\sigma))}{value(\sigma)}$.

Box 3 — Scenario "Railway Maintenance"

In a process for railway maintenance, an alarm should be raised when there is a risk that the railway may break down within a relatively short time range. Railway breakdowns can cause severe disruptions in the train transportation (i.e. trains could be canceled or delayed), thereby causing losses of reimbursing tickets to travelers.

Cost of intervention. The cost of an intervention increases with time because the more urgent the disruption, the more resources need to be allocated for handling it. We assume that the cost is at its minimum m at the beginning of a trace σ and grows exponentially with time: $c_{in}(k,\sigma,L) = m \cdot \beta \exp(\pi_{\mathcal{T}}(\sigma(k)))$ for some $\beta > 0$.

Cost of undesired outcome. Let P be the average total price of tickets sold per time unit; let i_d and i_m be the positions of the events in σ when the disruption took place and was resolved, respectively. The cost of the undesired outcome can be calculated as P multiplied by the length of the timeframe when the railway service was disrupted: $c_{out}(\sigma,L) = (\pi_{\mathcal{T}}(\sigma(i_m)) - \pi_{\mathcal{T}}(\sigma(i_d))) \cdot P$.

Cost of compensation. Assuming that performing (unnecessary) maintenance actions does not cause inconveniences to the customers, no cost of compensation is present: $c_{com}(\sigma,L) = 0$.

Mitigation effectiveness. A timely intervention fully avoids the undesired outcome: $\mathit{eff}(k,\sigma,L) = 1$ for any $k \in [1,|\sigma|]$.

Table 23: Cost of a case σ based on its outcome and whether an alarm was raised.

	undesired outcome	desired outcome
alarm raised	$c_{in}(k,\sigma,L) + (1 - \mathit{eff}(k,\sigma,L))c_{out}(\sigma,L)$	$c_{in}(k,\sigma,L) + c_{com}(\sigma,L)$
alarm not raised	$c_{out}(\sigma,L)$	0

7.2.2. Alarm-based prescriptive process monitoring system

An alarm-based prescriptive process monitoring system aims at alarming if an ongoing case is likely to end up with an undesired outcome. The case outcomes are represented by a labeling function $\mathit{out} : \mathcal{S} \to \{0,1\}$, s.t. given a case identified by a trace σ, $\mathit{out}(\sigma) = 1$ if the case has an undesired outcome, and $\mathit{out}(\sigma) = 0$ otherwise. At runtime, the outcome of a running case is not yet known and needs to be estimated using a predictive model trained on past executions that are recorded in an event log $L \subseteq \mathcal{S}$. A predictive model is a function $\widehat{\mathit{out}} : \mathcal{S}_* \to [0,1]$ returning the probability $\widehat{\mathit{out}}(\sigma')$ that the outcome of a case that starts with prefix σ' is undesired. We can define an alarm system as a function that returns 1 or 0 depending on whether an alarm is raised based on the predicted outcome or not.

Definition 7.2.2 (Alarm-Based Prescriptive Process Monitoring System)**.** Given an event log $L \subseteq \mathcal{S}$, let $\widehat{\mathit{out}}_L$ be a predictive model built from L. An *alarm-based prescriptive process monitoring system* is a function $\mathit{alarm}_{\widehat{\mathit{out}}_L} : \mathcal{S}_* \to \{0,1\}$. Given a running case identified by a trace σ and with current prefix σ', $\mathit{alarm}_{\widehat{\mathit{out}}_L}(\sigma')$ returns 1, if an alarm is raised based on the predicted outcome $\widehat{\mathit{out}}_L(\sigma')$, or 0, otherwise.

For simplicity, we omit the subscript L from $\widehat{\mathit{out}}_L$ and omit $\widehat{\mathit{out}}_L$ from $\mathit{alarm}_{\widehat{\mathit{out}}_L}$ when it is clear from the context. An alarm system can raise an alarm at most once per case, since we assume that already the first alarm triggers an intervention by the stakeholders.

The purpose of an alarm system is to minimize the cost of executing a case.

Table 23 summarizes how the cost of a case is determined based on a cost model (cf. Def. 7.2.1), on the case outcome, and on whether an alarm was raised or not.

Definition 7.2.3 (Cost of Case Execution)**.** Let $cm = (c_{in}, c_{out}, c_{com}, \mathit{eff})$ be an alarm-based cost model. Let $\mathit{out} : \mathcal{S} \to \{0,1\}$ be a labeling function. Let $\mathit{alarm} : \mathcal{S}_* \to \{0,1\}$ be an alarm-based prescriptive process monitoring system. Let $L \subseteq \mathcal{S}$ be the entire set of completed traces. Let $\sigma \in L$ be a completed trace. Let $\mathcal{I}(\sigma, \mathit{alarm})$ be the first index of the event in σ when the alarm was raised or zero if no alarm was raised:

$$\mathcal{I}(\sigma, \mathit{alarm}) = \begin{cases} 0 & \text{if } \nexists i \in \{1, \dots, |\sigma|-1\} : \mathit{alarm}(hd^i(\sigma)), \\ \min\{i \in \{1, \dots, |\sigma|-1\} : \mathit{alarm}(hd^i(\sigma))\} & \mathit{otherwise} \end{cases}$$

The *cost of execution of a case identified by a trace* σ supported by the alarm system is:

$$\mathit{cost}(\sigma, L, cm, \mathit{alarm}) = \begin{cases} c_{in}(\mathcal{I}(\sigma, \mathit{alarm}), \sigma, L) + (1 - \mathit{eff}(\mathcal{I}(\sigma, \mathit{alarm}), \sigma, L)) \cdot c_{out}(\sigma, L) & \mathit{out}(\sigma) \wedge \mathcal{I}(\sigma, \mathit{alarm}) > 0, \\ c_{in}(\mathcal{I}(\sigma, \mathit{alarm}), \sigma, L) + c_{com}(\sigma, L) & \neg \mathit{out}(\sigma) \wedge \mathcal{I}(\sigma, \mathit{alarm}) > 0, \\ c_{out}(\sigma, L) & \mathit{out}(\sigma) \wedge \mathcal{I}(\sigma, \mathit{alarm}) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Section 7.3 illustrates how an alarm-based prescriptive process monitoring system can be designed aiming at the minimization of the case execution costs (according to Def. 7.2.3).

7.2.3. Return on investment analysis

In this section, we provide an analysis and guidelines that suggest when it is valuable to invest in developing an alarm system, namely, when the return on investment (ROI) is positive. To this end, we compare the situation where the execution of a business process is supported by an alarm system with the *as-is* situation, where the business process is executed without this support. For this analysis, we consider a set of traces recorded in an event log L, where no interventions were done, and a cost model $cm = (c_{in}, c_{out}, c_{com}, \mathit{eff})$.

The *as-is* situation implies that no interventions are done in any of the cases that lead to an undesired outcome, yielding a cost $c_{out}(\sigma)$ for each of trace $\sigma \in L$ where $\mathit{out}(\sigma) = 1$. When applied to the entire log L, the *as-is* cost is:

$$\mathit{cost}_{\text{as-is}}(L) = \sum_{\sigma \in L \text{ s.t. } \mathit{out}(\sigma)} c_{out}(\sigma).$$

Instead, when an alarm system *alarm* is in effect, the cost for each trace can be calculated according to Def. 7.2.3:

$$\mathit{cost}_{\mathit{alarm}}(L) = \sum_{\sigma \in L} \mathit{cost}(\sigma, L, cm, \mathit{alarm}).$$

With this setting, the ROI of the system *alarm* is:

$$\mathit{ROI}(L, cm, \mathit{alarm}) = \mathit{cost}_{\text{as-is}}(L) - \mathit{cost}_{\mathit{alarm}}(L),$$

which must be positive to make deploying the system worthwhile.

The question that remains is: *how does the ROI depend on the cost model and the alarm system?* For the sake of simplicity, we assume in this analysis that every component of the cost model is constant, i.e. does not depend on the position in the trace where the alarm is raised. Furthermore, the initial investment costs are not considered because we assume the system to be fully operational already for a sufficiently long time, so that the the initial costs have been amortized. The above assumptions yield the following simplified cost of a case execution:

$$cost(\sigma, L, cm, alarm) = \begin{cases} c_{in} + (1 - eff)c_{out} & out(\sigma) \wedge \mathcal{I}(\sigma, alarm) > 0, \\ c_{in} + c_{com} & \neg out(\sigma) \wedge \mathcal{I}(\sigma, alarm) > 0, \\ c_{out} & out(\sigma) \wedge \mathcal{I}(\sigma, alarm) = 0, \\ 0 & \text{otherwise} \end{cases}$$

where c_{in}, c_{out}, c_{com}, and eff are constants. In order for the ROI to be positive, it is necessary that $cost_{as\text{-}is}(L) > cost_{alarm}(L)$, that is:

$$|L_{und}| \cdot c_{out} > |L_{und\&al}|(c_{in} + (1 - eff)c_{out}) + |L_{des\&al}|(c_{in} + c_{com}) + |L_{und\&nal}| \cdot c_{out}$$

where $L_{und\&al}, L_{des\&al}, L_{und\&nal}$ respectively consist of the traces in L related to the cases with an undesired outcome that would be alarmed, with a desired outcome that would still be alarmed, with an undesired outcome that would not be alarmed; also, $L_{und} = L_{und\&al} \cup L_{und\&nal}$. After simplification:

$$|L_{und\&al}|(eff\, c_{out} - c_{in}) > |L_{des\&al}|(c_{in} + c_{com}). \tag{7.1}$$

Because the right-hand side of Eq. 7.1 is non-negative, it follows as a corollary that $eff\, c_{out} > c_{in}$ is a necessary condition for yielding a positive ROI. In other words, it must be possible to avoid a cost that is higher than the cost of doing the intervention. This provides a validation of our framework: it complies with the *reasonableness condition* in the cost-sensitive learning literature [27], which states that the cost of labeling an example incorrectly should always be greater than the cost of labeling it correctly.

Eq. 7.1 also illustrates that the policy of always alarming does not yield a positive ROI unless the number of cases with undesired outcome and the cost of the undesired outcome are sufficiently high. When the number of cases with an undesired outcome is small (e.g. the unemployment benefits and the financial institution scenarios described in Boxes 1 and 2) and at the same time the cost of this undesired outcome is low, then the left-hand side of Eq. 7.1 is negligible, thus leading to condition $c_{in} + c_{com} < 0$, which can never hold.

So far we have assumed, for the sake of simplicity, that costs and mitigation effectiveness are constant, similarly to traditional cost-sensitive learning. However, the novelty of our formulation lays in the fact that costs are functions that depend on the time when an intervention is made. As a result, the reasonableness of the cost matrix would not be fixed, but potentially changes over time. Still,

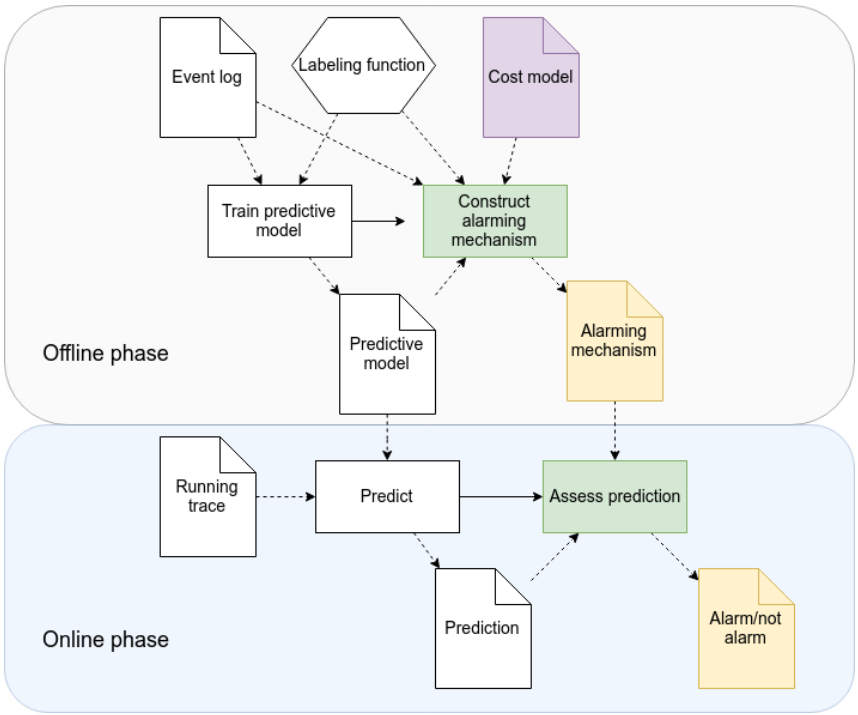


Figure 35: Alarm-based prescriptive process monitoring.

variable costs do not invalidate the ROI analysis. In fact, in order for the ROI to be positive, it is sufficient that the cost model is reasonable for a certain time period; otherwise, the alarm system would never raise alarms because of the cost model. Clearly, the longer the reasonable-cost period, the higher the chances of obtaining a positive ROI.

7.3. Alarming mechanisms and empirical thresholding

An alarm system as described above needs two components to minimize the costs of future cases: (1) a predictive model $\widehat{out}_L : \mathcal{S}_* \rightarrow [0,1]$ that estimates the probability of an undesired outcome for a running trace based on some historical observations L, and (2) an alarming mechanism that, for a given incomplete case, decides whether or not to raise an alarm based on the prediction made by $\widehat{out}_L$. Specifically, an alarming mechanism is a function $agent : [0,1] \rightarrow \{0,1\}$ that operates on the estimated probability of an undesired outcome, where the value 1 represents the decision to raise an alarm. Together, the two components form an *alarm system*, $alarm(hd^k(\sigma)) = agent(\widehat{out}_L(hd^k(\sigma)))$, which makes the decision on whether or not to raise an alarm based on the observed k events of trace σ. The offline and online workflows of an alarm system consisting of a predictive model and an alarming mechanism are illustrated in Figure 35.

The first component can be implemented using any predictive process moni-

toring technique and a training set L_{train}, yielding a predictive model $\widehat{out}_{L_{train}}$. It is easy to see that the decision on whether or not to raise an alarm should be dependent not only on $\widehat{out}_{L_{train}}(hd^k(\sigma))$, but also on the configuration of c_{in}, c_{out}, c_{com}, and *eff*. When c_{in} and c_{com} are very low compared to c_{out}, it might be beneficial to use a lower threshold for the estimated probability $\widehat{out}_{L_{train}}(hd^k(\sigma))$, while one would want to be more certain that the undesired outcome will happen when c_{in} or c_{com} is high.

We propose to implement the second component, *agent*, as an *alarming threshold*, i.e. a mechanism that alarms when the estimated probability of an undesired outcome is at least τ. We define function $alarm_\tau(hd^k(\sigma))$ to be the alarming function that uses the alarming mechanism $agent_\tau(\widehat{out}_{L_{train}}(hd^k(\sigma))) = \widehat{out}_{L_{train}}(hd^k(\sigma)) \geq \tau$. We aim at finding the optimal value $\overline{\tau}$ of the alarming threshold that minimizes the cost on a log L_{thres} consisting of historical observations such that $L_{thres} \cap L_{train} = \emptyset$ with respect to a given predictive model $\widehat{out}_{L_{train}}$ and cost model *cm*. The total cost of an alarming mechanism *alarm* on a log L is defined as $cost(L, cm, alarm) = \Sigma_{\sigma \in L} cost(\sigma, L, cm, alarm)$. Using this definition, we define $\overline{\tau} = \arg\min_{\tau \in [0,1]} cost(L_{thres}, cm, alarm_\tau)$. Finding the optimal threshold $\overline{\tau}$ with respect to a specified cost model can be done empirically (i.e. empirical thresholding) using a separate thresholding set L_{thres} and any hyperparameter optimization technique. The resulting approach can be considered to be a form of cost-sensitive learning, since the value $\overline{\tau}$ depends on how the cost model *cm* is specified.

Note that as an alternative to a single global alarming threshold $\overline{\tau}$ it is possible to optimize a separate threshold $\overline{\tau_k}$ for each prefix length k. Our initial experiments showed that a single global threshold $\overline{\tau}$ optimized on L_{thres} tends to outperform separate prefix-length-dependent thresholds $\overline{\tau_k}$ optimized on L_{thres}, therefore we propose using a single threshold.

After creating the fully functional alarm system by training a classifier on L_{train} and optimizing the alarming threshold on L_{thres} for the given cost model *cm*, the obtained alarming function *alarm* can be applied to the continuous stream of events coming from the executions of a business process, thereby reducing the processing costs of the running cases.

7.4. Evaluation

In this section, we describe the experimental setup for evaluating the proposed framework and the results of the evaluation. We address the following research questions:

RQ1 (Empirical thresholding) Can empirical thresholding find thresholds that consistently lead to a reduction in the average processing cost for different cost model configurations?

RQ2 (Mitigation effectiveness) Does the alarm system consistently yield a benefit over different values of the mitigation effectiveness?

Table 24: Statistics of the *unemployment* dataset.

dataset	# traces	min length	med length	max length	trunc length	# variants (after trunc)	pos class ratio	# event classes	# static attr-s	# dynamic attr-s	# static cat levels	# dynamic cat levels
unemployment	34627	1	21.0	1177	40	29689	0.2	21	6	13	218	27

RQ3 (Cost of compensation) Does the alarm system consistently yield a benefit over different values of the cost of compensation?

7.4.1. Approaches and baselines

We use XGBoost single classifier with aggregation encoding as the implementation of $\widehat{out}_{L_{train}}$, since it has shown to perform reliably over different datasets (see Chapters 4–6). For event logs containing unstructured data, we use the BoNG model to obtain numeric features from text. We apply the TPE optimization procedure for the alarming mechanism to find the optimal threshold $\overline{\tau}$.

We use several fixed thresholds as baselines. First, we compare with the *as-is* situation in which alarms are never raised. Secondly, we compare with the baseline $\tau = 0$, allowing us to compare with the situation where alarms are always raised directly at the start of a case. Finally, we compare with $\tau = 0.5$ enabling the comparison with the cost-insensitive scenario that simply alarms when an undesired outcome is expected.

7.4.2. Datasets

As evaluation datasets, we use the same event logs and preprocessing as introduced in Chapters 4 and 5. Additionally, we use an event log *unemployment* corresponding to the *Unemployment Benefits* scenario (Box 1). In this log, the undesired outcome occurs when a resident will receive more unemployment benefits than entitled, causing the need for a reclamation. Due to privacy constraints, this event log is not publicly available.

We use the same preprocessing on the *unemployment* dataset as described in Section 4.1. The statistics of this dataset are reported in Table 24.

7.4.3. Experimental setup

We start with the same experimental setup as in Chapters 4 and 5, i.e. we split each dataset into 80% training (precisely, $L_{train} \cup L_{thres}$) and 20% of testing (L_{test}) traces using a temporal split. For event logs consisting of only structured data the hyperparameters of the classifiers are optimized via 3-fold cross-validation using TPE on the training set. For event logs containing unstructured data, we apply the setup described in Chapter 5, i.e. the model selection is done on a single validation set. Then, in all event logs we randomly divide the traces in the training set into 80% (i.e. 64% of the total) for building the final model (L_{train}) and 20% (i.e. 16% of the total) for finding the optimal threshold (L_{thres}).

It is common in cost-sensitive learning to apply calibration techniques to the

Table 25: Cost model configurations.

	$c_{out}(\sigma,L)$	$c_{in}(k,\sigma,L)$	$c_{com}(\sigma,L)$	$eff(k,\sigma,L)$
RQ1	$\{1,2,3,5,10,20\}$	1	0	$1-k/\|\sigma\|$
RQ2	$\{1,2,3,5,10,20\}$	1	0	$\{0,0.1,0.2,...,1\}$
RQ3	$\{1,2,3,5,10,20\}$	1	$\{0,1/20,1/10,1/5,1/2,1,2,5,10,20\}$	$1-k/\|\sigma\|$

resulting classifier in order to obtain accurate probability estimates and, therefore, more accurate estimates of the expected cost [112]. However, in case of empirical thresholding for a single classifier, it is sufficient that the prediction scores are reasonably ordered, but not necessarily well calibrated. In particular, the only difference could occur in the exact value of the threshold, but not in its relative position among the returned scores. Therefore, we omit the calibration step in the given evaluation.

Table 25 shows the configurations of the cost model that we explore in the evaluation. To answer RQ1 (Can empirical thresholding find thresholds that consistently lead to a reduction in the average processing cost for different cost model configurations?), we vary the ratio between $c_{out}(\sigma,L)$ and $c_{in}(k,\sigma,L)$ (keeping $c_{com}(\sigma,L)$ and $eff(k,\sigma,L)$ unchanged). To answer RQ2 (Does the alarm system consistently yield a benefit over different values of the mitigation effectiveness?), we vary both $eff(k,\sigma,L)$ and the ratio between $c_{out}(\sigma,L)$ and $c_{in}(k,\sigma,L)$. To answer RQ3 (Does the alarm system consistently yield a benefit over different values of the cost of compensation?), we vary two ratios: 1) between $c_{out}(\sigma,L)$ and $c_{in}(k,\sigma,L)$ and 2) between $c_{in}(k,\sigma,L)$ and $c_{com}(\sigma,L)$.

As our main evaluation metric, we use the average processing cost per case in L_{test}, and aim at minimizing this cost. Additionally, we measure the *benefit* of the alarm system, i.e. the reduction in the average processing cost of a case when using the alarm system compared to the average processing cost when not using it. Also, we report the prediction accuracy in terms of F-score and the prediction earliness, calculated as $earliness = 1 - \frac{\sum_{\sigma \in L_{und\&al}} k/|\sigma|}{|L_{und\&al}|}$, where $L_{und\&al}$ is the set of evaluation traces where an alarm was raised and that were labeled with an undesired outcome (i.e. the true positives) and k is the length of the prefix when the alarm was raised. In case of F-score and earliness, higher values reflect better performance of the method.

Experiments were run using Python 3.5 and the scikit-learn library on a single core of an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz with 64GB of RAM.

7.4.4. Results

Figure 36 shows the average cost per case when increasing the ratio of $c_{out}(\sigma,L)$ and $c_{in}(k,\sigma,L)$ from left to right. When the ratio between these two costs is balanced (i.e. 1:1), the minimal cost is obtained by never alarming. This is in agreement with the ROI analysis, where we found $eff\,c_{out} > c_{in}$ to be a necessary

condition for having an advantage from an alarm system. When $c_{out} \gg c_{in}$ the best strategy is to always alarm. When c_{out} is slightly higher than c_{in} the best strategy is to sometimes alarm based on $\widehat{out}$. We find that the optimized $\overline{\tau}$ almost always outperforms the baselines, with the exception of *insurance_2* and a few specific ratios on other datasets. In particular, the optimized threshold tends to sometimes alarm on the ratio 1:1, where never alarming would be the optimal strategy (see, for instance, *production*, *bpic2015_4*, *bpic2017_3*).

Figure 37 provides more insights about the (optimized) alarming mechanism by plotting the resulting F-score and earliness. We can see that as the ratio between $c_{out}(\sigma, L)$ and $c_{in}(k, \sigma, L)$ increases, the alarming mechanism starts to raise alarms earlier in the process, while resulting in lower prediction accuracy. Note that when the alarms are never raised (mostly in case of the 1:1 ratio), both F-score and earliness are undefined.

In Figure 38, the average cost per case is plotted against different (fixed) thresholds. The optimized threshold is marked with a red cross and each line represents one particular cost ratio. We observe that, while the optimized threshold generally obtains minimal costs, there sometimes exist multiple optimal thresholds for a given cost model configuration. For instance, in the case of the 2:1 ratio in *bpic2017_3*, all thresholds are cost-wise almost equivalent. To answer RQ1 (Can empirical thresholding find thresholds that consistently lead to a reduction in the average processing cost for different cost model configurations?), we conclude that the empirical thresholding approach consistently finds a threshold that yields the lowest cost in a given event log and cost model configuration.

Figure 39 shows the benefit of having an alarm system compared to not having it for different (constant) mitigation effectiveness values. As expected, the benefit increases both with higher $eff(k, \sigma, L)$ and with higher $c_{out}(\sigma, L) : c_{in}(k, \sigma, L)$ ratio. Necessary conditions for receiving any benefit are $c_{out}(\sigma, L) > c_{in}(k, \sigma, L)$ and $eff(k, \sigma, L) > 0$. In most of the datasets the alarm system yields a benefit always when $eff(k, \sigma, L) > 0.5$ and $c_{out}(\sigma, L) > c_{in}(k, \sigma, L)$. In datasets with a larger number of cases with undesired outcome (i.e. with balanced class proportions), the potential benefit is much higher than in datasets with imbalanced class proportions (see the high benefit values in the top right corners of the plots for, e.g. *bpic2011_1*, *bpic2011_2*, *bpic2012_1*, *bpic2017_3*, *github*, *traffic*). At the same time, in these balanced datasets there is a risk that the alarm system yields a cost rather than a benefit. For instance, in the *production* dataset, if $c_{out}(\sigma, L) : c_{in}(k, \sigma, L) < 5$, using an alarm system is more costly than not using one. Similarly, in *bpic2011_4* if $eff(k, \sigma, L) < 0.3$ or when $c_{out}(\sigma, L) : c_{in}(k, \sigma, L) < 3$, the alarm system yields a cost instead of a benefit.

In slightly more imbalanced datasets (such as *bpic2012_2*, *bpic2017_2*, *unemployment*) the value of the benefit is small even in case both $eff(k, \sigma, L)$ and $c_{out}(\sigma, L) : c_{in}(k, \sigma, L)$ are high. The reason for this is that in these datasets the number of cases with undesired outcome is small and, therefore, the number of cases where c_{out} can be prevented by alarming is lower. At the same time, the risk

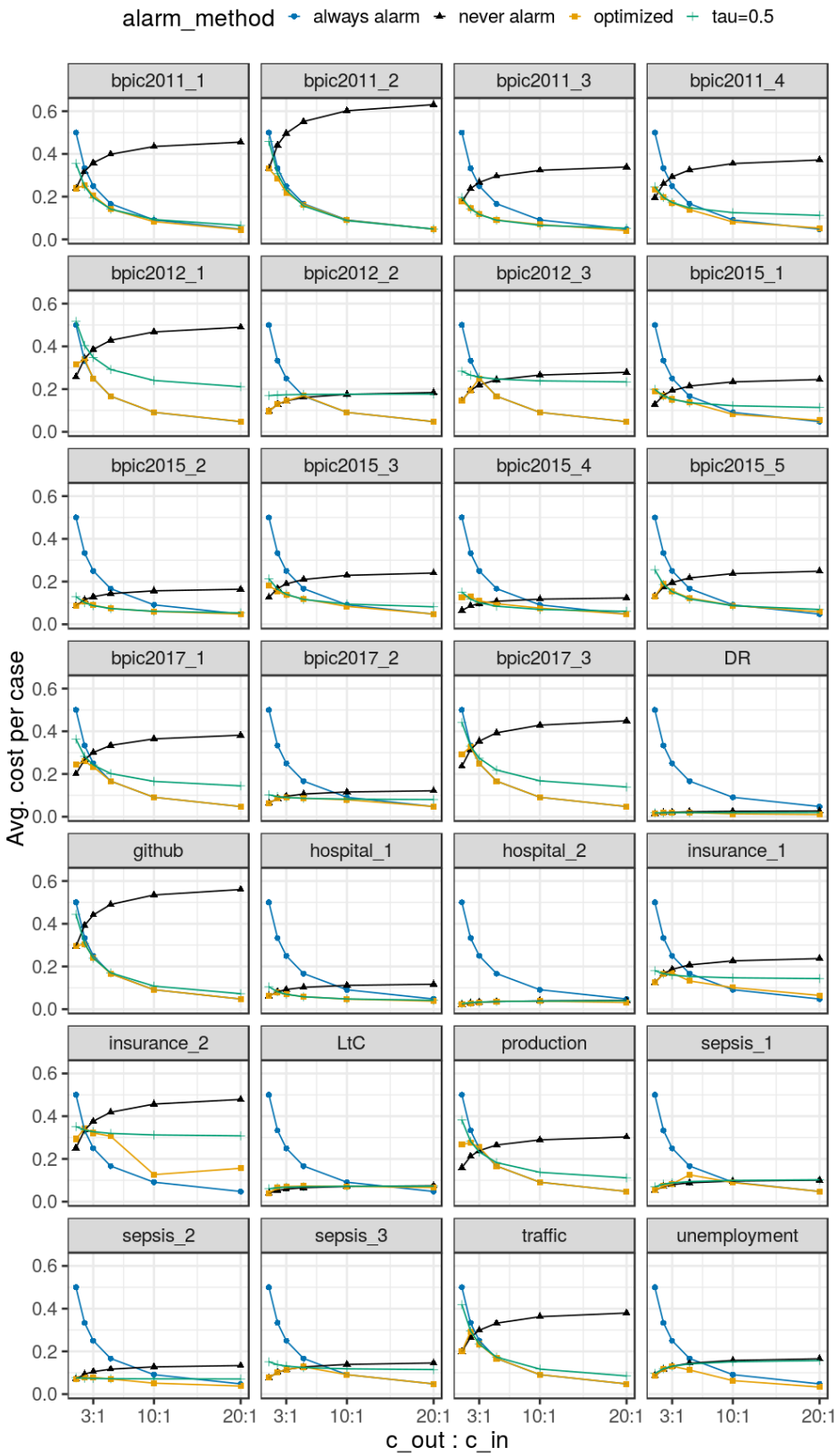


Figure 36: Cost over different ratios of $c_{out}(\sigma, L)$ and $c_{in}(k, \sigma, L)$.

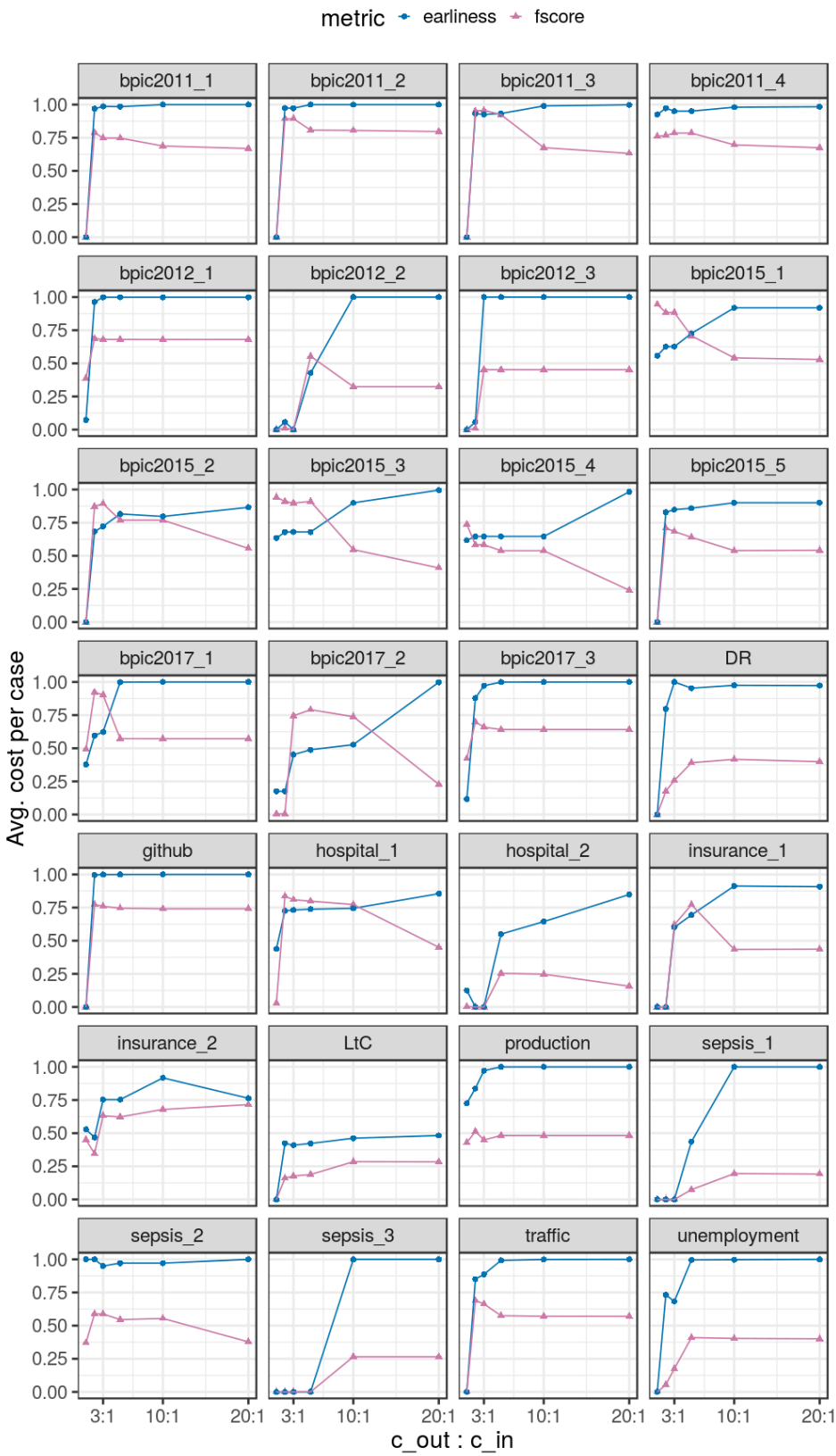


Figure 37: F-score and earliness over different ratios of $c_{out}(\sigma, L)$ and $c_{in}(k, \sigma, L)$.

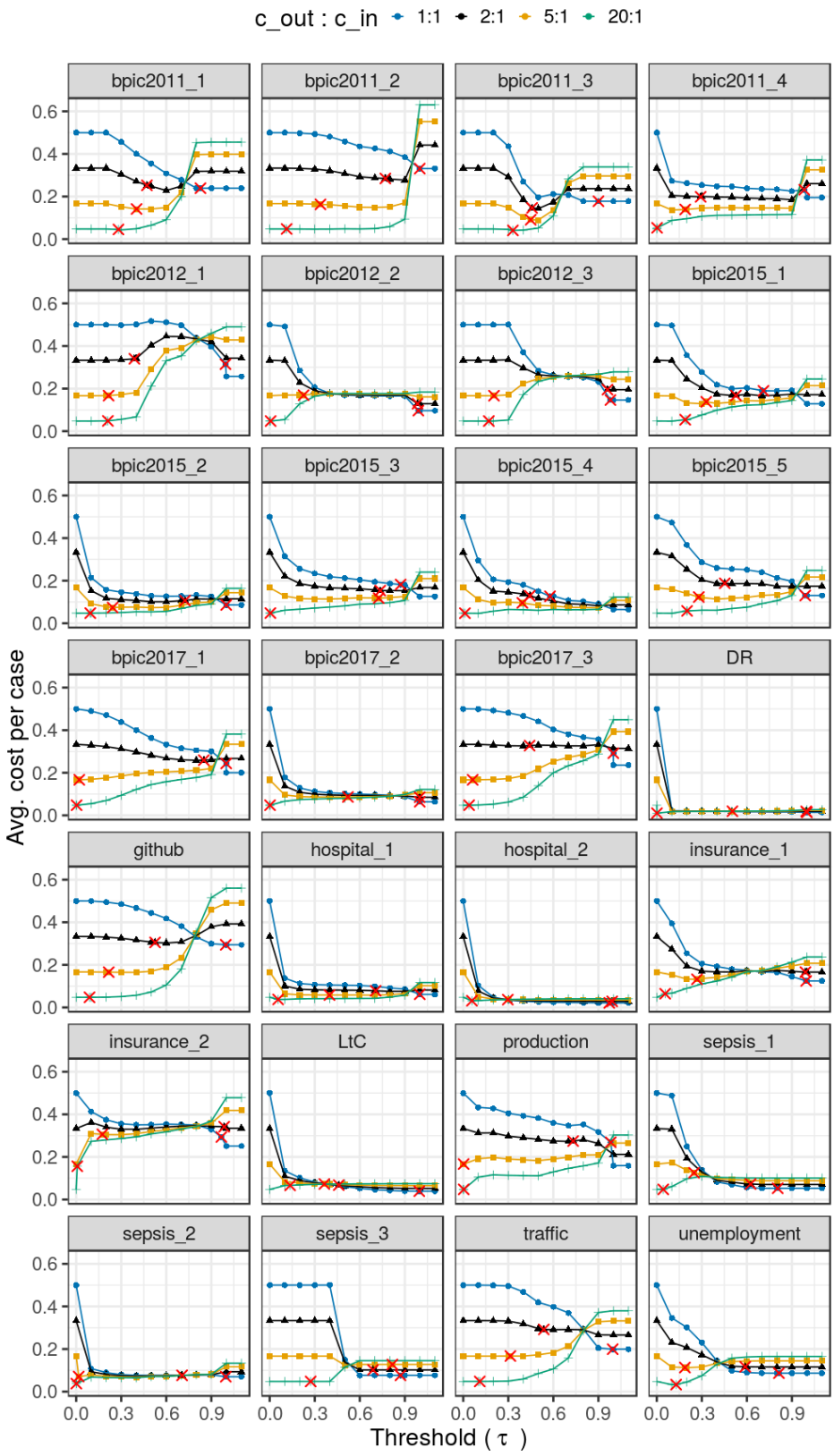


Figure 38: Cost over different thresholds ($\overline{\tau}$ is marked with a red cross).

of achieving a cost instead of a benefit is low — in smaller $eff(k,\sigma,L)$ values and $c_{out}(\sigma,L) : c_{in}(k,\sigma,L)$ ratios, the alarm system simply does not have an effect. In the case of very imbalanced datasets (such as *DR*, *hospital_2*, and *LtC*), the benefits (and costs) are almost non-existent.

We conducted analogous experiments with linear effectiveness decay, varying the maximum possible effectiveness (at the start of the case), which confirmed that the observed trends remain the same (see Figure 53 in Appendix). As answer to RQ2 (Does the alarm system consistently yield a benefit over different values of the mitigation effectiveness?), we have empirically confirmed our theoretical finding (Section 7.2.3) that $eff\,c_{out} > c_{in}$ is a necessary condition to obtain a benefit from using an alarm system, and have shown that a benefit is in practice also obtained under this condition when an optimized alarming threshold is used.

Similarly, the benefit of the alarm system is plotted in Figure 40 across different ratios of $c_{out}(\sigma,L) : c_{in}(k,\sigma,L)$ and $c_{in}(k,\sigma,L) : c_{com}(\sigma,L)$. We observe that when $c_{com}(\sigma,L)$ is high, the benefit decreases due to false alarms. For balanced datasets, such as *bpic2017_3*, a benefit is obtained almost always, except when $c_{out}(\sigma,L) : c_{in}(k,\sigma,L)$ is low (e.g. 2:1). For slightly more imbalanced datasets, such as *unemployment*, a benefit is obtained with fewer cost model configurations, e.g. when $c_{out}(\sigma,L) : c_{in}(k,\sigma,L) = 5:1$ and $c_{com}(\sigma,L)$ is smaller than $c_{in}(k,\sigma,L)$. For very imbalanced datasets (*DR*, *hospital_2*, and *LtC*), the alarm system has almost no effect. We conducted analogous experiments with linearly increasing cost of intervention, varying the maximum possible cost (at the end of the case), which confirmed that the trends described above remain the same (see Figure 54 in Appendix). To answer RQ3 (Does the alarm system consistently yield a benefit over different values of the cost of compensation?), we have empirically confirmed that the alarm system achieves a benefit as discussed in Section 7.2.3 in case the cost of the undesired outcome is sufficiently higher than the cost of the intervention and/or the cost of the intervention is sufficiently higher than the cost of compensation.

7.5. Summary

In this chapter, we outlined an alarm-based prescriptive process monitoring framework that extends existing predictive process monitoring approaches with the concepts of alarms, interventions, compensations, and mitigation effects. The framework incorporates a cost model to analyze the tradeoffs between the cost of intervention, the benefit of mitigating or preventing undesired outcomes, and the cost of compensating for unnecessary interventions induced by false alarms. The cost model allows one to estimate the benefits of deploying a prescriptive process monitoring system for the purposes of return on investment analysis. Additionally, the framework incorporates a technique to optimize the alarm generation mechanism with respect to a given configuration of the cost model and a given event log. An empirical evaluation on real-life logs showed the benefits of applying this

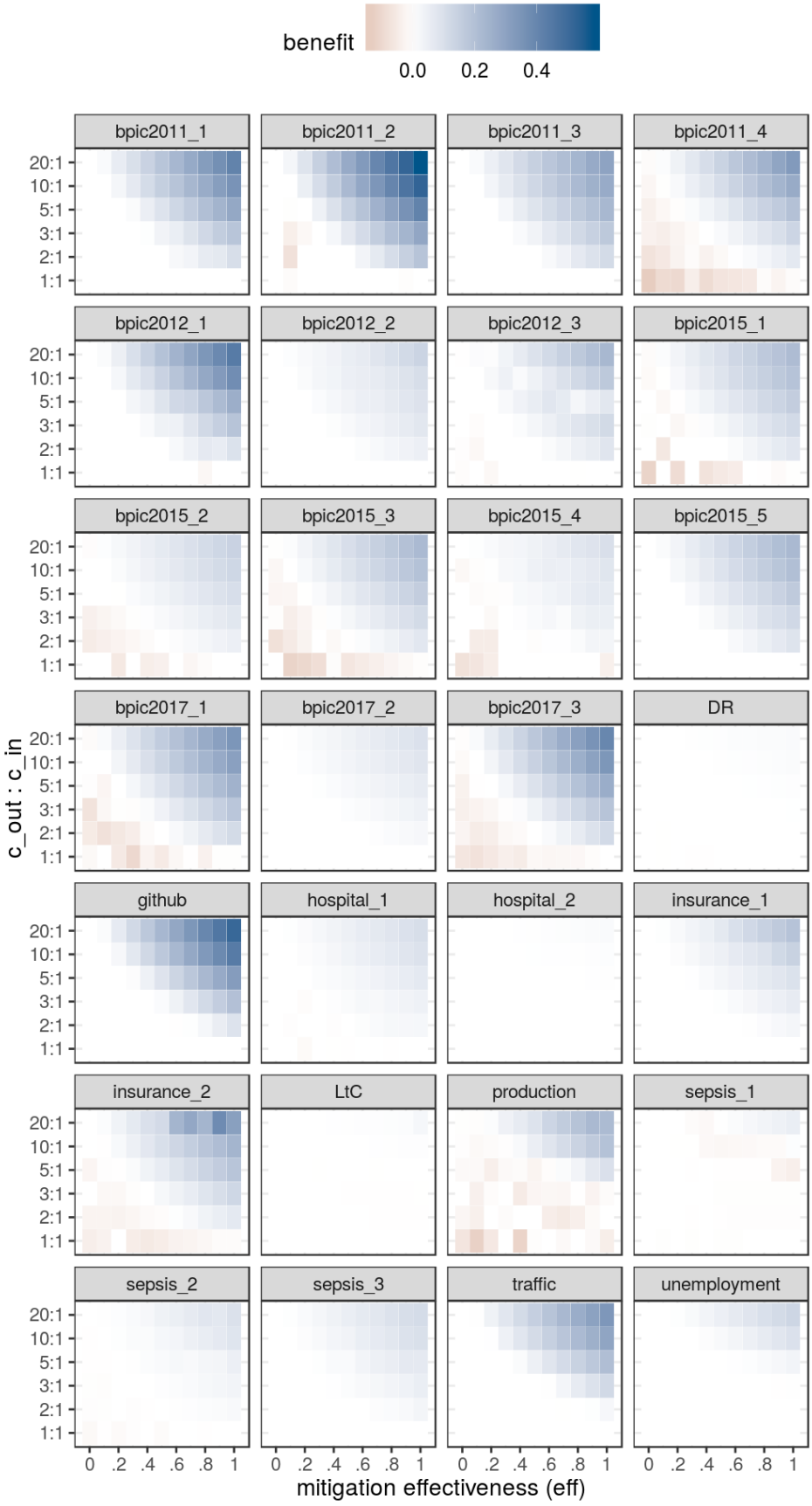


Figure 39: Benefit of the alarm system, varying $eff(k, \sigma, L)$.

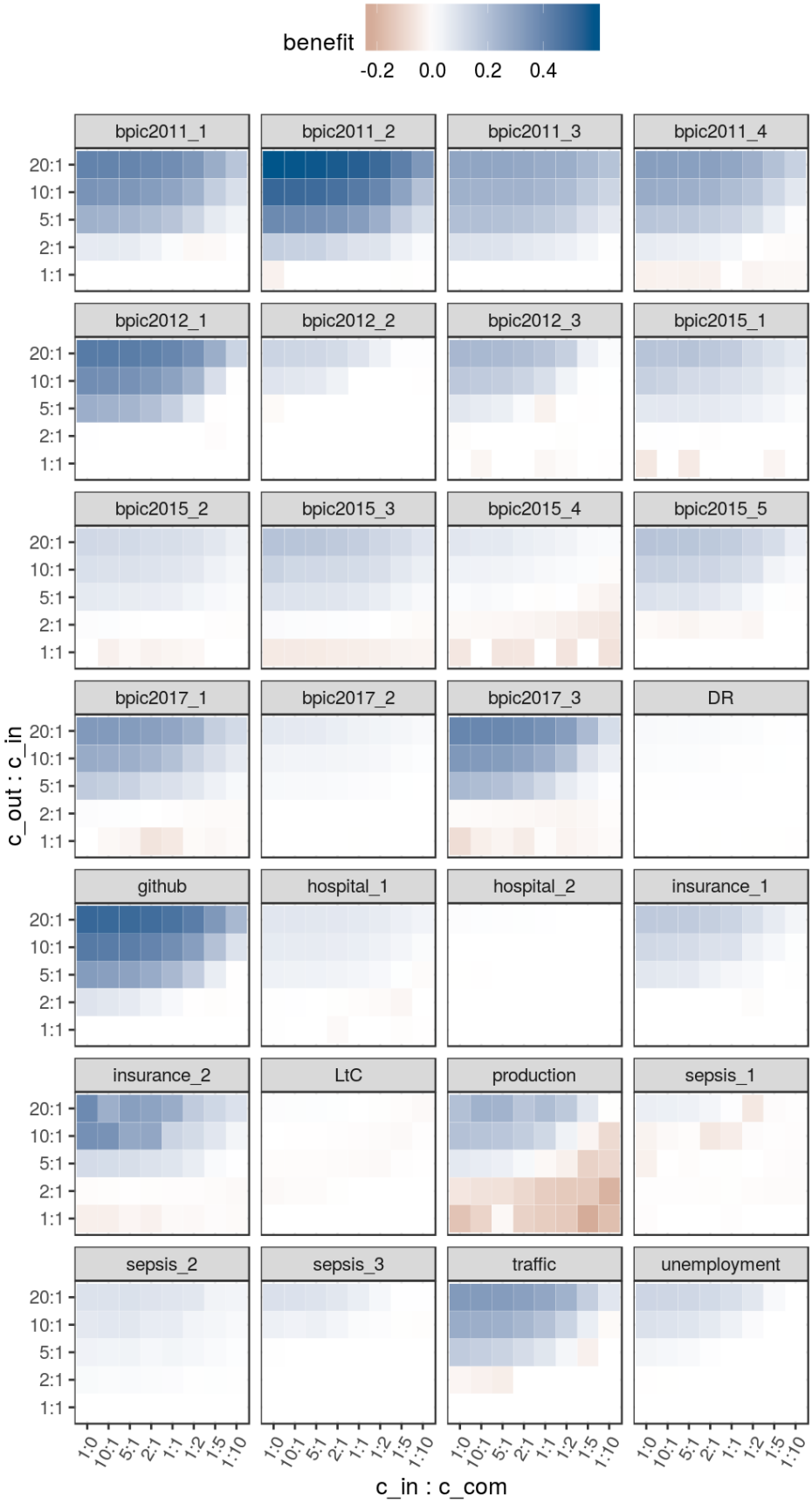


Figure 40: Benefit of the alarm system, varying $c_{com}(\sigma, L)$.

optimization versus a baseline where a fixed prediction score threshold is used to generate alarms, as considered in previous work in the field.

While the scenarios discussed in Boxes 1-3 show that the proposed framework is versatile enough to cover a variety of cases, the current version of the framework relies on two main assumptions. First, it assumes that an alarm always triggers an intervention, thus ignoring that a process worker might in some cases decide not to or be unable to intervene. Additionally, the current version of the framework considers each case in isolation, omitting the overall workload of the process workers, which in reality is an important factor for determining the number of alarms that can be acted upon. This limitation can be lifted by, e.g. combining the alarm system with [17], which proposes a recommender system that optimizes suggestions in case of concurrent process executions. A second limitation of the framework is that only one possible type of intervention is envisaged. This assumption can be lifted by extending the framework so that the cost of an intervention can vary depending on the specific action suggested by a recommender system.

Next to these limitations, we acknowledge the importance of further investigation on the applicability of the framework in practice. In particular, in the future, we aim at collaborating with companies and institutions to study whether process stakeholders are able to define the costs in a natural and simple way. Also, we plan to further investigate the consequences of incorrect and/or imprecise instantiations of the cost models. Furthermore, the current evaluation is limited to measuring the benefit of the alarm system in an offline manner, while a more thorough evaluation would consist in deploying the alarming mechanism in a real organization and making an end-to-end comparison of the costs before and after the deployment of the alarm system. However, this is a difficult task for two main reasons. First, companies need to be willing to let the technique really influence the process executions. Second, the end-to-end effectiveness analysis cannot be conducted without coupling the alarm system with a recommender system: if the system raises proper alarms, but inappropriate interventions are taken, the system would still be ineffective.

Another avenue for future work would be to explore alternative cost models and alarming mechanisms. In particular, the empirical thresholding approach is not restricted to the cost model proposed in this chapter and could be combined with any cost function. Furthermore, an alarm system does not necessarily need to consist of two components (a predictive model and an alarming mechanism). An alternative approach would be to learn an alarming policy directly using reinforcement learning techniques.

8. CONCLUSION AND FUTURE WORK

8.1. Summary of contributions

This thesis addressed the question of how to train, evaluate, and use machine learning models in the context of outcome-oriented predictive and prescriptive process monitoring.

By systematically retrieving and analyzing prior research proposals in this field, we found that existing proposals differ in two main aspects: 1) how the prefixes are divided into buckets (trace bucketing) and 2) how the data related to events in a prefix are transformed into fixed-length feature vectors (sequence encoding). For example, some techniques do not divide the traces in the event log into buckets (i.e. they use a single bucket), while others divide the traces into buckets using clustering or other approaches and then train one classifier per bucket. On the sequence encoding side, some techniques use a lossless encoding, while others use lossy encodings (of different types).

Based on these observations, we identified 11 representative methods in the field. We then designed a benchmark to empirically compare these techniques in a common setting. The benchmark is performed on the 11 identified techniques, executed using a unified experimental set-up and 24 predictive monitoring tasks constructed from 9 real-life event logs. To ensure a fair evaluation, all the selected techniques were implemented as a publicly available consolidated framework, which is designed to incorporate additional datasets and methods. The results of the benchmark show that the most reliable and accurate results (in terms of AUC) are obtained using a lossy (aggregation) encoding of the sequence, e.g. the frequencies of performed activites rather than the ordered activities. One of the main benefits of this encoding is that it enables to represent all prefix traces, regardless of their length, in the same number of features. This way, a single classifier can be trained over all of the prefix traces, allowing the classifier to derive meaningful patterns by itself. These results disprove the existing opinion in the literature about the superiority of a lossless encoding of the trace (index-based encoding [54]) that requires prefixes to be divided into buckets according to their length, while multiple classifiers are trained on each such subset of prefixes.

The review of previous work revealed that existing methods for training predictive models for outcome-oriented predictive process monitoring make use of only the structured (categorical or numeric) data available in an event log, neglecting the fact that real-life event logs often contain additional unstructured (textual) data. To address this gap, the thesis proposes a framework that combines text mining methods to extract features from textual documents, with existing predictive process monitoring techniques designed for structured data. We perform an experimental evaluation over different combinations of text mining, trace bucketing, sequence encoding, and classification techniques on three real-life datasets containing both structured and unstructured data. The evaluation confirms the im-

portance of including textual data when training predictive models and shows that a simple bag-of-ngrams encoding, in combination with a RF or XGBoost single classifier and aggregation encoding often outperforms other text modeling techniques.

The literature review also revealed that predictive process monitoring techniques are traditionally evaluated in terms of two main dimensions: 1) prediction accuracy, measuring how well the predictions correspond to the actual outcomes, and 2) prediction earliness, reflecting that a prediction is more useful if it uses only a short prefix of the trace, i.e. if it is made in the early stages of the execution of a process instance. These quality dimensions, however, neglect the "monitoring" aspect of predictive process monitoring, i.e. that for the same process instance multiple predictions are made given different prefixes of the same trace. In practice it is undesirable that a predictive model often changes its mind, i.e. that it outputs a highly volatile (unstable) sequence of predictions for the same case, since it would reduce the users' trust in the model and decrease their willingness to act upon any given prediction. To address these problems, the thesis introduces the notion of temporal stability for predictive process monitoring and propose a metric for measuring it. Temporal stability characterizes how much the prediction scores obtained for successive prefixes of the same trace differ from each other. For a temporally stable classifier, such successive prediction scores are similar to each other, resulting in a smooth time series, while in case of an unstable classifier, the resulting time series is volatile. We evaluate the temporal stability of seven existing predictive process monitoring methods, including single and multiclassifiers using random forest, XGBoost, and LSTM neural networks. The experiments conducted on 27 datasets show that the highest temporal stability is achieved by LSTM, followed by a single classifier approach with XGBoost (using either aggregation or index-based encoding). Furthermore, we investigate the effects of hyperparameter optimization on temporal stability. We compare the final classifiers constructed after selecting the best parameters based on 1) AUC over a single run for each parameter setting, 2) AUC over five runs for each setting, 3) combined AUC and inter-run stability over five runs for each setting. The results show that choosing the parameters based on five runs can increase both AUC and temporal stability. However, the improvement is small and is subject to the trade-off of five times more computations during validation. Finally, we explore how combining the predictions made for the same case via exponential smoothing affects the AUC and temporal stability. The results indicate that smoothing can be a reasonable approach for adjusting the predictions in applications where temporal stability is important at the expense of achieving slightly smaller AUC. Moreover, we observe that the multiclassifiers benefit the most from smoothing, in some cases even increasing both the temporal stability and the AUC at the same time. Therefore, when high temporal stability is required, it may be reasonable to use a multiclassifier approach with smoothing, achieving stable results with little or no loss in accuracy.

The purpose of a predictive process monitoring system is to provide operational decision support for process workers. Namely, based on a prediction, the user can decide to intervene in the process in order to mitigate the effects of a likely undesired outcome or to prevent it altogether. Despite that end goal, most of the existing works employ a continuous monitoring setting, where the users receive predictions after every event in a case, but are not advised if and how to act upon them. A handful of studies adopt an alarm-based monitoring setting, where the user is alerted only if a prediction is severe and/or reliable enough that it calls for an intervention action by the user. All of these works assume that the user is able to specify a threshold on the prediction scores that would trigger such alerts. However, in practice the optimal threshold depends on different types of costs related to the business process, such as the cost of an intervention and the cost of reaching an undesired outcome. The fifth contribution of the thesis is an alarm-based prescriptive process monitoring framework that extends existing predictive process monitoring approaches with the concepts of alarms, interventions, compensations, and mitigation effects. The framework incorporates a cost model to analyze the tradeoffs between the cost of intervention, the benefit of mitigating or preventing undesired outcomes, and the cost of compensating for unnecessary interventions induced by false alarms. The cost model allows one to estimate the benefits of deploying a prescriptive process monitoring system for the purposes of return on investment analysis. Additionally, the framework incorporates a technique to optimize the alarm generation mechanism with respect to a given configuration of the cost model and a given event log. An empirical evaluation on 28 real-life logs shows the benefits of applying this optimization versus a baseline where a fixed prediction score threshold is used to generate alarms, as considered in previous work in the field.

In order to support the reproducibility of the research results, the implementaions for all the experiments performed in this thesis are made available in code repositories (see Appendix A).

8.2. Future work

The contributions presented in this thesis pave the way for several directions for future work, outlined in the following paragraphs.

Multi-class outcomes. This thesis focuses specifically on the settings where the outcome of a business process is considered a binary variable (e.g. positive vs. negative outcomes). There are situations, however, where a case may have more than two outcomes. For example, a complaint-to-resolution process may end up in the case being (i) resolved; (ii) closed without resolution by the IT helpdesk; or (iii) closed by the customer. Many of the ideas presented in this thesis can also be applied to multiclass outcome-oriented predictive monitoring problems. However, we cannot say which sequence encodings, bucketing and classification techniques would perform better in this setting. Also, the problem of temporal prediction

stability and alarm-based prescriptive monitoring would need to be recast to fit this setting. This is a direction for future work.

Advanced text modeling. Recent years have witnessed a rapid growth of literature on methods for generating better deep-learning based word and sentence representations. A general trend emerging in these works is to use universal word embeddings that are pretrained on a large text corpus and fine-tune them to the specific task at hand via transfer learning [44]. Such embeddings are likely to yield more stable and comprehensive representations for textual data than training the embeddings separately for each data set. Another tendency is towards subword [5] or character-based [75] language models that are able to generate representations for out-of-vocabulary words, which were not seen in the training set. Regarding sentence representations, a simple yet effective approach proposed by Arora et al. represents sentences as weighted averages of word vectors [1]. Inspired by the well-known skip-gram model for generating word vectors [65], Kiros et al. proposed Skip-Thought Vectors, which lift the same idea to the sentence level [47]. Over the last year, several approaches for learning universal sentence represenations via multi-task learning have been developed [14, 89]. Incorporating these recent advances in text modeling to the framework proposed in Chapter 5 is a direction for future work.

Extending the notion of temporal stability. As another future direction, more robust notions of temporal stability could be developed, which would (still) require most of the successive differences in predictions to be small, but would not penalize the classifier for changing the prediction when an event with a relevant signal arrives. For instance, works on early sequence classification could possibly be helpful in developing an adaptive smoothing method that decreases volatility on subsequences without suppressing the relevant signal. Furthermore, the notion of temporal stability could be extended to other prediction tasks, such as multi-class predictions and regression. For instance, temporal stability could also be investigated in the context of predicting the remaining time of an ongoing case. While several methods have been developed with the goal of providing accurate remaining time estimations, using, e.g. non-parametric regression [105], support vector regression [78], or LSTM neural networks [94], none of these works has considered the stability of the predictions. Another avenue for future work is to incorporate the notion of stability into the training phase of the classifiers. For instance, in case of neural networks this could be achieved by adjusting the loss function to take into account both the accuracy and the stability of the predictions.

Alarming mechanism based on sequences of predictions. The alarm-based prescriptive process monitoring framework presented in Chapter 7 makes use of an alarming mechanism which decides whether to raise an alarm based on a single prediction. However, as observed in Chapter 6 there is benefit in combining the predictions made for different prefixes of the same trace. Based on this motivation, a possible avenue for future work would be to construct an alarming mechanism that takes into account the whole series of predictions made for the same case.

For instance, concepts of Bayesian statistics could be used to update the expected processing cost each time a new prediction is made, taking into account that the uncertainty in predictions made for longer prefixes is smaller than in predictions made for shorter ones.

Explainable predictions. If the process workers and analysts are able to understand why a predictive model yielded a particular prediction, they are able to make better founded decisions based on the prediction. Therefore, when deploying a predictive process monitoring system in practice, it is desirable that the system provides explainable predictions. However, as supported by the evidence from the experiments performed in this thesis, there is often a tradeoff between the accuracy and the explainability of a model. In particular, recurrent neural networks (e.g. LSTMs) and ensemble models (e.g. RF and XGBoost) that are considered to be difficult to interpret (or even black-box) achieve higher accuracy than simpler models like logistic regression or decision trees. A future direction would be to explore ways of extracting interpretable explanations from these high accuracy models (using techniques such as LIME [80]) and presenting them to the users of a predictive process monitoring system in an understandable way. A systematic overview of such techniques is provided by Guidotti et al. [39]. Alternatively, a broader range of techniques yielding explainable predictions can be tested, including sequential pattern mining approaches [58] and predictive clustering rules [113].

Active learning. Another avenue for future work is to extend the alarm-based prescriptive monitoring framework with active learning methods in order to incrementally tune the alarming mechanism based on feedback about the relevance of the alarms and the effectiveness of the interventions. Also, since specifying the cost model manually can be a difficult task for the user, active learning can be used to adjust the initial costs to better reflect the costs and benefits observed in practice.

BIBLIOGRAPHY

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. *International Conference on Learning Representations*, 2017.

[2] 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.

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

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[6] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[7] 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.

[8] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[10] Diego Calvanese, Marlon Dumas, Ülari Laurson, Fabrizio Maria Maggi, Marco Montali, and Irene Teinemaa. Semantics and analysis of DMN decision tables. 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 217–233. Springer, 2016.

[11] Diego Calvanese, Marlon Dumas, Ülari Laurson, Fabrizio Maria Maggi, Marco Montali, and Irene Teinemaa. Semantics, analysis and simplification of DMN decision tables. *Inf. Syst.*, 78:112–125, 2018.

[12] Malú Castellanos, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. A comprehensive and automated approach to intelligent business processes execution analysis. *Distributed and Parallel Databases*, 16(3):239–273, 2004.

[13] Malú Castellanos, Norman Salazar, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. Predictive business operations management. In Subhash Bhalla, editor, *Databases in Networked Information Systems, 4th International Workshop, DNIS 2005, Aizu-Wakamatsu, Japan, March 28-30, 2005, Proceedings*, volume 3433 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2005.

[14] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. Universal sentence encoder for english. In Eduardo Blanco and Wei Lu, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 169–174. Association for Computational Linguistics, 2018.

[15] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In 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*, pages 785–794. ACM, 2016.

[16] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, and Wil M. P. van der Aalst. Supporting risk-informed decisions during business process execution. 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 116–132. Springer, 2013.

[17] 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.

[18] Asma Dachraoui, Alexis Bondu, and Antoine Cornuéjols. Early classification of time series as a non myopic sequential decision making problem. In Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I*, volume 9284 of *Lecture Notes in Computer Science*, pages 433–447. Springer, 2015.

[19] 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.

[20] 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. *Inf. Syst.*, 56:235–257, 2016.

[21] Marcus Dees, Massimiliano de Leoni, and Felix Mannhardt. Enhancing process models to improve business performance: A methodology and case studies. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike P. Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*, volume 10573 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2017.

[22] Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

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

[24] Pedro M. Domingos. Metacost: A general method for making classifiers cost-sensitive. In Usama M. Fayyad, Surajit Chaudhuri, and David Madigan, editors, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*, pages 155–164. ACM, 1999.

[25] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

[26] André Elisseeff, Theodoros Evgeniou, and Massimiliano Pontil. Stability of randomized learning algorithms. *Journal of Machine Learning Research*, 6:55–79, 2005.

[27] Charles Elkan. The foundations of cost-sensitive learning. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 973–978. Morgan Kaufmann, 2001.

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

[29] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Mining predictive process models out of low-level multidimensional logs. In 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*, pages 533–547. Springer, 2014.

[30] Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi, and Irene Teinemaa. Clustering-based predictive process monitoring. *IEEE Transactions on Services Computing*, 2016. To appear.

[31] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[32] Mohamed F. Ghalwash and Zoran Obradovic. Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC Bioinformatics*, 13:195, 2012.

[33] Mohamed F. Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. Extraction of interpretable multivariate patterns for early diagnostics. In Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, pages 201–210. IEEE Computer Society, 2013.

[34] Johny Ghattas, Pnina Soffer, and Mor Peleg. Improving business process decision making based on past experience. *Decision Support Systems*, 59:93–107, 2014.

[35] Georgios Gousios. The ghtorent dataset and tool suite. In Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim, editors, *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, pages 233–236. IEEE Computer Society, 2013.

[36] 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.

[37] Daniela Grigori, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. Improving business process quality through exception understanding, prediction, and prevention. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 159–168. Morgan Kaufmann, 2001.

[38] Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. Prescriptive analytics for recommendation-based business process optimization. In Witold Abramowicz and Angelika I. Kokkinaki, editors, *Business Information Systems - 17th International Conference, BIS 2014, Larnaca, Cyprus,*

May 22-23, 2014. Proceedings, volume 176 of *Lecture Notes in Business Information Processing*, pages 25–37. Springer, 2014.

[39] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019.

[40] C.W. Gunther and H.M.W. Verbeek. *XES - standard definition*. BPM reports. BPMcenter.org, 2014.

[41] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 2017.

[42] Guoliang He, Yong Duan, Rong Peng, Xiaoyuan Jing, Tieyun Qian, and Lingling Wang. Early classification on multivariate time series. *Neurocomputing*, 149:777–787, 2015.

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

[44] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339. Association for Computational Linguistics, 2018.

[45] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In Miryung Kim, Romain Robbes, and Christian Bird, editors, *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 291–302. ACM, 2016.

[46] Miryung Kim, Romain Robbes, and Christian Bird, editors. *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*. ACM, 2016.

[47] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. 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 3294–3302, 2015.

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

[49] Julian Krumeich, Dirk Werth, and Peter Loos. Prescriptive control of busi-

ness processes - new potentials through predictive analytics of big data in the process manufacturing industry. *Business & Information Systems Engineering*, 58(4):261–280, 2016.

[50] Matjaz Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *ECAI*, pages 445–449, 1998.

[51] Meelis Kull and Peter A. Flach. Novel decompositions of proper scoring rules for classification: Score adjustment as precursor to calibration. In Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I*, volume 9284 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2015.

[52] Geetika T. Lakshmanan, Songyun Duan, Paul T. Keyser, Francisco Curbera, and Rania Khalaf. Predictive analytics for semi-structured case oriented business processes. In Michael zur Muehlen and Jianwen Su, editors, *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Hoboken, NJ, USA, September 13-15, 2010, Revised Selected Papers*, volume 66 of *Lecture Notes in Business Information Processing*, pages 640–651. Springer, 2010.

[53] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1188–1196. JMLR.org, 2014.

[54] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria 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.

[55] Yu-Feng Lin, Hsuan-Hsu Chen, Vincent S. Tseng, and Jian Pei. Reliable early classification on multivariate time series with numerical and categorical attributes. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu Bao Ho, David Wai-Lok Cheung, and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining - 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part I*, volume 9077 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2015.

[56] Charles X. Ling, Qiang Yang, Jianning Wang, and Shichao Zhang. Decision trees with minimal costs. In Carla E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML*

2004), Banff, Alberta, Canada, July 4-8, 2004, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[57] C. H. Bryan Liu, Benjamin Paul Chamberlain, Duncan A. Little, and Ângelo Cardoso. Generalising random forest parameter optimisation to include stability and cost. In Yasemin Altun, Kamalika Das, Taneli Mielikäinen, Donato Malerba, Jerzy Stefanowski, Jesse Read, Marinka Zitnik, Michelangelo Ceci, and Saso Dzeroski, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part III*, volume 10536 of *Lecture Notes in Computer Science*, pages 102–113. Springer, 2017.

[58] David Lo, Hong Cheng, Jiawei Han, Siau-Cheng Khoo, and Chengnian Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 557–566. ACM, 2009.

[59] Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil M. P. van der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Inf. Syst.*, 54:209–234, 2015.

[60] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In 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*, pages 457–472. Springer, 2014.

[61] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[62] Andreas Metzger and Felix Föcker. Predictive business process monitoring considering reliability estimates. 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 445–460. Springer, 2017.

[63] Andreas Metzger, Rod Franklin, and Yagil Engel. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *2012 Annual SRII Global Conference, San Jose, CA, USA, July 24-27, 2012*, pages 313–322. IEEE Computer Society, 2012.

[64] 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.

[65] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[66] Usue Mori, Alexander Mendiburu, Sanjoy Dasgupta, and José Antonio Lozano. Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE Trans. Neural Netw. Learning Syst.*, 29(10):4569–4578, 2018.

[67] Usue Mori, Alexander Mendiburu, Eamonn J. Keogh, and José Antonio Lozano. Reliable early classification of time series based on discriminating the classes over time. *Data Min. Knowl. Discov.*, 31(1):233–263, 2017.

[68] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[69] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 625–632. ACM, 2005.

[70] Randal S. Olson, William La Cava, Zairah Mustahsan, Akshay Varik, and Jason H. Moore. Data-driven advice for applying machine learning to bioinformatics problems. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Marylyn D. Ritchie, and Teri E. Klein, editors, *Biocomputing 2018: Proceedings of the Pacific Symposium, The Big Island of Hawaii, Hawaii, USA, January 3-7, 2018*, pages 192–203, 2018.

[71] Siim Orasmaa, Timo Petmanson, Alexander Tkachenko, Sven Laur, and Heiki-Jaan Kaalep. Estnltk - NLP toolkit for estonian. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*. European Language Resources Association (ELRA), 2016.

[72] Jason W. Osborne. Dealing with missing or incomplete data: Debunking the myth of emptiness. In *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*, pages 105–138. Sage Publications Thousand Oaks, CA, 2013.

[73] Nathan Parrish, Hyrum S. Anderson, Maya R. Gupta, and Dun-Yu Hsiao.

Classifying with confidence from incomplete information. *Journal of Machine Learning Research*, 14(1):3561–3589, 2013.

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[75] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018.

[76] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[78] 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.

[79] Sonja Pravilovic, Annalisa Appice, and Donato Malerba. Process mining to forecast the future of running cases. In Annalisa Appice, Michelangelo Ceci, Corrado Loglisci, Giuseppe Manco, Elio Masciari, and Zbigniew W. Ras, editors, *New Frontiers in Mining Complex Patterns - Second International Workshop, NFMCP 2013, Held in Conjunction with ECML-PKDD 2013, Prague, Czech Republic, September 27, 2013, Revised Selected Papers*, volume 8399 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2013.

[80] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In 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*, pages 1135–1144. ACM, 2016.

[81] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing de-

lays. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*, volume 8274 of *Lecture Notes in Computer Science*, pages 389–403. Springer, 2013.

[82] Andrii Rozumnyi. A dashboard-based predictive process monitoring engine. Master's thesis, University of Tartu, 2017.

[83] Tiago Santos and Roman Kern. A literature survey of early time series classification and deep learning. In Roman Kern, Gerald Reiner, and Olivia Bluder, editors, *Proceedings of the 1st International Workshop on Science, Application and Methods in Industry 4.0 co-located with (i-KNOW 2016), Graz, Austria, October 19, 2016.*, volume 1793 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[84] Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

[85] Bernd Schwegmann, Martin Matzner, and Christian Janiesch. A method and tool for predictive event-driven process analytics. In *11. Internationale Tagung Wirtschaftsinformatik, Leipzig, Germany, February 27 – March 1, 2013*, page 46, 2013.

[86] Bernd Schwegmann, Martin Matzner, and Christian Janiesch. precep: Facilitating predictive event-driven process analytics. In Jan vom Brocke, Riitta Hekkala, Sudha Ram, and Matti Rossi, editors, *Design Science at the Intersection of Physical and Virtual Design - 8th International Conference, DESRIST 2013, Helsinki, Finland, June 11-12, 2013. Proceedings*, volume 7939 of *Lecture Notes in Computer Science*, pages 448–455. Springer, 2013.

[87] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maria 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.

[88] Victor S. Sheng and Charles X. Ling. Thresholding for making classifiers cost-sensitive. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 476–481. AAAI Press, 2006.

[89] Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J. Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. *CoRR*, abs/1804.00079, 2018.

[90] Suriadi Suriadi, Moe Thandar Wynn, Chun Ouyang, Arthur H. M. ter Hofstede, and Nienke J. van Dijk. Understanding process behaviours in a large

insurance company in australia: A case study. 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 449–464. Springer, 2013.

[91] Romain Tavenard and Simon Malinowski. Cost-aware early classification of time series. In Paolo Frasconi, Niels Landwehr, Giuseppe Manco, and Jilles Vreeken, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I*, volume 9851 of *Lecture Notes in Computer Science*, pages 632–647. Springer, 2016.

[92] Niek Tax, Irene Teinemaa, and Sebastiaan J. van Zelst. An interdisciplinary comparison of sequence modeling methods for next-element prediction. *CoRR*, abs/1811.00062, 2018.

[93] 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.

[94] 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.

[95] Irene Teinemaa, Marlon Dumas, Anna Leontjeva, and Fabrizio Maria Maggi. Temporal stability in predictive process monitoring. *Data Min. Knowl. Discov.*, 32(5):1306–1338, 2018.

[96] Irene Teinemaa, Marlon Dumas, Fabrizio Maria 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.

[97] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Outcome-oriented predictive process monitoring: Review and

benchmark. *ACM Transactions on Knowledge Discovery from Data*, 2018. To appear.

[98] Irene Teinemaa, Anna Leontjeva, Marlon Dumas, and Riivo Kikas. Community-based prediction of activity change in skype. In Jian Pei, Fabrizio Silvestri, and Jie Tang, editors, *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, pages 73–80. ACM, 2015.

[99] Irene Teinemaa, Niek Tax, Massimiliano de Leoni, Marlon Dumas, and Fabrizio Maria 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.

[100] Peter D. Turney. Types of cost in inductive concept learning. *CoRR*, cs.LG/0212034, 2002.

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

[102] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011.

[103] Wil M. P. van der Aalst, Vladimir A. Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling*, 9(1):87–111, 2010.

[104] 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.

[105] 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.

[106] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria 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.

[107] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Irene Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *CoRR*, abs/1805.02896, 2018.

[108] Sida I. Wang and Christopher D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 2: Short Papers*, pages 90–94. The Association for Computer Linguistics, 2012.

[109] Zhengzheng Xing, Jian Pei, Guozhu Dong, and Philip S. Yu. Mining sequence classifiers for early prediction. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, pages 644–655. SIAM, 2008.

[110] Zhengzheng Xing, Jian Pei, and Eamonn J. Keogh. A brief survey on sequence classification. *SIGKDD Explorations*, 12(1):40–48, 2010.

[111] Zhengzheng Xing, Jian Pei, and Philip S. Yu. Early classification on time series. *Knowl. Inf. Syst.*, 31(1):105–127, 2012.

[112] Bianca Zadrozny and Charles Elkan. Learning and making decisions when costs and probabilities are both unknown. In Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant, editors, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001*, pages 204–213. ACM, 2001.

[113] Bernard Zenko, Saso Dzeroski, and Jan Struyf. Learning predictive clustering rules. In Francesco Bonchi and Jean-François Boulicaut, editors, *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*, pages 234–250. Springer, 2005.

Appendix A. CODE REPOSITORIES

In order to support the reproducibility of the research results, we have published the implementations of the proposed methods and the code required to run the experiments reported in the thesis in the following code repository: `https://github.com/irhete/predictive-monitoring-thesis`.

Due to our aim of applying a unified experimental setting throughout the thesis, the experiments reported in this thesis differ slightly from the ones reported in the original publications. Accordingly, the implementations and the code required to run the experiments reported in the the original publications can be found in the following code repositories:

I Benchmark of outcome-oriented predictive monitoring methods [97]: `https://github.com/irhete/predictive-monitoring-benchmark`

II Predictive business process monitoring with structured and unstructured data [96]: `https://github.com/irhete/PredictiveMonitoringWithText`

III Temporal stability in predictive process monitoring [95]: `https://github.com/irhete/stability-predictive-monitoring`

IV Alarm-based prescriptive process monitoring [99]: `https://github.com/TaXxER/AlarmBasedProcessPrediction`

Appendix B. ADDITIONAL EXPERIMENTS

This Appendix reports the following:

- The distributions of case lengths in different outcome classes (Figures 41-42);
- The optimal number of clusters (Table 26) and the optimal number of neighbors for KNN approaches (Table 27) found for each classifier;
- The distributions of bucket sizes for the different bucketing methods (Figures 43-44);
- The overall AUC and F-score values for RF (Table 28), logit (Table 29), and SVM (Table 30);
- The AUC scores across prefix lengths using XGBoost classifier and all of the compared methods (Figures 45-46);
- The AUC scores across prefix lengths, including long traces only, using the XGBoost classifier (Figure 47);
- The execution times for RF (Tables 31-32), logit(Tables 33-34), and SVM(Tables 35-36);
- The offline (Figure 48) and online (Figure 49) execution times and the overall AUC scores (Figure 50) when filtering the static categorical attribute domain, using the XGBoost classifier;
- The execution times for RF (Table 37) and logit(Table 38) over different text modeling approaches;
- Overall Brier scores for uncalibrated and calibrated classifiers (Table 39);
- Differences in Brier scores on uncalibrated vs. calibrated classifiers over different prefix lengths (Figures 51-52);
- Benefit of the alarm system, varying mitigation effectiveness with a linear decay (Figure 53) and varying the ratios of $c_{out}(\sigma, L) : c_{in}(k, \sigma, L)$ and $c_{in}(k, \sigma, L) : c_{com}(\sigma, L)$ with linearly increasing $c_{in}(k, \sigma, L)$ (Figure 54).

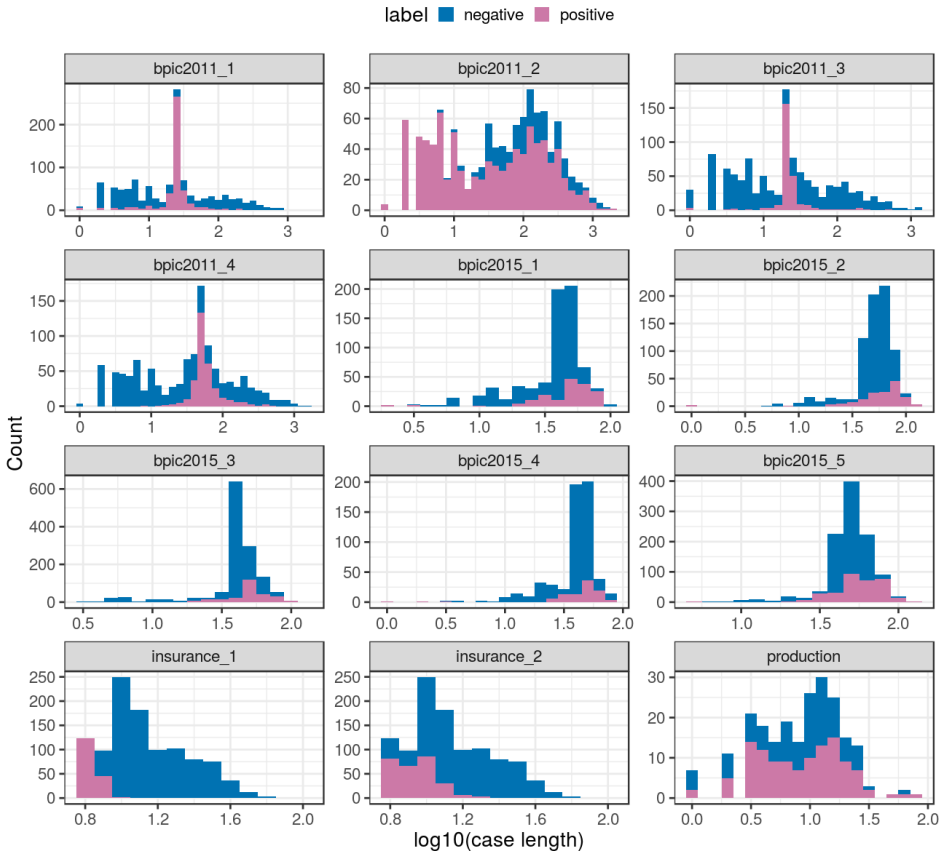


Figure 41: Case length histograms for positive and negative classes

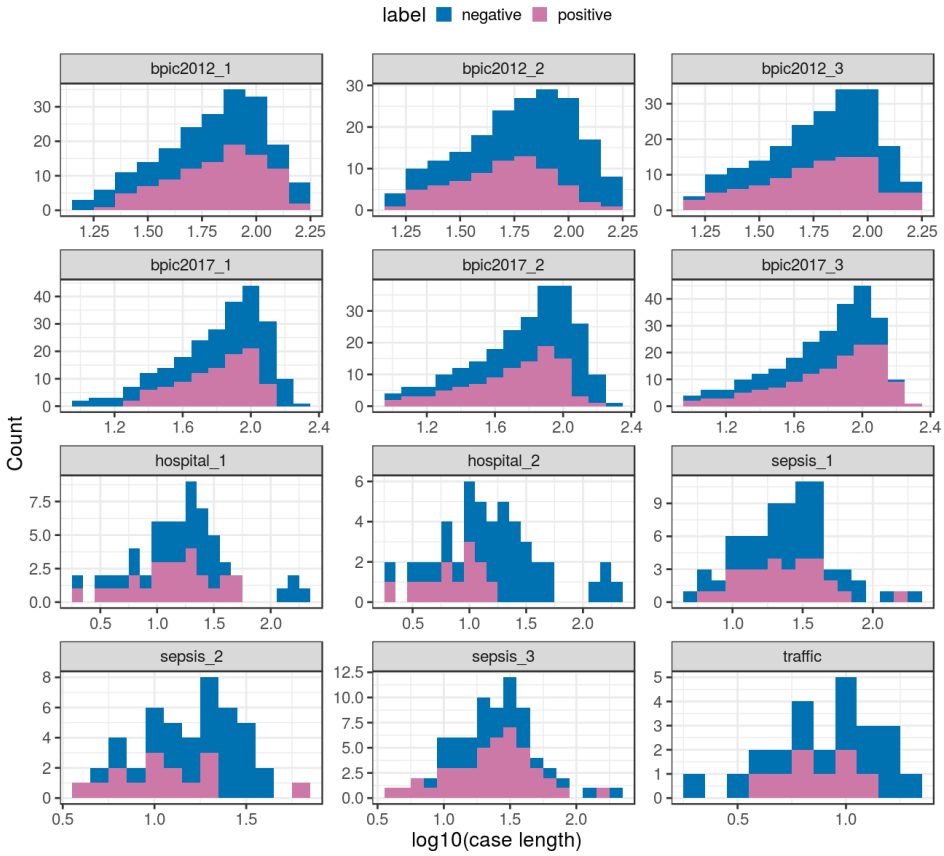


Figure 42: Case length histograms for positive and negative classes (continued)

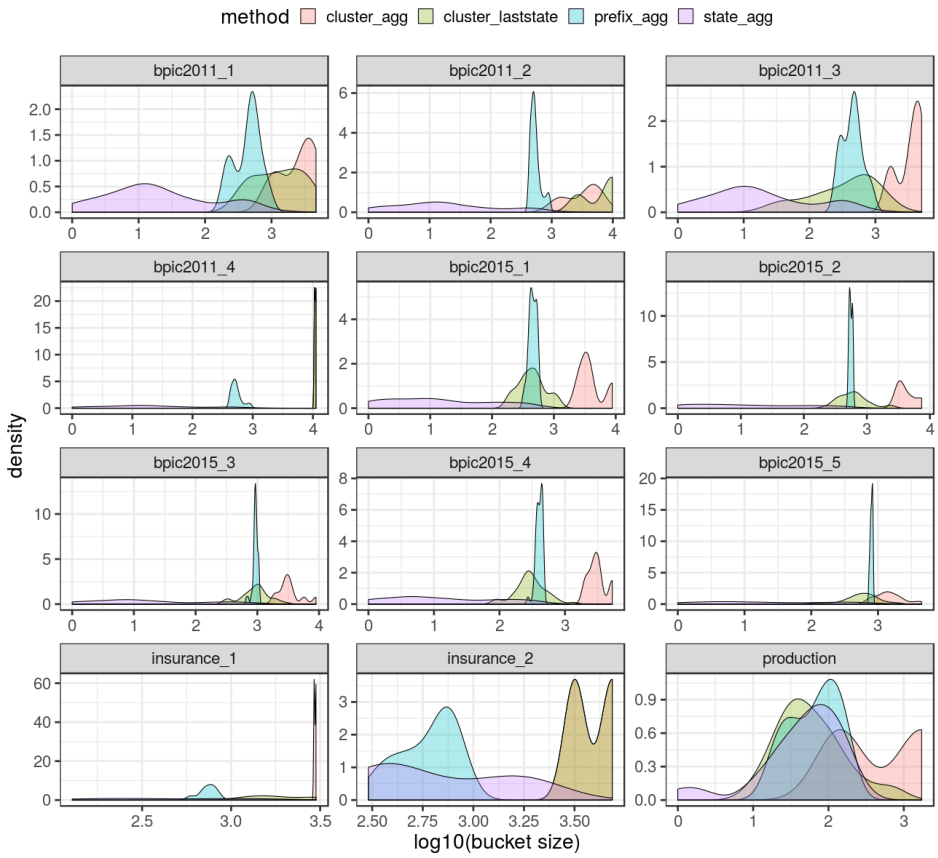


Figure 43: Bucket size distributions

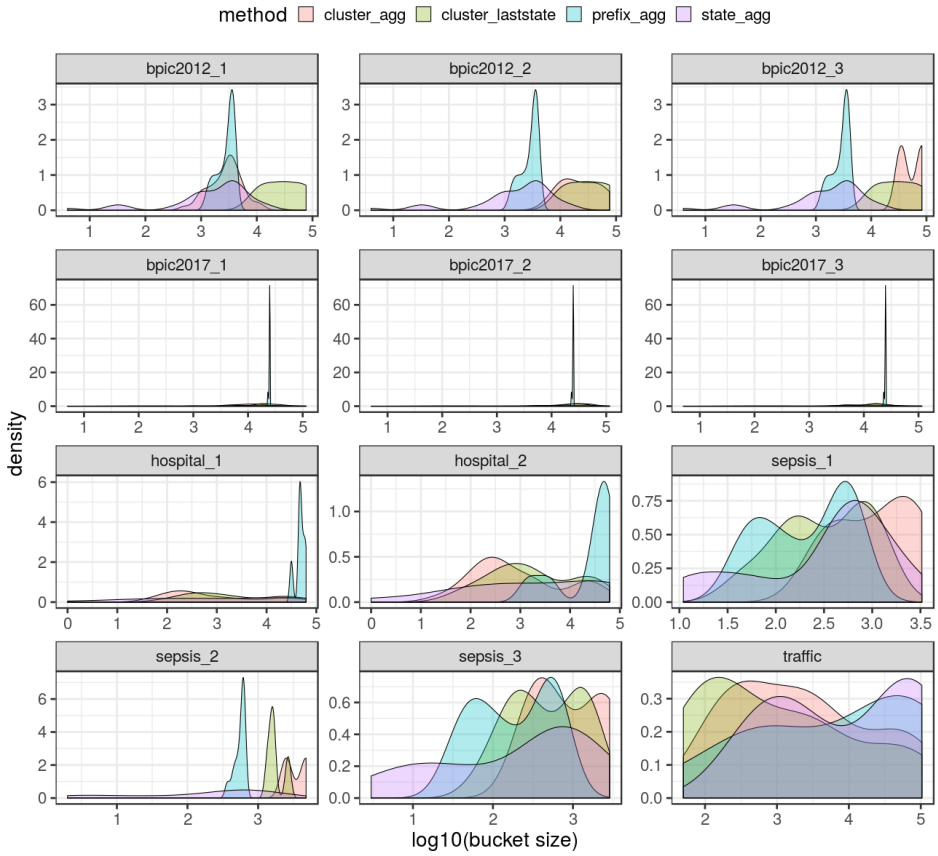


Figure 44: Bucket size distributions (continued)

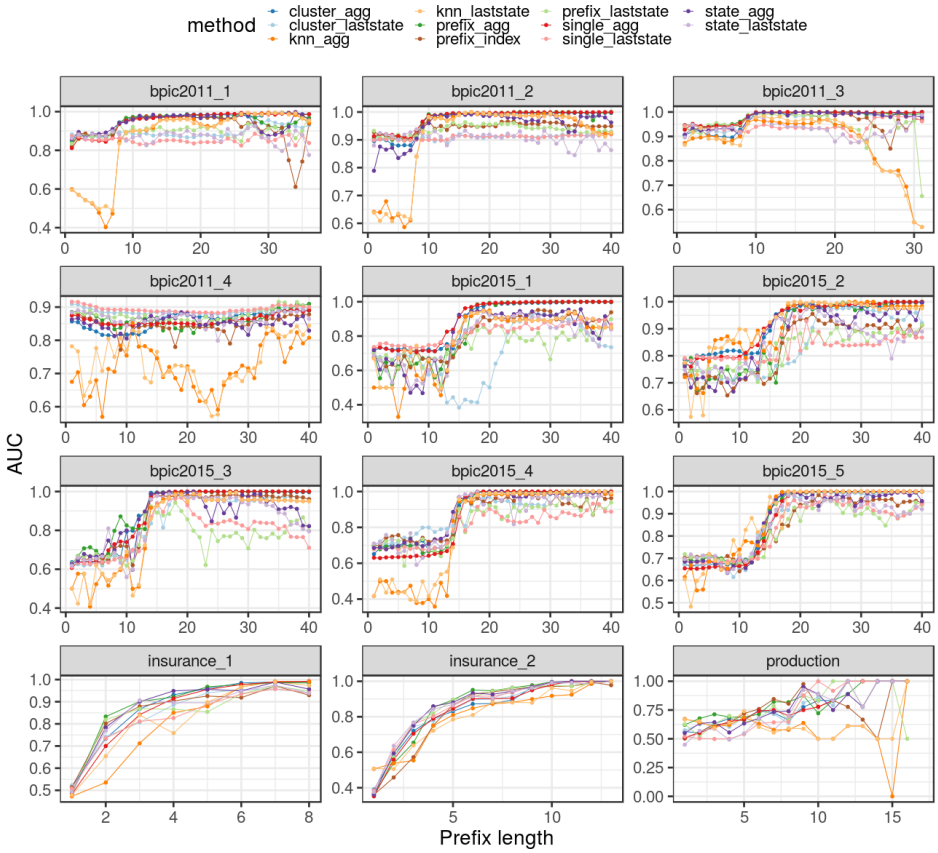


Figure 45: AUC across prefix lengths using **XGBoost**, all methods

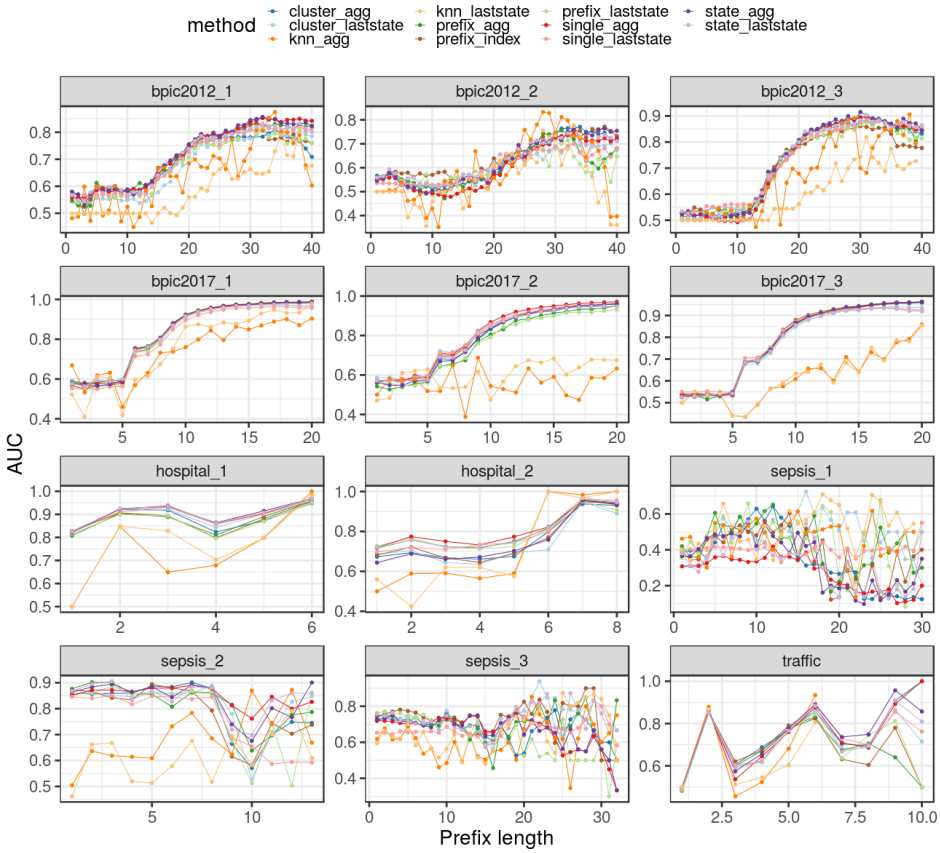


Figure 46: AUC across prefix lengths using **XGBoost**, all methods (continued)

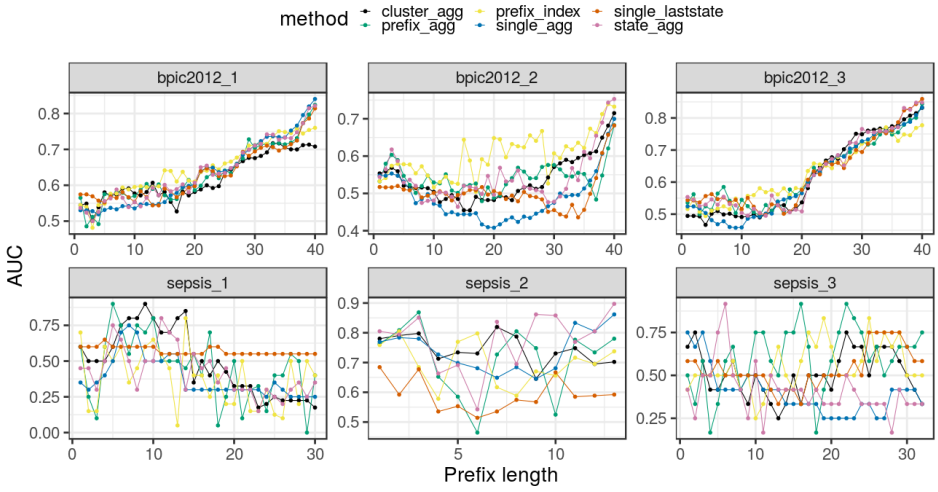


Figure 47: AUC across prefix lengths using **XGBoost**, long traces only

Table 26: Best number of clusters

	RF		XGBoost		Logit		SVM	
dataset	cluster_last	cluster_agg	cluster_last	cluster_agg	cluster_last	cluster_agg	cluster_last	cluster_agg
bpic2011_1	10	8	10	6	24	23	15	43
bpic2011_2	28	4	3	6	20	13	27	24
bpic2011_3	30	4	28	4	33	13	32	44
bpic2011_4	2	21	2	2	16	2	24	36
insurance_2	8	12	2	2	4	3	30	25
insurance_1	6	18	3	2	10	47	45	3
bpic2015_1	39	10	37	4	21	2	13	7
bpic2015_2	32	6	31	5	42	7	9	13
bpic2015_3	44	12	36	10	41	11	11	13
bpic2015_4	45	3	47	5	47	40	19	8
bpic2015_5	43	4	49	19	32	4	8	4
production	44	21	18	2	38	44	10	7
sepsis_1	38	14	19	6	39	41	9	29
sepsis_2	3	8	4	2	7	3	10	21
sepsis_3	2	7	13	7	7	23	7	3
bpic2012_1	22	7	3	35	3	3	8	49
bpic2012_2	9	9	3	4	7	9	15	3
bpic2012_3	10	26	3	2	13	8	22	15
bpic2017_1	39	30	22	43	4	34	39	19
bpic2017_2	11	10	20	15	31	27	40	4
bpic2017_3	29	30	32	34	19	47	21	35
traffic	42	43	29	23	42	36	9	13
hospital_1	35	2	33	48	10	8	48	32
hospital_2	19	48	33	45	11	8	34	28

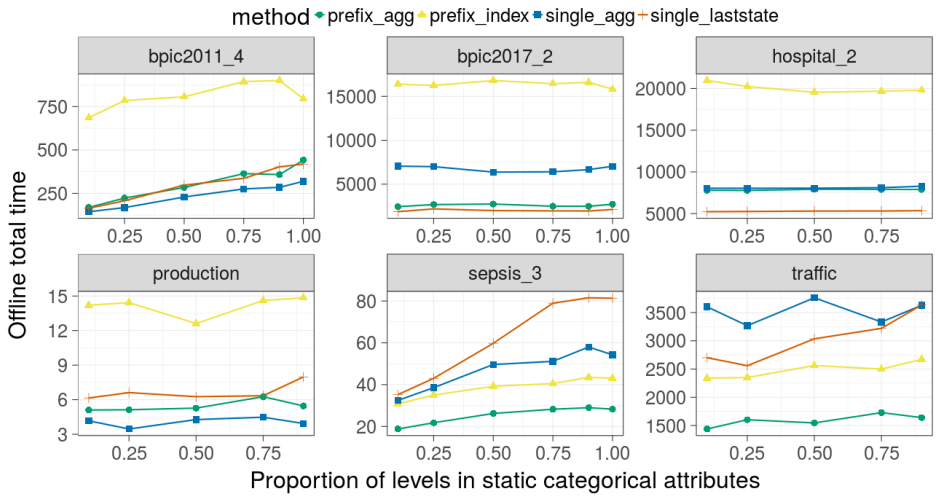


Figure 48: Offline times across different filtering proportions of **static** categorical attribute levels (XGBoost)

Table 27: Best number of neighbors

	RF		XGBoost		Logit		SVM	
dataset	knn_last	knn_agg	knn_last	knn_agg	knn_last	knn_agg	knn_last	knn_agg
bpic2011_1	47	45	50	50	46	48	49	39
bpic2011_2	45	47	50	46	26	21	42	40
bpic2011_3	50	46	50	46	45	32	44	42
bpic2011_4	40	41	43	46	44	50	16	32
insurance_2	46	47	50	45	48	49	32	44
insurance_1	45	49	44	50	29	36	16	12
bpic2015_1	31	49	50	45	32	17	12	3
bpic2015_2	48	50	46	46	41	12	11	2
bpic2015_3	29	48	46	46	40	49	2	3
bpic2015_4	30	43	50	36	13	9	3	38
bpic2015_5	30	37	46	50	27	47	2	2
production	10	19	19	16	46	14	15	21
sepsis_1	50	49	47	32	32	26	32	43
sepsis_2	50	41	47	49	47	49	46	39
sepsis_3	47	48	50	50	32	50	49	29
bpic2012_1	2	50	9	17	6	45	37	33
bpic2012_2	50	50	14	50	3	42	32	39
bpic2012_3	50	50	19	3	9	25	22	22
bpic2017_1	50	50	50	50	50	50	4	50
bpic2017_2	50	50	50	50	50	50	50	50
bpic2017_3	50	50	50	50	50	50	50	50
traffic	50	50	14	25	10	10	31	42
hospital_1	50	50	26	22	29	50	38	3
hospital_2	50	50	50	50	36	6	31	24

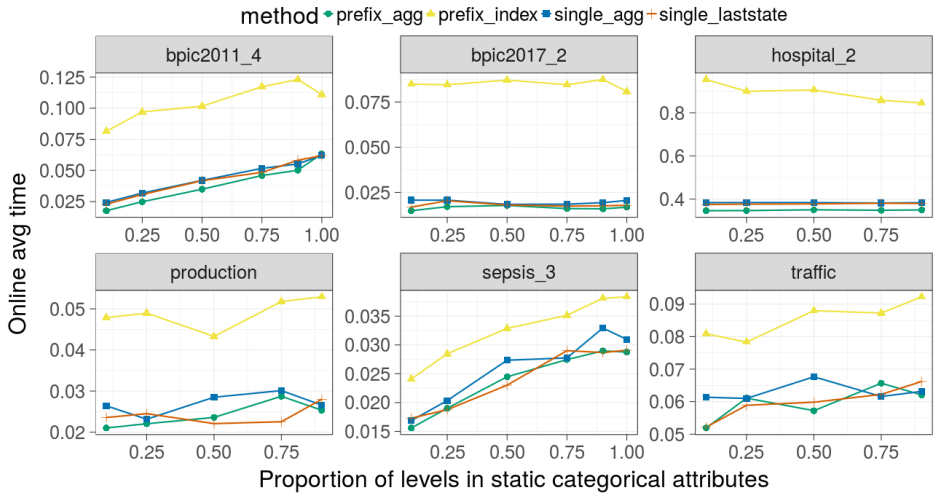


Figure 49: Online times across different filtering proportions of **static** categorical attribute levels (XGBoost)

Table 28: Overall AUC (F-score) for **random forest**

	bpic2011_1	**bpic2011_2**	**bpic2011_3**	**bpic2011_4**	**insurance_1**	**insurance_2**
single_laststate	0.87 (0.73)	0.92 (0.83)	0.94 (0.79)	**0.9** (**0.82**)	0.88 (0.53)	0.83 (0.47)
single_agg	**0.94** (0.86)	**0.98** (**0.95**)	**0.98** (**0.94**)	0.89 (0.8)	**0.91** (**0.65**)	0.82 (0.48)
knn_laststate	0.92 (0.85)	0.96 (0.92)	0.92 (0.85)	0.79 (0.7)	0.85 (0.55)	0.77 (0.5)
knn_agg	0.87 (0.8)	0.94 (0.9)	0.9 (0.81)	0.78 (0.7)	0.87 (0.63)	0.79 (0.54)
state_laststate	0.88 (0.74)	0.92 (0.86)	0.94 (0.76)	0.88 (0.8)	0.88 (0.55)	0.84 (**0.61**)
state_agg	0.92 (0.85)	0.96 (0.93)	0.97 (0.87)	0.87 (0.77)	0.9 (0.63)	**0.85** (0.59)
cluster_laststate	0.9 (0.75)	0.91 (0.87)	**0.98** (0.92)	0.88 (0.79)	0.89 (0.54)	0.82 (0.46)
cluster_agg	0.91 (0.82)	0.96 (0.94)	**0.98** (0.93)	0.89 (0.79)	0.9 (0.61)	0.82 (0.58)
prefix_index	0.93 (0.8)	0.96 (0.88)	0.97 (0.73)	0.85 (0.76)	0.88 (0.56)	0.79 (0.37)
prefix_laststate	0.9 (0.75)	0.94 (0.87)	0.97 (0.72)	0.88 (0.78)	0.87 (0.5)	0.84 (0.57)
prefix_agg	**0.94** (**0.88**)	0.97 (0.94)	**0.98** (0.78)	0.88 (0.78)	0.9 (0.59)	0.83 (0.58)
	bpic2015_1	**bpic2015_2**	**bpic2015_3**	**bpic2015_4**	**bpic2015_5**	**production**
single_laststate	0.83 (0.43)	0.84 (0.5)	0.76 (0.49)	0.83 (0.51)	0.84 (0.59)	0.65 (0.55)
single_agg	**0.88** (**0.73**)	**0.91** (**0.74**)	**0.91** (0.72)	0.86 (0.62)	0.88 (0.76)	0.63 (0.54)
knn_laststate	0.8 (0.44)	0.87 (0.63)	0.87 (0.67)	**0.88** (0.56)	0.85 (0.71)	0.62 (0.55)
knn_agg	0.78 (0.53)	0.85 (0.63)	0.87 (0.73)	0.87 (**0.67**)	0.85 (0.69)	0.66 (0.58)
state_laststate	0.75 (0.52)	0.84 (0.57)	0.84 (0.58)	0.85 (0.57)	0.86 (0.68)	0.62 (0.56)
state_agg	0.79 (0.64)	0.87 (0.69)	0.89 (**0.74**)	0.87 (0.66)	0.88 (0.75)	0.68 (0.58)
cluster_laststate	0.75 (0.43)	0.86 (0.61)	0.85 (0.67)	0.87 (0.63)	**0.89** (0.74)	0.6 (0.53)
cluster_agg	0.85 (0.69)	0.89 (0.73)	**0.91** (0.71)	0.87 (0.63)	0.88 (0.76)	0.74 (**0.66**)
prefix_index	0.82 (0.52)	0.85 (0.46)	0.89 (0.67)	0.85 (0.59)	0.86 (0.68)	**0.76** (0.57)
prefix_laststate	0.75 (0.37)	0.84 (0.4)	0.79 (0.44)	0.85 (0.38)	0.84 (0.61)	0.72 (0.58)
prefix_agg	0.82 (0.65)	0.87 (0.7)	0.9 (**0.74**)	0.87 (**0.67**)	0.88 (**0.77**)	0.69 (0.59)
	sepsis_1	**sepsis_2**	**sepsis_3**	**bpic2012_1**	**bpic2012_2**	**bpic2012_3**
single_laststate	**0.49** (0.01)	0.78 (**0.45**)	0.67 (0.3)	0.67 (0.63)	0.6 (0.11)	0.69 (**0.42**)
single_agg	0.41 (0.0)	0.79 (0.39)	0.66 (0.34)	**0.69** (**0.64**)	**0.61** (0.17)	**0.7** (**0.42**)
knn_laststate	0.46 (**0.05**)	0.77 (0.26)	0.64 (0.16)	0.59 (0.59)	0.58 (0.11)	0.66 (0.41)
knn_agg	0.48 (0.04)	0.74 (0.2)	0.61 (0.17)	0.64 (0.59)	0.57 (0.14)	0.67 (**0.42**)
state_laststate	0.46 (0.0)	0.79 (0.41)	**0.71** (0.26)	0.68 (0.63)	0.6 (0.16)	0.69 (0.4)
state_agg	0.43 (0.0)	0.79 (0.44)	0.7 (0.3)	0.68 (**0.64**)	0.59 (0.17)	**0.7** (**0.42**)
cluster_laststate	0.48 (0.01)	**0.8** (0.41)	**0.71** (**0.35**)	0.66 (0.61)	**0.61** (0.09)	0.68 (0.39)
cluster_agg	0.43 (0.0)	**0.8** (0.44)	0.69 (0.3)	0.67 (**0.64**)	0.59 (0.16)	0.68 (0.41)
prefix_index	0.47 (0.0)	0.75 (0.41)	**0.71** (0.19)	0.67 (0.61)	0.6 (**0.22**)	0.67 (0.39)
prefix_laststate	0.46 (0.01)	**0.8** (0.43)	**0.71** (0.21)	0.66 (0.62)	0.59 (0.14)	0.68 (0.4)
prefix_agg	0.46 (0.02)	0.76 (0.41)	0.7 (0.26)	0.68 (**0.64**)	0.59 (0.16)	**0.7** (0.41)
	bpic2017_1	**bpic2017_2**	**bpic2017_3**	**traffic**	**hospital_1**	**hospital_2**
single_laststate	**0.83** (0.7)	**0.81** (0.47)	0.79 (0.72)	0.66 (0.67)	**0.88** (**0.66**)	**0.72** (0.11)
single_agg	**0.83** (0.71)	0.8 (**0.48**)	**0.8** (**0.73**)	0.65 (0.66)	**0.88** (0.65)	0.7 (0.11)
knn_laststate	0.78 (0.65)	0.61 (0.09)	0.78 (0.67)	**0.67** (**0.7**)	0.81 (0.51)	0.58 (0.05)
knn_agg	0.78 (0.65)	0.57 (0.13)	0.76 (0.66)	0.66 (**0.7**)	0.81 (0.44)	0.56 (0.04)
state_laststate	**0.83** (0.7)	0.8 (0.46)	**0.8** (0.72)	0.65 (0.66)	**0.88** (0.65)	0.71 (**0.12**)
state_agg	**0.83** (**0.72**)	0.8 (0.47)	**0.8** (**0.73**)	0.66 (0.66)	**0.88** (0.64)	0.7 (0.07)
cluster_laststate	**0.83** (0.71)	0.8 (0.46)	**0.8** (0.72)	0.66 (0.66)	**0.88** (0.65)	0.7 (**0.12**)
cluster_agg	**0.83** (0.71)	0.79 (0.46)	**0.8** (**0.73**)	0.66 (0.66)	**0.88** (0.65)	0.69 (0.07)
prefix_index	**0.83** (**0.72**)	0.8 (0.46)	0.79 (**0.73**)	0.66 (0.66)	**0.88** (0.64)	0.69 (0.1)
prefix_laststate	**0.83** (0.7)	0.8 (0.46)	0.79 (0.72)	0.65 (0.66)	**0.88** (0.64)	0.71 (0.11)
prefix_agg	**0.83** (0.71)	0.8 (0.47)	**0.8** (**0.73**)	0.66 (0.66)	**0.88** (0.63)	0.7 (0.09)

Table 29: Overall AUC (F-score) for **logistic regression**

	bpic2011_1	**bpic2011_2**	**bpic2011_3**	**bpic2011_4**	**insurance_1**	**insurance_2**
single_laststate	0.9 (0.82)	0.9 (0.83)	0.92 (0.75)	0.88 (0.76)	0.84 (0.47)	**0.83** (0.56)
single_agg	0.92 (0.83)	0.94 (0.9)	0.96 (0.86)	0.87 (0.75)	0.79 (0.46)	0.77 (0.49)
knn_laststate	0.91 (0.79)	0.94 (**0.92**)	0.92 (0.87)	0.81 (**0.82**)	0.77 (0.49)	0.65 (0.46)
knn_agg	0.82 (0.73)	0.86 (0.86)	0.87 (0.77)	0.75 (0.72)	0.79 (0.5)	0.7 (0.57)
state_laststate	0.91 (0.8)	0.89 (0.86)	0.91 (0.76)	0.87 (0.79)	0.82 (0.48)	0.8 (**0.62**)
state_agg	0.91 (0.82)	0.94 (0.91)	0.96 (0.86)	0.85 (0.77)	0.82 (0.49)	0.75 (0.51)
cluster_laststate	0.91 (0.8)	0.9 (0.88)	0.95 (0.86)	**0.89** (0.8)	0.82 (0.53)	0.77 (0.53)
cluster_agg	**0.94** (**0.84**)	**0.95** (**0.92**)	**0.97** (**0.88**)	0.87 (0.77)	0.82 (**0.59**)	0.76 (0.58)
prefix_index	0.91 (0.81)	0.92 (0.88)	0.94 (0.83)	0.8 (0.73)	0.82 (0.55)	0.67 (0.53)
prefix_laststate	0.9 (0.79)	0.89 (0.83)	0.95 (0.82)	0.86 (0.77)	0.83 (0.5)	0.74 (0.55)
prefix_agg	0.92 (0.83)	**0.95** (0.91)	0.96 (0.79)	0.86 (0.77)	**0.85** (0.57)	0.75 (0.6)
	bpic2015_1	**bpic2015_2**	**bpic2015_3**	**bpic2015_4**	**bpic2015_5**	**production**
single_laststate	0.8 (0.5)	0.87 (0.51)	0.79 (0.45)	0.86 (0.5)	0.81 (0.58)	0.68 (0.59)
single_agg	0.81 (0.61)	0.9 (0.68)	**0.89** (0.7)	0.87 (0.51)	**0.86** (0.74)	0.67 (0.58)
knn_laststate	0.76 (0.52)	0.84 (**0.73**)	0.85 (**0.72**)	0.85 (0.57)	0.81 (0.68)	**0.71** (0.58)
knn_agg	0.75 (0.47)	0.81 (0.67)	0.84 (**0.72**)	0.81 (0.51)	0.82 (0.7)	0.57 (0.46)
state_laststate	0.75 (0.48)	0.83 (0.53)	0.81 (0.53)	0.87 (0.52)	0.82 (0.62)	0.65 (0.56)
state_agg	0.79 (0.57)	0.88 (0.67)	0.87 (0.7)	0.89 (0.64)	0.84 (0.72)	0.58 (0.38)
cluster_laststate	0.46 (0.19)	0.81 (0.58)	0.84 (0.61)	0.85 (0.63)	0.85 (0.7)	0.66 (0.59)
cluster_agg	**0.86** (0.61)	**0.91** (0.69)	0.88 (0.7)	0.86 (0.63)	0.85 (0.74)	**0.71** (0.59)
prefix_index	0.84 (0.51)	0.87 (0.64)	0.86 (0.69)	0.89 (0.57)	0.83 (0.69)	**0.71** (**0.6**)
prefix_laststate	0.73 (0.43)	0.81 (0.45)	0.79 (0.49)	0.82 (0.45)	0.83 (0.62)	0.68 (0.56)
prefix_agg	0.85 (**0.63**)	0.89 (0.69)	**0.89** (0.71)	**0.9** (**0.66**)	**0.86** (**0.76**)	0.67 (**0.6**)
	sepsis_1	**sepsis_2**	**sepsis_3**	**bpic2012_1**	**bpic2012_2**	**bpic2012_3**
single_laststate	0.43 (0.0)	0.88 (0.44)	**0.74** (0.34)	0.65 (0.57)	0.58 (0.09)	**0.7** (0.32)
single_agg	**0.57** (0.09)	0.86 (0.47)	0.73 (0.37)	0.65 (0.53)	**0.59** (0.13)	0.69 (0.3)
knn_laststate	0.43 (0.12)	0.76 (0.35)	0.58 (0.27)	0.6 (0.49)	0.54 (**0.17**)	0.64 (**0.52**)
knn_agg	0.5 (**0.13**)	0.76 (0.35)	0.59 (0.27)	0.57 (0.47)	0.55 (**0.17**)	0.6 (0.45)
state_laststate	0.47 (0.01)	**0.9** (0.43)	0.71 (0.34)	0.65 (0.56)	**0.59** (0.09)	0.69 (0.34)
state_agg	0.55 (**0.13**)	**0.9** (0.43)	0.7 (0.38)	0.66 (0.58)	**0.59** (0.13)	**0.7** (0.36)
cluster_laststate	0.47 (0.05)	0.86 (0.43)	0.67 (0.32)	0.64 (0.55)	0.58 (0.1)	0.68 (0.33)
cluster_agg	0.51 (0.11)	0.87 (0.44)	0.7 (0.35)	0.65 (0.55)	0.58 (0.13)	0.69 (0.34)
prefix_index	0.5 (0.12)	0.88 (0.48)	0.7 (0.84)	0.66 (**0.59**)	0.56 (0.16)	0.67 (0.39)
prefix_laststate	0.42 (0.11)	0.88 (0.45)	0.7 (0.86)	0.65 (0.56)	0.58 (0.09)	0.69 (0.34)
prefix_agg	0.55 (0.12)	0.87 (**0.49**)	0.73 (**0.87**)	**0.67** (0.58)	**0.59** (0.13)	0.69 (0.34)
	bpic2017_1	**bpic2017_2**	**bpic2017_3**	**traffic**	**hospital_1**	**hospital_2**
single_laststate	0.82 (0.67)	**0.81** (**0.46**)	0.79 (0.73)	0.65 (0.64)	**0.88** (0.58)	0.73 (0.05)
single_agg	**0.83** (0.67)	0.79 (0.23)	0.79 (**0.74**)	0.66 (**0.65**)	0.87 (0.6)	0.72 (0.04)
knn_laststate	0.7 (0.59)	0.62 (0.3)	0.77 (0.7)	0.6 (**0.65**)	0.78 (0.45)	0.59 (0.07)
knn_agg	0.73 (0.61)	0.63 (0.28)	0.75 (0.67)	0.63 (0.63)	0.8 (0.48)	0.57 (0.02)
state_laststate	0.82 (0.67)	0.72 (0.3)	0.79 (0.73)	0.67 (0.64)	**0.88** (**0.64**)	0.73 (0.09)
state_agg	0.82 (0.68)	0.8 (0.45)	**0.8** (**0.74**)	**0.68** (0.64)	**0.88** (0.63)	0.71 (0.09)
cluster_laststate	0.82 (0.67)	0.78 (0.41)	0.79 (**0.74**)	**0.68** (0.64)	**0.88** (0.63)	0.72 (0.08)
cluster_agg	0.81 (0.67)	0.77 (0.39)	0.79 (0.73)	**0.68** (**0.65**)	**0.88** (0.62)	0.7 (0.09)
prefix_index	0.82 (0.68)	0.78 (0.41)	0.79 (0.73)	**0.68** (0.64)	0.87 (0.57)	0.73 (**0.1**)
prefix_laststate	0.82 (0.67)	0.8 (0.45)	0.79 (**0.74**)	0.67 (0.64)	**0.88** (0.59)	**0.74** (0.08)
prefix_agg	**0.83** (**0.69**)	0.8 (0.41)	0.79 (**0.74**)	**0.68** (0.64)	**0.88** (0.55)	0.73 (0.07)

Table 30: Overall AUC (F-score) for **SVM**

	bpic2011_1	**bpic2011_2**	**bpic2011_3**	**bpic2011_4**	**insurance_1**	**insurance_2**
single_laststate	0.89 (0.65)	0.9 (0.76)	0.92 (0.0)	0.86 (0.0)	0.78 (0.38)	**0.82** (0.39)
single_agg	0.87 (0.73)	**0.95** (**0.91**)	**0.96** (0.0)	0.87 (0.35)	0.81 (0.24)	0.78 (0.42)
knn_laststate	**0.92** (0.83)	**0.95** (**0.91**)	0.93 (**0.81**)	0.76 (0.46)	0.74 (**0.46**)	0.63 (0.35)
knn_agg	0.88 (**0.85**)	0.94 (0.9)	0.92 (0.66)	0.72 (0.29)	0.77 (**0.46**)	0.7 (0.15)
state_laststate	0.88 (0.54)	0.89 (0.82)	0.86 (0.55)	0.8 (**0.62**)	0.79 (**0.46**)	0.77 (**0.46**)
state_agg	0.91 (0.75)	**0.95** (0.87)	0.94 (0.61)	0.82 (**0.62**)	0.83 (0.42)	0.73 (0.0)
cluster_laststate	0.9 (0.78)	0.89 (0.79)	0.94 (0.58)	**0.88** (0.5)	0.68 (0.23)	0.68 (0.37)
cluster_agg	**0.92** (0.75)	0.94 (0.84)	0.93 (0.78)	0.86 (0.49)	0.74 (0.27)	0.76 (0.35)
prefix_index	0.91 (0.66)	0.92 (0.77)	0.93 (0.52)	0.82 (0.36)	0.83 (0.26)	0.65 (0.0)
prefix_laststate	0.87 (0.59)	0.9 (0.81)	0.93 (0.38)	0.84 (0.0)	0.8 (0.09)	0.74 (0.41)
prefix_agg	**0.92** (0.64)	0.94 (0.89)	0.95 (0.12)	0.86 (0.0)	**0.85** (0.13)	0.77 (0.33)
	bpic2015_1	**bpic2015_2**	**bpic2015_3**	**bpic2015_4**	**bpic2015_5**	**production**
single_laststate	0.78 (0.54)	0.8 (0.5)	0.82 (0.43)	0.86 (0.41)	0.81 (0.55)	0.71 (0.6)
single_agg	**0.82** (**0.56**)	**0.88** (0.57)	**0.88** (0.67)	0.85 (**0.56**)	0.85 (0.64)	0.66 (0.54)
knn_laststate	0.72 (0.48)	0.76 (0.6)	0.8 (0.66)	0.76 (**0.56**)	0.78 (0.6)	0.6 (0.41)
knn_agg	0.7 (0.49)	0.78 (**0.62**)	0.81 (**0.68**)	0.82 (**0.56**)	0.78 (0.63)	0.63 (0.51)
state_laststate	0.68 (0.44)	0.76 (0.5)	0.75 (0.42)	0.8 (0.42)	0.8 (0.57)	0.67 (0.54)
state_agg	0.63 (0.34)	0.85 (**0.62**)	0.85 (0.53)	0.86 (0.52)	0.85 (0.57)	0.63 (0.56)
cluster_laststate	0.78 (0.37)	0.8 (0.46)	0.82 (0.61)	0.85 (0.47)	0.85 (0.57)	**0.74** (**0.62**)
cluster_agg	0.79 (0.0)	0.84 (0.0)	**0.88** (0.49)	**0.88** (0.5)	0.83 (0.17)	**0.74** (0.59)
prefix_index	0.77 (0.15)	0.83 (0.23)	0.85 (0.51)	0.86 (0.22)	0.83 (0.52)	0.72 (0.57)
prefix_laststate	0.67 (0.07)	0.69 (0.04)	0.75 (0.0)	0.75 (0.09)	0.82 (0.55)	0.67 (0.49)
prefix_agg	0.81 (0.01)	0.86 (0.53)	**0.88** (0.54)	0.86 (0.34)	**0.86** (**0.66**)	0.66 (0.53)
	sepsis_1	**sepsis_2**	**sepsis_3**	**bpic2012_1**	**bpic2012_2**	**bpic2012_3**
single_laststate	0.5 (0.0)	0.78 (**0.41**)	**0.72** (0.26)	0.63 (0.52)	0.56 (0.09)	0.68 (0.18)
single_agg	0.49 (0.0)	0.82 (0.0)	**0.72** (0.0)	0.63 (0.38)	0.55 (0.0)	**0.7** (0.2)
knn_laststate	0.47 (**0.09**)	0.7 (0.11)	0.62 (0.0)	0.57 (0.5)	0.51 (0.06)	0.59 (**0.3**)
knn_agg	0.48 (0.0)	0.71 (0.01)	0.59 (0.0)	0.63 (**0.56**)	0.53 (0.06)	0.58 (0.28)
state_laststate	0.5 (0.0)	0.75 (0.0)	0.69 (0.05)	0.63 (0.42)	0.52 (0.09)	0.61 (0.13)
state_agg	**0.54** (0.0)	0.81 (0.0)	0.65 (**0.27**)	**0.64** (0.45)	0.53 (0.09)	0.65 (0.27)
cluster_laststate	0.49 (0.0)	0.81 (0.0)	0.67 (0.0)	**0.64** (0.52)	0.54 (0.08)	0.65 (**0.3**)
cluster_agg	0.5 (0.0)	0.79 (0.4)	0.68 (0.0)	0.6 (0.28)	0.55 (**0.12**)	0.67 (0.25)
prefix_index	**0.54** (0.0)	**0.84** (0.33)	0.65 (0.0)	0.61 (0.39)	**0.57** (0.06)	0.66 (0.05)
prefix_laststate	0.46 (0.0)	0.8 (0.22)	0.66 (0.18)	**0.64** (0.46)	0.56 (0.04)	0.67 (0.23)
prefix_agg	0.51 (0.01)	0.81 (0.0)	0.68 (0.01)	**0.64** (0.31)	**0.57** (0.02)	0.67 (0.0)
	bpic2017_1	**bpic2017_2**	**bpic2017_3**	**traffic**	**hospital_1**	**hospital_2**
single_laststate	0.75 (0.14)	0.59 (0.0)	0.71 (0.67)	0.64 (0.63)	**0.85** (0.49)	0.51 (0.0)
single_agg	**0.79** (**0.67**)	0.71 (0.0)	0.64 (0.0)	0.67 (0.5)	0.73 (0.0)	**0.78** (0.0)
knn_laststate	0.55 (0.28)	0.54 (0.0)	0.68 (0.51)	0.62 (**0.69**)	0.78 (0.48)	0.5 (0.0)
knn_agg	0.63 (0.06)	0.57 (0.0)	0.71 (0.54)	0.63 (0.17)	0.61 (0.43)	0.5 (0.04)
state_laststate	0.57 (0.11)	0.52 (0.0)	0.62 (0.06)	0.65 (0.6)	0.84 (0.56)	0.57 (0.03)
state_agg	**0.79** (0.58)	0.57 (0.03)	0.55 (0.4)	0.66 (0.59)	0.84 (**0.59**)	0.57 (0.01)
cluster_laststate	0.59 (0.07)	0.56 (0.0)	**0.77** (**0.68**)	0.66 (0.58)	0.82 (0.57)	0.61 (0.07)
cluster_agg	0.64 (0.13)	0.73 (0.0)	0.73 (0.58)	**0.73** (0.58)	0.82 (0.54)	0.59 (0.0)
prefix_index	0.68 (0.0)	0.73 (0.28)	0.68 (0.0)	0.59 (0.27)	0.82 (0.46)	0.59 (0.0)
prefix_laststate	0.76 (0.34)	**0.77** (**0.46**)	0.74 (0.58)	0.65 (0.6)	**0.85** (0.52)	0.54 (0.02)
prefix_agg	0.6 (0.01)	0.73 (0.28)	0.73 (0.0)	0.65 (0.6)	0.84 (0.52)	0.66 (**0.09**)

Table 31: Execution times for **random forest**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	42.88 ± 0.78	76 ± 108	38.81 ± 0.27	67 ± 106	34.12 ± 0.17	78 ± 107
single_agg	51.92 ± 0.85	76 ± 109	369.75 ± 2.56	68 ± 107	57.62 ± 1.04	79 ± 109
knn_laststate	6.24 ± 0.24	168 ± 251	9.75 ± 0.44	143 ± 228	4.15 ± 0.09	169 ± 257
knn_agg	6.14 ± 0.12	176 ± 264	10.2 ± 0.35	151 ± 240	4.22 ± 0.22	190 ± 288
state_laststate	112.57 ± 0.39	**58 ± 80**	145.86 ± 0.49	**53 ± 82**	81.7 ± 0.19	**60 ± 79**
state_agg	265.71 ± 0.58	69 ± 95	226.93 ± 0.82	64 ± 100	153.58 ± 0.14	72 ± 95
cluster_laststate	35.88 ± 0.23	73 ± 122	73.3 ± 0.18	56 ± 105	42.61 ± 0.12	64 ± 114
cluster_agg	211.72 ± 0.86	76 ± 124	439.57 ± 0.68	73 ± 122	58.2 ± 0.1	80 ± 122
prefix_index	410.74 ± 5.03	126 ± 79	465.87 ± 10.13	127 ± 71	331.3 ± 12.36	121 ± 75
prefix_laststate	98.34 ± 0.22	66 ± 98	129.95 ± 0.5	57 ± 93	61.51 ± 0.07	69 ± 97
prefix_agg	173.76 ± 0.29	70 ± 101	189.64 ± 0.57	61 ± 95	122.13 ± 0.15	73 ± 100

	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	50.65 ± 0.21	68 ± 106	22.29 ± 0.54	26 ± 40	60.39 ± 0.53	24 ± 38
single_agg	135.39 ± 7.21	68 ± 108	102.42 ± 0.65	26 ± 41	86.86 ± 0.26	24 ± 40
knn_laststate	11.42 ± 2.19	151 ± 236	8.35 ± 0.48	126 ± 226	11.01 ± 0.16	116 ± 225
knn_agg	9.3 ± 0.2	152 ± 237	9.33 ± 0.13	138 ± 240	11.24 ± 0.54	115 ± 225
state_laststate	144.21 ± 4.36	**54 ± 83**	107.37 ± 0.18	31 ± 54	100.87 ± 0.27	33 ± 55
state_agg	227.05 ± 4.78	65 ± 100	135.72 ± 0.47	34 ± 57	132.69 ± 0.32	37 ± 58
cluster_laststate	428.27 ± 8.7	75 ± 121	40.05 ± 0.58	43 ± 69	55.96 ± 2.26	40 ± 66
cluster_agg	125.97 ± 0.29	**54 ± 87**	52.37 ± 0.83	30 ± 46	73.58 ± 2.67	32 ± 52
prefix_index	1121.44 ± 25.31	126 ± 71	504.84 ± 0.61	62 ± 12	1137.08 ± 2.07	60 ± 14
prefix_laststate	196.68 ± 0.02	57 ± 93	70.06 ± 0.24	**14 ± 19**	75.49 ± 0.13	**12 ± 16**
prefix_agg	306.88 ± 0.64	61 ± 95	97.72 ± 0.08	17 ± 21	159.75 ± 0.25	15 ± 18

	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	138.37 ± 2.95	26 ± 42	20.86 ± 0.6	24 ± 36	37.74 ± 0.9	22 ± 35
single_agg	95.39 ± 0.2	27 ± 43	70.51 ± 0.32	25 ± 38	105.67 ± 0.64	23 ± 36
knn_laststate	19.94 ± 0.64	113 ± 226	7.66 ± 0.02	120 ± 228	15.2 ± 0.33	115 ± 221
knn_agg	19.15 ± 0.68	124 ± 239	7.77 ± 0.35	127 ± 237	18.78 ± 0.24	122 ± 231
state_laststate	147.65 ± 0.1	35 ± 59	89.11 ± 0.16	31 ± 51	114.35 ± 0.4	29 ± 48
state_agg	192.76 ± 0.73	39 ± 62	93.86 ± 0.31	34 ± 54	157.16 ± 6.82	32 ± 52
cluster_laststate	121.76 ± 0.41	43 ± 73	41.47 ± 0.19	37 ± 72	72.07 ± 0.35	38 ± 66
cluster_agg	73.09 ± 0.4	31 ± 49	31.47 ± 0.27	35 ± 55	64.21 ± 0.58	31 ± 49
prefix_index	1531.4 ± 15.22	71 ± 17	176.56 ± 0.22	51 ± 10	366.0 ± 2.63	60 ± 13
prefix_laststate	100.27 ± 0.28	**13 ± 18**	62.28 ± 0.02	**12 ± 17**	99.77 ± 0.71	**12 ± 17**
prefix_agg	172.12 ± 4.21	16 ± 19	94.56 ± 0.16	15 ± 18	156.44 ± 3.28	14 ± 18

	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	2.99 ± 0.06	43 ± 37	12.51 ± 0.09	55 ± 44	9.65 ± 0.09	47 ± 42
single_agg	8.5 ± 0.05	47 ± 40	17.82 ± 0.12	57 ± 47	11.85 ± 0.08	48 ± 45
knn_laststate	0.89 ± 0.05	307 ± 338	0.61 ± 0.03	204 ± 206	0.89 ± 0.04	217 ± 173
knn_agg	1.04 ± 0.01	338 ± 333	0.65 ± 0.04	223 ± 228	0.94 ± 0.04	224 ± 178
state_laststate	17.34 ± 0.07	**41 ± 34**	14.36 ± 0.04	**48 ± 35**	16.63 ± 0.01	**40 ± 34**
state_agg	17.2 ± 0.21	47 ± 40	16.88 ± 0.12	57 ± 44	15.36 ± 0.18	49 ± 43
cluster_laststate	18.89 ± 0.26	45 ± 43	10.9 ± 0.1	50 ± 50	19.69 ± 0.06	48 ± 44
cluster_agg	15.18 ± 0.13	49 ± 47	24.89 ± 0.09	50 ± 49	31.68 ± 0.1	51 ± 48
prefix_index	24.93 ± 0.11	70 ± 27	27.69 ± 0.22	107 ± 10	28.91 ± 0.4	103 ± 11
prefix_laststate	18.32 ± 0.89	**41 ± 36**	15.16 ± 0.0	51 ± 37	22.93 ± 0.1	43 ± 37
prefix_agg	26.86 ± 0.08	57 ± 48	16.36 ± 0.03	**48 ± 35**	24.44 ± 0.38	**40 ± 33**

Table 32: Execution times for **random forest** (continued)

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	87.17 ± 2.69	38 ± 43	10.85 ± 0.19	**46 ± 45**	8.81 ± 0.03	40 ± 44
single_agg	43.23 ± 0.31	39 ± 45	14.38 ± 0.14	49 ± 48	46.89 ± 0.22	43 ± 46
knn_laststate	2.78 ± 0.06	242 ± 271	0.99 ± 0.01	305 ± 301	1.9 ± 0.05	261 ± 279
knn_agg	2.79 ± 0.06	255 ± 282	1.04 ± 0.05	307 ± 305	1.89 ± 0.04	272 ± 288
state_laststate	71.76 ± 0.18	41 ± 46	23.6 ± 0.06	50 ± 50	22.51 ± 0.33	43 ± 46
state_agg	140.28 ± 0.2	42 ± 47	19.23 ± 0.03	51 ± 51	123.09 ± 0.17	45 ± 48
cluster_laststate	41.22 ± 0.15	**36 ± 44**	25.36 ± 0.21	49 ± 48	36.94 ± 0.26	41 ± 45
cluster_agg	77.48 ± 0.11	38 ± 45	81.31 ± 0.18	50 ± 50	47.18 ± 0.08	43 ± 46
prefix_index	165.12 ± 0.03	51 ± 39	114.1 ± 0.09	58 ± 42	137.86 ± 0.18	52 ± 39
prefix_laststate	78.81 ± 0.18	39 ± 44	44.0 ± 0.02	48 ± 46	51.21 ± 0.25	41 ± 44
prefix_agg	141.0 ± 0.06	37 ± 41	57.12 ± 0.03	**46 ± 44**	73.75 ± 0.01	**39 ± 42**
	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	258.83 ± 4.05	13 ± 20	113.69 ± 1.3	13 ± 20	221.25 ± 0.69	13 ± 20
single_agg	395.73 ± 2.98	14 ± 21	358.35 ± 1.11	13 ± 21	389.96 ± 7.22	14 ± 21
knn_laststate	29.02 ± 4.34	107 ± 213	30.21 ± 0.59	501 ± 593	27.96 ± 0.55	491 ± 577
knn_agg	29.96 ± 0.59	506 ± 592	30.54 ± 0.6	509 ± 599	29.9 ± 0.59	513 ± 605
state_laststate	139.79 ± 1.06	14 ± 19	215.16 ± 0.66	14 ± 19	129.01 ± 0.82	14 ± 19
state_agg	697.14 ± 6.11	15 ± 21	547.46 ± 4.22	15 ± 20	394.35 ± 6.53	15 ± 21
cluster_laststate	147.92 ± 5.76	15 ± 23	125.69 ± 0.15	16 ± 24	175.72 ± 2.12	15 ± 22
cluster_agg	535.28 ± 10.75	16 ± 25	196.31 ± 2.84	16 ± 24	276.72 ± 3.67	15 ± 23
prefix_index	7198.38 ± 70.16	43 ± 10	8059.82 ± 147.53	41 ± 9	4076.35 ± 70.84	43 ± 10
prefix_laststate	277.58 ± 0.87	**11 ± 13**	253.66 ± 0.28	**10 ± 12**	265.56 ± 0.7	**10 ± 12**
prefix_agg	1326.15 ± 4.76	**11 ± 14**	548.0 ± 1.36	11 ± 14	833.38 ± 3.9	12 ± 15
	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	1289.72 ± 25.31	26 ± 32	1625.87 ± 31.91	30 ± 36	3172.95 ± 62.27	26 ± 31
single_agg	2880.68 ± 56.53	31 ± 38	4371.05 ± 85.78	27 ± 33	8649.74 ± 169.75	28 ± 34
knn_laststate	143.14 ± 2.81	1757 ± 1700	138.09 ± 2.71	1792 ± 1727	117.39 ± 2.3	1532 ± 1475
knn_agg	136.12 ± 2.67	1784 ± 1715	132.3 ± 2.6	1642 ± 1582	130.03 ± 2.55	1679 ± 1612
state_laststate	1239.01 ± 24.32	25 ± 30	1247.27 ± 24.48	29 ± 35	795.83 ± 15.62	25 ± 30
state_agg	12366.43 ± 242.69	31 ± 34	5671.91 ± 111.31	33 ± 39	13165.58 ± 258.37	27 ± 31
cluster_laststate	2325.38 ± 45.64	**24 ± 31**	1018.82 ± 19.99	25 ± 29	1367.58 ± 26.84	**23 ± 28**
cluster_agg	1535.54 ± 30.14	26 ± 34	3979.78 ± 78.1	27 ± 32	1728.18 ± 33.92	25 ± 31
prefix_index	25283.44 ± 496.19	88 ± 12	22481.59 ± 441.2	91 ± 12	19949.41 ± 391.51	86 ± 14
prefix_laststate	2270.56 ± 29.0	25 ± 28	789.43 ± 15.49	**22 ± 24**	4245.86 ± 60.81	26 ± 30
prefix_agg	5933.72 ± 36.32	27 ± 32	5003.87 ± 98.2	25 ± 28	10723.16 ± 41.97	**23 ± 27**
	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	2553.51 ± 175.77	**86 ± 48**	5508.58 ± 108.11	401 ± 263	2974.02 ± 58.37	456 ± 300
single_agg	2253.99 ± 12.52	94 ± 53	11667.12 ± 228.97	470 ± 309	131453.83 ± 2579.78	411 ± 271
knn_laststate	424.08 ± 56.11	543 ± 392	123.42 ± 2.42	463 ± 362	368.05 ± 7.22	435 ± 401
knn_agg	439.44 ± 59.76	560 ± 404	115.82 ± 2.27	497 ± 387	362.69 ± 7.12	453 ± 419
state_laststate	1222.57 ± 25.28	94 ± 53	1329.31 ± 26.09	371 ± 298	1491.67 ± 29.27	**309 ± 255**
state_agg	1362.58 ± 48.94	97 ± 54	959.94 ± 18.84	414 ± 270	1663.75 ± 32.65	370 ± 236
cluster_laststate	1129.09 ± 44.03	90 ± 52	2402.26 ± 47.14	**341 ± 305**	1794.1 ± 35.21	332 ± 324
cluster_agg	1261.5 ± 3.47	96 ± 55	5956.26 ± 116.89	465 ± 305	1267.46 ± 24.87	393 ± 295
prefix_index	2051.23 ± 27.11	116 ± 26	3566.4 ± 69.99	890 ± 90	6309.37 ± 123.82	930 ± 174
prefix_laststate	1365.6 ± 8.31	91 ± 50	1950.52 ± 38.28	402 ± 281	2085.55 ± 16.28	337 ± 235
prefix_agg	1601.71 ± 11.23	99 ± 55	17359.15 ± 340.67	431 ± 251	7652.09 ± 2.19	374 ± 219

Table 33: Execution times for **logistic regression**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	8.14 ± 0.29	68 ± 97	11.09 ± 0.4	61 ± 96	5.55 ± 0.13	70 ± 96
single_agg	11.85 ± 0.39	69 ± 99	16.89 ± 0.17	62 ± 98	6.58 ± 0.05	71 ± 97
knn_laststate	5.69 ± 0.03	**29 ± 42**	9.86 ± 0.22	**23 ± 37**	4.12 ± 0.12	**29 ± 41**
knn_agg	5.93 ± 0.14	32 ± 46	9.69 ± 0.05	26 ± 42	4.28 ± 0.13	**29 ± 43**
state_laststate	8.64 ± 0.1	51 ± 70	11.97 ± 0.15	47 ± 73	6.57 ± 0.01	52 ± 68
state_agg	11.49 ± 0.05	66 ± 91	15.76 ± 0.59	57 ± 88	7.77 ± 0.01	61 ± 80
cluster_laststate	19.61 ± 0.1	56 ± 102	27.12 ± 0.79	53 ± 100	16.11 ± 0.07	55 ± 102
cluster_agg	19.1 ± 0.7	53 ± 76	27.15 ± 0.2	58 ± 109	13.86 ± 0.21	65 ± 107
prefix_index	28.51 ± 0.71	122 ± 71	44.66 ± 2.52	124 ± 65	18.37 ± 0.5	106 ± 62
prefix_laststate	7.52 ± 0.13	58 ± 87	10.34 ± 0.12	51 ± 83	5.32 ± 0.07	60 ± 85
prefix_agg	9.28 ± 0.11	61 ± 88	13.22 ± 0.14	54 ± 83	7.34 ± 0.08	64 ± 86

	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	11.13 ± 0.71	62 ± 97	7.97 ± 0.29	20 ± 30	7.04 ± 0.05	18 ± 29
single_agg	20.21 ± 2.74	62 ± 98	16.57 ± 0.41	22 ± 34	48.14 ± 0.66	20 ± 32
knn_laststate	8.55 ± 0.43	**26 ± 40**	8.98 ± 0.07	23 ± 37	10.59 ± 0.04	24 ± 39
knn_agg	8.56 ± 0.36	**26 ± 41**	8.16 ± 0.27	24 ± 39	11.9 ± 0.11	26 ± 45
state_laststate	11.71 ± 0.08	47 ± 72	8.52 ± 0.2	26 ± 45	9.65 ± 0.2	27 ± 46
state_agg	15.1 ± 0.04	55 ± 85	9.8 ± 0.36	29 ± 48	11.07 ± 0.08	31 ± 49
cluster_laststate	25.99 ± 0.08	54 ± 106	24.07 ± 0.07	22 ± 35	37.74 ± 0.67	36 ± 61
cluster_agg	26.49 ± 0.23	68 ± 112	19.51 ± 0.27	33 ± 52	27.18 ± 0.34	27 ± 45
prefix_index	41.41 ± 0.35	118 ± 60	27.99 ± 0.6	56 ± 12	38.47 ± 0.45	54 ± 15
prefix_laststate	10.81 ± 0.11	51 ± 82	7.09 ± 0.06	**7 ± 9**	7.67 ± 0.28	**6 ± 6**
prefix_agg	14.14 ± 0.08	54 ± 84	7.53 ± 0.03	8 ± 9	9.77 ± 0.07	7 ± 7

	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	38.41 ± 0.18	21 ± 33	6.57 ± 0.09	18 ± 27	18.67 ± 0.14	17 ± 26
single_agg	29.69 ± 0.72	21 ± 34	9.17 ± 0.24	18 ± 29	24.9 ± 0.24	17 ± 27
knn_laststate	19.6 ± 0.62	26 ± 42	7.6 ± 0.34	21 ± 35	14.0 ± 0.14	20 ± 33
knn_agg	18.5 ± 0.67	31 ± 49	7.76 ± 0.36	23 ± 38	14.56 ± 0.54	26 ± 42
state_laststate	14.56 ± 0.3	30 ± 50	7.17 ± 0.08	26 ± 42	12.4 ± 0.09	23 ± 39
state_agg	18.28 ± 0.07	33 ± 53	8.37 ± 0.09	29 ± 45	14.39 ± 0.3	26 ± 42
cluster_laststate	60.56 ± 1.47	36 ± 63	27.27 ± 0.18	34 ± 61	38.51 ± 0.45	18 ± 31
cluster_agg	44.96 ± 0.59	26 ± 41	26.35 ± 0.24	33 ± 60	48.48 ± 0.84	25 ± 40
prefix_index	65.17 ± 0.35	64 ± 19	25.3 ± 0.3	43 ± 8	46.49 ± 0.74	53 ± 13
prefix_laststate	12.02 ± 0.08	**7 ± 8**	6.37 ± 0.07	**6 ± 7**	11.19 ± 0.02	**6 ± 7**
prefix_agg	17.16 ± 0.1	8 ± 8	6.65 ± 0.04	7 ± 7	12.4 ± 0.13	7 ± 7

	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	0.89 ± 0.01	25 ± 21	1.41 ± 0.01	37 ± 30	2.88 ± 0.01	32 ± 29
single_agg	1.11 ± 0.13	28 ± 25	1.04 ± 0.05	39 ± 33	2.68 ± 0.09	34 ± 32
knn_laststate	0.87 ± 0.03	31 ± 28	0.62 ± 0.03	**20 ± 18**	0.92 ± 0.01	**24 ± 16**
knn_agg	0.93 ± 0.05	31 ± 31	0.63 ± 0.01	24 ± 21	0.89 ± 0.01	27 ± 18
state_laststate	1.47 ± 0.12	**23 ± 19**	1.29 ± 0.04	31 ± 22	2.43 ± 0.01	27 ± 21
state_agg	1.68 ± 0.06	30 ± 26	1.54 ± 0.03	43 ± 32	2.03 ± 0.02	37 ± 32
cluster_laststate	5.16 ± 0.08	28 ± 29	4.96 ± 0.08	32 ± 31	5.34 ± 0.13	34 ± 31
cluster_agg	5.72 ± 0.05	33 ± 34	5.83 ± 0.11	27 ± 28	8.5 ± 0.11	36 ± 34
prefix_index	3.03 ± 0.08	51 ± 10	2.89 ± 0.02	89 ± 5	3.86 ± 0.03	90 ± 4
prefix_laststate	1.15 ± 0.01	**23 ± 19**	1.26 ± 0.0	33 ± 23	2.33 ± 0.01	28 ± 22
prefix_agg	1.67 ± 0.0	34 ± 28	1.29 ± 0.01	32 ± 22	1.75 ± 0.01	27 ± 21

Table 34: Execution times for **logistic regression** (continued)

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	3.21 ± 0.15	27 ± 31	1.31 ± 0.03	**33 ± 33**	2.3 ± 0.04	29 ± 31
single_agg	21.44 ± 0.28	29 ± 33	1.48 ± 0.03	35 ± 35	2.54 ± 0.04	31 ± 33
knn_laststate	2.82 ± 0.06	28 ± 32	0.98 ± 0.04	35 ± 35	1.97 ± 0.05	31 ± 33
knn_agg	2.88 ± 0.11	31 ± 34	1.05 ± 0.06	40 ± 39	1.85 ± 0.06	34 ± 36
state_laststate	5.74 ± 0.17	29 ± 33	1.88 ± 0.03	36 ± 36	3.04 ± 0.02	31 ± 34
state_agg	22.54 ± 0.11	32 ± 36	2.1 ± 0.02	39 ± 39	3.18 ± 0.02	35 ± 37
cluster_laststate	9.51 ± 0.11	**26 ± 32**	5.52 ± 0.18	35 ± 34	6.52 ± 0.15	31 ± 34
cluster_agg	11.73 ± 0.04	27 ± 33	4.98 ± 0.09	38 ± 38	7.58 ± 0.03	30 ± 34
prefix_index	16.74 ± 0.07	39 ± 26	3.14 ± 0.02	45 ± 28	6.1 ± 0.02	40 ± 26
prefix_laststate	4.8 ± 0.07	27 ± 30	2.1 ± 0.07	**33 ± 32**	3.25 ± 0.03	**28 ± 30**
prefix_agg	8.01 ± 0.07	28 ± 32	2.27 ± 0.05	35 ± 34	3.64 ± 0.06	30 ± 32

	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	30.51 ± 0.38	7 ± 10	29.15 ± 0.79	7 ± 10	30.31 ± 1.76	7 ± 10
single_agg	31.1 ± 0.27	8 ± 12	60.4 ± 3.11	8 ± 12	74.28 ± 0.31	8 ± 12
knn_laststate	30.24 ± 0.38	55 ± 62	28.85 ± 0.08	27 ± 30	27.84 ± 0.83	74 ± 82
knn_agg	29.06 ± 0.46	350 ± 374	28.32 ± 0.08	310 ± 333	31.05 ± 2.0	192 ± 208
state_laststate	24.21 ± 0.29	8 ± 10	24.41 ± 0.1	8 ± 10	23.68 ± 0.37	8 ± 10
state_agg	29.84 ± 0.39	10 ± 12	28.8 ± 1.11	9 ± 11	30.25 ± 0.17	9 ± 12
cluster_laststate	38.2 ± 0.11	10 ± 15	37.28 ± 0.57	9 ± 13	39.4 ± 0.72	9 ± 13
cluster_agg	108.75 ± 1.51	10 ± 17	64.86 ± 2.16	9 ± 14	79.46 ± 6.91	9 ± 14
prefix_index	67.45 ± 2.49	36 ± 12	65.61 ± 1.1	34 ± 11	62.86 ± 1.03	36 ± 12
prefix_laststate	26.0 ± 0.87	**4 ± 3**	25.84 ± 0.39	**4 ± 3**	23.42 ± 0.3	**4 ± 3**
prefix_agg	38.74 ± 0.91	5 ± 5	42.84 ± 0.41	5 ± 5	47.5 ± 0.28	5 ± 5

	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	88.86 ± 1.74	19 ± 23	155.13 ± 3.04	19 ± 23	107.82 ± 2.12	18 ± 21
single_agg	213.25 ± 4.18	21 ± 26	143.2 ± 2.81	21 ± 27	209.74 ± 4.12	19 ± 23
knn_laststate	137.49 ± 2.7	1575 ± 1484	129.35 ± 2.54	1518 ± 1426	135.54 ± 2.66	1567 ± 1473
knn_agg	127.0 ± 2.49	1474 ± 1387	129.12 ± 2.53	1524 ± 1432	121.37 ± 2.38	1413 ± 1326
state_laststate	92.96 ± 1.82	19 ± 20	74.44 ± 1.46	18 ± 24	84.76 ± 1.66	17 ± 20
state_agg	472.57 ± 9.27	20 ± 22	244.76 ± 4.8	22 ± 24	114.88 ± 2.25	19 ± 22
cluster_laststate	154.87 ± 3.04	21 ± 25	172.13 ± 3.38	**15 ± 18**	104.51 ± 2.05	16 ± 19
cluster_agg	368.51 ± 7.23	19 ± 24	431.13 ± 8.46	17 ± 21	163.87 ± 3.22	16 ± 20
prefix_index	539.24 ± 10.58	72 ± 9	512.71 ± 10.06	72 ± 9	340.34 ± 6.68	72 ± 9
prefix_laststate	95.98 ± 0.95	**15 ± 16**	112.1 ± 5.26	**15 ± 16**	75.28 ± 0.25	**14 ± 15**
prefix_agg	281.04 ± 0.84	**15 ± 17**	551.86 ± 26.98	17 ± 20	325.93 ± 0.41	16 ± 17

	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	95.22 ± 0.54	**62 ± 34**	171.88 ± 3.37	381 ± 250	105.77 ± 2.08	395 ± 259
single_agg	273.55 ± 1.81	66 ± 37	361.41 ± 7.09	405 ± 267	269.08 ± 5.28	397 ± 262
knn_laststate	384.89 ± 5.26	75 ± 42	108.0 ± 0.13	**43 ± 24**	361.86 ± 7.1	**88 ± 64**
knn_agg	361.35 ± 8.72	81 ± 51	110.14 ± 1.45	78 ± 54	352.76 ± 6.92	94 ± 65
state_laststate	68.11 ± 0.53	66 ± 37	274.97 ± 5.4	349 ± 287	144.61 ± 2.84	334 ± 282
state_agg	151.29 ± 4.43	75 ± 42	274.89 ± 5.39	368 ± 230	308.15 ± 6.05	402 ± 249
cluster_laststate	93.99 ± 6.84	66 ± 39	225.33 ± 4.42	330 ± 284	124.98 ± 2.45	280 ± 244
cluster_agg	111.74 ± 2.5	68 ± 39	259.91 ± 5.1	386 ± 264	681.42 ± 13.37	369 ± 246
prefix_index	331.76 ± 1.14	89 ± 11	263.88 ± 5.18	964 ± 112	229.74 ± 4.51	848 ± 100
prefix_laststate	80.76 ± 0.95	**62 ± 34**	122.85 ± 1.04	363 ± 262	112.29 ± 0.47	313 ± 223
prefix_agg	186.94 ± 2.77	63 ± 35	166.22 ± 5.34	394 ± 232	173.75 ± 0.52	353 ± 204

Table 35: Execution times for **SVM**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	503.86 ± 1.75	70 ± 100	457.58 ± 4.1	64 ± 100	173.03 ± 1.95	73 ± 99
single_agg	40.4 ± 0.35	69 ± 100	381.13 ± 0.35	63 ± 100	156.46 ± 0.44	73 ± 100
knn_laststate	6.0 ± 0.22	**29 ± 42**	9.79 ± 0.44	**27 ± 42**	4.26 ± 0.23	**29 ± 41**
knn_agg	5.91 ± 0.26	**29 ± 42**	9.98 ± 0.28	28 ± 44	4.25 ± 0.25	31 ± 44
state_laststate	12.44 ± 0.07	52 ± 72	19.77 ± 0.13	49 ± 75	9.52 ± 0.04	53 ± 70
state_agg	16.44 ± 0.08	61 ± 84	21.67 ± 0.04	58 ± 90	10.44 ± 0.08	63 ± 83
cluster_laststate	35.72 ± 0.22	62 ± 110	44.74 ± 0.14	45 ± 75	18.37 ± 0.28	48 ± 76
cluster_agg	30.12 ± 0.1	57 ± 99	46.53 ± 0.29	56 ± 106	21.49 ± 0.11	50 ± 74
prefix_index	39.94 ± 0.49	123 ± 72	64.98 ± 3.13	125 ± 65	25.33 ± 0.46	107 ± 62
prefix_laststate	16.31 ± 0.17	60 ± 90	23.17 ± 0.1	52 ± 84	11.99 ± 0.04	62 ± 87
prefix_agg	17.82 ± 0.08	62 ± 89	24.89 ± 0.04	54 ± 84	11.54 ± 0.1	65 ± 88

	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	549.47 ± 23.94	65 ± 102	86.12 ± 0.31	21 ± 33	111.39 ± 0.15	20 ± 32
single_agg	130.1 ± 15.04	63 ± 99	68.34 ± 6.06	22 ± 34	88.97 ± 1.6	21 ± 33
knn_laststate	8.93 ± 0.43	**23 ± 35**	8.32 ± 0.43	23 ± 37	11.31 ± 0.37	23 ± 39
knn_agg	8.84 ± 0.41	26 ± 40	8.4 ± 0.33	21 ± 38	11.4 ± 0.34	18 ± 28
state_laststate	18.98 ± 0.03	48 ± 74	8.87 ± 0.11	26 ± 45	9.74 ± 0.07	28 ± 46
state_agg	25.72 ± 0.83	58 ± 89	10.33 ± 0.21	28 ± 47	11.49 ± 0.15	31 ± 48
cluster_laststate	46.62 ± 0.44	51 ± 103	22.02 ± 0.1	22 ± 34	29.91 ± 0.11	25 ± 41
cluster_agg	43.22 ± 0.12	49 ± 92	25.47 ± 0.13	25 ± 39	33.13 ± 0.22	25 ± 40
prefix_index	60.85 ± 0.68	119 ± 60	34.55 ± 0.4	52 ± 12	41.09 ± 0.26	52 ± 14
prefix_laststate	21.82 ± 0.04	52 ± 85	7.45 ± 0.05	**7 ± 9**	9.34 ± 0.1	**6 ± 6**
prefix_agg	22.69 ± 0.19	54 ± 85	9.16 ± 0.12	9 ± 10	10.71 ± 0.04	8 ± 7

	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	363.33 ± 16.46	21 ± 32	45.01 ± 0.31	17 ± 27	567.21 ± 11.07	16 ± 25
single_agg	178.69 ± 12.55	21 ± 34	28.26 ± 0.59	18 ± 28	144.11 ± 1.39	17 ± 27
knn_laststate	18.82 ± 0.62	21 ± 37	7.6 ± 0.05	20 ± 35	14.34 ± 0.32	19 ± 32
knn_agg	19.6 ± 0.81	26 ± 45	7.07 ± 0.11	23 ± 38	14.37 ± 0.04	19 ± 34
state_laststate	15.56 ± 0.21	30 ± 50	7.3 ± 0.05	26 ± 43	13.02 ± 0.16	23 ± 39
state_agg	19.63 ± 0.09	34 ± 54	9.29 ± 0.07	29 ± 46	16.39 ± 0.11	27 ± 43
cluster_laststate	51.08 ± 0.33	25 ± 39	20.86 ± 0.1	21 ± 33	43.79 ± 0.08	21 ± 35
cluster_agg	52.3 ± 1.32	25 ± 40	19.61 ± 0.3	24 ± 37	140.02 ± 1.08	25 ± 41
prefix_index	77.94 ± 0.32	61 ± 18	27.1 ± 0.09	42 ± 8	66.81 ± 1.85	50 ± 12
prefix_laststate	16.14 ± 0.1	**7 ± 8**	6.4 ± 0.12	**6 ± 7**	13.36 ± 0.18	**6 ± 7**
prefix_agg	22.12 ± 0.07	9 ± 9	9.14 ± 0.03	10 ± 10	22.26 ± 0.07	10 ± 10

	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	1.46 ± 0.05	26 ± 22	4.42 ± 0.05	38 ± 30	8.06 ± 0.05	33 ± 30
single_agg	1.48 ± 0.05	28 ± 24	3.63 ± 0.05	39 ± 33	7.16 ± 0.09	34 ± 32
knn_laststate	0.88 ± 0.05	26 ± 25	0.63 ± 0.02	**19 ± 17**	0.9 ± 0.02	**22 ± 15**
knn_agg	0.93 ± 0.05	33 ± 31	0.64 ± 0.01	22 ± 44	1.02 ± 0.07	27 ± 18
state_laststate	1.32 ± 0.06	**22 ± 19**	1.64 ± 0.03	31 ± 22	2.37 ± 0.01	27 ± 21
state_agg	1.67 ± 0.1	30 ± 26	1.78 ± 0.03	43 ± 33	3.01 ± 0.02	37 ± 32
cluster_laststate	4.84 ± 0.14	31 ± 29	5.48 ± 0.09	24 ± 25	6.56 ± 0.07	27 ± 26
cluster_agg	5.04 ± 0.13	37 ± 34	5.37 ± 0.1	39 ± 37	6.83 ± 0.03	30 ± 26
prefix_index	3.5 ± 0.01	61 ± 12	2.85 ± 0.03	89 ± 4	5.98 ± 0.04	89 ± 4
prefix_laststate	1.17 ± 0.01	23 ± 19	1.8 ± 0.0	33 ± 23	2.45 ± 0.01	28 ± 22
prefix_agg	1.56 ± 0.18	31 ± 26	1.55 ± 0.23	32 ± 23	2.12 ± 0.02	26 ± 21

Table 36: Execution times for **SVM** (continued)

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	62.48 ± 0.54	28 ± 32	6.46 ± 0.02	34 ± 34	12.88 ± 0.09	30 ± 32
single_agg	15.52 ± 0.22	29 ± 33	5.81 ± 0.06	36 ± 36	14.88 ± 0.08	31 ± 34
knn_laststate	3.14 ± 0.07	29 ± 33	1.06 ± 0.01	38 ± 37	1.91 ± 0.06	31 ± 33
knn_agg	2.72 ± 0.12	30 ± 34	1.02 ± 0.02	37 ± 37	1.82 ± 0.03	33 ± 35
state_laststate	5.14 ± 0.09	29 ± 33	2.08 ± 0.02	35 ± 36	4.23 ± 0.02	31 ± 34
state_agg	9.5 ± 0.16	33 ± 37	2.61 ± 0.03	40 ± 40	5.08 ± 0.02	35 ± 38
cluster_laststate	10.03 ± 0.09	**26 ± 32**	6.02 ± 0.11	34 ± 34	7.7 ± 0.07	**29 ± 32**
cluster_agg	10.37 ± 0.09	28 ± 34	6.84 ± 0.03	**33 ± 35**	7.95 ± 0.06	31 ± 34
prefix_index	10.45 ± 0.03	36 ± 24	3.16 ± 0.01	41 ± 26	6.67 ± 0.05	38 ± 25
prefix_laststate	6.63 ± 0.02	28 ± 31	2.49 ± 0.03	34 ± 33	4.78 ± 0.04	30 ± 32
prefix_agg	6.61 ± 0.09	30 ± 33	2.85 ± 0.04	36 ± 35	4.96 ± 0.02	31 ± 34

	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	12067.81 ± 399.76	10 ± 14	14557.12 ± 780.58	8 ± 11	2746.74 ± 133.03	9 ± 13
single_agg	3143.14 ± 24.75	10 ± 16	1336.26 ± 5.35	9 ± 14	3940.96 ± 508.12	10 ± 15
knn_laststate	28.92 ± 0.47	279 ± 299	31.47 ± 0.55	254 ± 274	30.7 ± 2.47	174 ± 188
knn_agg	33.21 ± 1.1	258 ± 277	29.0 ± 0.93	297 ± 319	31.11 ± 0.58	171 ± 189
state_laststate	108.84 ± 0.29	8 ± 10	156.77 ± 2.77	8 ± 10	128.66 ± 0.22	8 ± 10
state_agg	168.48 ± 1.15	10 ± 12	88.35 ± 1.29	10 ± 12	191.19 ± 3.22	10 ± 13
cluster_laststate	414.04 ± 3.83	9 ± 14	139.3 ± 0.17	8 ± 13	181.63 ± 2.76	8 ± 12
cluster_agg	186.63 ± 2.21	8 ± 9	2409.21 ± 27.71	11 ± 18	273.16 ± 6.91	10 ± 15
prefix_index	538.77 ± 6.5	37 ± 12	420.91 ± 13.0	34 ± 11	344.04 ± 8.18	34 ± 11
prefix_laststate	77.87 ± 0.94	**4 ± 3**	211.0 ± 9.02	**4 ± 3**	100.44 ± 2.24	**5 ± 4**
prefix_agg	100.43 ± 0.25	5 ± 5	288.14 ± 4.56	5 ± 5	66.23 ± 0.75	**5 ± 4**

	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	86469.6 ± 1696.97	38 ± 46	91457.8 ± 1923.76	42 ± 56	64770.83 ± 1271.13	32 ± 39
single_agg	89619.4 ± 1736.87	40 ± 47	23877.21 ± 468.59	30 ± 38	68407.03 ± 1323.42	35 ± 42
knn_laststate	117.67 ± 2.31	118 ± 108	133.43 ± 2.62	1515 ± 1424	119.49 ± 2.35	1375 ± 1292
knn_agg	122.36 ± 2.4	1408 ± 1332	118.44 ± 2.32	1412 ± 1329	139.45 ± 2.74	1587 ± 1497
state_laststate	18343.13 ± 359.98	23 ± 26	1905.91 ± 37.4	20 ± 24	37822.2 ± 742.26	17 ± 23
state_agg	10257.06 ± 201.29	25 ± 28	58203.39 ± 1142.24	23 ± 26	10121.8 ± 198.64	24 ± 26
cluster_laststate	8744.81 ± 171.62	17 ± 22	7006.73 ± 137.51	17 ± 21	3026.05 ± 59.39	18 ± 22
cluster_agg	74691.13 ± 1465.81	22 ± 27	8903.74 ± 174.74	23 ± 28	5264.25 ± 103.31	18 ± 21
prefix_index	17933.49 ± 351.94	76 ± 9	39670.6 ± 778.54	87 ± 10	24417.21 ± 479.19	89 ± 11
prefix_laststate	16596.88 ± 1364.46	**16 ± 17**	3553.0 ± 3.84	**14 ± 15**	7667.05 ± 53.16	**15 ± 16**
prefix_agg	21941.34 ± 37.13	**16 ± 18**	5395.36 ± 587.31	16 ± 18	4509.12 ± 372.24	18 ± 20

	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	52218.27 ± 1024.78	103 ± 57	163070.85 ± 3200.27	487 ± 319	100195.84 ± 1966.34	482 ± 311
single_agg	58867.16 ± 1155.27	114 ± 62	65398.88 ± 1283.45	436 ± 281	291765.77 ± 5725.9	545 ± 353
knn_laststate	394.55 ± 7.74	69 ± 40	109.75 ± 1.32	81 ± 55	312.82 ± 6.14	**62 ± 49**
knn_agg	482.25 ± 9.46	96 ± 54	102.69 ± 2.02	**42 ± 23**	321.04 ± 6.3	85 ± 68
state_laststate	31579.01 ± 1040.68	78 ± 48	33549.22 ± 658.4	320 ± 268	110178.73 ± 2162.26	365 ± 307
state_agg	71510.15 ± 1403.39	87 ± 53	31458.64 ± 617.38	359 ± 222	18274.1 ± 358.63	385 ± 236
cluster_laststate	7466.95 ± 149.95	72 ± 46	23090.77 ± 453.16	313 ± 292	19699.97 ± 386.61	285 ± 255
cluster_agg	97876.03 ± 1920.82	**45 ± 40**	46246.86 ± 907.59	324 ± 239	22553.25 ± 442.61	325 ± 242
prefix_index	28902.06 ± 145.18	125 ± 20	53322.05 ± 1046.45	1093 ± 132	82296.52 ± 1615.07	1013 ± 214
prefix_laststate	12316.56 ± 14.1	72 ± 43	39203.83 ± 1030.35	360 ± 255	76378.26 ± 118.46	344 ± 244
prefix_agg	24735.14 ± 425.56	75 ± 45	52526.59 ± 857.39	393 ± 234	57426.99 ± 22.74	364 ± 215

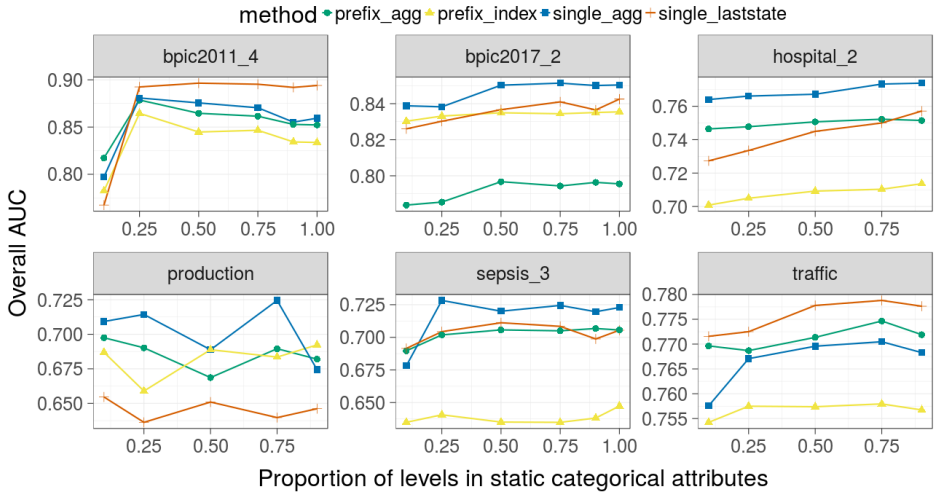


Figure 50: AUC across different filtering proportions of **static** categorical attribute levels (XGBoost)

Table 37: Execution times for **RF** with unstructured data.

	DR		**LtC**		**Github**	
single_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	19.83 ± 0.58	0.21 ± 0.07	1648.58 ± 82.5	0.09 ± 0.11	1043.73 ± 40.63	**0.07 ± 0.05**
BoNG	29.66 ± 0.84	0.37 ± 0.12	22761.75 ± 456.53	0.11 ± 0.13	2286.96 ± 241.26	0.56 ± 0.34
NB	81.03 ± 1.46	0.23 ± 0.07	18208.0 ± 0.0	0.11 ± 0.13	2456.88 ± 189.07	0.14 ± 0.09
LDA	134.16 ± 2.14	0.22 ± 0.07	10303.79 ± 482.55	**0.05 ± 0.06**	2325.06 ± 21.31	0.14 ± 0.08
PV	34.4 ± 8.0	0.25 ± 0.27	3855.07 ± 174.3	0.11 ± 0.79	945.45 ± 17.32	0.21 ± 0.24
	DR		**LtC**		**Github**	
single_laststate	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	**14.06 ± 0.43**	0.2 ± 0.06	**950.4 ± 25.6**	0.08 ± 0.1	**251.42 ± 16.26**	**0.07 ± 0.05**
BoNG	38.35 ± 2.09	0.23 ± 0.07	3499.5 ± 302.86	0.08 ± 0.1	1424.13 ± 33.39	0.53 ± 0.33
NB	34.92 ± 1.76	0.22 ± 0.07	3779.14 ± 258.75	**0.05 ± 0.06**	1035.07 ± 52.62	0.08 ± 0.05
LDA	85.16 ± 5.79	0.22 ± 0.07	62506.71 ± 2453.04	**0.05 ± 0.06**	1756.59 ± 27.88	**0.07 ± 0.05**
PV	103.6 ± 12.36	0.32 ± 0.46	1554.43 ± 138.73	0.07 ± 0.83	792.51 ± 10.35	0.16 ± 0.26
	DR		**LtC**		**Github**	
prefix_index_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	25.98 ± 0.52	0.21 ± 0.06	2016.05 ± 247.85	0.19 ± 0.06	265.76 ± 3.44	0.08 ± 0.05
BoNG	71.38 ± 2.52	0.26 ± 0.1	12098.83 ± 1956.86	0.22 ± 0.1	4931.93 ± 15.19	0.27 ± 0.08
NB	135.0 ± 0.78	0.25 ± 0.08	8087.82 ± 1378.65	0.19 ± 0.06	3253.13 ± 9.82	0.16 ± 0.13
LDA	585.43 ± 1.69	**0.13 ± 0.05**	67104.63 ± 8767.19	0.21 ± 0.08	137211.56 ± 68.25	0.15 ± 0.09
PV	685.96 ± 25.44	0.5 ± 4.14	21406.83 ± 4621.16	0.28 ± 1.49	6624.45 ± 150.46	0.18 ± 0.26
	DR		**LtC**		**Github**	
prefix_index_last	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	25.98 ± 0.52	0.21 ± 0.06	2016.05 ± 247.85	0.19 ± 0.06	265.76 ± 3.44	0.08 ± 0.05
BoNG	47.35 ± 0.64	0.25 ± 0.11	7743.85 ± 1267.65	0.21 ± 0.07	1558.44 ± 6.77	0.24 ± 0.1
NB	56.77 ± 0.6	**0.13 ± 0.05**	8385.81 ± 326.55	0.2 ± 0.07	1709.9 ± 69.61	0.09 ± 0.07
LDA	292.75 ± 2.71	**0.13 ± 0.04**	59552.84 ± 9298.44	0.2 ± 0.06	10590.26 ± 24.73	0.08 ± 0.05
PV	502.11 ± 69.74	0.4 ± 5.03	18544.0 ± 4078.02	0.28 ± 1.37	2821.29 ± 52.41	0.17 ± 0.26

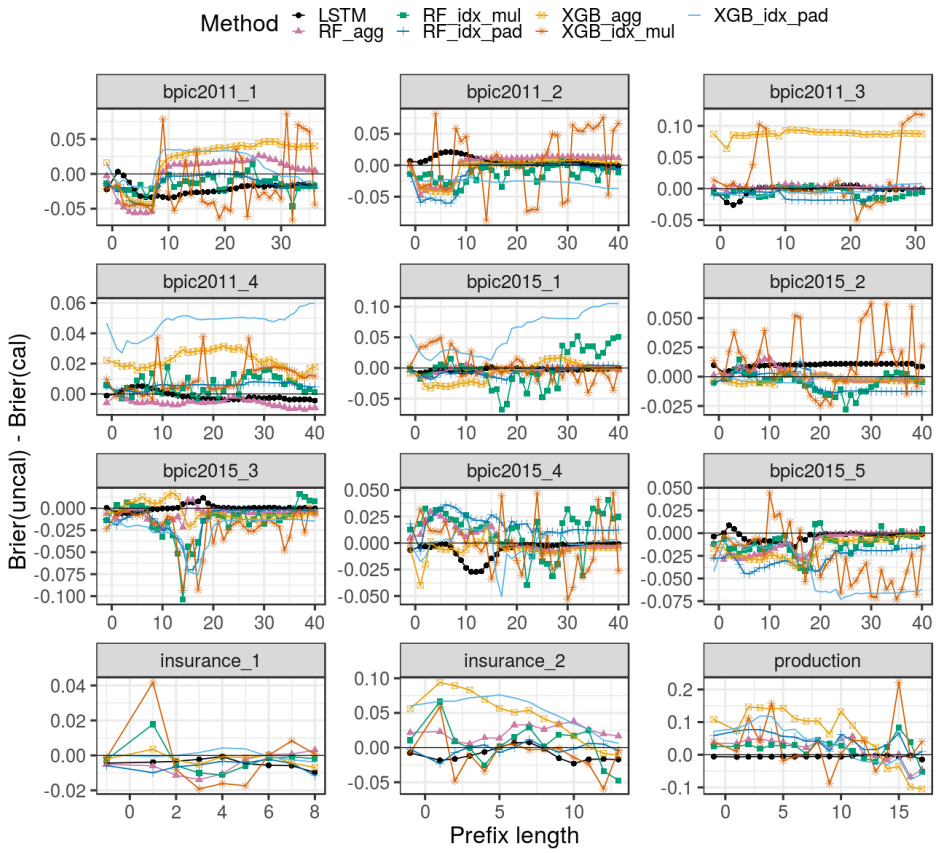


Figure 51: Differences in Brier scores on uncalibrated vs. calibrated classifiers over different prefix lengths. Positive scores show that calibration (Platt scaling) helped to make the classifier better calibrated.

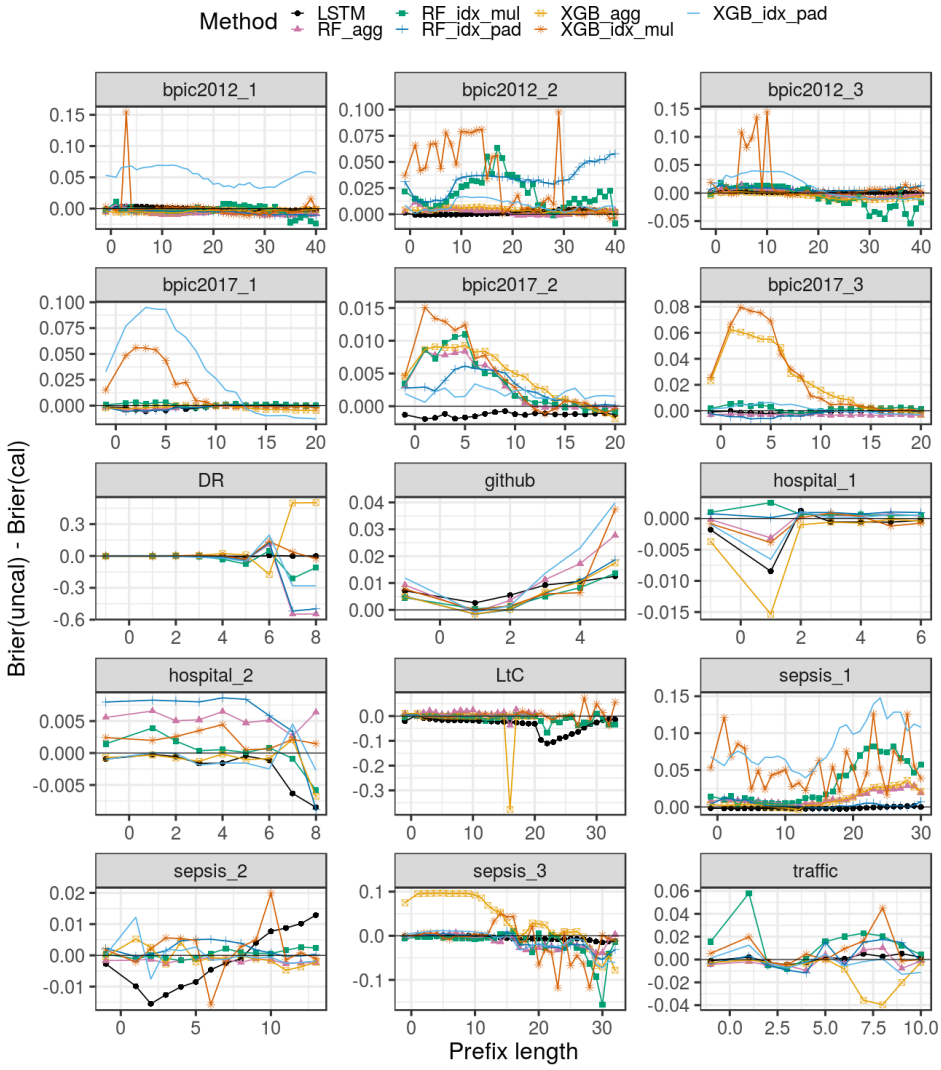


Figure 52: Differences in Brier scores (continued).

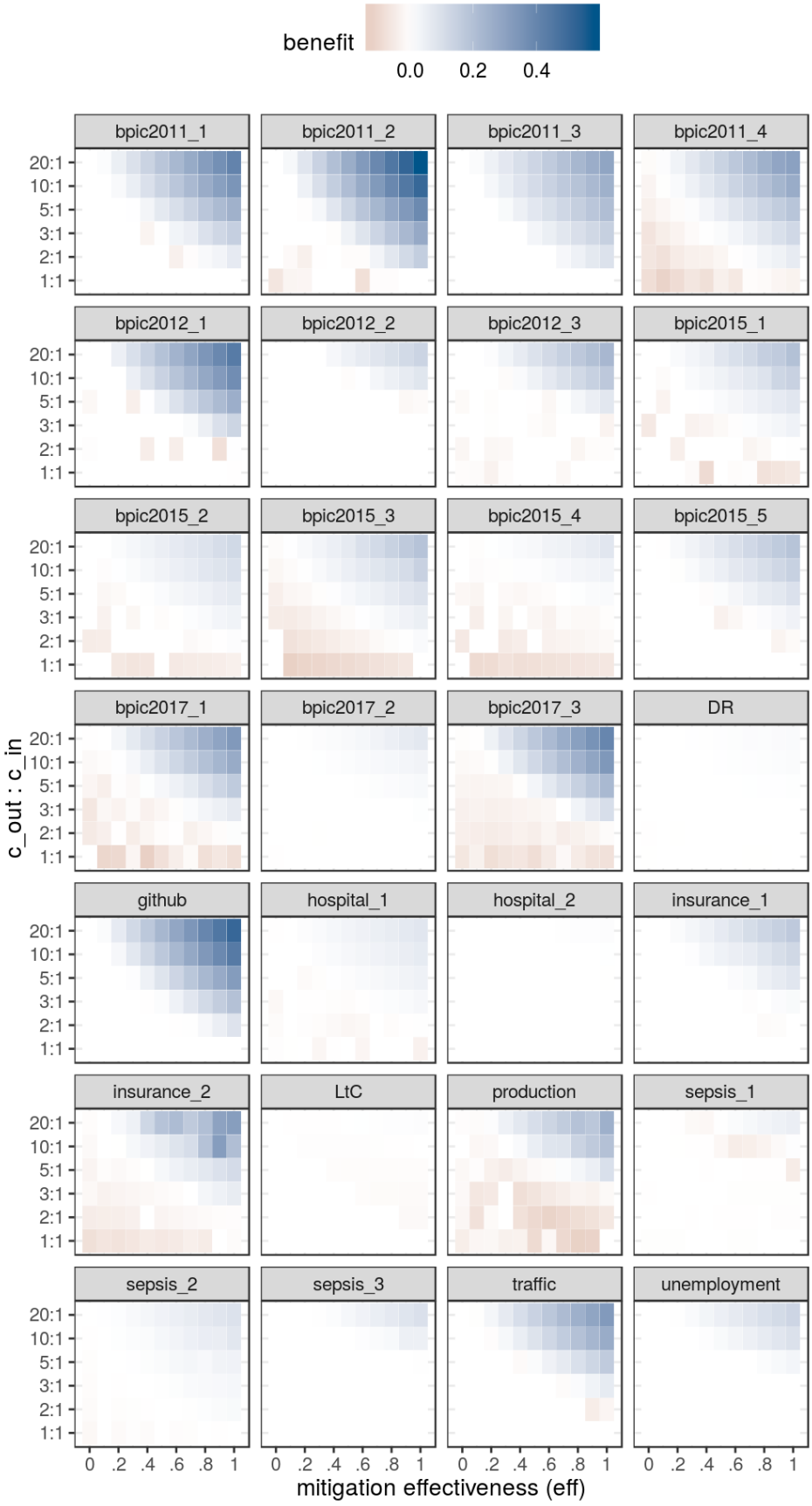


Figure 53: Benefit of the alarm system, varying $eff(k, \sigma, L)$ with a linear decay.

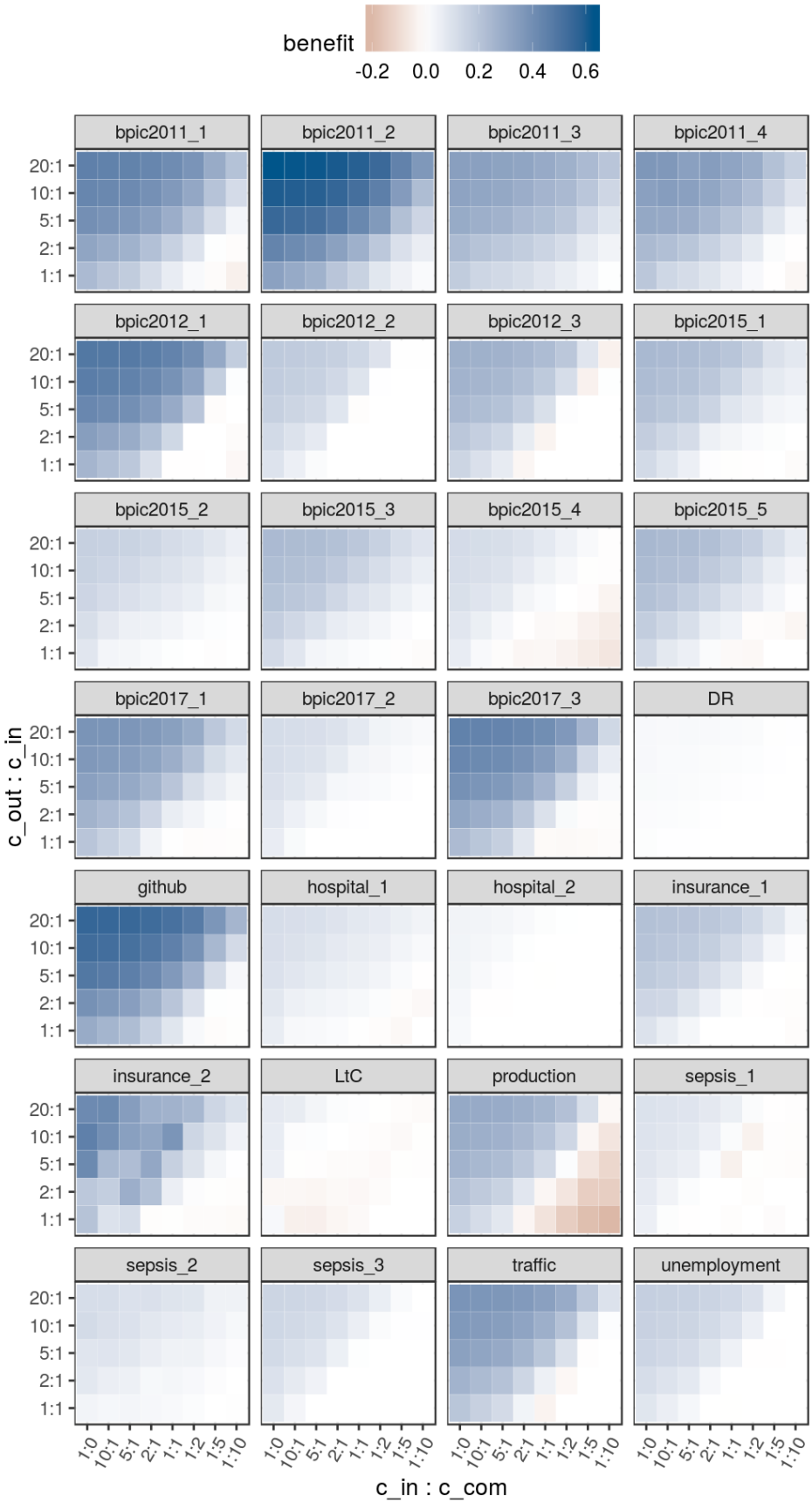


Figure 54: Benefit of the alarm system, varying $c_{com}(\sigma, L)$; $c_{in}(k, \sigma, L)$ is increasing linearly from $1/|\sigma|$ to 1.

Table 38: Execution times for **logit** with unstructured data.

	DR		LtC		Github	
single_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	**4.64 ± 0.09**	**0.02 ± 0.01**	1135.82 ± 2.18	**0.02 ± 0.02**	327.34 ± 4.34	0.01 ± 0.01
BoNG	30.29 ± 1.17	0.18 ± 0.06	12411.58 ± 0.0	0.09 ± 0.11	1143.93 ± 15.48	0.42 ± 0.25
NB	20.9 ± 0.14	0.05 ± 0.02	6047.49 ± 1285.66	0.05 ± 0.06	899.01 ± 8.47	0.02 ± 0.03
LDA	55.24 ± 0.76	0.04 ± 0.01	2493.09 ± 36.36	**0.02 ± 0.03**	6325.6 ± 11.63	0.01 ± 0.01
PV	49.78 ± 3.24	0.06 ± 0.09	2892.62 ± 67.85	0.05 ± 0.52	1480.71 ± 23.16	0.11 ± 0.21
	DR		**LtC**		**Github**	
single_laststate	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	5.51 ± 0.09	**0.02 ± 0.01**	**233.54 ± 0.63**	**0.02 ± 0.02**	**39.91 ± 0.46**	**0.0 ± 0.0**
BoNG	10.5 ± 0.16	0.04 ± 0.01	1481.22 ± 20.49	0.06 ± 0.07	548.76 ± 3.59	0.39 ± 0.23
NB	16.88 ± 0.29	0.04 ± 0.01	1539.13 ± 150.79	**0.02 ± 0.03**	560.12 ± 2.21	0.01 ± 0.01
LDA	87.36 ± 0.23	0.03 ± 0.01	70902.31 ± 466.62	**0.02 ± 0.03**	11261.53 ± 26.37	0.01 ± 0.01
PV	22.36 ± 2.27	0.13 ± 0.45	685.1 ± 29.85	0.06 ± 0.37	663.12 ± 23.53	0.12 ± 0.21
	DR		**LtC**		**Github**	
prefix_index_agg	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	6.44 ± 0.04	0.03 ± 0.01	546.52 ± 25.14	0.09 ± 0.01	54.78 ± 1.42	0.01 ± 0.0
BoNG	65.83 ± 0.61	0.09 ± 0.02	7020.3 ± 63.33	0.12 ± 0.02	1857.56 ± 30.7	0.2 ± 0.12
NB	64.31 ± 0.63	0.05 ± 0.01	6577.28 ± 51.08	0.09 ± 0.01	1785.06 ± 19.56	0.02 ± 0.02
LDA	481.8 ± 2.41	0.05 ± 0.01	66009.73 ± 6023.06	0.09 ± 0.01	42738.68 ± 741.82	0.02 ± 0.01
PV	154.51 ± 5.81	0.33 ± 3.49	17449.61 ± 3765.84	0.16 ± 0.08	3015.11 ± 134.45	0.12 ± 0.21
	DR		**LtC**		**Github**	
prefix_index_last	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
no text	6.44 ± 0.04	0.03 ± 0.01	546.52 ± 25.14	0.09 ± 0.01	54.78 ± 1.42	0.01 ± 0.0
BoNG	38.29 ± 0.65	0.06 ± 0.03	5865.15 ± 10.65	0.12 ± 0.02	1161.78 ± 49.62	0.36 ± 0.35
NB	40.73 ± 0.64	0.03 ± 0.01	5470.95 ± 48.09	0.09 ± 0.01	2048.33 ± 117.08	0.02 ± 0.01
LDA	406.3 ± 5.15	0.03 ± 0.02	73917.56 ± 9302.16	0.09 ± 0.01	64183.27 ± 1736.86	0.01 ± 0.01
PV	281.64 ± 73.6	0.3 ± 2.89	20248.29 ± 6224.33	0.21 ± 1.23	3139.05 ± 86.97	0.12 ± 0.21

Table 39: Overall Brier scores for uncalibrated and calibrated classifiers. Best scores for each classifier are marked in bold.

	RF_agg		XGB_agg		RF_idx_mul		XGB_idx_mul		RF_idx_pad		XGB_idx_pad		LSTM	
	uncal	cal	uncal	cal	uncal	cal	uncal	cal	uncal	cal	uncal	cal	uncal	cal
bpic2011_1	**0.11**	**0.11**	0.13	**0.11**	**0.11**	0.12	0.13	0.15	0.13	0.15	0.15	0.13	0.14	0.16
bpic2011_2	**0.05**	**0.05**	**0.05**	0.06	0.12	0.13	0.2	0.2	0.12	0.13	0.13	0.16	0.14	0.13
bpic2011_3	0.04	0.04	0.12	**0.03**	0.06	0.07	0.08	0.07	0.06	0.07	0.06	0.06	0.13	0.14
bpic2011_4	**0.13**	0.14	0.18	0.16	0.16	0.15	0.17	0.16	0.16	0.15	0.21	0.16	0.16	0.16
bpic2015_1	**0.09**	0.1	**0.09**	0.1	0.15	0.16	0.15	0.15	0.14	0.15	0.19	0.13	0.12	0.13
bpic2015_2	0.06	0.06	**0.05**	**0.05**	0.09	0.1	0.12	0.11	0.08	0.08	0.09	0.09	0.14	0.13
bpic2015_3	**0.07**	**0.07**	0.08	0.08	0.09	0.1	0.1	0.12	0.08	0.1	0.09	0.12	0.09	0.09
bpic2015_4	0.07	0.07	0.07	0.07	0.1	0.09	0.08	0.08	0.1	0.08	0.1	0.1	**0.06**	**0.06**
bpic2015_5	**0.08**	0.09	**0.08**	0.1	0.11	0.12	0.13	0.15	0.11	0.14	0.13	0.17	0.09	0.1
production	0.27	0.23	0.35	0.24	0.26	0.23	0.28	0.24	0.29	0.24	0.29	**0.22**	0.28	0.29
insurance_1	**0.09**	0.1	0.12	0.12	0.11	0.11	0.12	0.12	0.12	0.13	0.14	0.14	0.14	0.15
insurance_2	0.23	0.21	0.29	0.23	0.25	0.24	**0.2**	0.21	0.27	0.28	0.34	0.28	0.25	0.26
sepsis_1	0.12	0.12	0.12	0.11	0.12	**0.1**	0.15	**0.1**	0.12	0.11	0.17	**0.1**	**0.1**	**0.1**
sepsis_2	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	**0.06**	0.07	0.07
sepsis_3	**0.14**	0.15	0.22	0.15	**0.14**	0.15	0.15	0.15	0.15	0.15	0.16	0.16	0.16	0.16
bpic2012_1	**0.21**	0.22	**0.21**	**0.21**	0.22	0.22	0.22	0.22	0.23	0.24	0.29	0.24	0.22	0.22
bpic2012_2	**0.15**	**0.15**	**0.15**	**0.15**	0.17	**0.15**	0.18	**0.15**	0.19	0.16	0.16	0.16	**0.15**	**0.15**
bpic2012_3	0.17	**0.16**	**0.16**	0.17	0.18	0.18	0.18	**0.16**	0.19	0.18	0.2	0.19	**0.16**	**0.16**
bpic2017_1	**0.12**	**0.12**	0.13	0.13	**0.12**	**0.12**	0.14	0.13	0.13	0.13	0.21	0.17	**0.12**	**0.12**
bpic2017_2	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.09	0.09	**0.07**	0.08
bpic2017_3	**0.15**	**0.15**	0.19	0.16	**0.15**	**0.15**	0.18	0.16	**0.15**	0.16	0.17	0.17	**0.15**	**0.15**
traffic	0.19	0.2	0.19	0.19	0.2	**0.18**	0.19	**0.18**	0.2	0.21	0.2	0.2	**0.18**	**0.18**
hospital_1	**0.03**	**0.03**	**0.03**	0.04	**0.03**	**0.03**	**0.03**	**0.03**	0.04	**0.03**	0.04	0.04	0.04	0.04
hospital_2	0.05	**0.04**	**0.04**	**0.04**	**0.04**	**0.04**	**0.04**	**0.04**	0.05	**0.04**	**0.04**	**0.04**	**0.04**	**0.04**
DR	0.02	**0.01**	0.02	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
github	0.21	**0.2**	**0.2**	**0.2**	0.21	**0.2**	0.21	**0.2**	0.21	**0.2**	0.21	**0.2**	0.23	0.23
LtC	0.12	**0.11**	0.12	**0.11**	**0.11**	0.12	**0.11**	**0.11**	**0.11**	**0.11**	**0.11**	**0.11**	0.12	0.14

ACKNOWLEDGEMENT

I am very grateful to my supervisors Marlon Dumas and Fabrizio Maria Maggi for guiding me in my research and for their constant support and feedback throughout the years.

I would also like to express my gratitude to my thesis reviewers Myra Spiliopoulou, Donato Malerba, and Meelis Kull for their positive feedback and insightful comments that helped me to improve the thesis further.

I would like to thank all my co-authors for helping me in fine-tuning the ideas and executing them into research papers. I am very thankful to Marcello la Rosa and Massimiliano de Leoni for their time while hosting me in their research groups at Queensland University of Technology and Eindhoven University of Technology.

I am grateful to my friends and family without whom I cannot imagine going through the PhD years. I would like to give a special thanks to Anna Leontjeva, for always being up for the idea-generating and pair-programming sessions. To Karl-Oskar Masing for always being there for me when needed and providing the very necessary moral support. And to Niek Tax, for convincing me that anything is possible and for motivating me to aim higher; your constant encouragement and positivity have been key factors in helping me finish writing this thesis.

I would also like to acknowledge the Estonian Research Council, the Doctoral School of Information and Communication Technology (IKTDK), and the European Regional Development Fund for funding my studies. The experiments provided in the thesis were performed using the resources from the University of Tartu High Performance Computing Center.

SISUKOKKUVÕTE

Äriprotsessi tulemuste ennustav ja korralduslik seire

Viimastel aastatel on erinevates valdkondades tegutsevad ettevõtted üles näidanud kasvavat huvi masinõppel põhinevate rakenduste kasutusele võtmiseks. Muuhulgas otsitakse võimalusi oma äriprotsesside efektiivsuse tõstmiseks, kasutades ennustusmudeleid protsesside jooksvaks seireks. Sellised ennustava protsessiseire meetodid võtavad sisendiks sündmuslogi, mis koosneb hulgast lõpetatud äriprotsessi juhtumite sündmusjadadest, ning kasutavad masinõppe algoritme ennustusmudelite treenimiseks. Saadud mudelid teevad ennustusi lõpetamata (antud ajahetkel aktiivsete) protsessijuhtumite jaoks, võttes sisendiks sündmuste jada, mis selle hetkeni on toimunud ning ennustades kas järgmist sündmust antud juhtumis, juhtumi lõppemiseni jäänud aega või instantsi lõpptulemust. Lõpptulemusele orienteeritud ennustava protsessiseire meetodid keskenduvad ennustamisele, kas protsessijuhtum lõppeb soovitud või ebasoovitava lõpptulemusega. On oluline, et sellised ennustussüsteemid teeksid ennustusi võimalikult täpselt ning võimalikult varajastes protsessistaadiumites, teisisõnu, võttes sisendiks võimalikult vähe sündmusi. Süsteemi kasutaja saab ennustuste alusel otsustada, kas sekkuda antud protsessijuhtumisse või mitte, eesmärgiga ära hoida ebasoovitavat lõpptulemust või leevendada selle negatiivseid tagajärgi. Erinevalt puhtalt ennustavatest süsteemidest annavad korralduslikud protsessiseire meetodid kasutajale ka soovitusi, kas ja kuidas antud juhtumisse sekkuda, eesmärgiga optimeerida mingit kindlat kasulikkusfunktsiooni.

Käesolev doktoritöö uurib, kuidas treenida, hinnata ja kasutada ennustusmudeleid äriprotsesside lõpptulemuste ennustava ja korraldusliku seire raames. Kuigi kirjanduses eksisteerib mitmeid ennustava protsessiseire meetodeid, on eri autorid kasutanud erinevat terminoloogiat, katsete ülesehitust, andmestikke ja võrdlusaluseid, mistõttu puudub selge ülevaade, kuidas antud meetodid üksteisega kõrvutuvad. Selle tühimiku täitmiseks analüüsib antud doktoritöö olemasolevaid ennustava protsessiseire meetodeid ning pakub välja taksonoomia nende klassifitseerimiseks. Lisaks võrdleb doktoritöö olemasolevaid meetodeid katseliselt, võttes aluseks 24 ennustusülesannet, mis põhinevad üheksal reaalsel sündmuslogil. Eksperimentide tulemused panevad kahtluse alla varasema hüpoteesi, mille kohaselt on soovitatav treenida iga võimaliku jada pikkuse jaoks eraldi klassifitseerija, kasutades kadudeta (indeksipõhist) jada kodeerimist, eelistades seda võrrelduna kadudega (agregeeritud) kodeerimise ning üheainsa klassifitseerija treenimisega.

Varasemate meetodite analüüsist selgub, et olemasolevad meetodid keskenduvad vaid struktureeritud andmetele, jättes tähelepanuta struktureerimata (tekstilised) andmed, mis reaalsetes sündmuslogides tihtipeale esinevad. Käesolev doktoritöö täidab antud tühimiku, pakkudes välja raamistiku, mis kasutab tekstikaeve tehnikaid, ekstraheerimaks tunnuseid tekstilistest andmetest, ning kombineerib need struktureeritud andmetest saadud tunnustega, eesmärgiga treenida täpsemaid

ennustusmudeleid. Katsed näitavad, et lihtne sõnahulga-põhine kodeering töötab enamasti paremini kui teised tekstikaeve meetodid.

Ennustava protsessiseire tehnikate kvaliteeti mõõdetakse enamasti ennustuste täpsuse ning varasuse seisukohast, eirates ennustuste stabiilsust olukorras, kus ennustused on tehtud sama protsessijuhtumi jaoks, kasutades erinevat hulka sündmusi. Selle tühimiku täitmiseks pakub käesolev doktoritöö välja ennustuste ajalise stabiilsuse mõiste ning võrdleb olemasolevaid ennustava protsessiseire tehnikaid antud mõõdiku alusel. Tulemuste kohaselt on kõige stabiilsemad rekurrentsetel närvivõrkudel ning otsustuspuude võimendamisel põhinevad klassifitseerijad (LSTM ja XGBoost).

Varasemad teadustööd ennustava protsessiseire vallas keskenduvad ennustuste genereerimisele, kuid ei anna kasutajale soovitusi nende ennustuste kasutamise osas või eeldavad, et kasutaja määrab ise ennustusskoori suuruse osas lävendi, mille ületamisel saadab süsteem häire. Käesolev doktoritöö pakub välja raamistiku häirepõhiseks korralduslikuks protsessiseireks, mis leiab optimaalse lävendi empiiriliselt, võttes arvesse äriprotsessiga seonduvaid kulusid, nt protsessijuhtumisse sekkumise kulu, ebasoovitava lõpptulemuse kulu ning ebasoovitava lõpptulemuse leevendamise efektiivsust, juhul kui protsessi sekkutakse. Katsed näitavad, et väljapakutud lähenemine võimaldab järjepidevalt leida lävendeid, mis minimeerivad protsessijuhtumite töötlemise kogukulusid.

CURRICULUM VITAE

Personal data

Name:	Irene Teinemaa
Date of Birth:	22.08.1990
Citizenship:	Estonian

Education

2015–2019	University of Tartu, Faculty of Science and Technology, doctoral studies, specialty: Computer Science.
2012–2014	University of Tartu, Faculty of Mathematics and Computer Science, master's studies, specialty: Software Engineering.
2009–2012	University of Tartu, Faculty of Law, bachelor's studies, specialty: Law.

Employment

2018	University of Tartu, Data Analyst
2013–2017	STACC, Research Engineer
2015	University of Tartu, Teaching Assistant
2014–2015	Teleport Inc., Data Scientist

Scientific work

Main fields of interest:

- machine learning
- data mining
- process mining

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Irene Teinemaa
Sünniaeg:	22.08.1990
Kodakondsus:	Eesti

Haridus

2015–2019	Tartu Ülikool, Loodus- ja täppisteaduste valdkond, doktoriõpe, eriala: Informaatika.
2012–2014	Tartu Ülikool, Matemaatika-informaatikateaduskond, magistriõpe, eriala: Tarkvaratehnika.
2009–2012	Tartu Ülikool, Õigusteaduskond, bakalaureuseõpe, eriala: Õigusteadus.

Teenistuskäik

2018	Tartu Ülikool, andmeanalüütik
2013–2017	STACC, teadur
2015	Tartu Ülikool, praktikumijuhendaja
2014–2015	Teleport Inc., andmeteadlane

Teadustegevus

Peamised uurimisvaldkonnad:

- masinõpe
- andmekaeve
- protsessikaeve

LIST OF ORIGINAL PUBLICATIONS

Publications in the scope of the thesis

I Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Transactions on Knowledge Discovery from Data*, 2018. To appear

Lead author. The author performed the implementation and the analysis of the experiments, most of the literature review, and contributed substantially to the ideas and the writing.

II Irene Teinemaa, Marlon Dumas, Fabrizio Maria 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

Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

III Irene Teinemaa, Marlon Dumas, Anna Leontjeva, and Fabrizio Maria Maggi. Temporal stability in predictive process monitoring. *Data Min. Knowl. Discov.*, 32(5):1306–1338, 2018

Lead author. The author performed the implementation and the analysis of the experiments, and contributed substantially to the ideas and the writing.

IV Irene Teinemaa, Niek Tax, Massimiliano de Leoni, Marlon Dumas, and Fabrizio Maria 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

Lead author. The author performed the implementation and the analysis of the experiments and contributed substantially to the ideas and the writing.

Publications out of the scope of the thesis

1. Irene Teinemaa, Anna Leontjeva, Marlon Dumas, and Riivo Kikas. Community-based prediction of activity change in skype. In Jian Pei, Fabrizio Silvestri, and Jie Tang, editors, *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, pages 73–80. ACM, 2015

2. Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi, and Irene Teinemaa. Clustering-based predictive process monitoring. *IEEE Transactions on Services Computing*, 2016. To appear
3. Diego Calvanese, Marlon Dumas, Ülari Laurson, Fabrizio Maria Maggi, Marco Montali, and Irene Teinemaa. Semantics and analysis of DMN decision tables. 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 217–233. Springer, 2016
4. Diego Calvanese, Marlon Dumas, Ülari Laurson, Fabrizio Maria Maggi, Marco Montali, and Irene Teinemaa. Semantics, analysis and simplification of DMN decision tables. *Inf. Syst.*, 78:112–125, 2018
5. 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
6. Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Irene Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *CoRR*, abs/1805.02896, 2018
7. Niek Tax, Irene Teinemaa, and Sebastiaan J. van Zelst. An interdisciplinary comparison of sequence modeling methods for next-element prediction. *CoRR*, abs/1811.00062, 2018

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

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 multi-party 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.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.