Fire Now, Fire Later: Alarm-Based Systems for Prescriptive Process Monitoring

Stephan A. Fahrenkrog-Petersen · Niek
Tax · Irene Teinemaa · Marlon Dumas ·
Massimiliano de Leoni · Fabrizio Maria
Maggi · Matthias Weidlich

Abstract Predictive process monitoring is a family of techniques to analyze events produced during the execution of a business process in order to predict the future state or the final outcome of running process instances. Existing techniques in this field are able to predict, at each step of a process instance, the likelihood that it will lead to an undesired outcome. These techniques, however, focus on generating predictions and do not prescribe when and how process workers should intervene to decrease the cost of undesired outcomes. This paper proposes a framework for prescriptive process monitoring, which extends predictive monitoring with the ability to generate alarms that trigger interventions to prevent an undesired outcome or mitigate its effect. The framework incorporates a parameterized cost model to assess the cost-benefit

S. Fahrenkrog-Petersen · M. Weidlich
Humboldt-Universität zu Berlin, Berlin, Germany
E-mail: {stephan.fahrenkrog-petersen, matthias.weidlich}@hu-berlin.de

I. Teinemaa · M. Dumas
University of Tartu, Tartu, Estonia
E-mail: {irene.teinemaa, marlon.dumas}@ut.ee

N. Tax
Facebook, London, United Kingdom
E-mail: niek@fb.com

M. de Leoni
University of Padua, Padua, Italy
E-mail: deleoni@math.unipd.it

F. Maggi
Free University of Bozen-Bolzano, Bolzano, Italy
E-mail: maggi@inf.unibz.it

trade-off of generating alarms. We show how to optimize the generation of alarms given an event log of past process executions and a set of cost model parameters. The proposed approaches are empirically evaluated using a range of real-life event logs. The experimental results show that the net cost of undesired outcomes can be minimized by changing the threshold for generating alarms, as the process instance progresses. Moreover, introducing delays for triggering alarms, instead of triggering them as soon as the probability of an undesired outcome exceeds a threshold, leads to lower net costs.

1 Introduction

Process mining is a family of techniques to discover, monitor and improve business processes by extracting knowledge from logs of process executions (herein called *event logs*) recorded by information systems [1]. In this context, an event log is a set of event records (or events for short) such that each event captures a state change in the execution of an activity that occurs within a given instance of the process (herein called a *case*). For example, an event in a log may capture the fact that a worker has completed an activity "Check purchase order" as part of the execution of a case of an order-to-cash process. Over the last ten years, such event logs have become more widely available due to the adoption of standardized enterprise systems such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems.

Predictive process monitoring [17,19] is a sub-family of process mining techniques designed to predict the future state of ongoing cases of a business process. A predictive monitoring technique may, for example, predict the remaining execution time of each ongoing case, or the next activity to be executed in each case, or the final outcome of each ongoing case, with respect to a set of possible case outcomes. This article is related to the latter type of predictive process monitoring, which we call *outcome-oriented* [30]. For example, in an order-to-cash process, an outcome-oriented technique may predict whether a case will end in a timely and correct product delivery (desired outcome) or not (undesired outcome). More specifically, an outcome-oriented predictive process monitoring technique predicts, after each event of a case, the probability that the case will end in an undesired outcome.

Prescriptive process monitoring techniques go beyond predictive ones. Instead of merely predicting future (undesired) states, prescriptive monitoring techniques use such predictions to recommend or prescribe interventions designed to prevent a case from reaching an undesired future state. A prescriptive process monitoring technique is a function that maps a stream of events of ongoing cases to a stream of recommended or prescribed interventions that, if executed, minimize a given net cost function (or equivalently to maximize a utility function). This cost function depends on the number of cases that end in a negative outcome as well as other parameters such as the cost of executing the interventions, or the timing of such interventions.

A naive approach to turn a predictive process monitoring technique into a prescriptive one is by triggering an alarm whenever the probability that a case will lead to an undesired outcome is above a fixed threshold (e.g., 90%). This alarm leads to an intervention, such as calling the customer, offering a discount, etc. To distinguish between different types of interventions, the alarms generated by a prescriptive system may be associated with a type. For example, an alarm of type A may lead to one type of intervention (calling the customer) whereas an alarm of type B may lead to a different type of intervention (offering a discount). This naive approach may be far from optimal, as it does not take into account the cost induced by each intervention (e.g. the time spent by workers in the intervention or forgone revenue) nor its effect (e.g. preventing the undesired outcome altogether or only partially mitigating it). This cost and this effect is likely to be different depending on the type of intervention (e.g. offering a discount might be more expensive than calling the customer). Moreover, the cost and the effect of an intervention may vary as the case advances. Late interventions may be less effective and more costly than earlier interventions, in such a way that the net cost of intervening varies (possibly non-monotonically) as the case advances.

This article proposes a prescriptive process monitoring approach based on the above notions of predictions, alarms and interventions. An overview of the proposed approach is provided in Figure 1. The starting point of the approach is an event log of completed executions of a business process, together with a function that assigns a boolean outcome (positive or negative) to every case in the log. This labeled event log is used to train a predictive model (specifically, an outcome estimator). An outcome estimator is a function that given given a (partial) trace of a ongoing case, produces an estimate of the probability that this ongoing case will be labeled with a negative outcome upon its completion. This probability estimate is fed into an alarm system that decides whether or not to send an alarm to a human worker or a software bot (or other automated system) to prompt them to perform a given intervention. Each generated alarm has a type, which determines which intervention will be performed. Alarms are generated in a way that optimizes a net cost function, which depends on several parameters such as the cost of an intervention and the cost of an undesired outcome. These parameters are encapsulated in a *cost model*, which consists of parameters that may be tuned by a manager to capture the cost structure and business goals of the organization in which the process is executed. In this setting, the goal of the proposed framework is to generate alarms in a way that minimizes the expected net cost, given an event log, a set of ongoing cases, and a cost model.

This article is an extended and revised version of a previous conference paper [31]. The conference version focused on the scenario where there is a single type of alarm leading to a single type of intervention (e.g. calling the customer). This article enhances the scope of the framework to consider multiple alarm types, each one leading to a different intervention. For example, one alarm type may lead to calling the customer while another one leads to offering a discount by email. Additionally, this article considers further factors
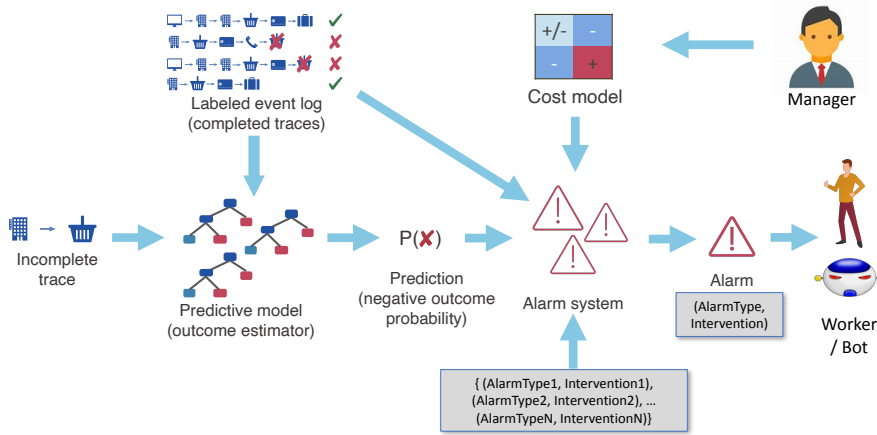
Fig. 1: Overview of the prescriptive process monitoring approach

that influence the effectiveness of alarms in practice. First, we add the possibility that the probability threshold above which an alarm is triggered may vary depending on how far the case has progressed. Second, we introduce the possibility of delaying the firing of an alarm to reduce false alarms stemming from instability in the predictive model.

The article is structured as follows. Section 2 discusses related work. Next, Section 3 presents the prescriptive process monitoring framework. Section 4 outlines the approach to optimize the alarm generation mechanism, while Section 5 reports on an empirical evaluation of the proposed approach. Section 6 concludes the paper and spells out directions for future work.

2 Related Work

As stated above, a naive approach to turn a model for predicting undesired process execution outcomes into a prescriptive model is to raise an alarm whenever the predictive model estimates that the probability of a negative outcome is above a given threshold. This, in turn, raises the question of determining an optimal alarm threshold. The problem of determining an optimal alarm threshold with respect to a given cost function is closely related to the problem of *cost-sensitive learning*. Cost-sensitive learning seeks to find an optimal prediction when different types of misclassifications have different costs and different types of correct classifications have different benefits [10]. In [23], the authors investigate cost-sensitive learning for predictive process monitoring through assigning different (cost) weights to classes. A non-cost-sensitive classifier can be turned into a cost-sensitive one by *stratification* (rebalancing the ratio of positive and negative training samples) [10], by learning a *meta-classifier* after relabeling the training samples according to their estimated cost-minimizing class label [9], or via empirical thresholding [28]. In this ar-

ticle, we adopt the latter approach, which has been shown to perform better than other cost-sensitive learning approaches [28].

While the above-cited cost-sensitive learning approaches provide a starting point for turning prediction models (e.g., classifiers) for triggering alarms, they are not designed to tackle the problem of prescriptive process monitoring. In particular, the above approaches target the scenario where the predictions made by a model are immediately used to make a decision (e.g., triggering an intervention). In particular, these approaches do not consider the possibility of delaying the decision. In this article, we deal with a different problem formulation, where the costs may depend on the time when the decision is made, i.e., one may delay the decision to accumulate further information.

Cost-sensitive learning for sequential decision-making has been approached using reinforcement learning (RL) techniques [26]. This work differs from ours in three ways. First, instead of making a sequence of decisions, we aim at finding an optimal time to make a single decision (the decision to raise an alarm). Second, RL assumes that actions affect the observed state once they are triggered. In our case, there are two possible actions at each step: 1) raising one of the available alarms, or 2) delaying the decision. In the latter case, we will wait until we observe the next event. But this event is not at all affected by the selected action (i.e., to "wait"). In other words, the environment is not affected by the delay in the decision. Finally, we train our model only based on observed data, while reinforcement learning requires the existence of a simulator, that imitates the environment, in our case the business process.

Predictive and prescriptive process monitoring are also related to Early Classification of Time Series (ECTS), which aims at accurate classification of a (partial) time series as early as possible [34]. Common solutions for ECTS find an optimal *trigger function* that decides on whether to output the prediction or to delay the decision and wait for another observation in the time series. To this end, the approach presented in [34] identifies the minimum prediction length when the memberships assigned by the nearest neighbor classifier become stable, the one presented in [25] estimates the probability that the label assigned based on the current prefix is the same as the one assigned based on the complete time series, and, similarly, the one presented in [21,22] compares the accuracy achieved based on a prefix to the one achieved based on a complete trace. Recently, a few non-myopic methods have been proposed [5,29]. Yet, these approaches assume a-priori knowledge of the length of the sequence, which is not given in the context of traces of business processes. As such, the problem is different, as delaying the decision comes with a risk that the case will end before the next possible decision point. The methods outlined in [21, 22,5,29] are the only ECTS methods that try to balance accuracy-related and earliness-related costs. However, they 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.

We are aware of five previous studies related to alarm-based prescriptive process monitoring [18,7,20,14,15]. [18] study the effect of different likelihood

thresholds on the total intervention cost (called adaptation cost) and the misclassification penalties. However, they assume that the alarms can only be generated at one pre-determined point in the process that can be located as a state in a process model. Hence, their approach is restricted to scenarios where: (i) there is a process model that perfectly captures all cases; (ii) the costs and rewards implied by alarms are not time-varying. Also, their approach relies on a mechanism with a user-defined threshold, as opposed to our empirical thresholding approach. The latter remark also applies to [7, 20], which generate a prediction when the likelihood returned by a trace prefix classifier first exceeds a given threshold. Also, the study proposed in [7] is not cost-sensitive, whereas the one introduced in [20] is based on a static cost model that does not change over time. [14] provide recommendations during the execution of business processes to avoid a predicted performance deviation. Yet, this approach does not take into account the notion of earliness, i.e., the fact that firing an alarm earlier has a different effect than firing it at a later point. In the context of prescriptive process monitoring, in [33], the authors, instead of alarms, as addressed in this paper, provide recommendations in the form of actions to be executed next. Finally, [15] propose a general architecture for prescriptive process monitoring. However, this architecture does not incorporate an alarm model nor does it propose a method for cost optimization.

3 Prescriptive Process Monitoring Framework

This section introduces a framework for an alarm-based prescriptive process monitoring system. We first introduce the notion of event log in Section 3.1, before turning to the cost model of our framework in Section 3.2. For the sake of clarity, this model is first introduced for scenarios that allow for one type of alarm. The latter assumption is dropped in Section 3.3 extending the cost model to scenarios with multiple alarms. Finally, Section 3.4 builds on the cost-model to formalize the concept of an alarm system.

3.1 Event Log

Executions of process instances (a.k.a. cases) are recorded in so-called *event logs*. At the most abstract level, an event log can be represented in a tabular form. Figure 2 shows an excerpt of an event log that refers to the execution of the unemployment-benefit process in the Netherlands. Scenario 1 describes how the process is usually executed, which serves for illustration in the remainder of the paper. The description is the result of several interactions of one of the authors (see, e.g., [6]) with the company's stakeholders.

Scenario 1 (Unemployment Benefit) *In the Netherlands, UWV is the institution that provides social-security insurances for Dutch residents. UWV provides several insurances. One of the most relevant insurances is the provision of unemployment benefit. When residents (hereafter customers) become*

Customer ID	Event	Event date	Gender	Age category	Max entitlement	Entitled amount
25879	Initialize income form	2017-01-15	Female	55 years or older	15 months	€ 1,034
25879	Receive income form	2017-02-03	Female	55 years or older	15 months	€ 1,034
25879	Pay benefits	2017-02-05	Female	55 years or older	15 months	€ 1,034
25879	Inititialize income form	2017-02-15	Female	55 years or older	15 months	€ 1,034
25879	Check income form	2017-03-02	Female	55 years or older	15 months	€ 1,034
25879	Receive income form	2017-03-02	Female	55 years or older	15 months	€ 1,034
25879	Pay benefits	2017-03-04	Female	55 years or older	15 months	€ 1,034
25879	...					
164453	Initialize income form	2017-06-15	Male	26-35 years	3 months	€ 712
164453	Receive income form	2017-07-01	Male	26-35 years	3 months	€ 712
164453	Pay benefits	2017-07-05	Male	26-35 years	3 months	€ 712
164453	Initialize income form	2017-07-15	Male	26-35 years	3 months	€ 712
164453	Check income form	2017-08-02	Male	26-35 years	3 months	€ 712
164453	Receive income form	2017-08-02	Male	26-35 years	3 months	€ 712
164453	Detect reclamation	2017-08-02	Male	26-35 years	3 months	€ 712
164453	Pay benefits	2017-08-06	Male	26-35 years	3 months	€ 712
164453	Handle reclamation	2017-09-02	Male	26-35 years	3 months	€ 712
164453	...					

Fig. 2: A fragment of an event log for two UWV's customers, as explained in Scenario 1. Each row is an event; events with the same customer id are grouped into traces.

unemployed, they are usually entitled to monthly monetary benefits for a certain period of time. UWV executes a process to determine the amount of these benefits, to manage the interaction with the customers, and to perform the monthly payments. These payments are stopped when the customer reports that he/she has found a new job or when the time period in which the customer is entitled to unemployment benefits ends.

Each row in Figure 2 corresponds to an event. An event represents the execution of an activity for a case with a certain identifier that occurred at a specific moment in time and was performed by a given resource. The case identifier can vary depending on the process: for the example in question, it coincides with the customer id. Events can be grouped by case identifier and ordered by timestamp, thus obtaining a sequence of events, a.k.a. a trace.

Figure 2 shows that events can be associated with various properties, namely the different columns of the event table; we assume that the activity name and the timestamp are properties that are always present. Given an event e of a log, $\pi_{\mathcal{T}}(e)$ and $\pi_{\mathcal{A}}(e)$ return the timestamp and the activity name of an event e, respectively. As an example, the trace of customer id 25879 in Figure 2 is $\langle e_1, e_2, \ldots, e_n \rangle$ where, e.g., it holds that $\pi_{\mathcal{T}}(e_1) = \texttt{2017-01-15}$ and $\pi_{\mathcal{A}}(e_1) = $ '*Initialize income form*'. In the remainder, $\langle \rangle$ is the empty sequence, and $\sigma_1 \cdot \sigma_2$ indicates the concatenation of sequence σ_1 and σ_2. Given a sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, for any $0 < k \leq n$, $hd^k(\sigma) = \langle a_1, a_2, \ldots, a_k \rangle$ is the prefix of length k $(0 < k < n)$ of sequences σ, and $\sigma(k) = a_k$ is the k-th event of σ.

With these concepts at hand, a trace is defined as a sequence of events, and an event log as a set of traces:

Definition 1 (Trace, Event Log) Let $\mathcal{E}$ be the universe of all events. A *trace* is a finite non-empty sequence of events $\sigma \in \mathcal{E}^+$ such that each event appears

at most once and time is non-decreasing, i.e., for $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$ and $\pi_\mathcal{T}(\sigma(i)) \leq \pi_\mathcal{T}(\sigma(j))$. An *event log* is a set of traces $L \subset \mathcal{E}^+$ such that each event appears at most once in the entire log.

3.2 Single-Alarm Cost Model

An alarm-based prescriptive process monitoring system (*alarm system* for short) is a monitoring system that can raise an alarm in relation to a running case of a business process in order to indicate that the case is likely to lead to some undesired outcome. These alarms are handled by process workers who intervene in the process instance by performing an action (e.g., by calling a customer or blocking a credit card), thereby preventing the undesired outcome or mitigating its effect. These actions may have a cost, which we call *cost of intervention*. When a case does end in a negative outcome, this leads to the *cost of undesired outcome*. Clearly, intervening is beneficial if its cost is lower than the cost of reaching an undesired outcome.

For many practical application scenarios, however, it is not sufficient to consider solely a trade-off between the cost of intervention and the cost of undesired outcome. This happens when interventions that turn out to be superfluous induce further costs, ranging from financial damage to losing customers. We capture these effects by a *cost of compensation* in case the intervention was unnecessary, i.e., if the case was only suspected to lead to an undesired outcome, but this suspicion was wrong.

A further aspect to consider is the *mitigation effectiveness*, i.e., the relative benefit of raising an alarm at a certain point in time. Consider again the process of handling unemployment benefits at UWV as detailed in Scenario 1. In case of unlawful benefits, the longer UWV postpones an intervention, the lower its effectiveness, since the amount paid already to the customer cannot always be claimed back successfully.

Having introduced the various costs to consider when interfering in the execution of a process instance, we need to clarify the granularity at which these costs are modeled. In general, an alarm system is intended to continuously monitor the cases of the business process. However, in many application scenarios, such continuous monitoring is expensive (resources to interfere need to be continuously available). Hence, an approximation may be employed that assumes that significant cost changes are always correlated with the availability of new events (i.e., new information) in a case. Following this line, alarms can only be raised after the occurrence of an event.

In the remainder, each case is identified by a trace σ that is (eventually) recorded in an event log. For a prefix of such a trace, the above characteristics of an alarm, i.e., its cost of intervention, cost of compensation, and mitigation effectiveness, are captured by an alarm model. These characteristics may depend on the position in the case at which the alarm is raised and/or on other cases being executed. Hence, they are defined as functions over the number of already recorded events and the entire set of cases being executed, as follows:

Definition 2 (Alarm Model) An *alarm model* is a tuple (c_{in}, c_{com}, eff) consisting of:

- a function $c_{in} \in \mathbb{N} \times \mathcal{E}^+ \times 2^{\mathcal{E}^+} \to \mathbb{R}_0^+$ modeling the *cost of <u>in</u>tervention*: given a trace σ belonging to an event log L, $c_{in}(k, \sigma, L)$ indicates the cost of an intervention for a running case with trace σ when the intervention takes place after the k-th event;
- a function $c_{com} \in \mathcal{E}^+ \times 2^{\mathcal{E}^+} \to \mathbb{R}_0^+$ modeling the *cost of <u>com</u>pensation*: given a trace σ belonging to an event log L, $c_{com}(\sigma, L)$ indicates the cost to be incurred in when the intervention was put in place for a running case with trace σ that did not finally required it;
- a function $eff \in \mathbb{N} \times \mathcal{E}^+ \times 2^{\mathcal{E}^+} \to [0, 1]$ modeling the *mitigation <u>eff</u>ectiveness* of an intervention: for a trace σ of an event log L, $eff(k, \sigma, L)$ indicates the mitigation effectiveness of an intervention in σ when the intervention takes place after the k-th event.

Combining the above properties of an alarm with the cost of the undesired outcome that shall be prevented, we define a single-alarm cost model.

Definition 3 (Single-Alarm Cost Model) A *single-alarm cost model* is a tuple $(c_{in}, c_{out}, c_{com}, eff)$ consisting of:

- an alarm model (c_{in}, c_{com}, eff);
- a function $c_{out} \in \mathcal{E}^+ \times 2^{\mathcal{E}^+} \to \mathbb{R}_0^+$ to model the *cost of undesired <u>out</u>come*: for a trace σ of an event log L, $c_{out}(\sigma, L)$ is the cost associated with the undesired outcome for a running case with trace σ, if an effective intervention was not put in place.

Next, we illustrate the above model with exemplary application scenarios.

Scenario 1 (Continued from p.6) *While receiving unemployment benefits at UWV, several customers omit to inform UWV about the fact that they had found a job and, thus, keep receiving benefits that they are not entitled to. If the salary of the new job is lower than the salary of the previous job, then the customer is still entitled to a benefit proportional to the difference in salary. In fact, customers often mistakenly declare the wrong salary for their new job and are then expected to return a certain amount of received benefits. In practice, however, this rarely happens. UWV would therefore benefit from an alarm system that informs about customers who are likely to be receiving unentitled benefits. The* cost of an intervention *stems from the hourly rate of UWV's employees; investigations using different IT systems and verifications with the old and new employer of a customer take a significant amount of time. The* cost of undesired outcome *is the monetary value of the benefits that the customer received unlawfully.*

Let $unt(\sigma)$ denote the amount of unentitled benefits received in a case corresponding to trace σ. Based on discussions with UWV, we designed the following cost model.

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. Thus, the cost of intervention is the total labour cost of the employee for the amount of time they spend executing an intervention. Let S be the employee's average salary rate per time unit; let d be the amount of time it takes for the employee to intervene. Then, the cost of an intervention can be modeled as: $c_{in}(k, \sigma, L) = d \cdot S$.

Cost of undesired outcome. *The 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, i.e., $eff(k, \sigma, L) = \frac{unt(\sigma) - unt(hd^k(\sigma))}{unt(\sigma)}$.*

Due to the lack of competition for public administration, the above example does not include compensation costs. The following example from finance, however, illustrates the importance of being able to incorporate the cost of compensation.

Scenario 2 (Financial Institute) *Suppose that the customers of a financial institute use their credit cards to make payments online. Each of such a transaction is associated with a risk of fraud, e.g., through a stolen or cloned card. In this scenario, an alarm system shall determine whether the card needs to be blocked, due to a high risk of fraud. However, superfluous blocking of the card causes discomfort to the customer, who may then switch to a different financial institute.*

Cost of intervention. *The card is automatically blocked by the system. Therefore, the intervention costs consists in those to manufacture and send a new credit card to the customer by mail, so that $c_{in}(k, \sigma, L)$ is a constant value.*

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

Cost of compensation. *It is defined as the expected asset of the customer that would be lost, namely the actual asset multiplied by a certain probability $p \in [0, 1]$, which is the fraction of customers who left the institute within a short time after the card was wrongly blocked. Denoting the asset value of a customer (the amount of the investment portfolio, the account balance, etc.) with $asset(\sigma)$, the cost of compensation is estimated as: $c_{com} = p \cdot asset(\sigma)$.*

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

It is very important to highlight that the intervention is not modelled explicitly because it is not relevant for the scope of the framework. Conversely, we model its consequences in terms of alarm-cost models.

3.3 Multi-Alarm Cost Model

The above formalization of the cost model assumes that an alarm system supports a single type of alarm. For some processes, however, alternative alarms exist, each associated to a different mitigation effectiveness and model of costs of intervention and compensation.

Consider Scenario 2, the scenario of blocking a credit card to prevent fraud. An alternative intervention is to call the credit card owner by phone to verify the suspicious transaction. A phone call certainly has a higher cost of intervention, in comparison to automatically blocking the credit card. Yet, a phone call has a lower cost of compensation because, in case of falsely suspected fraud, it prevents the inconvenience of unnecessarily blocking a card.

The choice of which alarm to employ might depend on the likelihood of achieving an undesired outcome, if no action is put in place. For instance, for the financial institute, if fraud is assessed to be a possibility, but not a near-certainty, then it is preferable to call the customer. If it is assessed to be fraud with near-certainty, it is safer to preventively block the card.

To handle scenarios with multiple alarms, the following definition generalizes our earlier single-alarm cost model.

Definition 4 (Multi-Alarm Cost Model) Let C be the universe of alarm models. An *multi-alarm cost model* is a tuple (A, c_{out}) consisting of:
− a set $A \subset C$ of alarm models,
− a function $c_{out} \in \mathcal{E}^+ \times 2^{\mathcal{E}^+} \to \mathbb{R}_0^+$ to model the *cost of undesired __outcome__*;

3.4 Alarm-Based Prescriptive Process Monitoring System

An alarm-based prescriptive process monitoring system is driven by the outcome of the cases. In the remainder, the outcome of the cases is represented by a function $out : \mathcal{E}^+ \to \{\mathsf{true}, \mathsf{false}\}$: given a case identified by a trace σ, if the case has an undesired outcome, $out(\sigma) = \mathsf{true}$; otherwise, $out(\sigma) = \mathsf{false}$. In reality, during the execution of a case, its outcome is not yet known and needs to be estimated based on past executions that are recorded in an event log $L \subset \mathcal{E}^+$. The outcome estimator is a function $\widehat{out}_L : \mathcal{E}^+ \to [0, 1]$ predicting the likelihood $\widehat{out}_L(\sigma')$ that the outcome of a case that starts with prefix σ' is undesired. We define an alarm system as a function that returns true or false depending on whether an alarm is raised based on the predicted outcome.

Definition 5 (Alarm-Based Prescriptive Process Monitoring System)
Given an event log $L \subset \mathcal{E}^+$, let $\widehat{out}_L$ be an outcome estimator built from L. Let A be the set of alarms that can be raised. An *alarm-based prescriptive process monitoring system* is a function $alarm_{\widehat{out}_L} : \mathcal{E}^+ \to A \cup \{\bot\}$ Given a running case with current prefix σ', $alarm_{\widehat{out}_L}(\sigma')$ returns the alarm raised after σ', or $\bot$ if no alarm is raised.

For simplicity, we omit subscript L from $\widehat{out}_L$, and the entire subscript $\widehat{out}_L$ from $alarm_{\widehat{out}_L}$ when it is clear from the context. An alarm system can only

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

Table 1: Net cost of a case with trace σ based on its outcome and whether an alarm was raised. If the alarm is raised, i_σ indicates the index of σ when the alarm occurred.

raise one of the alarms and only once per case because the first alarm is expected to trigger an intervention by process actors.

Table 1 illustrates how the net cost of a case is determined based on a cost model on the basis of a multi-alarm cost model (A, c_{out}). In the table, i_σ indicates the index of the event in σ when the alarm was raised, namely the smallest $i \in [1, |\sigma|]$ such that $alarm(hd^{i_\sigma}(\sigma)) \in A$.

The above definition provides a framework for alarm-based prescriptive process monitoring systems. Next, we turn to the problem of tuning a prescriptive process monitoring system in order to optimize its net cost.

4 Alarm Systems and Empirical Thresholding

This section introduces four types of alarm systems. In Section 4.1, we introduce a basic mechanism based on empirical thresholding. In Section 4.2, we enhance this approach with the idea of delaying alarms. An enhancement based on prefix-length-dependent thresholds is introduced in Section 4.3. Finally, multiple possible alarms are handled in Section 4.4.

4.1 Basic Alarm System

We first consider a basic alarm system, in which there is only one alarm a, which is triggered as soon as the estimated probability of an undesired outcome exceeds a given threshold τ. Given this simple alarm system, we aim at finding an optimal value for the alarming threshold τ, which minimizes the net cost on a log L_{thres} comprising historical traces such that $L_{thres} \cap L_{train} = \emptyset$ with respect to a given probability estimator $\widehat{out}_{L_{train}}$ and alarm model a.

The total net cost of a mechanism to raise alarms $alarm$ on a log L is defined as $cost(L, a, alarm) = \Sigma_{\sigma \in L} cost(\sigma, L, a, alarm)$. Based thereon, we define an optimal threshold as $\tau = \arg\min_{\tau' \in [0,1]} cost(L_{thres}, a, alarm_{\tau'})$. Optimizing a threshold τ on a separate thresholding set is called *empirical thresholding* [28]. The search for such a threshold τ wrt. a specified alarm model a and log L_{thres} is done through any hyperparameter optimization technique, such as Tree-structured Parzen Estimator (TPE) optimization [3]. The resulting approach is a form of cost-sensitive learning, since the value τ depends on how the alarm model a is specified.

4.2 Delayed Firing System

The basic alarm system introduced above fires an alarm as soon as the probability of an undesired outcome is higher than the threshold τ. This may lead to firing an alarm too soon. Consider a scenario in which, after observing event e_i, the probability is above τ, whereas it drops below τ and stays below τ after the subsequent event e_{i+1} has been recorded. The basic alarm system would fire the alarm even if the probability of an undesired outcome is above τ for one single event and then drops below τ in subsequent events in a case.

An alternative alarming system (herein called *delayed firing*) is to fire an alarm only if the probability of a negative outcome remains above threshold τ for κ consecutive events, where κ is the *firing delay*. One would expect this delayed firing system to be more robust to instabilities in the predictive model (e.g., robust to high levels of variations in the probability calculated by the predictive model for consecutive events in a trace).

When building a delayed alarming mechanism, we need to consider both, the firing delay κ as well as the threshold τ, for the hyperparameter optimization. Note that the basic alarm system is a special case of the delayed firing system with a firing delay of $\kappa = 1$.

4.3 Prefix-length-dependent Threshold System

To cope with scenarios with dynamic costs, we propose to not rely only on a single global alarming threshold τ, but to use different thresholds depending on the length of the prefix. That is, a separate threshold τ_k is optimized for each prefix length k or, more generally, thresholds $\tau_{a \to b}$ are optimized for certain intervals of prefix lengths $[a, b]$ for $a, b \in \mathbb{N}$. For instance, $\tau_{1 \to 4}$ denotes the optimal threshold for prefix lengths 1 to 4, while $\tau_{5 \to \infty}$ denotes the optimal threshold from length 5 to the end of the trace. To divide the whole range of prefix lengths into n non-overlapping intervals, we define n *splitting prefixes* $\rho_i, 1 \leq i \leq n$, where ρ_i refers to the start point of interval i. For instance, in the previous example, $\rho_1 = 1, \rho_2 = 5$. A single global threshold and prefix-length-based thresholds can be seen as special cases of interval-based thresholds.

The splitting prefixes can either be user-defined or optimized over L_{train}. In a system with multiple thresholds, all the thresholds can be optimized simultaneously. It is also possible to treat the splitting prefixes for prefix intervals as hyperparameters and optimize them over the event log L together with the thresholds. Prefix-length dependent thresholds $\tau_{a \to b}$ may be combined with a fire delay ϵ to build an alarm system. Then, the fire delay ϵ and the prefix-length dependent thresholds $\tau_{a \to b}$ are trained at the same time.

4.4 Multi-alarm systems

To optimize the decision of firing an alarm in a setting with multiple possible alarms A we consider each alarm, as well as the option to not firing an alarm,

as a potential decisions. Therefore we formulate the problem similar to a multi-class classification task [32], with $|A \cup \{\perp\}|$ different classes: one class for no alarm and one class per alarm. This might look like a traditional multi-class classification task, however it is not. Because in such a task a perfect system would assign multiple labels and therefore we would also use a training set with multiple labels. In our scenario a perfect system would only fire the cheapest alarm, with the lowest cost of intervention and/or mitigation effect, for all cases that will end in an undesired outcome. For all other cases no alarm would be fired. In conclusion a perfect solution would not require multiple alarms and the training set would only consist of two labels. Therefore, our can not be solved with a lot of multi-class classification techniques like one-vs-all approaches.

However, one way of solving a multi-class classification problem is to break it into several binary classification problems (i.e., the *one-vs-one* approach) [2]. Then, all classes are tested against each other with a binary classification algorithm. The set of binary decisions leads to a number of votes, that are assigned to the classes that won the binary classifications. The input data is ascribed the class with the most votes. Our optimization problem can modelled simultaneously like an *one-vs-one* approach.

Optimizing the alarms against the class of not firing an alarm is done by applying the basic model to each alarm independently. In a scenario with two possible alarms this gives us the thresholds $\tau_{False-vs.-a_1}$ and $\tau_{False-vs.-a_2}$. All alarms are then trained against each other with empirical thresholding. In our example, that implies training the threshold $\tau_{a_1-vs.-a_2}$. However, for training $\tau_{a_1-vs.-a_2}$, we use only the prefixes σ that have a higher likelihood probability than both thresholds for these alarms (e.g., $\widehat{out}_{L_{train}}(hd^k(\sigma)) > \tau_{False-vs.-a_1}$ and $\widehat{out}_{L_{train}}(hd^k(\sigma)) > \tau_{False-vs.-a_2}$). While this might seem contrary to the one-vs-one approach (the threshold is not trained over the whole dataset), it is necessary for the following reasons. First, the question of 'should we fire an alarm?' shall be separated from the question of 'which alarm to fire?'. The decision on the alarm is taken only over the subset of prefixes where alarming was found necessary. Training the threshold over all prefixes would fit the threshold to irrelevant prefixes. Also, it is more important to fire any alarm for a case with an undesired outcome than to choose the best alarm. Since we train the thresholds to decide between alarms in a hierarchical order, first the alarm vs. no alarm thresholds and then the alarms against each other, we call this approach *hierarchical thresholding*.

Figure 3 illustrates the general idea for a system with two alarms. It represents the probability of an undesired outcome on the horizontal axis. The three aforementioned thresholds are visualized as vertical lines. Different colors illustrate the slides of the likelihood probability that will lead to different actions by the alarm system: firing no alarm, or firing one of the two alarms a_1, a_2.
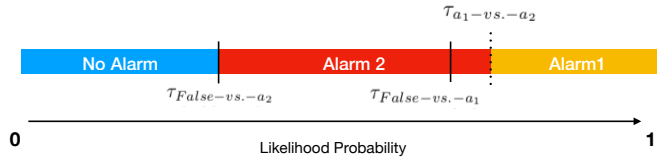
Fig. 3: Likelihood probability in hierarchical thresholding.

5 Evaluation

In this section, we report on an experimental evaluation of the proposed framework. Specifically, our evaluation addresses the following research questions related to the overall effectiveness of alarm-based prescriptive process monitoring. We first explore the effectiveness of the basic model that employs simple empirical thresholding, considering the goodness of the identified thresholds as well as the impact of the problem setting:

RQ1 *How good are the thresholds identified by empirical thresholding in terms of the reduction of the average processing cost for different alarm model configurations?*

RQ2 *How does the mitigation effectiveness affect the benefit obtained by an alarm system?*

RQ3 *How does the cost of compensation affect the benefit obtained by an alarm system?*

Moreover, we go beyond simple empirical thresholding and investigate the effectiveness of the more elaborated models for thresholding. In particular, our research questions relate to the aforementioned design choices involved when deciding on when to fire an alarm: *How is the average processing cost per case affected:*

RQ4 *When training a parameter for the minimum number of events that exceed the threshold?*

RQ5 *When using more than one threshold interval?*

RQ6 *When increasing the number of prefix-length-dependent thresholds?*

RQ7 *When combining prefix-length dependent thresholds with a firing delay, compared to the a single-alarm system?*

Finally, we turn to the effectiveness of a system that supports multiple alarms. Here, we compare such a multi-alarm system against one that features solely a single alarm.

RQ8 *How is the average processing cost per case affected when using multiple alarms compared to the best system with only one alarm?*

In the remainder, we first discuss the real-world datasets used in our evaluation (Section 5.1), before turning to the experimental setup (Section 5.2). We then report on our evaluation results in detail and close with an overview of our answers to the above research questions (Section 5.3).

Table 2: Dataset statistics

dataset name	# traces	class ratio	min length	med length	(trunc.) max length	# events
bpic2017_refused	31 413	0.12	10	35	60	1 153 398
bpic2017_cancelled	31 413	0.47	10	35	60	1 153 398
traffic_fines	129 615	0.46	2	4	5	445 959
unemployment	34 627	0.20	1	21	79	1 010 450

5.1 Datasets

We use the following real-world datasets to evaluate the alarm system:

BPIC2017. This log contains traces of a loan application process in a Dutch bank.[1] It was split into two sub-logs, denoted with *bpic2017_refused* and *bpic2017_cancelled*. In the first one, the undesired cases refer to the process executions in which the applicant has refused the final offer(s) by the financial institution. In the second one, the undesired cases consist of those cases where the financial institution has cancelled the offer(s).

Road traffic fines. This log originates from a Italian police unit and relates to a process to collect traffic fines.[2] The desired outcome is that a fine is paid, while in the undesired cases the fine needs to be sent for credit collection.

Unemployment. This event log corresponds to the *Unemployment Benefits* process run by the UWV in the Netherlands, introduced already as Scenario 1 in Section 3. Due to privacy constraints, this event log is not publicly available. The undesired outcome of the process is that a resident will receive more benefits than entitled, causing the need for a reclamation.

Table 2 describes the characteristics of the event logs used. These logs cover diverse evaluation settings, along several dimensions. The classes are well balanced in *bpic2017_cancelled* and *traffic_fines*, while the undesired outcome is more rare in *unemployment* and *bpic2017_refused*. In *traffic_fines*, the traces are very short, while in the other datasets the traces are generally longer.

For each event log, we use all available data attributes as input to the classifier. Additionally, we extract the *event number*, i.e., the index of the event in the given case, the *hour, weekday, month, time since case start*, and *time since last event*. Infrequent values of categorical attributes (occurring less than 10 times in the log) are replaced with value 'other', to avoid a massive blow-up of the considered number of dimensions. Missing attributes are imputed with the respective most recent (preceding) value of that attribute in the same trace when available, otherwise with zero. Traces are cut before the labeling of the case becomes trivially known and are truncated at the 90th percentile of all case lengths to avoid bias from very long traces.

---

[1] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

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

5.2 Experimental Setup

We split the aforementioned datasets temporally, as follows. We order cases by their start time and randomly select 80% of the first 80% of the cases (i.e., 64% of the total) for L_{train}; 20% of the first 80% of the cases (i.e., 16% of the total) for L_{thres}; and use the remaining 20% as the test set L_{test}. The events in cases in L_{train} and L_{thres} that overlap in time with L_{test} are discarded in order to not use any information that would not be available yet in a real setting.

Based on L_{train}, we build the classifier $\widehat{out}$ using random forest (RF) and gradient boosted trees (GBT). Both algorithms have been shown to work well on a variety of classification tasks [13, 24]. Features for a given prefix are obtained using the aggregation encoding [16], which is known to be effective for logs [30].

The configuration of the alarming mechanism depends on the setup chosen for a specific research question. For RQ1 to RQ3, we rely on the basic model, introduced in Section 4.1, and determine an optimal alarming threshold $\overline{\tau}$ based on L_{thres}. We employ TPE optimization [3] with 3-fold cross validation. The resulting alarm system is then compared against several baselines: First, we compare with the *as-is* situation, in which alarms are never raised. Second, define a baseline with $\tau = 0$, which enables us to compare with the situation where alarms are always raised directly at the start of a case. Finally, setting $\tau = 0.5$, we consider a comparison with the cost-insensitive scenario that simply raises alarms when an undesired outcome is expected.

To answer RQ4 to RQ7, we consider the advanced models introduced in Section 4.2 and Section 4.3. Again, the parameters are optimized using TPE. For RQ4, this includes the threshold $\overline{\tau}$ and the firing delay $\kappa \in \{1, \ldots, 7\}$. For RQ5, we train two prefix-length-interval-based thresholds $\overline{\tau_{1 \to \rho}}$, $\overline{\tau_{\rho \to \infty}}$ and optimize them together with parameter ρ. For RQ6, we define three different systems with 1 to 3 prefix length intervals. We rely on user-defined intervals for the prefix length, as follows. We set the length for each interval, except for the last, to one prefix and put all intervals after each other staring at prefix length 1. This results in three systems with $\{\overline{\tau_{1 \to \infty}}\}$, $\{\overline{\tau_{1 \to 2}}, \overline{\tau_{2 \to \infty}}\}$, and $\{\overline{\tau_{1 \to 2}}, \overline{\tau_{2 \to 3}}, \overline{\tau_{3 \to \infty}}\}$. Finally, to test RQ7, we build an alarming mechanism by training thresholds $\overline{\tau_{1 \to \rho}}$, $\overline{\tau_{\rho \to \infty}}$, the interval-section point ρ, and the firing delay $\kappa \in \{1, \ldots, 7\}$.

To explore RQ8, we derive a multi-alarm system that optimizes the alarming threshold $\overline{\tau}$, using TPE, for two different alarms independently (Section 4.4). For these alarms, we multiply the factors given in Table 3 with the cost of intervention c_{in} and the cost of compensation c_{com}. We compare the resulting system against a baseline that always uses one of the alarms, i.e., the one that is better in terms of average processing cost per case.

It is common in cost-sensitive learning to apply calibration techniques to the resulting classifier [35]. Yet, we found that calibration using Platt scaling [27] does not consistently improve the estimated likelihood of the undesired outcome on our data and, thus, we did not apply calibration.

Table 3: Factors for the different alarms, that are multiplied with the respective costs

	factor c_{in}	factor c_{com}
Alarm 1	1	1
Alarm 2	1.2	0.5

Table 4: Alarm model configurations

	$c_{out}(\sigma, L)$	$c_{in}(k, \sigma, L)$	$c_{com}(\sigma, L)$	$\mathit{eff}(k, \sigma, L)$
RQ1	$1, 2, 3, 5, 10, 20$	1	0	$1 - k/|\sigma|$
RQ2	$1, 2, 3, 5, 10, 20$	1	0	$0, 0.1, 0.2, \ldots, 1$
RQ3	$1, 2, 3, 5, 10, 20$	1	$\{0, 1/20, 1/10, 1/5, 1/2, 1, 2, 5, 10, 20\}$	$1 - k/|\sigma|$
RQ4–RQ7	10	$1, 2, 3, 4, 5$	$0, 1, 2, 3, 4, 5, 10, 15, 20$	1
	10	$\{1, 2, 3, 4, 5\} \cdot k/|\sigma|$	$0, 1, 2, 3, 4, 5, 10, 20$	$1 - k/|\sigma|$
	10	$\{1, 2, 3, 4, 5\} * (1 - \frac{\min(a, k-1)}{b})$	$\{0, 1, 2, 3, 4, 5, 10, 20\} * (1 - \frac{\min(k-1, c)}{d})$	$1 - \min(\frac{\min(e, k-1)}{f})$
RQ8	10	$1, 2, 3, 4, 5$	$1, 2, 3, 4, 5, 10, 20, 30, 40$	1

Next, we turn to the alarm models used in our evaluation, summarized in Table 4. For RQ1, we vary the ratio between the cost of the undesired outcome c_{out} and the cost of intervention c_{in}, keeping the cost of compensation c_{com} and the mitigation effectiveness eff unchanged. The same is done for RQ2 and, in addition, the mitigation effectiveness eff is varied. For RQ3, we vary two ratios: the one between c_{out} and c_{in}, and the one between c_{in} and c_{com}.

In the experiments related to RQ4-RQ7, we consider three types of configurations, each corresponding to one row in Table 4. We vary the cost of intervention c_{in} and the cost of compensation c_{com} from values that render them insignificant compared to c_{out}, to values that yield a significant impact. Specifically, the first type of alarm model, coined constant cost configurations, assigns constant costs over the whole trace. A second type, linear cost configurations, assigns costs that increase with longer trace prefixes. The third type, non-monotonic cost configurations, changes costs non-linearly over the length of the trace. Constants for this setting are introduced as listed in Table 5. We assign values similar to the minimum case length to constants used as a numerator and values similar to the medium case length for constants used as a divisor. This yields a non-monotonic cost development over the trace length for most cases. For instance, for *traffic_fines*, with $a = 3$ and $b = 5$ (Table 5), the cost of intervention $c_{in}(k, \sigma, L)$ under this cost model (Table 4), assumes values from $(1 - \frac{\min(3, k-1)}{5})$ to $5 * (1 - \frac{\min(3, k-1)}{5})$.

For RQ8, we largely follow the same setup. However, we define our test scenarios such that the alarm that is more expensive for true positives is at least 10% cheaper for false positives compared to the other alarm. Thereby, both alarms show a certain difference in the induced cost trade-off.

To evaluate the success of prescriptive process monitoring, we measure the *average cost per case* using the test set L_{test} derived for each dataset as discussed above. This cost shall be minimal. Moreover, we measure the *benefit* of the alarm system, i.e., the reduction in the average cost of a case when using the alarm system compared to the average cost when not using it.

Table 5: Constants for non-monotonic cost configurations

Dataset	a	b	c	d	e	f
bpic2017_cancelled	10	35	13	32	18	40
bpic2017_refused	8	33	15	34	20	35
traffic_fines	3	5	2	5	3	4
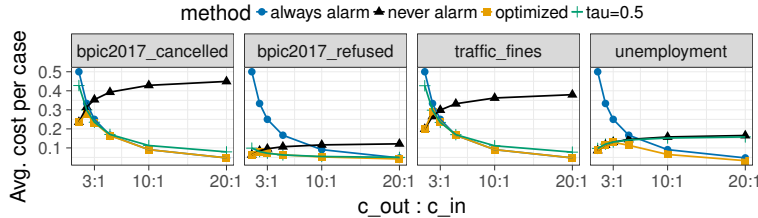
Fig. 4: Cost over different ratios of c_{out} and c_{in} (GBT)

Additionally, we use the f-score to test the accuracy of our systems, in terms of how often they fire an alarm correctly. It is calculated as the harmonic mean of precision and recall and ranges from 0 (worst) to 1 (best).

The above experimental setup has been implemented in Python based on Scikit-Learn and LightGBM, with the prototype being publicly available online.[3]

5.3 Results

We evaluate our basic model of an alarming system that is based on empirical thresholding by exploring whether it consistently reduces the average processing cost under different alarm models (RQ1). Figure 4 shows the average cost per case when varying the ratio of c_{out} and c_{in}. We only present the results obtained with GBT, which slightly outperform those with RF. When the ratio between the two costs is balanced, the minimal cost is obtained by never alarming. When $c_{out} \gg c_{in}$, one shall always raise an alarm. However, when c_{out} is slightly higher than c_{in}, the best strategy is to sometimes raise an alarm based on $\widehat{out}$. We found that the optimized $\overline{\tau}$ always outperforms the baselines. An exception is ratio 2:1 for the *traffic_fines* dataset, where never alarming is slightly better.

The impact of the threshold for firing an alarm is further explored in Figure 5. The optimized threshold is marked with a red cross and each line represents one particular cost ratio. While the optimized threshold generally obtains minimal costs, there sometimes exist multiple optimal thresholds for a given alarm model. For instance, for the 5:1 ratio in *bpic2017_cancelled*, all thresholds between 0 and 0.4 are cost-wise equivalent. Hence, empirical thresholding consistently finds a threshold that yields the lowest cost for a given log and cost model configuration.
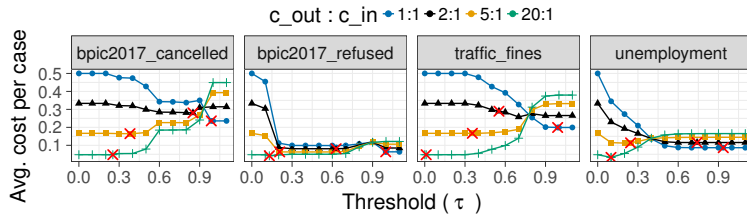
---

[3] https://github.com/samadeusfp/alarmBasedPrescriptiveProcessMonitoring/

Fig. 5: Cost over different thresholds ($\overline{\tau}$ is marked with a red cross)

(a) Varying *eff*

(b) Varying c_{com}

Fig. 6: Benefits achieved by a basic empirical thresholding systems over no alarm system for different alarm model configurations

Turning to RQ2, we evaluate how the mitigation effectiveness influences the results. Figure 6a shows the benefit of having an alarm system compared to not having it for different (constant) effectiveness values. As the results are similar for logs with similar class ratios, hereinafter, we show the results for *bpic2017_cancelled* (balanced classes) and *unemployment* (imbalanced classes). As expected, the benefit increases both with higher *eff* and with higher $c_{out} : c_{in}$ ratios. For *bpic2017_cancelled*, the alarm system yields a benefit when $c_{out} : c_{in}$ is high and $eff > 0$. Also, a benefit is always obtained when $eff > 0.5$ and $c_{out} > c_{in}$. In the case of the *unemployment* dataset, the average benefits are smaller, since there are fewer cases with undesired outcome and, therefore, the number of cases where c_{out} can be prevented by alarming is lower. In this case, a benefit is obtained when both *eff* and $c_{out} : c_{in}$ are high. We conducted analogous experiments with a linear decay in effectiveness, varying the maximum possible effectiveness (at the start of the case), which confirmed that the observed patterns remain the same. As such, we have confirmed empirically that an alarm system yields a benefit over different values of mitigation effectiveness.

Next, we consider the influence of the cost of compensation (RQ3). As above, the benefit of the alarm system is plotted in Figure 6b across different ratios of $c_{out} : c_{in}$ and $c_{in} : c_{com}$. When the cost of compensation c_{com} is high, the benefit decreases due to false alarms. For *bpic2017_cancelled*, a benefit is obtained almost always, except when $c_{out} : c_{in}$ is low (e.g., 2:1) and c_{com} is high (i.e., higher than c_{in}). For *unemployment*, fewer configurations are beneficial, e.g., when $c_{out} : c_{in} = 5 : 1$ and c_{com} is smaller than c_{in}. We conducted analogous experiments with a linearly increasing cost of intervention, varying the maximum possible cost, which confirmed the above trends. In sum, we confirmed empirically that the alarm system achieves a benefit if
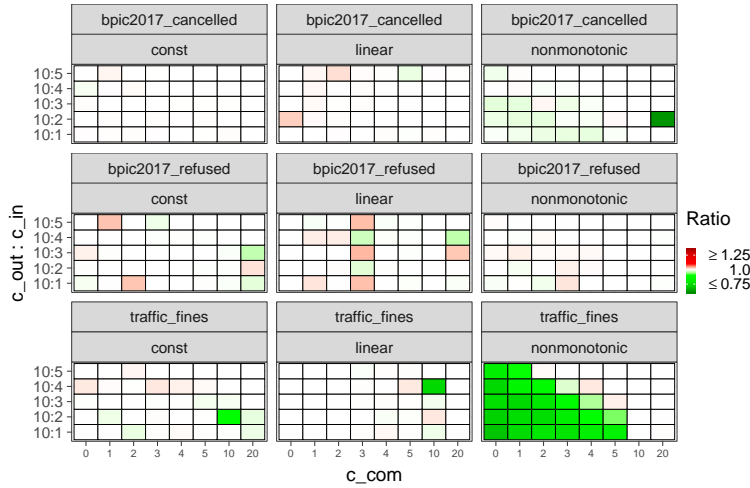
Fig. 7: Ratio of the average processing costs for a systems with fiiring delay compared to a system without firing delay

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.

Next, we evaluate the design choices involved when deciding on when to fire an alarm. We explore the reduction in the average processing cost under a firing delay, the use of multiple threshold intervals or prefix-length-dependent thresholds, and the combination of the latter thresholds and a firing delay.

To answer RQ4 on the effectiveness of a firing delay, we calculated the ratio of the average cost per case for the system with the firing delay as an additional hyperparameter, divided by the system without a firing delay. Figure 7 shows nearly no cost settings, in which the system that fires immediately outperforms the system with the firing delay (red cells). Many times, the firing delay reduces the costs (green cells), especially for the non-monotonic scenarios. Sometimes the costs are reduced by 20% or more. However, in the vast majority of cases, both systems produce the same results. This is due to the firing delay parameter being set to 1 in many scenarios. In addition to the ratio, we also visualized the benefit in terms of f-score that the system with firing delay delivers, see Figure 8. The results provide evidence for our hypothesis that waiting a certain number of events before firing an alarm improves the classification of traces. Overall, our results suggest the following answer to RQ4: The firing delay parameter may reduce the average processing cost per case.

To visualize relative benefits of multiple threshold intervals (RQ5), we calculate the ratio of the average processing costs between our baseline, a system based on the basic model, and our system with two threshold intervals. In Figure 9, we plot this ratio for all used cost configurations. Green and red coloring represents an improvement by using interval-based thresholds or by
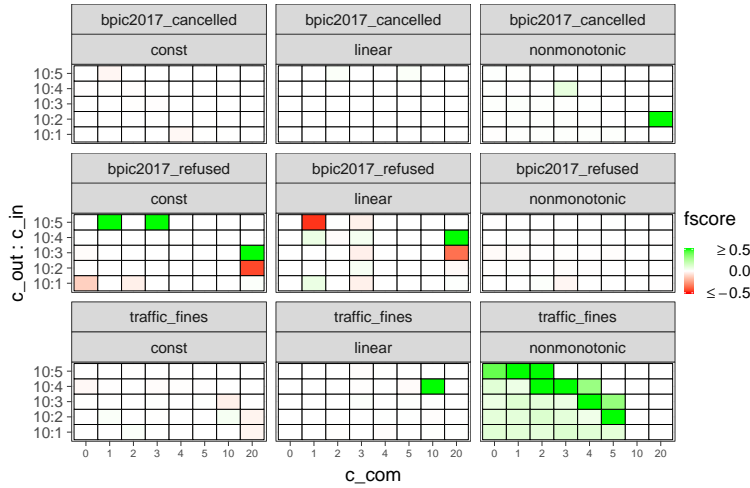
Fig. 8: Absolute F-score benefit of a system with firing delay over a system without firing delay
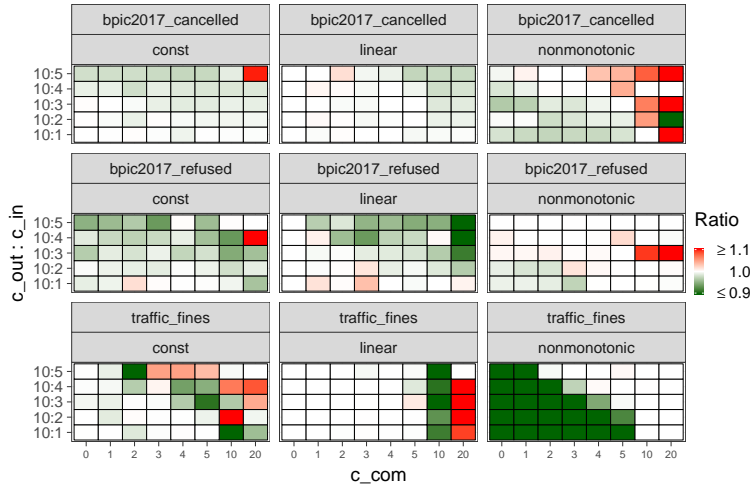

Fig. 9: Ratio of average processing costs of a systems with single threshold compared to a system with two thresholds

using a single global threshold, respectively. Contrary to our expectation, in some scenarios the system with a global threshold outperforms a system with intervals. However, improvements are observed for most scenarios.

Consistent improvements materialize for scenarios with non-monotonic changing costs and the cost of compensation being lower than the cost of undesired outcome. The improvements are the highest for the *traffic fines* dataset, that has the shortest traces. This leads also to shorter intervals. The smallest improvement in this group is seen in the *bpic2017 refused* dataset, which shows the lowest ratio of traces with an undesired outcome. Due to this low class
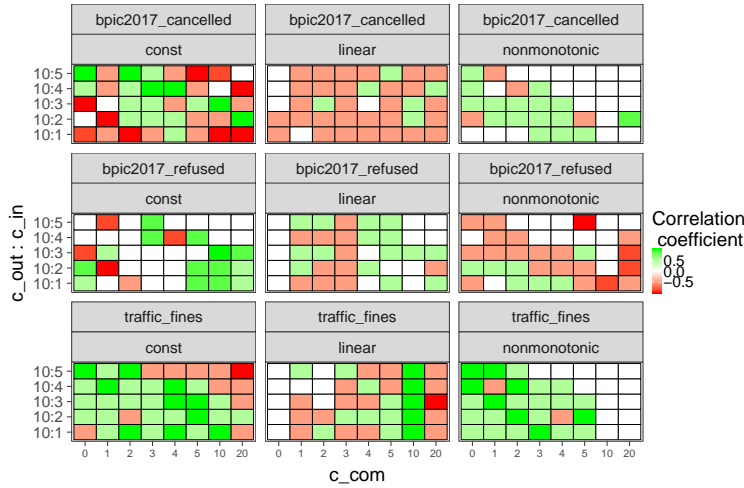
Fig. 10: Correlation coefficient between number of intervals for thresholds in an alarm system and the average cost per case

ratio, the potential room for improvement is also the smallest, because for a smaller number of cases it is necessary to intervene.

Moreover, we observe consistent improvements for the two *bpic2017* datasets for linear changing costs and constant costs.

The *traffic fines* dataset with linear costs is an interesting outlier. It shows nearly no improvement for scenarios, in which the cost of compensation is lower than the cost of undesired outcome. If the cost of undesired outcome has the same value as the cost of compensation the system with threshold intervals performs significantly better than the system with a global threshold. If the cost of compensation is higher than the cost of undesired outcome, the system with a single global threshold is significantly better. We investigated why the interval systems performs worse than the single threshold system and found out that, in the linear scenario, the single threshold system set the threshold to such a high level, that the threshold is never reached and no alarm is fired. However, this is not true for the system with two threshold intervals, this system fire alarms in a variety of cases. This approach underperforms here compared to not firing an alarm at all. The reason for this seems to be overfitting.

In general, considering RQ5, our results indicate that prefix-length dependent thresholds based on intervals may improve a prescriptive alarm system.

We further consider the use of several prefix-length-dependent thresholds. Figure 10 depicts the ranking correlation between the number of thresholds a system has in ascending order and the corresponding average cost per case in descending order. As in the previous experiment, an increasing number of thresholds should not lead to a higher, average cost per case. If there is no benefit to additional thresholds, all thresholds could just be set to the same value. This is not the case in our experiments, though, which we attribute to
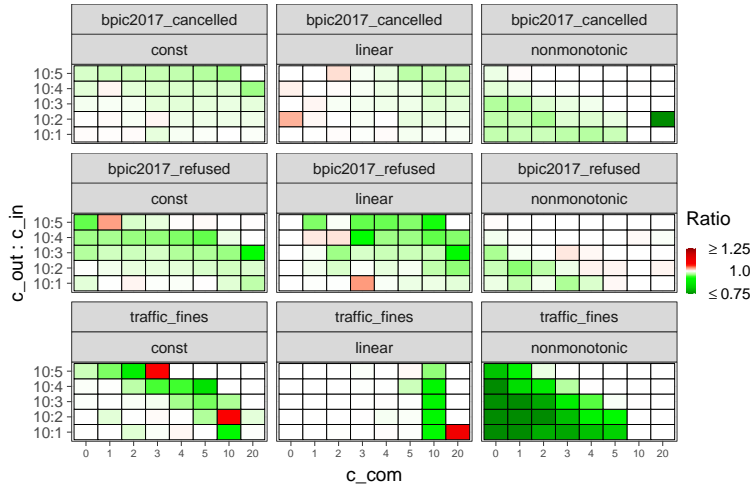
Fig. 11: Ratio for system that fires immediately and global threshold divided by system with firing delay and prefix-length dependent threshold

overfitting. An example for overfitting are situations, where the basic model does not fire an alarm for any case, while the prefix-length dependent approach uses the second or third threshold to fire alarms for lengthy traces. This leads to better results on the training set, but not on the test set.

In the light of RQ6, the above results do not generally suggest that there is a positive influence of an increased number of prefix-length-dependent thresholds. However, we see consistent improvements or at least no declines for the non-monotonic versions of the datasets *bpic2017 cancelled* and *traffic fines*. These datasets have both a balanced class ratio, whereas the dataset *bpic2017 refused* has an imbalanced class ratio (see Table 2). This leads to the conclusion that it is possible to improve the average cost per case with an increasing number of thresholds for datasets with a balanced class ratio in non-monotonic cost scenarios. A limitation to this is that we only observe improvements if the cost of an undesired outcome is larger than the cost of compensation. As expected, the improvements in the traffic fines dataset are higher. Since this dataset has shorter traces, multiple thresholds imply a high coverage of possible prefix-lengths.

As mentioned in RQ7, we also investigate, if the combination of prefix-length dependent thresholds and a firing delay is better than the basic model. To this end, we first check, if the combined approach outperforms a system with a global threshold and without firing delay. The results in Figure 11 confirm that the proposed approach leads to equivalent or better ratios of average processing cost under nearly all tested cost configurations. However, for three cost configurations in the *traffic fines* dataset, we observe negative results. Exploring these experiments in more detail, we found the negative results for two of them being due to overfitting. In these scenarios, the basic model approach builds a system that never fires an alarm, while the proposed
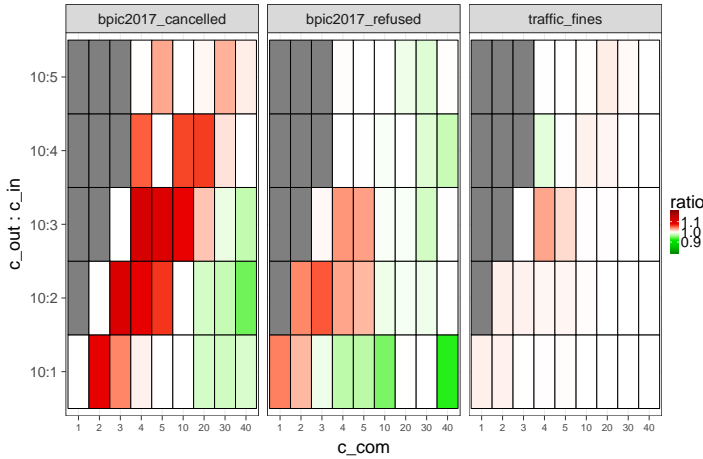
Fig. 12: Ratio of average processing cost for a system with multiple alarm based on hierarchical thresholding and the best single alarm system

approach builds a system that fires an alarm for some traces. In the third case, the sophisticated approach sets the fire delay to 1 and sets the later prefix-length dependent threshold to 1 and the first threshold to a similar value like the threshold of the basic approach. With more training runs, therefore, the sophisticated approach would probably end up with the same behavior as the basic model.

Finally, we turn to the evaluation of a multi-alarm system (RQ8). Our results in Figure 12 indicate that hierarchical thresholding may, but does not necessarily outperform the best single-alarm system. Specifically, we observe a stair-like border for scenarios with high compensation cost, in which hierarchical thresholding outperforms the best-single alarm system. This aligns with the relative benefit for false positive for alarm 1 compared to alarm 2. We visualize this benefit by calculating the ratio of firing alarm 1 for false positive and alarm 2 in Figure 13. Based thereon, we conclude that a multi-alarm system using hierarchical thresholding can outperform the best single-alarm system for scenarios in which the more expensive alarm, in terms of cost of intervention, has a huge cost benefit in the case of a false alarm, compared to a cheaper alarm.

A summary of the experimental findings is given in Table 6.

6 Conclusion

This article presented a prescriptive process monitoring framework that extends existing approaches for predictive process monitoring with a mechanism to raise alarms, and hence trigger interventions, to prevent or mitigate the effects of undesired outcomes. The framework incorporates a cost model to capture the trade-offs between the cost of intervention, the benefit of mitigating or preventing undesired outcomes, and the cost of compensating for
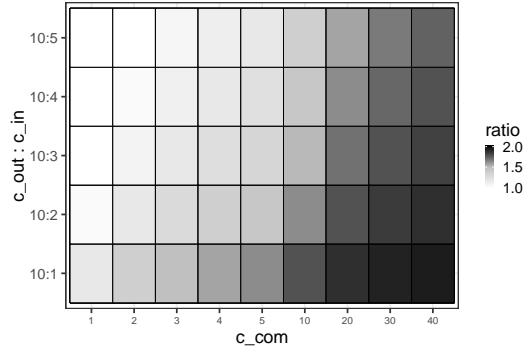
Fig. 13: Ratio of false positives of alarm 1 and alarm 2

Research Question	Answer
1 *How good are the thresholds identified by empirical thresholding in terms of the reduction of the average processing cost for different alarm model configurations?*	Thresholds yield a consistent benefit for virtually all configurations
2 *How does the mitigation effectiveness affect the benefit obtained by an alarm system?*	Consistent benefit, largely independent of the mitigation effectiveness
3 *How does the cost of compensation affect the benefit obtained by an alarm system?*	Consistent benefit, largely independent of the cost of compensation
How is the average processing cost per case affected:	
4 *When training a parameter for the minimum number of events that exceed the threshold?*	Benefit for virtually all configurations
5 *When using more than one threshold interval?*	Benefit for most configurations
6 *When increasing the number of prefix-length-dependent thresholds?*	Benefit for some configurations, depends on class ratio and costs
7 *When combining prefix-length dependent thresholds with a firing delay, compared to the a single-alarm system?*	Consistent benefit for virtually all configurations
8 *How is the average processing cost per case affected when using multiple alarms compared to the best system with only one alarm?*	Benefit for large cost differences of alarms

Table 6: Summary of our findings with respect to the research questions

unnecessary interventions. We also showed how to optimize the threshold(s) for generating alarms, with respect to a given configuration of the alarm model and event log.

An empirical evaluation on real-life logs showed significant benefits in optimizing the alarm threshold, relative to the baseline case where an alarm is raised when the likelihood of a negative outcome exceeds a pre-determined value. We also highlighted that under some conditions, it is preferable to use

multiple alarm thresholds. Finally, it became apparent that additional cost reductions can be obtained, in some configurations, by delaying the firing of an alarm. These findings provide insights into how alarm-based systems for prescriptive process monitoring shall be configured in practice.

Our work opens various directions for future research. We plan to lift the assumption of an alarm always triggering an intervention (regardless of, for example, the workload of process workers). To this end, our framework shall be extended with notions of resource capacity and utilization, possibly drawing on our previous work on risk-aware resource allocation across concurrent cases [4]. We also strive for incremental tuning of the alarming mechanism based on feedback about the alarm relevance and the intervention effectiveness. We foresee that active learning methods could be applied in this context. Our work could also be integrated with approaches like the ones presented in [11, 12] or those reviewed in [8] in the case where there is an abstraction gap between the low-level events recorded in the log and the high-level activities performed during the process enactments.

Acknowledgement

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Allwein, E.L., Schapire, R.E., Singer, Y.: Reducing multiclass to binary: A unifying approach for margin classifiers. Journal of machine learning research **1**(Dec), 113–141 (2000)
3. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Proc. of NIPS, pp. 2546–2554 (2011)
4. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M., ter Hofstede, A.H.: A recommendation system for predicting risks across multiple business process instances. Decision Support Systems **69**, 1–19 (2015)
5. Dachraoui, A., Bondu, A., Cornuéjols, A.: Early classification of time series as a non myopic sequential decision making problem. In: ECML PKDD, pp. 433–447. Springer (2015)
6. Dees, M., de Leoni, M., Mannhardt, F.: Enhancing process models to improve business performance: A methodology and case studies. In: Proc. of CoopIS, pp. 232–251. Springer (2017)
7. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Services Computing (2017)
8. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **10**(3) (2020). DOI 10.1002/widm.1346. URL https://doi.org/10.1002/widm.1346
9. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 155–164. ACM (1999)
10. Elkan, C.: The foundations of cost-sensitive learning. In: Proc. of IJCAI, pp. 973–978. Morgan Kaufmann (2001)
11. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: Efficiently interpreting traces of low level events in business process logs. Inf. Syst. **73**, 1–24 (2018)

12. Fazzinga, B., Folino, F., Furfaro, F., Pontieri, L.: An ensemble-based approach to the security-oriented classification of low-level log traces. Expert Syst. Appl. **153**, 113,386 (2020)
13. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we need hundreds of classifiers to solve real world classification problems. JMLR **15**(1), 3133–3181 (2014)
14. Gröger, C., Schwarz, H., Mitschang, B.: Prescriptive analytics for recommendation-based business process optimization. In: Proc. of BIS, pp. 25–37. Springer (2014)
15. Krumeich, J., Werth, D., Loos, P.: Prescriptive control of business processes. BISE **58**(4), 261–280 (2016)
16. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems **56**, 235–257 (2016)
17. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Proc. of CAiSE, pp. 457–472. Springer (2014)
18. Metzger, A., Föcker, F.: Predictive business process monitoring considering reliability estimates. In: Proc. of CAiSE, pp. 445–460. Springer (2017)
19. Metzger, A., Leitner, P., Ivanovic, D., Schmieders, E., Franklin, R., Carro, M., Dustdar, S., Pohl, K.: Comparing and combining predictive business process monitoring techniques. IEEE Trans. Systems, Man, and Cybernetics: Systems **45**(2), 276–290 (2015)
20. Metzger, A., Neubauer, A., Bohn, P., Pohl, K.: Proactive process adaptation using deep learning ensembles. In: Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings, pp. 547–562 (2019)
21. Mori, U., Mendiburu, A., Dasgupta, S., Lozano, J.A.: Early classification of time series by simultaneously optimizing the accuracy and earliness. IEEE transactions on neural networks and learning systems (99), 1–10 (2017)
22. Mori, U., Mendiburu, A., Keogh, E., Lozano, J.A.: Reliable early classification of time series based on discriminating the classes over time. Data Mining and Knowledge Discovery **31**(1), 233–263 (2017)
23. Nguyen, A., Chatterjee, S., Weinzierl, S., Schwinn, L., Matzner, M., Eskofier, B.M.: Time matters: Time-aware lstms for predictive business process monitoring. In: S.J.J. Leemans, H. Leopold (eds.) Process Mining Workshops - ICPM 2020 International Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers, *Lecture Notes in Business Information Processing*, vol. 406, pp. 112–123. Springer (2020)
24. Olson, R.S., La Cava, W., Mustahsan, Z., Varik, A., Moore, J.H.: Data-driven advice for applying machine learning to bioinformatics problems. In: Proc. of Biocomputing, pp. 192–203. World Scientific (2017)
25. Parrish, N., Anderson, H.S., Gupta, M.R., Hsiao, D.Y.: Classifying with confidence from incomplete information. The Journal of Machine Learning Research **14**(1), 3561–3589 (2013)
26. Pednault, E., Abe, N., Zadrozny, B.: Sequential cost-sensitive decision making with reinforcement learning. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 259–268. ACM (2002)
27. Platt, J., et al.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers **10**(3), 61–74 (1999)
28. Sheng, V.S., Ling, C.X.: Thresholding for making classifiers cost-sensitive. In: AAAI, pp. 476–481 (2006)
29. Tavenard, R., Malinowski, S.: Cost-aware early classification of time series. In: ECML PKDD, pp. 632–647. Springer (2016)
30. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. Knowl. Discov. Data **13**(2), 17:1–17:57 (2019)
31. Teinemaa, I., Tax, N., de Leoni, M., Dumas, M., Maggi, F.M.: Alarm-based prescriptive process monitoring. In: Proceedings of the Business Process Management Forum (BPM Forum), pp. 91–107. Springer (2018)
32. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. International Journal of Data Warehousing and Mining (IJDWM) **3**(3), 1–13 (2007)
33. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: D. Fahland, C. Ghidini, J. Becker,

M. Dumas (eds.) Business Process Management Forum - BPM Forum 2020, Seville, Spain, September 13-18, 2020, Proceedings, *Lecture Notes in Business Information Processing*, vol. 392, pp. 193–209. Springer (2020)
34. Xing, Z., Pei, J., Philip, S.Y.: Early classification on time series. KAIS **31**(1), 105–127 (2012)
35. Zadrozny, B., Elkan, C.: Learning and making decisions when costs and probabilities are both unknown. In: Proc. of KDD, pp. 204–213. ACM (2001)