Clustering-Based Predictive Process Monitoring

Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi and Irene Teinemaa

Abstract—The enactment of business processes is generally supported by information systems that record data about each process execution (a.k.a. case). This data can be analyzed via a family of methods broadly known as process mining. Predictive process monitoring is a process mining technique concerned with predicting how running (uncompleted) cases will unfold up to their completion. In this paper, we propose a predictive process monitoring framework for estimating the probability that a given predicate will be fulfilled upon completion of a running case. The framework takes into account both the sequence of events observed in the current trace, as well as data attributes associated to these events. The prediction problem is approached in two phases. First, prefixes of previous (completed) cases are clustered according to control flow information. Secondly, a classifier is built for each cluster using event data attributes to discriminate between cases that lead to a fulfillment of the predicate under examination and cases that lead to a violation within the cluster. At runtime, a prediction is made on a running case by mapping it to a cluster and applying the corresponding classifier. The framework has been implemented in the ProM toolset and validated on a log pertaining to the treatment of cancer patients in a large hospital.

Index Terms—process mining, predictive monitoring, sequence classification, clustering

1 Introduction

Process mining is a family of methods to analyze business processes based on their observed behavior recorded in *event logs*. In this setting, an event log is a collection of *traces*, each representing one execution of the process (a.k.a. a *case*). A trace consists of a sequence of timestamped events, each capturing the execution of an activity. Each event may carry a payload consisting of attribute-value pairs such as the resource(s) involved in the execution of the activity, or other data recorded with the event.

Predictive business process monitoring [1] is a category of process mining methods that aims at predicting at runtime and as early as possible the outcome of ongoing cases of a process given their uncompleted traces. In this context, an outcome of a case may be the fulfillment of a constraint on the cycle time of the case, the validity of a temporal logic constraint, or any predicate over a completed case. For example, in a sales process, a possible outcome may be the placement of a purchase order by a potential customer, while the corresponding negative outcome is the non-placement of a purchase order. Meanwhile, in a medical treatment process, a possible outcome is the recovery of the patient upon completion of the treatment, while the corresponding negative outcome is the non-recovery of the patient after that given treatment. These examples illustrate that the problem of predictive process monitoring is a problem of early sequence classification [2]: given a trace of an uncompleted case (which can be modeled as a sequence of events with data payloads), we seek to predict as early as possible whether the outcome of the case will fall into a positive class or a negative class.[1]

In previous work [1], we presented a predictive process monitoring method where a classifier is constructed at runtime to predict the outcome of an ongoing case of the process. However, while achieving a relatively high level of accuracy, this method incurs a significant runtime overhead – in the order of seconds or even minutes per prediction – making it inapplicable in settings with high throughput or when instantaneous response times are required. The slow response times are due to the fact that the classifier used for predicting the outcome of a given case is built at runtime.

This paper presents an alternative approach that significantly reduces the runtime overhead while maintaining a comparable accuracy. The crux of the approach is to build the classifiers offline, so that the runtime step consists simply in matching an uncompleted trace (prefix) to a given classifier and applying the latter to make a prediction. The offline component follows a two-phase approach. First, prefixes of traces of historical cases are extracted and clustered without looking at the event payloads, so that existing methods for trace clustering can be applied. Secondly, for each such cluster, a classifier is constructed now taking into account the payload (i.e., the data attributes) associated to the events in the trace prefixes. The classifier is targeted at discriminating between trace prefixes that lead to fulfillments of the monitored predicate vs. those that lead to violations. Finally, the online phase consists in taking the uncompleted trace of a running case, matching it to a cluster, and applying the corresponding classifier to estimate the probability of fulfillment of the monitored predicate.

The proposed approach is wrapped as a generic framework that can be instantiated by selecting three input methods: (i) a method for encoding traces in the event log as feature vectors; (ii) a clustering method; and (iii) a

- *C. Di Francescomarino is with FBK-IRST, Via Sommarive 18, 38050 Trento, Italy.*
  E-mail: dfmchiara@fbk.eu
- *M. Dumas, F. M. Maggi, I. Teinemaa are with the University of Tartu, Liivi 2, 50409 Tartu, Estonia.*
  E-mail: {marlon.dumas, f.m.maggi, irheta}@ut.ee

1. In principle, we could also consider a larger set of classes (not just two). In this paper, we focus on binary classification, but the same techniques could be applied to N-ary classification.

classification method. The proposed *PM Framework* has been implemented in the ProM process mining toolset, specifically in the Operational Support (OS) environment [3], [4], [5]. This latter environment takes as input a stream of events (e.g., produced by an enterprise system) and updates a set of predictions for each new incoming event.

Using an event log of a patient treatment process in a hospital, we have evaluated four instantiations of the *PM Framework*, corresponding to two feature encodings (frequency-based and sequence-based) , two clustering methods (model-based clustering and DBSCAN) and two classification methods (decision trees and random forests). These instantiations of the framework are compared in terms of their ability to consistently produce predictions early, accurately and with low runtime overhead.

The paper is structured as follows. Section 2 introduces the proposed framework. Section 3 presents the experimental evaluation. Section 4 discusses related work and Section 5 draws conclusions and directions for future work.

2 Predictive Monitoring Framework

The goal of predictive monitoring is to determine if a current running trace will reach a given outcome based on historical traces. Hence, like other process monitoring techniques, our framework (herein called the *PM Framework*) requires a *labeling function* that, given a trace, tells us if it is normal or deviant. Accordingly, the *PM Framework* takes as input both an event log and a labeling function f_c. The labeling function can be defined, for example, using Linear Temporal Logic (LTL) rules, as discussed later.

In this section, we first introduce an illustrative example. We then describe an "on-the-fly" approach for predictive monitoring presented in [1], which we use as baseline. Finally, we introduce the proposed *PM Framework*.

2.1 Running Example

In Fig. 1, we show examples of executions pertaining to a patient diagnosis process. In trace t_1, a message with a diagnosis request (M) arrives. The request contains a list of patient's symptoms, e.g., $pain_A$. The patient is required to do some clinical analysis (A) and, once the results of the analysis are received, the reception is confirmed (C). Then, a diagnosis is made by the doctor (D), e.g., d_1, and again two times some new clinical analysis is required. After that, the hospital fee is paid by the patient (P), a new diagnosis is made by the doctor and the reception of the analysis is confirmed. Finally the patient recovers from the disease (R).

Data consumed and produced in the process is globally visible throughout the whole process execution in the form of *attribute:value* pairs carried by event payloads. The *payload* of an event, signaling the execution of an activity A, contains the values of each attribute after the execution of A as well as the values of attributes for each activity that occurred before activity A in the trace. For example, in Fig. 1, the payload associated to the doctor diagnosis activity (D) contains the values associated to attributes of the diagnosis activity, e.g., the diagnosis (dia) and possibly the prescription (pre) of the doctor, as well as those associated to past activities, e.g., patient's symptoms (sym). We write $P(D) = \{sym : pain_A, dia : d_1, pre : p_1, ...\}$ to refer to the payload of activity D.

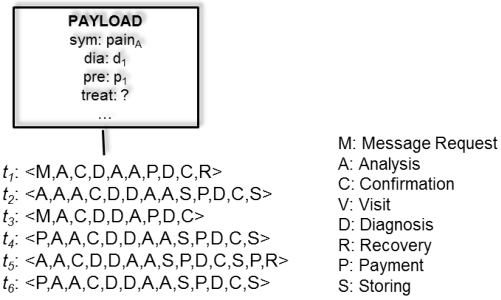

Fig. 1: Patient diagnosis example.

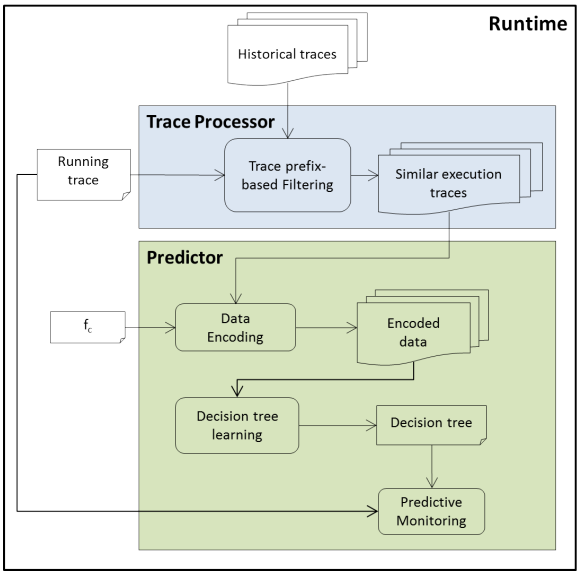

Fig. 2: On-the-fly predictive monitoring.

2.2 On-the-fly Predictive Monitoring

In this section, we report the details of the approach presented in [1]. This approach builds classification models on-the-fly at runtime based on historical trace prefixes of completed cases to provide predictions about the fulfillments of a predicate in a running trace. In the following, we report an overview of the approach and of its implementation.

Fig. 2 sketches the approach. It relies on two main modules: a *Trace Processor* module to filter (past) execution traces and a *Predictor* module, which uses the information contained in the *Trace Processor* output as training data to provide predictions. Both modules operate at runtime.

The *Trace prefix-based Filtering* submodule of the *Trace Processor* module extracts from the set of historical traces only those traces having a prefix control flow similar to the one of the running trace (up to the current event). The filtering is needed since traces with similar prefixes are more likely to have, eventually in the future, a similar behavior. The similarity between two traces is evaluated based on their string-edit distance. We use this abstraction (instead of considering traces with a prefix that perfectly matches the current partial trace) to guarantee a sufficient number of examples to be used for the decision tree learning. In particular, a *similarity threshold* can be specified to include more traces in the training set (by considering also the ones

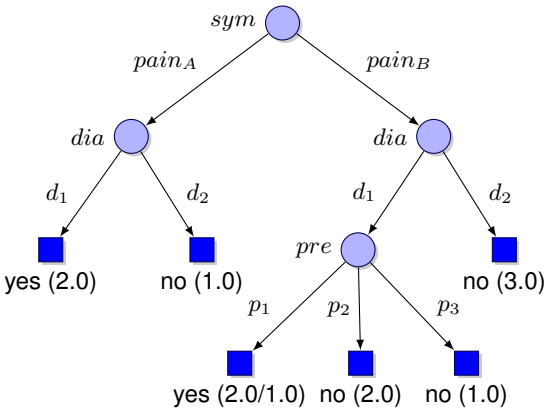

Fig. 3: Example decision tree.

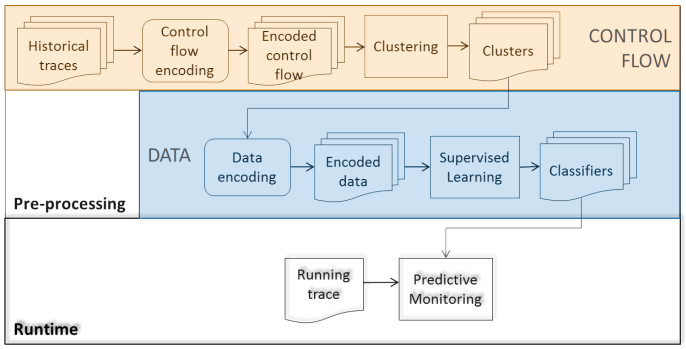

Fig. 4: *PM Framework*.

that are less similar to the running trace).

The traces of the training set (and the corresponding selected prefixes) are then passed to the *Data Encoding* submodule, in charge of preparing them to be used for training a decision tree. Specifically, the submodule (i) classifies each (completed) trace based on whether the desired predicate is satisfied or not (this is done by using the input classification function f_c); (ii) identifies for each trace prefix the *payload* containing the assignment of values for each attribute corresponding to the last event in the prefix. The encoding of the trace is then obtained by combining the value of the classification function on the specific trace and the trace payload. For example, given the trace t_1 in Fig. 1, and its prefix $\langle M, A, C, D\rangle$, the payload of D is taken. In particular, assuming that we have the vector of data attributes $\langle sym, dia, pre, treat\rangle$, the encoding for the specific trace prefix will be the vector $\langle pain_A, d_1, p_1, ?, yes\rangle$, where the question mark is used to identify not available data values, while the last value represents the value of the classification function f_c on the specific case. In this example, it represents the fact that the patient in the historical trace t_1 recovered from the disease.

Once the relevant traces and, therefore, the corresponding payloads, are classified and encoded, they are passed to the *Decision tree learning* module, in charge of deriving the learned decision tree. The decision tree is queried using the payload of the running trace to derive a prediction. Fig. 3 shows a decision tree related to our running example. The non-terminal nodes of the tree contain the decision points for the prediction (the data attributes in our case), while the arcs are labeled with possible values assigned to the attribute in the node. The leaves, instead, represent the value of the classification function on the specific path tree. The number of data training examples (with values of the input variables following the path from the root to each leaf), respectively correctly and non-correctly classified, is reported on the corresponding leaf of the tree. For example, if the payload of the current execution trace corresponds to values $pain_B$, d_1 and p_1, the resulting class is the formula satisfaction ("yes"), with 2 examples of the training set following the same path correctly classified (class support) and 1 non-correctly classified, i.e., with a class probability $prob = \frac{2}{2+1} = 0.66$. Therefore, in this case, the *Predictor* will predict the satisfaction of the formula with a class probability $prob = 0.66$. Note that if a path from the root to a leaf of the tree cannot be identified starting from the payload of the current execution trace (e.g., if some data is missing) no prediction can be returned.

2.3 Clustering-Based Predictive Monitoring

Differently from the approach described in the previous section, in the proposed framework, the on-the-fly construction of the decision tree can be avoided by applying a simple pre-processing phase. In such a phase, state-of-the-art approaches for clustering and classification are applied to the historical data in order to (i) identify and group historical trace prefixes with a similar control flow (clustering from a control flow perspective); and (ii) get a precise classification based on the data of traces with similar control flow (data-based classification). At runtime, the classification of the historical trace prefixes is used to classify new traces during their execution and predict how they will behave in the future. The overall picture of the framework is illustrated in Fig. 4. In the following, we describe each framework component in detail.

2.3.1 Control flow encoding

Before applying state-of-the-art techniques for clustering and classification, two propaedeutical steps are applied: (i) the selection of the historical trace prefixes to consider; and (ii) their encoding. In particular, prefixes of past execution traces are selected (rather than the entire trace or all the prefixes for a trace). The reason behind this choice is twofold: on the one side, taking all the prefixes could become very expensive in terms of efficiency. On the other side, we are interested in early predictions, when still reparative actions can be undertaken to prevent violations. In this light, considering only the initial parts of the historical traces seems to be a reasonable choice. For example, given the 6 traces $t_1, \ldots, t_6$ of Fig. 1, only a selection of k prefixes for each trace will be considered. Different approaches can be used for the selection of these k prefixes. For example, the first k prefixes of each historical trace can be selected or alternatively k prefixes, one every g events. In the latter case, two prefixes differ one from another for a *gap* of g events.[2] Different approaches can also be taken to perform the encoding of trace prefixes for clustering. Just to name a

2. k and g are user-specified parameters.

few, a trace (prefix) can be encoded as a sequence of events or in terms of the frequency of the occurrence of sequence patterns in the trace. The simplest case is the one related to the occurrence of unary patterns, i.e., patterns composed of a single log event. For example, in the scenario in Fig. 1, we can represent the alphabet of the events as an ordered vector $L = \langle A, C, D, M, P, R, S, V \rangle$. In this case, trace t_1 will be encoded as a vector of frequencies $\langle 3, 2, 2, 1, 1, 1, 0, 0 \rangle$, obtained by replacing each symbol of the alphabet in vector L by its frequency in trace t_1. Trace prefixes encoded in this way are used as input of the clustering phase.

2.3.2 Clustering

In the clustering phase, a selection of prefixes of the historical traces with the same (control flow) characteristics is grouped together based on some distance notion. Examples of distances are Euclidean distance and the string-edit distance. The historical traces contained in each cluster are then used to generate a classifier, that is exploited, in turn, to make predictions on running traces, once identified their matching cluster. For example, the execution traces in Fig. 1 could be grouped by a clustering algorithm in two clusters c_1 and c_2, according to the similarities in their control flow, so that $c1$ contains traces t_1 and t_3 (which have a very similar control flow), and c_2 contains the remaining four traces.

2.3.3 Trace encoding

Trace prefixes in each cluster are used as input for supervised learning. In this case, the data perspective is taken into consideration. Historical execution traces are encoded using the available data attributes in the event payloads, i.e., prefixes clustered based on control flow are now analyzed from a data perspective. Similarly to the on-the-fly approach, each prefix is encoded as a feature vector that includes elements corresponding to the data assignments contained in the payload associated to the last event of the prefix. In addition, each prefix in a cluster is classified based on whether the corresponding completed trace is "normal" or "deviant" with respect to the input labeling function f_c.

2.3.4 Supervised learning

Each cluster is used as training set of a supervised learning technique (e.g., decision tree learning, random forest) to generate a classifier that allows for discriminating between deviant and normal behaviors. For example, given two clusters c_1 and c_2, for each of them a classifier is built.

2.3.5 Predictive monitoring

At runtime, the set of classifiers generated during the pre-processing phase is used to make predictions about how the behavior of a current running trace will develop in the future. At any point in time, the current prefix of the running trace is classified as part of one of the clusters identified during the pre-processing phase. This is done by considering the cluster containing the prefix with the minimum distance from the current prefix. Based on the selected cluster (and, therefore, based on the control flow characteristics of the current prefix) the corresponding classifier is selected. This classifier is queried using the payload of the last event of the current prefix (exploiting the data perspective of the current prefix). For example, given a partial execution trace $t_p : \langle M, A, C, D \rangle$ and the predicate "the patient will recover eventually", we first identify the cluster to which the partial trace belongs, e.g., c_1, and then the classifier associated to the cluster (e.g., the decision tree in Fig. 3) is exploited in order to predict whether the predicate will be verified or not.

3 Evaluation

We implemented the proposed *PM Framework* as a so-called "Operational Support (OS) provider" on top of the ProM framework.[3] In this way, the framework can be used in a "streaming" mode, meaning that it can take as input a stream of events coming from an external system. Specifically, the *PM Framework* uses the Weka implementation of the clustering and classification algorithms. Using such an implementation, we conducted an evaluation of different configurations of the proposed framework on a real-life dataset as reported below.

3.1 Dataset

We conducted experiments using the BPI challenge 2011 [6] event log. This log records the execution of a cancer treatment process in a Dutch academic hospital over a three-years period. The log contains $1,143$ traces and $150,291$ events. Each trace in the log refers to the treatment of one patient. Each event represents an execution of one among 623 activities. Each event contains a timestamp, an event type (i.e., an activity lifecycle state like *start* or *complete*), a case (i.e., patient) identifier, and a number of domain-specific attributes (e.g., *Age*, *Diagnosis*, and *Treatment code*). There are 15 domain-specific attributes in total.

Since the goal of predictive monitoring is to classify a case as normal or deviant, we need to define a notion of deviance (i.e., a *labeling function*). We experimented with 4 labeling functions corresponding to the following LTL rules:

- $\varphi_1 = \mathbf{F}(\text{“}tumor\ marker\ CA - 19.9\text{”}) \vee \mathbf{F}(\text{“}ca - 125\ using\ meia\text{”})$,
- $\varphi_2 = \mathbf{G}(\text{“}CEA - tumor\ marker\ using\ meia\text{”} \rightarrow \mathbf{F}(\text{“}squamous\ cell\ carcinoma\ using\ eia\text{”}))$,
- $\varphi_3 = (\neg\text{“}histological\ examination - biopsies\ nno\text{”})\ \mathbf{U}(\text{“}squamous\ cell\ carcinoma\ using\ eia\text{”})$, and
- $\varphi_4 = \mathbf{F}(\text{“}histological\ examination - big\ resectiep\text{”})$.

For a given LTL rule, if a case violates the LTL rule, it is labeled as deviant (herein called a "positive" case); otherwise it is labeled as normal (herein called a "negative" case). Each of these labeling functions captures a business rule. Specifically, φ_1 assesses that either the diagnostic test for the tumor marker CA-19.9 or for the tumor marker ca-125 has to be performed. φ_2 states that every time the diagnostic test for the CEA tumor marker is performed, then the eia test for the squamous cell cancer has also to be performed eventually. φ_3 assesses that no histological examination can be performed until the eia test for the squamous cell cancer is performed and, finally, φ_4 states that the resection for the histological examination has to be performed eventually.

3. http://processmining.org

The distribution of positive and negative cases in the event log is: 458 negative vs. 682 positive for φ_1, 893 negative vs. 247 positive for φ_2, 259 negative vs. 881 positive for φ_3, and 319 negative vs. 821 positive for φ_4.

3.2 Experimental Settings

The experimentation workflow is outlined in Fig. 5. First, we order the traces in the log based on the time at which the first event of each trace has occurred. Then, we split the log temporally into two parts: 80%-20%. We used the first part as training log, i.e., we used these traces as historical data to construct clusters and build the classification models for prediction. Then, we implemented a log replayer to simulate the execution of the remaining traces (the testing log) by pushing them as an event stream to the implementation of the *PM Framework* and making predictions for each case during this replay.

The first step in the offline component of the framework is to extract a set of prefixes from the historical traces and to encode each prefix as a vector in order to calculate clusters of prefixes. For the experiments, we extracted all prefixes of historical traces starting from prefixes of length 1 and up to length 21 in steps of g, where $g \in \{3, 5, 10\}$. For example, for $g = 5$, we extracted all prefixes of lengths 1, 6, 11, 16, and 21 for each trace in the training set.

The second step in the framework is to construct clusters of prefixes. For this, we used two popular clustering methods, namely model-based clustering [7] and DBSCAN [8]. In model-based clustering, we need to calculate the center point and covariance matrix of clusters. These parameters can only be computed if we use an Euclidean distance. The string-edit distance – which is otherwise a natural distance in the context of traces – is not an Euclidean distance. Hence, we applied model-based clustering using the Euclidean distance over the frequency-encoded prefixes. On the other hand, in DBSCAN, we just need to calculate the distance between two points and this can be done using the edit distance. Accordingly, for DBSCAN, we used edit distance over sequence-encoded prefixes.

For the model-based clustering method, it is necessary to set the number of clusters to be created (parameter k). Meanwhile, DBSCAN requires the minimum number of points in a cluster (parameter $minPoints$) and the minimum radius of a cluster (parameter ϵ). For each of the three datasets of prefixes ($g \in \{3, 5, 10\}$), we identified the optimal parameters for each of these clustering methods. In the case of model-based clustering, we applied model-based clustering with $k = 15$ to 35 clusters and chose the value of k that achieved the highest Bayesian Information Criteria (BIC). Meanwhile, the DBSCAN optimal parameters were estimated by using the *sorted k-dist graph* [8]. This led us to set $minPoints = 4$ and $\epsilon = 0.125$.

The third step in the framework is to build a classifier for each cluster of trace prefixes. Specifically, each cluster is used as training set of a supervised learning technique to generate a classifier that discriminates between deviant and normal cases. A range of classification algorithms has been proposed in the literature. In this paper, we use decision trees, which are known for the interpretability of the models they generate, and random forests, which apply similar principles as decision trees but are designed to maximize accuracy rather than interpretability.

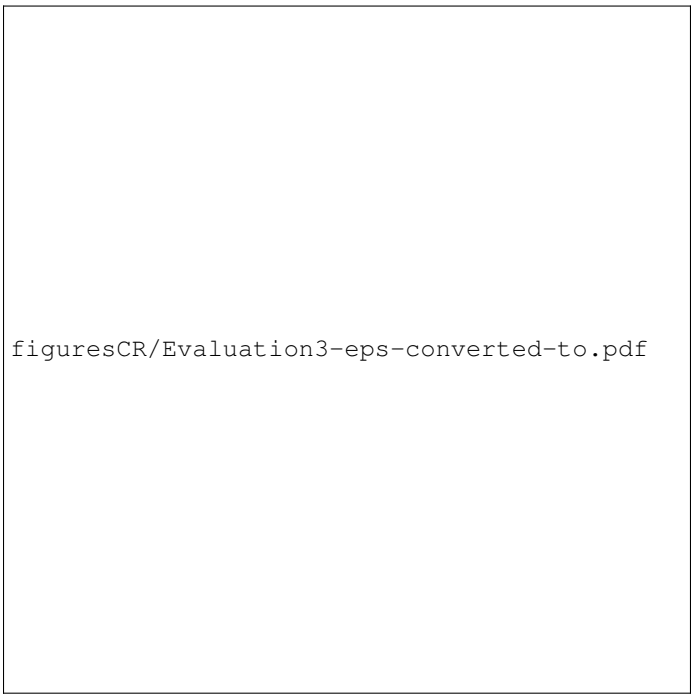

Fig. 5: Experimentation workflow.

Combining the two clustering and the two classification techniques, we obtain the following four *PM Framework* instances:

- *mbased_dt*: model-based clustering and decision trees;
- *dbscan_dt*: DBSCAN clustering and decision trees;
- *mbased_rf*: model-based clustering and random forests;
- *dbscan_rf*: DBSCAN clustering and random forests.

We replayed each trace in the testing set and produced a prediction of the outcome of the case every 5 events (starting from the first event in each trace). Each prefix of each running trace is encoded in the same way as the historical traces and assigned to the closest cluster. In case of model-based clustering, the closest cluster is the one with the minimum Euclidean distance from the current prefix, while for DBSCAN the closest cluster is the cluster containing the prefix with the minimum edit distance from the current prefix. We use the classifier associated to the closest cluster to predict the label for the current running case. To consider a prediction reliable, the corresponding class support and class probability need to be above a given minimum class support and minimum class probability threshold. In our experiments, the minimum class support is set to $s = 6$. The minimum class probability thresholds considered are $prob \in \{0.6, 0.7, 0.8, 0.9\}$. The trace in the testing set is replayed until either a satisfactory prediction is achieved or the end of the trace is reached. The latter situation is herein called a *prediction failure*.

For the on-the-fly approach, the minimum class support is also set to $s = 6$. The minimum class probability thresholds considered are $prob \in \{0.6, 0.7, 0.8, 0.9\}$. In this case, we extracted all prefixes of historical traces starting from prefixes of length 1 and up to length 21, one every $g = 10$ events. As for the *PM Framework*, also in this case,

we replayed each trace in the testing set and produced a prediction every 5 events.

3.3 Research Questions and Metrics

The goal of the evaluation is focused on two aspects: the performance of the approach in terms of the quality of the results and the performance of the approach in terms of the time required to provide predictions.

In particular, we are interested in answering the following two research questions:

RQ1 How *effective* is the *PM Framework* in providing *accurate* results as *early* as possible?

RQ2 How *efficient* is the *PM Framework* in providing results?

To answer **RQ1**, we evaluated the performance of the approach in terms of its prediction *accuracy*. However, as highlighted by Salfner et al. in related work in the field of failure prediction methods [9], using accuracy measures as sole indicators of the effectiveness of a prediction technique may be misleading. Accordingly, we also included metrics associated to the *earliness* of the prediction, to reflect the desire to make a prediction with sufficient confidence as early as possible. Furthermore, since the proposed method may sometimes fail to make a prediction at all for a given trace, we also measured the percentage of cases where a prediction is not made (*failure-rate*). Finally, to answer **RQ2**, we measured the *computation time* required to provide a prediction. Below, we discuss in more detail how each of these performance dimensions was measured.

F1-score: This measure is defined with respect to a *gold standard* that indicates the correct labeling of each trace. In our experiments, we extracted the gold standard by evaluating the input predicate on each completed trace in the testing set. Given the gold standard, we classify predictions made at runtime into four categories: *i*) true-positive (T_P: positive outcomes correctly predicted); *ii*) false-positive (F_P: negative outcomes predicted as positive); *iii*) true-negative (T_N: negative outcomes correctly predicted); *iv*) false-negative (F_N: positive outcomes predicted as negative). F1-score, which intuitively represents the proportion of correctly classified positive results with respect to all the possible cases, is defined as:

$$\textit{F1-score} = \frac{2T_P}{2T_P + F_P + F_N} \tag{1}$$

Earliness: As already mentioned, during the replay of each trace in the testing log, we give a prediction every 5 events (starting from the first event in the trace). While replaying a trace, we consider a prediction reliable and we stop the replay when the corresponding class probability is above the given minimum class probability threshold. For this reason, the earliness is directly dependent on the chosen class probability threshold. The lower the class probability threshold is, the earlier a prediction is given. The earliness of the prediction is defined as 1 minus the ratio between the index indicating the position of the last evaluation point (the one corresponding to the reliable prediction) and the size of the trace under examination. Earliness is a crucial measure since, during the execution of a business process, the stakeholders must be provided with predictions as soon as possible to apply possible reparative actions in case there is high probability of deviance in the future.

Failure-rate: Sometimes it happens that, when replaying a trace of the testing set, the end of the trace is reached and no prediction has been made with a sufficiently high class probability. In this case, the answer of the predictor is "maybe" to indicate that it was not possible to provide a reliable prediction. The percentage of traces in the log that lead to a failure in the prediction is called failure-rate. Like earliness, also the failure-rate is directly dependent on the chosen minimum class probability threshold. If we set the class probability threshold to 0.5, a prediction would always be made right away from the first event in a trace – except in the very unlikely case where the classifier gives a class probability of exactly 0.5 to each class. With a slightly higher threshold (e.g., 0.6), a prediction is very likely to be made at some point in each trace, and hence the failure-rate is very low. This latter setting is suitable if the user considers that "no prediction" (i.e., failure) is equivalent to a "wrong prediction".

Computation time: We estimate three different types of computation times required for providing a prediction:

- *init time*: the time required for pre-processing, i.e., for clustering and supervised learning;
- *processing time*: the total time required for processing the entire testing set;
- *average prediction time*: the average time required to the predictor for returning an answer at each evaluation point.

3.4 Results

Fig. 6 reports *F1-score*, *failure-rate* and *earliness* obtained by applying the baseline on-the-fly approach for each of the four investigated predicates ($\varphi_1 - \varphi_4$) with different minimum class probability thresholds. By looking at the plots, the four metrics seem not to be particularly affected by the differences in terms of minimum class probability thresholds. Overall, for the four predicates, the F1-score seems to depend on the formula under analysis, ranging between a minimum value of 0.47 for φ_1 to a maximum of 0.93 for for φ_4. The failure-rate is generally high, varying from a value of 0.36 for φ_3 up to 0.6 for φ_1. The earliness for φ_1 is slightly lower than for the other predicates.

Fig. 7 plots, for each of the four predicates, *F1-score*, *failure-rate* and *earliness* obtained by instantiating the *PM Framework* with the model-based clustering and with the decision tree classification (*mbased_dt*) for different minimum class probability thresholds and prefix gaps. The plots show that *mbased_dt* reaches peaks of F1-score of 0.92 for φ_1 with a threshold for minimum class probability ($prob = 0.8$). A high failure-rate influences the F1-score since, in its computation, we can rely on a lower number of predictions. On the other hand, a high earliness can also negatively affect the F1-score. Indeed, a too high earliness results into very little information carried by the running trace in terms of control flow. Focusing only on the results with a reasonably low failure-rate (e.g., with failure-rate lower than 0.25) and an earliness not excessively high (e.g., lower than 0.9), *mbased_dt* still guarantees to find, for each predicate, a parameter configuration resulting in a good F1-score. The F1-score values range, indeed, between 0.58 and 0.92 for the four predicates.

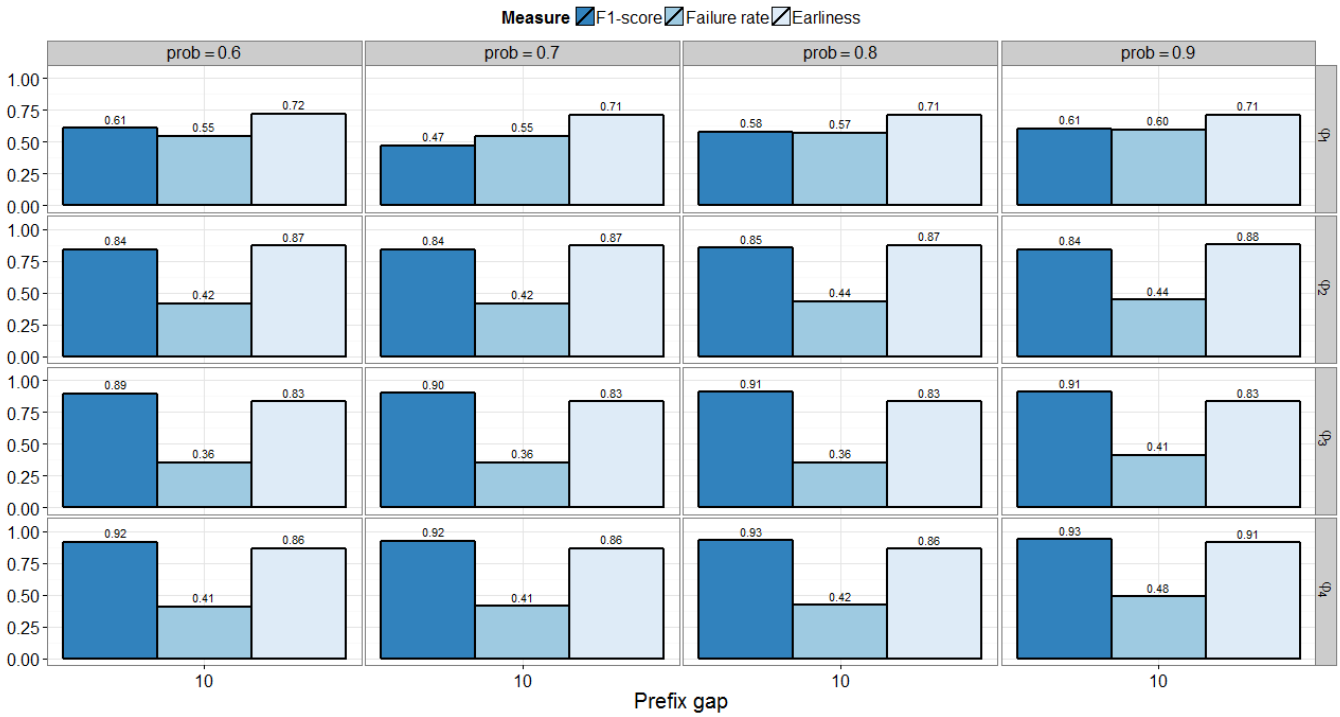

Fig. 6: On-the-fly predictive monitoring - F1-score, failure-rate and earliness.

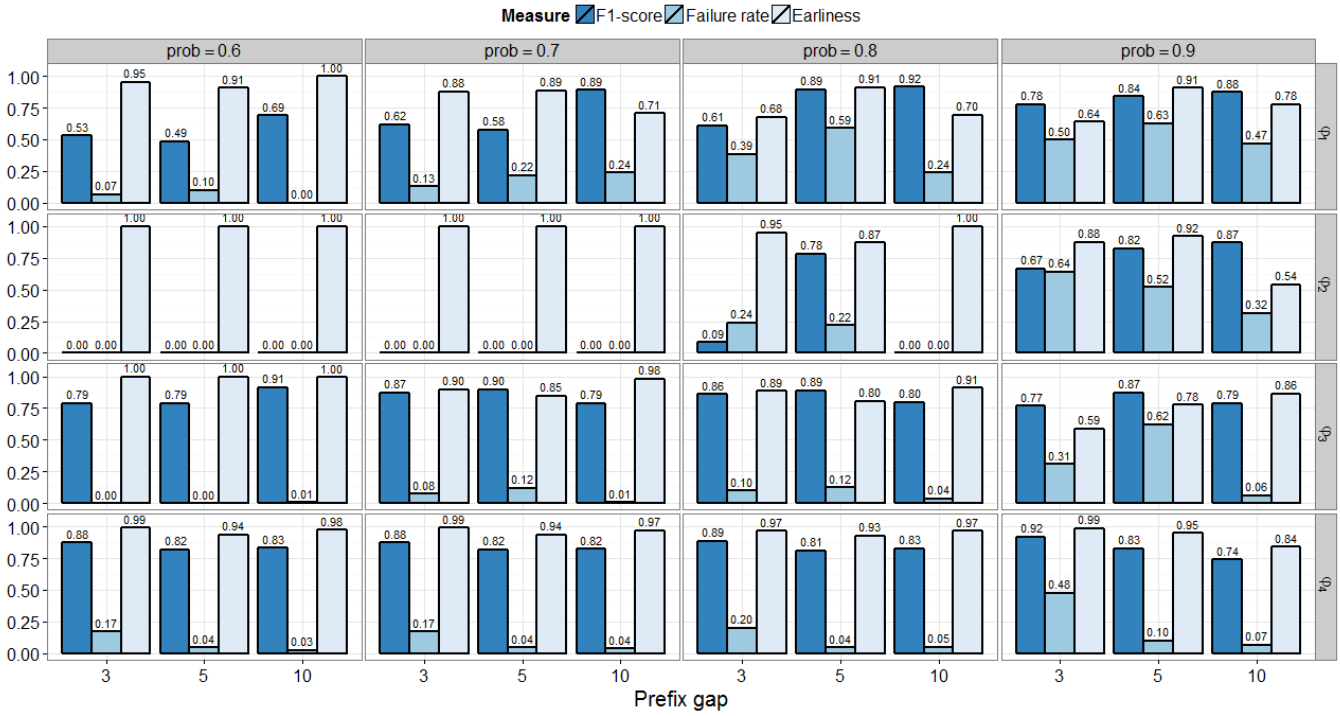

Fig. 7: Model-based clustering and decision tree classification (*mbased_dt*) - F1-score, failure-rate and earliness.

In general, by opportunely selecting the minimum class probability threshold, it is possible to meet different needs and preferences. For example, for *mbased_dt*, it seems that $prob = 0.9$ ensures good values of F1-score, whereas for the other class probability thresholds, often the predictions are given too early, thus resulting in a poor F1-score. This is especially true for φ_2, the only predicate among the ones under examination for which the number of deviant cases is much lower than the number of normal cases.

Fig. 8 shows the same type of plots of Fig. 7 obtained by instantiating the *PM Framework* with DBSCAN clustering and with the decision tree classification (*dbscan_dt*). In this

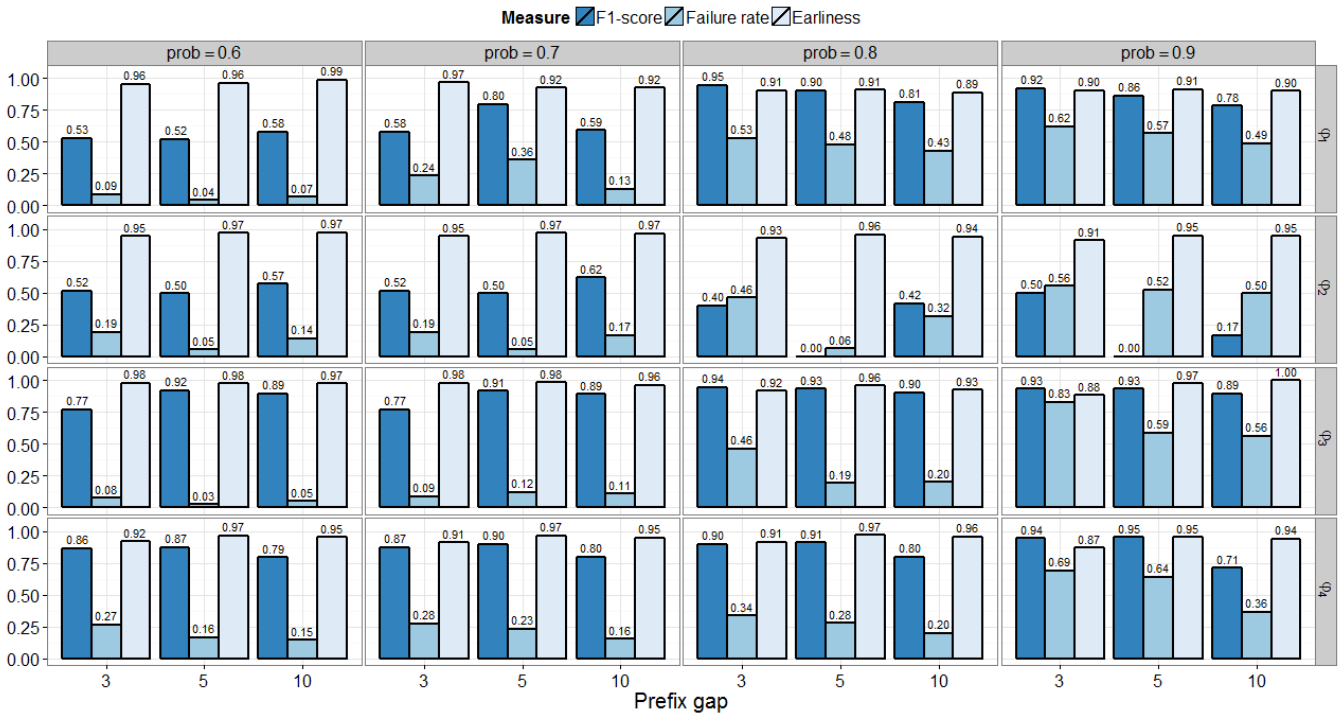

Fig. 8: DBSCAN clustering and decision tree classification (*dbscan_dt*) - F1-score, failure-rate and earliness.

case, the earliness is always very high and this leads to low values for F1-score for the two predicates with a low number of deviant cases, i.e., for φ_1 and especially for φ_2. For class probability thresholds 0.8 and 0.9, the F1-score reaches values higher than 0.9 for three of the four predicates under examination (φ_1, φ_3, and φ_4), whereas for φ_2 it is not able to perform better than 0.5.

Fig. 9 shows the values of F1-score, failure-rate and earliness for the model-based clustering and the random forest classification (*mbased_rf*). For $prob = 0.9$, the F1-score reaches values higher than 0.8 for three of the four predicates under examination (φ_1, φ_3, and φ_4), whereas for φ_2 is equal to 0 for all the prefix gaps. By looking at the only results with failure-rate lower than 0.25 and earliness lower than 0.9, the F1-score values for all predicates lie in the range 0.42-0.86.

Fig. 10 reports the values of the three metrics for the last instance of the *PM Framework*, i.e., the one using DBSCAN clustering and random forest classification (*dbscan_rf*). We note that, also in this case, like in *dbscan_dt*, the earliness is extremely high (for all the considered configurations it is always higher than 0.95). This leads to low values for F1-score for the two predicates with a low number of deviant cases, i.e., φ_1 and φ_2. For $prob = 0.9$, the F1-score reaches values higher than 0.7 for three of the four predicates under examination (φ_1, φ_3, and φ_4), whereas for φ_2 is equal to 0 for all the prefix gaps.

To quantify the difference in terms of the various evaluation metrics of the *PM Framework* with respect to the on-the-fly approach, we (i) evaluated whether a statistically significant difference exists between each of the four *PM Framework* instances (with prefix gap 10) and the on-the-fly approach; and (ii) measured the strength of the difference. Specifically, we applied a two-tailed paired statistical test - the non-parametric Wilcoxon test [10] - on each of the evaluation metrics and on a global measure g obtained by combining the three metrics based on the formula $g = F1\_score * (1 - failure\_rate) * earliness$. The idea is that the higher the value of g is, the better the technique overall performs. Then, we also computed the $Cohen-d$ effect-size [11] to identify whether the magnitude of the difference is small (~ 0.2), medium (~ 0.5), or large (> 0.8). Table 1 reports, for each *PM Framework* instance, the corresponding p-value and effect-size. The p-values related to the statistically significant[4] differences are reported in bold in Table 1. The table shows that there is no statistically significant difference between the on-the-fly approach and the *PM Framework* instances *mbased_dt* and *dbscan_rf* in terms of F1-score. Conversely, the *PM Framework* instances *mbased_rf* and *dbscan_dt* perform statistically significantly worse in terms of F1-score than the on-the-fly approach. On the other hand, by looking at the failure-rate, we can observe that a statistically significant difference exists for three of the four *PM Framework* instances under examination with respect to the on-the-fly approach. In particular, the failure-rate is lower for the *PM Framework* instances. Similarly, we found that in two of the four *PM Framework* instances under examination, we have a statistical difference in terms of earliness with respect to the on-the-fly approach; the earliness is higher for the *PM Framework* instances, which, in those case, perform better than the on-the-fly approach. By looking at the global measure g, we find that on average the *PM Framework* instances perform overall better than

4. The analysis is performed with a level of confidence of 95% (p-value < 0.05), i.e., there is only 5% probability that the results are obtained by chance.

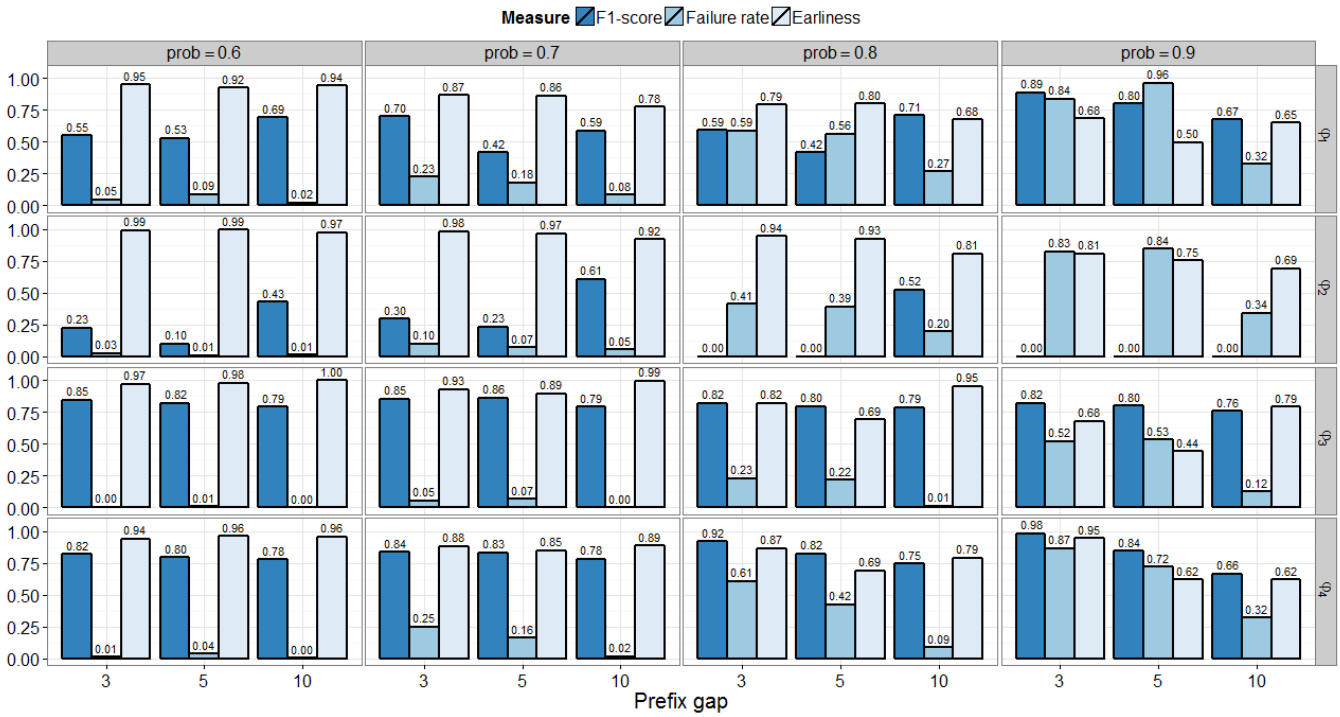

Fig. 9: Model-based clustering and random forest classification (*mbased_rf*) - F1-score, failure-rate and earliness.

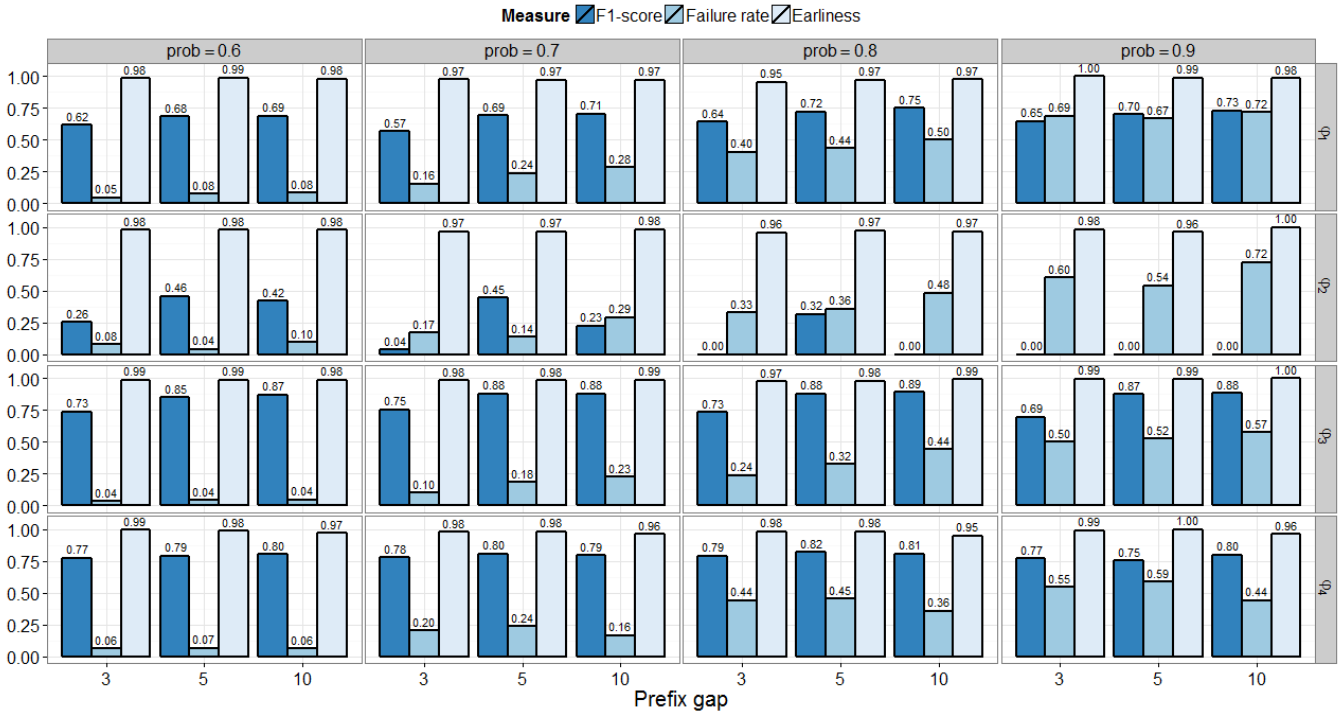

Fig. 10: DBSCAN clustering and random forest classification (*dbscan_rf*) - F1-score, failure-rate and earliness.

the on-the-fly approach. While for *mbased_dt* and *dbscan_rf* there is no statistically significant difference between the two approaches, in the case of *dbscan_dt* and *mbased_rf* the *PM Framework* instances outperform the on-the-fly approach with a statistically significant difference.

In Fig. 11, we show the results obtained with DBSCAN clustering and decision tree classification for different proportions of training and testing data, i.e., choosing as training logs 60%, 70%, and 80% of the traces of the original log, and as testing logs, 40%, 30% and 20% of the traces, respectively. The results show that, as expected, F1-score slightly improves with more training data. However, the

Fmwk instance	F1-score				failure-rate				earliness				g			
	avg on-the-fly	*avg fmwk*	*p-value*	*|Cohen-d|*	*avg on-the-fly*	*avg fmwk*	*p-value*	*|Cohen-d|*	*avg on-the-fly*	*avg fmwk*	*p-value*	*|Cohen-d|*	*avg on-the-fly*	*avg fmwk*	*p-value*	*|Cohen-d|*
mbased_dt	0.809	0.672	0.187	0.523	0.449	0.078	**<0.001**	3.1	0.824	0.889	0.024	0.595	0.382	0.516	0.187	0.58
dbscan_dt	0.809	0.702	**0.044**	0.604	0.449	0.251	**<0.001**	1.518	0.824	0.839	**<0.001**	2.383	0.382	0.505	**0.018**	0.766
mbased_rf	0.809	0.645	**0.004**	0.924	0.449	0.017	**<0.001**	3.115	0.824	0.839	0.562	0.149	0.382	0.504	**0.034**	0.676
dbscan_rf	0.809	0.641	0.052	0.699	0.449	0.343	0.083	0.631	0.824	0.977	**<0.001**	3.125	0.382	0.429	0.495	0.24

TABLE 1: Statistical comparison between each *PM Framework* instance and the on-the-fly approach

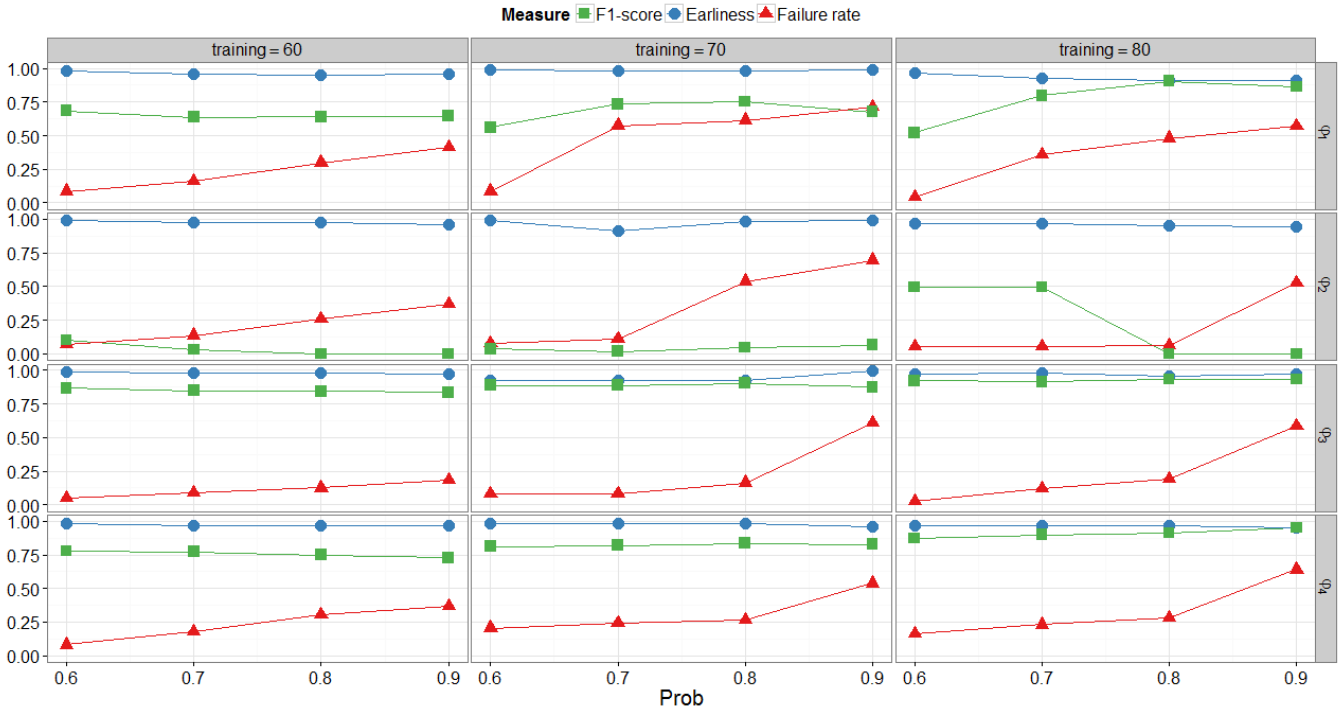

Fig. 11: DBSCAN clustering and decision tree classification - F1-score, failure-rate and earliness- for different proportions of training and testing data.

Formula	F1-score	failure-rate	earliness
φ_1	0.046	0.018	0.989
φ_2	0.481	0.004	0.98
φ_3	0	0	1
φ_4	0.107	0.013	0.971

TABLE 2: Only-control-flow baseline results

effectiveness of the approach is not significantly affected in case of little training data available.

In Fig. 12, we show the results obtained with model-based clustering and decision tree classification for different number of clusters. It is relevant to highlight that, without clustering (number of clusters equal to 1), the failure-rate is higher than in the other cases. This is probably due to the fact that the heterogeneity of the traces considered all together in only one cluster does not allow us to obtain predictions with a class probability sufficiently high. Moreover, by choosing the optimal number of clusters among those greater than or equal to 15, we obtain slightly better performances in terms of F1-score than for smaller numbers of clusters.

Furthermore, in order to provide a hint of the benefits of using an approach that exploits both control flow and data flow, we also compared the *PM Framework* instances against an only-control-flow baseline. Such a baseline was obtained by applying frequency-based encoding to individual activities and discriminative patterns computed according to the approach described in [12]. We show the results obtained with minimum class probability threshold equal to 0.7 and prefix gap equal to 5. In addition, we use random forest as classifier to get the predictions. We obtained similar results for the other configurations. The results show that the only-control-flow baseline is slightly more accurate than *mbased_rf* for φ_2 and has a comparable F1-score with respect to *dbscan_rf*. The F1-score is significantly lower in all the other cases. The baseline performs extremely well in terms of failure-rate and earliness. The considerations and the plots discussed provide an answer to **RQ1**.

To answer **RQ2**, we focus on two of the four *PM Framework* instances, the ones based on decision tree classification, since the computational time required by the corresponding random forest ones, do not present significant differences. Table 3 reports the *init time* (which is null for the on-the-fly approach), the *processing time* and the *average prediction time* obtained by applying the on-the-fly approach for each of the four investigated predicates with different minimum class probability thresholds. The results in the table show that processing a trace with such an approach is very expensive in terms of time required (the processing time ranges between $\sim$ 100 to $\sim$ 130 hours), which makes the approach difficult to be used in runtime scenarios. The average predic-

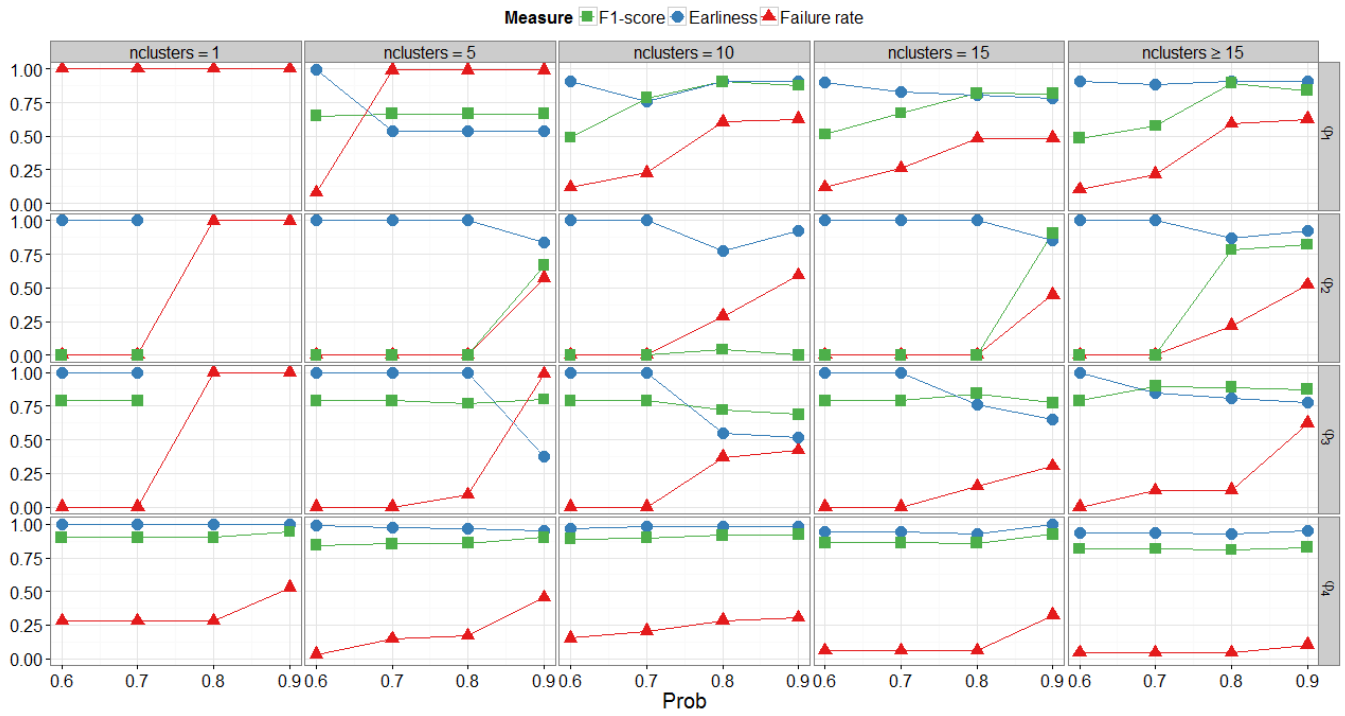

Fig. 12: Model-based clustering and decision tree classification - F1-score, failure-rate and earliness- for different numbers of clusters.

φ	prob=0.6			prob=0.7			prob=0.8			prob=0.9		
	init	processing	avg. pred.	init	processing	avg. pred.	init	processing	avg. pred.	init	processing	avg. pred.
φ_1		453946.354	165.84		456517.477	164.116		466850.858	171.528		474879.207	177.843
φ_2		368796.069	25.532		366606.233	25.36		368895.547	26.622		371021.759	27.94
φ_3		395761.404	45.773		396113.806	45.835		395810.652	45.797		384078.402	40.978
φ_4		234582.674	38.239		238284.95	40.224		247649.381	46.305		290813.698	25.854

TABLE 3: On-the-fly predictive monitoring - Processing and average prediction time (in seconds)

tion time, i.e., the time required at each evaluation point to provide a prediction, strongly depends on the investigated predicate, and ranges from about 25 seconds for φ_2 to about 3 minutes for φ_1.

Table 4 reports the computational time, and specifically the *init time*, *processing time* and *average prediction time*, required for providing a prediction for each of the four predicates with different minimum class probability thresholds and considering different prefix gaps for *mbased_dt*. By observing the table, it seems that no significant differences exist in terms of pre-processing time (*init-time*) for different predicates and minimum class probability thresholds. On the other hand, the pre-processing time depends on the considered trace prefix gap: the higher the prefix gap is, the less prefix traces are in the training set, and the less time is required to process them. In general, very small differences can be observed in terms of average prediction time (always ~ 10 milliseconds), while more significant differences occur in terms of the time required for processing the entire testing set, ranging from a minimum value of 1 second to a maximum value of about 67 seconds. These values are, however, still reasonable to be considered for providing predictions at runtime. The differences in terms of processing time depend on the predicate, on the prefix gap as well as on the minimum class probability threshold. More in general, the processing time is related to the failure-rate and the earliness: a higher failure-rate (and earliness) for a specific settings leads to a higher number of evaluation points that have to be processed (so that the processing time increases).

Finally, Table 5 reports the *init time*, the *processing time* and the *average prediction time* required to the *PM Framework* instance obtained combining the DBSCAN clustering and the decision tree classification techniques (*dbscan_dt*) for providing predictions. Also in this case, the initialization is constant for different minimum class probability thresholds and predicates, while it depends on the prefix gap (ranging from about 1.25 minutes to about 5 minutes). Small differences exist in terms of average prediction times (ranging from 8 to 31 milliseconds), while more significant ones hold in terms of processing time. The processing time, indeed, ranges between 1 to 30 minutes, when the failure-rate is very high (φ_3, $prob = 0.9$, $gap = 3$).

By comparing the performance of the proposed *PM Framework* instances with the on-the-fly approach, it clearly comes out that, with comparable results in terms of F1-score, failure-rate and earliness, the time required for processing a trace and providing a prediction is much lower with the clustering-based approach. The processing time is indeed about 300 times lower for *dbscan_dt* and $16,000$

φ	prob=0.6								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	2844.829	2.54	0.007	1660.859	5.824	0.01	713.558	3.429	0.009
φ_2	2844.444	1.366	0.006	1662.347	7.536	0.01	714.918	5.615	0.009
φ_3	2843.995	1.378	0.006	1661.454	11.959	0.01	713.166	2.697	0.008
φ_4	2844.803	10.95	0.006	1661.599	18.301	0.01	713.522	4.644	0.009
φ	prob=0.7								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	2842.966	8.138	0.007	1661.364	34.657	0.01	714.605	15.411	0.009
φ_2	2843.775	1.344	0.006	1660.577	9.807	0.011	713.023	7.059	0.009
φ_3	2843.078	4.861	0.007	1661.214	20.782	0.011	713.106	47.958	0.009
φ_4	2845.012	10.938	0.007	1661.828	39.13	0.01	713.915	13.48	0.009
φ	prob=0.8								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	2843.349	22.655	0.007	1662.209	15.328	0.01	714.142	10.584	0.009
φ_2	2843.716	16.625	0.006	1661.085	26.023	0.011	713.067	12.912	0.009
φ_3	2845.504	6.267	0.007	1660.463	8.854	0.011	713.077	4.618	0.009
φ_4	2844.309	13.942	0.007	1661.118	44.609	0.01	713.349	16.78	0.01
φ	prob=0.9								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	2844.202	25.645	0.007	1660.841	67.494	0.01	713.522	17.333	0.009
φ_2	2844.342	37.222	0.006	1661.668	15.94	0.011	714.25	6.388	0.009
φ_3	2844.619	17.646	0.007	1661.371	4.877	0.01	714.319	3.412	0.009
φ_4	2846.026	29.548	0.007	1660.566	9.377	0.01	713.438	7.065	0.009

TABLE 4: Model-based clustering and decision tree classification (*mbased_dt*)- Init, processing and average prediction time (in seconds)

φ	prob=0.6								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	296.707	77.535	0.03	174.26	43.927	0.023	94.956	30.457	0.01
φ_2	296.474	470.605	0.024	174.093	65.692	0.018	76.567	83.382	0.01
φ_3	299.63	61.476	0.02	172.858	32.376	0.019	82.567	20.599	0.011
φ_4	301.507	349.717	0.025	172.904	260.02	0.02	76.932	103.178	0.011
φ	prob=0.7								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	297.478	720.306	0.026	172.587	816.293	0.025	78.03	61.288	0.013
φ_2	299.725	473.14	0.024	173.853	66.565	0.0187	75.541	149.526	0.011
φ_3	298.888	74.922	0.021	173.233	479.454	0.018	76.189	107.485	0.012
φ_4	297.841	350.111	0.027	174.746	603.756	0.019	78.221	102.639	0.012
φ	prob=0.8								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	296.225	1313.315	0.031	175.039	937.048	0.025	75.784	308.192	0.015
φ_2	298.736	1567.935	0.026	173.587	70.065	0.019	79.096	591.503	0.014
φ_3	294.316	1110.0883	0.027	174.532	545.689	0.022	77.114	176.959	0.015
φ_4	301.411	784.481	0.028	175.386	642.789	0.019	75.418	128.4	0.011
φ	prob=0.9								
	gap=3			gap=5			gap=10		
	init	processing	avg. prediction	init	processing	avg. prediction	init	processing	avg. prediction
φ_1	296.36	1395.725	0.029	174.147	1047.62	0.025	75.997	355.925	0.015
φ_2	299.012	1577.679	0.028	173.56	1294.726	0.023	78.354	619.202	0.012
φ_3	297.687	2057.379	0.027	173.599	1018.924	0.019	76.989	420.525	0.008
φ_4	294.808	1300.652	0.027	176.473	1041.599	0.018	78.387	213.356	0.013

TABLE 5: DBSCAN clustering and decision tree (*dbscan_dt*) - Init, processing and average prediction time (in seconds)

times lower for *mbased_dt*. Even taking into account the init time required by the clustering-based predictive monitoring approaches, which is computed once per all traces, such a time is anyway lower than the processing time of the on-the-fly approach. Similarly, the average prediction time for the cluster-based approaches is even $5,000$ times smaller than the average prediction time of the on-the-fly approach: the average prediction time for *dbscan_dt* and *mbased_dt* is of the order of few milliseconds, while for the on-the-fly approach it can also reach two minutes. Compared to the baseline the *PM Framework* is considerably more efficient and can be used for runtime prediction under high throughput (**RQ2**).

3.5 Discussion

The observations and the analysis performed so far allow us to draw some conclusions and guidelines. The solutions provided by the different instances of the *PM Framework* offer the possibility to meet different types of needs, by opportunely setting the available configuration parameters.

For instance, in settings in which users are more interested in getting predictions at an early stage of a trace execution, low minimum class probability thresholds should be preferred. The same type of thresholds should also be preferred to have a prediction even if not always correct rather than a non-prediction. Indeed, low minimum class probability thresholds would allow users to get an almost null failure-rate with an acceptable F1-score in many cases.

For the choice of the clustering and the classification technique to use for instantiating the *PM Framework*, *mbased_rf* presents an average F1-score lower than the other instances. In general, the instances based on random forests seem to perform slightly worse than the ones based on decision trees. Furthermore, *mbased_dt* outperforms *dbscan_dt*. The instances based on DBSCAN present a very high earliness. In all the cases, the failure-rate increases with the minimum class probability threshold.

The choice of the configuration values also depends on the predicate under consideration. Predicates with a lower number of positive cases in the historical dataset lead to a lower F1-score. For instance, in the investigated settings, the F1-scores derived for φ_1 and φ_2 are generally worse than the ones derived for φ_3 and φ_4.

3.6 Threats to Validity

One of the main threats to the external validity of the evaluation is the application of the *PM Framework* to a single event log. The use of more logs would clearly allow for more general results. However, such a threat is mitigated by the fact that the considered log is a real-life log with real data chronologically ordered so as to simulate a realistic scenario. A second threat is the choice of the predicates used for the evaluation. Also in this case, we limited ourselves to 4 predicates. However, they are realistic business rules covering all the LTL constructs.

4 Related Work

The problem of predictive process monitoring has been addressed in several previous studies. For example, an approach for prediction of abnormal termination of business processes is presented in [13]. Here, a fault detection algorithm (local outlier factor) is used to estimate the probability of a fault occurring. Alarms are provided for early notification of probable abnormal terminations. In [14], [15], a technique is presented to predict "late show" events in transportation processes by applying standard statistical techniques to find correlations between "late show" events and external variables related to weather conditions or road traffic. In [16], the authors evaluate three types of predictive monitoring techniques and combinations of them on an industrial case study in the area of transport and logistics.

A key difference between the above predictive monitoring techniques and our technique is that they rely either on the control-flow or on the data perspective, whereas we take both perspectives into consideration. In addition, we provide a general framework for predictive process monitoring, which is flexible and can be instantiated with specific clustering and classification techniques to fit different scenarios.

Kang et al. [17] propose another predictive process monitoring technique, which starts by constructing a transition system from the event log. A state in this transition system represents the set of events that have occurred in the prefix of a case. Transitions are annotated with probabilities. The resulting transition system is used at runtime to predict Key Performance Indicators (KPIs) of a running case. Unlike our proposal, this approach does not take into account event payloads. Also, the prediction target is different. Our framework predicts binary outcomes (compliant vs. non-compliant) whereas in [17], the goal is to predict KPIs. Other proposals for predicting numerical KPIs of ongoing cases include [18], [19] and [20], while [21] and [22] deal with the problem of predicting the remaining time of a case.

Another body of related work focuses on estimating risks during the execution of a business process. In [23], the authors present a technique to support process participants in making risk-informed decisions, with the aim of reducing the materialization of certain predefined risks. Their technique constructs a decision tree for every decision point in the process, and uses this model to determine the probability that a given risk materializes for each branch of the decision point. The framework uses both control-flow features and event payloads. Specifically, each activity in the process is treated as a boolean feature in order to construct the decision tree: the feature corresponding to activity X is true iff activity X has occurred in the prefix of the ongoing case. The *PM Framework* differs from this proposal in that it applies clustering prior to constructing the classifier – in this way each classifier is only constructed for groups of similar traces. Also, the classifier is constructed using only data attributes, as the control-flow information is already taken into account during the clustering step.

The idea of clustering traces prior to building models from them has been considered in the field of specification mining in [24]. In this latter work, sets of traces produced by program executions are first clustered. For each cluster, a specification (specifically a probabilistic finite state machine) is constructed. Finally, the finite state machines produced for each cluster are merged together into a single specification.

5 Conclusion

We presented a framework for predictive monitoring of business processes that exploits data from past execution traces (both control flow and data attributes associated to events) to estimate the probability that a given predicate will be fulfilled upon completion of a running case. The framework achieves relatively low runtime overhead by constructing classification models offline – one per cluster of prefixes of historical traces. At runtime the prediction is made by matching the running case to a cluster, and applying the corresponding classification model to extract a prediction. Compared to a previous method that computes classification models at runtime [1], this leads to comparable results in terms of accuracy (measured by F1-score) and to a significant improvement in terms of response times. Experimental results show that the framework can achieve high levels of earliness (i.e., predictions are made early during the running case) and low failure-rates (i.e., low number of cases where predictions cannot be made with sufficiently high class probability).

In separate work, we have investigated the use of alternative early sequence classification techniques for predictive process monitoring, such as Hidden Markov Models (HMMs) [25]. We are also investigating the idea of assigning the uncompleted trace of a running case to multiple clusters (and thus using multiple classifiers) for making predictions instead of assigning each uncompleted trace to one single

cluster [26]. These and similar ideas can help to derive new variants of the proposed *PM Framework*.

As future work, we plan to investigate the application of techniques for extracting sequence patterns to achieve further accuracy and earliness improvements. Several types of sequence pattern extraction techniques have been developed in related fields, which might be applicable for predictive process monitoring, including predictive sequence patterns [12] (which emphasize earliness), discriminative (dyadic) patterns [27], [28] (which emphasize discriminative power) and iterative patterns [29] (which have been successfully applied to anomaly monitoring).

As the experiments presented in this paper are based on a single log, the results have low generalizability. Therefore, another avenue for future work is to conduct further experiments with logs with different characteristics and from different application domains.

Acknowledgments. This research is partly funded by the Estonian Research Council.

REFERENCES

[1] F. M. Maggi, C. Di Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *Proc. of CAiSE*, 2014, pp. 457–472.
[2] Z. Xing, J. Pei, and E. J. Keogh, "A brief survey on sequence classification," *SIGKDD Explorations*, vol. 12, no. 1, pp. 40–48, 2010.
[3] W. M. P. van der Aalst, M. Pesic, and M. Song, "Beyond process mining: From the past to present and future," in *Proc. of CAiSE*, 2010, pp. 38–52.
[4] M. Westergaard and F. M. Maggi, "Modelling and Verification of a Protocol for Operational Support using Coloured Petri Nets," in *Proc. of ATPN*, 2011, pp. 169–188.
[5] F. M. Maggi and M. Westergaard, "Designing software for operational decision support through coloured petri nets," *Enterprise Information Systems*, to appear.
[6] 3TU Data Center, "BPI Challenge 2011 Event Log," 2011, doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.
[7] C. Fraley and A. E. Raftery, "Enhanced model-based clustering, density estimation, and discriminant analysis software: MCLUST," *Journal of Classification*, vol. 20, pp. 263–286, September 2003.
[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of KDD*, 1996, pp. 226–231.
[9] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, 2010.
[10] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
[11] *Statistical Power Analysis for the Behavioral Sciences (2nd Edition)*, 2nd ed. Routledge, Jul. 1988.
[12] Z. Xing, J. Pei, G. Dong, and P. S. Yu, "Mining sequence classifiers for early prediction," in *Proc. of the SIAM International Conference on Data Mining (SDM)*. SIAM, 2008, pp. 644–655.
[13] B. Kang, D. Kim, and S.-H. Kang, "Real-time business process monitoring method for prediction of abnormal termination using KNNI-based LOF prediction," *Expert Syst. Appl.*, vol. 39, no. 5, pp. 6061–6068, 2012.
[14] A. Metzger, R. Franklin, and Y. Engel, "Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case," in *Proc. of SRII*, 2012.
[15] Z. Feldman, F. Fournier, R. Franklin, and A. Metzger, "Proactive event processing in action: a case study on the proactive management of transport processes," in *Proc. of DEBS*, 2013.
[16] A. Metzger, P. Leitner, D. Ivanovic, E. Schmieders, R. Franklin, M. Carro, S. Dustdar, and K. Pohl, "Comparing and combining predictive business process monitoring techniques." *IEEE T. Systems, Man, and Cybernetics: Systems*, vol. 45, no. 2, pp. 276–290, 2015.
[17] B. Kang, J. Jung, N. W. Cho, and S. Kang, "Real-time business process monitoring using formal concept analysis," *Industrial Management and Data Systems*, vol. 111, no. 5, pp. 652–674, 2011.
[18] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann, "Monitoring and analyzing influential factors of business process performance," in *Proc. of EDOC*. IEEE Computer Society, 2009, pp. 141–150.
[19] M. Castellanos, N. Salazar, F. Casati, U. Dayal, and M.-C. Shan, "Predictive business operations management," in *Proc. of DNIS*, 2005, pp. 1–14.
[20] F. Folino, M. Guarascio, and L. Pontieri, "Discovering context-aware models for predicting business process performances," in *Proc. of OTM*, 2012, vol. 7565, pp. 287–304.
[21] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Syst.*, vol. 36, no. 2, pp. 450–475, 2011.
[22] A. Rogge-Solti and M. Weske, "Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays," in *Proc. of ICSOC*, 2013, pp. 389–403.
[23] R. Conforti, M. de Leoni, M. La Rosa, and W. M. P. van der Aalst, "Supporting risk-informed decisions during business process execution," in *Proc. of CAiSE*, 2013, pp. 116–132.
[24] D. Lo and S. Khoo, "Smartic: towards building an accurate, robust and scalable specification miner," in *Proceedings of ACM SIGSOFT FSE*. ACM, 2006, pp. 265–275.
[25] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, and F. M. Maggi, "Complex symbolic sequence encodings for predictive monitoring of business processes," in *Proc. of BPM*. Springer, 2015, pp. 297–313.
[26] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, and C. Di Francescomarino, "Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring," in *Business Process Management Workshops - BPM 2015*, 2015, pp. 218–229.
[27] D. Lo, H. Cheng, J. Han, S. Khoo, and C. Sun, "Classification of software behaviors for failure detection: a discriminative pattern mining approach," in *Proceedings of ACM SIGKDD*. ACM, 2009, pp. 557–566.
[28] D. Lo, H. Cheng, and Lucia, "Mining closed discriminative dyadic sequential patterns," in *Proc. of EDBT*. Springer, 2011, pp. 21–32.
[29] N. A. Milea, S. Khoo, D. Lo, and C. Pop, "NORT: runtime anomaly-based monitoring of malicious behavior for windows," in *Prov. of Runtime Verification (RV)*. Springer, 2012, pp. 115–130.

bio/pictures/chiara.jpg

Chiara Di Francescomarino is a researcher at Fondazione Bruno Kessler (FBK) in the Shape and Evolve Living Knoweldge (SHELL) unit. She received her PhD in Information and Communication Technologies from the University of Trento, working on business process modeling and reverse engineering from execution logs. Her current research interests include business process modeling, collaborative modelling and the evaluation of tools and techniques for its support, as well as business process monitoring.

Marlon Dumas is Professor of Software Engineering at University of Tartu, Estonia. Prior to this appointment he was faculty member at Queensland University of Technology and visiting researcher at SAP Research, Australia. His research interests span across the fields of software engineering, information systems and business process management. He is co-author of the textbook Fundamentals of Business Process Management (Springer, 2013).

bio/pictures/fabrizio.png

Fabrizio M. Maggi received his Ph.D. degree in Computer Science in 2010 from the University of Bari and after a period at the Architecture of Information Systems (AIS) research group - Department of Mathematics and Computer Science - Eindhoven University of Technology. He is currently Senior Researcher at the Software Engineering Group at University of Tartu. His research interest span across the fields of business process management, data mining and service-oriented computing.

Irene Teinemaa is a PhD student at the University of Tartu working on predictive monitoring of business processes. Additionally, she is researcher at the Software Technology and Applications Competence Center (STACC), where she works on various industrial data mining projects. She received her Master degree in Software Engineering in 2014. Her current research interests include data mining, machine learning, and process mining.