Outcome-Oriented Predictive Process Monitoring: Review and Benchmark

IRENE TEINEMAA, University of Tartu
MARLON DUMAS, University of Tartu
MARCELLO LA ROSA, The University of Melbourne
FABRIZIO MARIA MAGGI, University of Tartu

Predictive business process monitoring refers to the act of making predictions about the future state of ongoing cases of a business process, based on their incomplete execution traces and logs of historical (completed) traces. Motivated by the increasingly pervasive availability of fine-grained event data about business process executions, the problem of predictive process monitoring has received substantial attention in the past years. In particular, a considerable number of methods have been put forward to address the problem of outcome-oriented predictive process monitoring, which refers to classifying each ongoing case of a process according to a given set of possible categorical outcomes – e.g., Will the customer complain or not? Will an order be delivered, canceled or withdrawn? Unfortunately, different authors have used different datasets, experimental settings, evaluation measures and baselines to assess their proposals, resulting in poor comparability and an unclear picture of the relative merits and applicability of different methods. To address this gap, this article presents a systematic review and taxonomy of outcome-oriented predictive process monitoring methods, and a comparative experimental evaluation of eleven representative methods using a benchmark covering 24 predictive process monitoring tasks based on nine real-life event logs.

CCS Concepts: •**Applied computing** → **Business process monitoring;**

Additional Key Words and Phrases: business process, predictive monitoring, sequence classification

1 INTRODUCTION

A business process is a collection of inter-related events, activities, and decision points that involve a number of actors and objects, which collectively lead to an outcome that is of value to a customer [11]. A typical example is an order-to-cash process: a process that starts when a purchase order is received and ends when the product/service is delivered and the payment is confirmed. An execution of a business process is called a *case*. In an order-to-cash process, each purchase order gives rise to a case.

Business process monitoring is the act of analyzing events produced by the executions of a business process at runtime, in order to understand its performance and its conformance with respect to a set of business goals [11]. Traditional process monitoring techniques provide dashboards and reports showing the recent performance of a business process in terms of key performance indicators such as mean execution time, resource utilization or error rate with respect to a given notion of error.

Predictive (business) process monitoring techniques go beyond traditional ones by making predictions about the future state of the executions of a business process (i.e. the cases). For example, a predictive monitoring technique may seek to predict the remaining execution time of each ongoing case of a process [33], the next activity that will be executed in each case [12], or the final outcome of a case, with respect to a possible set of business outcomes [25–27]. For instance, in an order-to-cash process , the possible outcomes of a case may be that the purchase order is closed satisfactorily (i.e., the customer accepted the products and paid) or unsatisfactorily (e.g., the

order was canceled or withdrawn). Another set of possible outcomes is that the products were delivered on time (with respect to a maximum acceptable delivery time), or delivered late.

Recent years have seen the emergence of a rich field of proposed methods for predictive process monitoring in general, and predictive monitoring of (categorical) case outcomes in particular – herein called *outcome-oriented predictive process monitoring*. Unfortunately, there is no unified approach to evaluate these methods. Indeed, different authors have used different datasets, experimental settings, evaluation measures and baselines.

This paper aims at filling this gap by (i) performing a systematic literature review of outcome-oriented predictive process monitoring methods; (ii) providing a taxonomy of existing methods; and (iii) performing a comparative experimental evaluation of eleven representative methods, using a benchmark of 24 predictive monitoring tasks based on nine real-life event logs.

The contribution of this study is a categorized collection of outcome-oriented predictive process monitoring methods and a benchmark designed to enable researchers to empirically compare new methods against existing ones in a unified setting. The benchmark is provided as an open-source framework that allows researchers to run the entire benchmark with minimal effort, and to configure and extend it with additional methods and datasets.

The rest of the paper is structured as follows. Section 2 introduces some basic concepts and definitions. Section 3 describes the search and selection of relevant studies. Section 4 surveys the selected studies and provides a taxonomy to classify them. Section 5 reports on benchmark evaluation of the selected studies while Section 6 discusses threats to validity. Finally, Section 7 summarizes the findings and outlines directions for future work.

2 BACKGROUND

The starting point of predictive process monitoring are *event records* representing the execution of activities in a business process. An event record has a number of attributes. Three of these attributes are present in every event record, namely the *event class* (a.k.a. *activity name*) specifying which activity the event refers to, the *timestamp* specifying when did the event occur, and the *case id* indicating which case of the process generated this event. For example, in an order-to-cash process, the purchase order identifier is the case id, since every event occurring in an execution of this process is associated with a purchase order. In other words, every event represents the occurrence of an activity at a particular point in time and in the context of a given case. An event record may carry additional attributes in its payload. These are called *event-specific attributes* (or *event attributes* for short). For example, in an order-to-cash process, the amount of the invoice may be recorded as an attribute of an event referring to activity "Create invoice". Other attributes, namely *case attributes*, belong to the case and are hence shared by all events generated by the same case. For example in an order-to-cash process, the customer identifier is likely to be a case attribute. If so, this attribute will appear in every event of every case of the order-to-cash process, and it has the same value for all events generated by a given case. In other words, the value of a case attribute is static, i.e., it does not change throughout the lifetime of a case, as opposed to attributes in the event payload, which are dynamic as they change from an event to the other.

Formally, an event record is defined as follows:

Definition 2.1 (Event). An *event* is a tuple $(a, c, t, (d_1, v_1), \ldots, (d_m, v_m))$ where a is the activity name, c is the case id, t is the timestamp and $(d_1, v_1) \ldots, (d_m, v_m)$ (where $m \geq 0$) are the event or case attributes and their values.

Herein, we use the term *event* as a shorthand for *event record*. The universe of all events is hereby denoted by $\mathcal{E}$.

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

σ_1 = [(consultation, 1, 10:30AM, (age, 33), (gender, female), (amountPaid, 10), (department, radiotherapy)), …,
(ultrasound, 1, 10:55AM, (age, 33), (gender, female), (amountPaid, 15), (department, NursingWard))]
σ_2 = [(order blood, 2, 12:30PM, (age, 56), (gender, male), (department, GeneralLab), …,
(payment, 2, 2:30PM, (age, 56), (gender, male), (amountPaid, 100), (deparment, FinancialDept))]

Fig. 1. Extract of an event log.

Definition 2.2 (Trace). A *trace* is a non-empty sequence $\sigma = [e_1, \ldots, e_n]$ of events such that $\forall i \in [1..n], e_i \in \mathcal{E}$ and $\forall i, j \in [1..n]\ e_i.c = e_j.c$. In other words, all events in the trace refer to the same case.

The universe of all possible traces is denoted by $\mathcal{S}$.

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

As a running example, we consider a simple log of a patient treatment process containing two cases (cf. Figure 1). The activity name of the first event in trace σ_1 is *consultation*, it refers to case 1 and occurred at *10:30AM*. The additional event attributes show that the cost of the procedure was 10 and the activity was performed in the *radiotherapy* department. These two are event attributes. Note that not all events carry every possible event attribute. For example, the first event of trace σ_2 does not have the attribute *amountPaid*. In other words, the set of event attributes can differ from one event to another even within the same trace. The events in each trace also carry two case attributes: the age of the patient and the gender. The latter attributes have the same value for all events of a trace.

An event or a case attribute can be of numeric, categorical, or of textual data type. Each data type requires different preprocessing to be usable by the classifier. With respect to the running example, possible event and case attributes and their type are presented in Table 1.

Table 1. Data attributes in the event log

Type	Example
Case (static)	
categorical	patient's gender
numeric	patient's age
textual	description of the application
Event (dynamic)	
categorical	activity, resource
numeric	amount paid
textual	patient's medical history

In predictive process monitoring, we aim at making predictions for traces of incomplete cases, rather than for traces of completed cases. Therefore, we make use of a function that returns the first l events of a trace of a (completed) case.

Definition 2.3 (Prefix function). Given a trace $\sigma = [e_1, \ldots, e_n]$ and a positive integer $l \leq n$, $prefix(\sigma, l) = [e_1, \ldots, e_l]$.

Given a trace, outcome-oriented predictive process monitoring aims at predicting its *class label* (expressing its outcome according to some business goal), given a set of completed cases with their known class labels.

Definition 2.4 (Labeling function). A labeling function $y : \mathcal{S} \rightarrow \mathcal{Y}$ is a function that maps a trace σ to its class label $y(\sigma) \in \mathcal{Y}$ with $\mathcal{Y}$ being the domain of the class labels. For outcome predictions, $\mathcal{Y}$ is a finite set of categorical outcomes. For example, for a binary outcome $\mathcal{Y} = \{0, 1\}$.

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

Definition 2.5 (Sequence/trace encoder). A sequence (or trace) encoder $f : \mathcal{S} \rightarrow \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ is a function that takes a (partial) trace σ and transforms it into a feature vector in the p-dimensional vector space $\mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ with $\mathcal{X}_j \subseteq \mathbb{R}, 1 \leq j \leq p$ being the domain of the j-th feature.

The features extracted from a trace may encode information on activities performed during the execution of a trace and their order (herein called *control-flow* features), and features that correspond to event/case attributes (herein referred to as *data payload* features).

A classifier is a function that assigns a class label to a feature vector.

Definition 2.6 (Classifier). A classifier $cls : \mathcal{X}_1 \times \cdots \times \mathcal{X}_p \rightarrow \mathcal{Y}$ is a function that takes an encoded p-dimensional sequence and estimates its class label.

The construction of a classifier (a.k.a. classifier *training*) for outcome-oriented predictive process monitoring is achieved by applying a classification algorithm over a set of prefixes of an event log. Accordingly, given a log L, we define its *prefix log* log L^* to be the event log that contains all prefixes of L, i.e., $L^* = \{prefix(\sigma, l) : \sigma \in L, 1 \leq l \leq |\sigma|\}$. Since the main aim of predictive process monitoring is to make predictions as early as possible (rather than when a case is about to complete), we often focus on the subset of the prefix log containing traces of up to a given length. Accordingly, we define the *length-filtered prefix log* L^*_k to be the subset of L^* containing only prefixes of size less than or equal to k.

With respect to the broader literature on machine learning, we note that predictive process monitoring corresponds to a problem of *early sequence classification*. In other words, given a set of labeled sequences, the goal is to build a model that for a sequence prefix predicts the label this prefix will get when completed. A survey on sequence classification presented in [46] provides an overview of techniques in this field. This latter survey noted that, while there is substantial literature on the problem of sequence classification for simple symbolic sequences (e.g., sequences of events without payloads), there is a lack of proposals addressing the problem for complex symbolic sequences (i.e., sequences of events with payloads). The problem of outcome-oriented predictive process monitoring can be seen as an early classification over complex sequences where each element has a timestamp, a discrete attribute referring to an activity, and a payload made of a heterogeneous set of other attributes.

3 SEARCH METHODOLOGY

In order to retrieve and select studies for our survey and benchmark, we conducted a *Systematic Literature Review* (SLR) according to the approach described in [20]. We started by specifying the research questions. Next, guided by these goals, we developed relevant search strings for querying a database of academic papers. We applied inclusion and exclusion criteria to the retrieved studies in order to filter out irrelevant ones, and last, we divided all relevant studies into primary and subsumed ones based on their contribution.

3.1 Research questions

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

In line with the selected scope, the survey focuses specifically on the following research question:

RQ0 Given an event log of completed business process execution cases and the final outcome (class) of each case, how to train a model that can accurately and efficiently predict the outcome of an incomplete (partial) trace, based on the given prefix only?

We then decomposed this overarching question into the following subquestions:

RQ1 What methods exist for predictive outcome-oriented monitoring of business processes?
RQ2 How to categorize these methods in a taxonomy?
RQ3 What is the relative performance of these methods?

In the following subsections, we describe our approach to identifying existing methods for predictive outcome-oriented process monitoring (RQ1). Subsequent sections address the other two research questions.

3.2 Study retrieval

First, we came up with relevant keywords according to the research question of predictive outcome-oriented process monitoring (RQ1) and our knowledge of the subject. We considered the following keywords relevant:

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

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

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

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

3.3 Study selection

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

The assessment of each study was performed independently by two authors of this paper, and the results were compared to resolve inconsistencies with the mediation of a third author.

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

IN1 The study is concerned with predictions in the context of business processes (this criterion was assessed by reading title and abstract).
IN2 The study is cited at least five times.

The application of these inclusion criteria to the original set of retrieved papers resulted in eight relevant studies. We proceeded with one-hop-snowballing, i.e., we retrieved the papers that are related to (cite or are cited by) these eight studies and applied the same inclusion criteria. This procedure resulted in 545 papers, of which we retained 70 unique papers after applying the inclusion criteria.[1]

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

EX1 The study does not actually propose a predictive process monitoring *method*.
EX2 The study does not concern *outcome*-oriented prediction.
EX3 The technique proposed in the study is tailored to a specific labeling function.
EX4 The study does not take an *event log* as input.

The EX1 criterion excludes overview papers, as well as studies that, after a more thorough examination, turned out to be focusing on some research question other than predictive process monitoring. EX2 excludes studies where the prediction target is something other than the final outcome. Common examples of other prediction targets that are considered irrelevant to this study are remaining time and next activity prediction. Using EX3, we excluded studies that are not directly about classification, i.e., that do not follow a black-box prediction of the case class. For example, studies that predict deadline violations by means of setting a threshold on the predicted remaining time, rather than by directly classifying the case as likely to violate the deadline or not. The reason for excluding such studies is that, in essence, they predict a numeric value, and are thus not applicable for predicting an arbitrarily defined case outcome. EX4 concerns studies that propose methods that do not utilize at least the following essential parts of an event log: the case identifier, the timestamp and the event classes. For instance, we excluded methods that take as input numerical time series without considering the heterogeneity in the control flow (event classes). In particular, this is the case in manufacturing processes which are of linear nature (a process chain). The reason for excluding such studies is that the challenges when predicting for a set of cases of heterogenous length are different from those when predicting for linear processes. While methods designed for heterogenous processes are usually applicable to those of linear nature, it is not so vice versa. Moreover, the linear nature of a process makes it possible to apply other, more standard methods that may achieve better performance.

The application of the exclusion criteria resulted in 14 relevant studies out of the 70 studies selected in the previous step.

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

3.4 Primary and subsumed studies

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

Specifically, a study is considered subsumed if:

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

This procedure resulted in seven primary and seven subsumed studies, listed in Table 2. In the next section we present the primary studies in detail, and classify them using a taxonomy.

Table 2. Primary and subsumed studies

Primary study	Subsumed studies
de Leoni et al. [8]	de Leoni et al. [7]
Maggi et al. [25]	
Lakshmanan et al. [21]	Conforti et al. [5, 6]
di Francescomarino et al. [10]	
Leontjeva et al. [22]	van der Spoel et al. [43]
Verenich et al. [45]	
Castellanos et al. [4]	Schwegmann et al. [36, 37], Ghattas et al.[18]

4 ANALYSIS AND TAXONOMY

In this section we present a taxonomy to classify the seven primary studies that we selected through our SLR. Effectively, with this section we aim at answering RQ1 (What methods exist?) and RQ2 (How to categorize them?) – cf. Section 3.1. The taxonomy is framed upon a general workflow for predictive process monitoring, which we derived by studying all the methods surveyed. This workflow is divided into two phases: offline, to train a prediction model based on historical cases, and online, to make predictions on running process cases. The *offline* phase, shown in Fig. 2, consists of four steps. First, given an event log, case prefixes are extracted and filtered (e.g., to retain only prefixes up to a certain length). Next, the identified prefixes are divided into buckets (e.g., based on process states or similarities among prefixes) and features are encoded from these buckets for classification. Finally, each bucket of encoded prefixes is used to train a classifier.

The *online* phase, shown in Fig. 3, concerns the actual prediction for a running trace, by reusing the elements (buckets, classifiers) built in the offline phase. Specifically, given a running trace and a set of buckets of historical prefixes, the correct bucket is first determined. Next, this information is used to encode the features of the running trace for classification. In the last step, a prediction is extracted from the encoded trace using the correct classifier for the determined bucket.

We note that there is an exception among the surveyed methods that does not perfectly fit the presented workflow. Namely, the KNN approach proposed by Maggi et al. [25] omits the

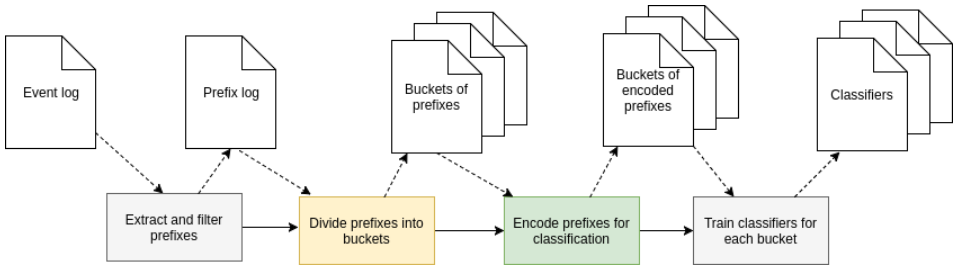


Fig. 2. predictive process monitoring workflow (offline phase)

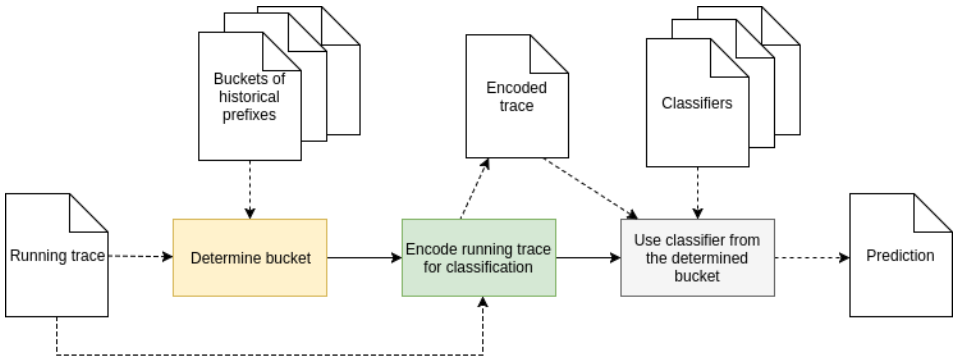


Fig. 3. predictive process monitoring workflow (online phase)

offline phase. Instead, in this approach the bucket (a set of similar traces from the training set) is determined and a classifier is trained during the online phase, separately for each running case.

Table 3 lists the seven primary studies identified in our SLR, and shows their characteristics according to the four steps of the offline phase (prefix selection and filtering, trace bucketing, sequence encoding and classification algorithm). In the rest of this section we survey these studies based on these characteristics, and use this information to build a taxonomy that allows us to classify the studies.

Table 3. Classification of the seven primary studies according to the four steps of the offline phase.

	Prefix extraction and		Sequence encoding		
Primary study	filtering	Trace bucketing	Control flow	Data	Classification algorithm
de Leoni et al. [8]	all	Single	agg, last state	agg, last state	DT
Maggi et al. [25]	all	KNN	agg	last state	DT
Lakshmanan et al. [21]	all	State	last state	last state	DT
di Francescomarino et al. [10]	prefix length 1-21, with gap 3, 5, or 10	Cluster	agg	last state	DT, RF
Leontjeva et al. [22]	prefix length 2-20	Prefix length	index	index	DT, RF, GBM, SVM
			index	last state	RF
			agg	-	RF
Verenich et al. [45]	prefix length 2-20	Prefix length + cluster	index	index	RF
Castellanos et al. [4]	all	Domain knowledge	*unknown*	*unknown*	DT

4.1 Prefix extraction and filtering

After analyzing the identified studies, we found that all of them take as input a prefix log (as defined in Section 2) to train a classifier. This choice is natural given that at runtime, we need to make predictions for partial traces rather than completed ones. Using a prefix log for training ensures that our training data is comparable to the testing data. For example, for a complete trace consisting of a total of 5 events, we could consider up to 4 prefixes: the partial trace after executing the first event, the partial trace after executing the first and the second event, and so on.

Using all possible prefixes raises multiple problems. Firstly, the large number of prefixes as compared to the number of traces considerably slows down the training of the prediction models. Secondly, if the length of the original cases is very heterogenous, the longer traces produce much more prefixes than shorter ones and, therefore, the prediction model is biased towards the longer cases. Accordingly, it is common to consider prefixes up to a certain number of events only. For example, Di Francescomarino et al. [10] limit the maximum prefix length to 21, while Leontjeva et al. [22] use prefixes of up to 20 events only. In other words, in their training phase, these approaches take as input the length-filtered prefix log L_k for $k = 21$ and $k = 20$.

Di Francescomarino et al. [10] propose a second approach to filter the prefix log using so-called *gaps*. Namely, instead of retaining all prefixes of up to a certain length, they retain prefixes whose length is equal to a base number (e.g., 1) plus a multiple of a gap (e.g., 1, 6, 11, 16, 21 for a gap of 5) . This approach helps to keep the prefix log sufficiently small for applications where efficiency of the calculations is a major concern.

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

4.2 Trace bucketing

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

4.2.1 Single bucket. All prefix traces are considered to be in the same bucket. A single classifier is trained on the whole prefix log and applied directly to the running cases. The single bucket approach has been used in the work by de Leoni et al. [8].

4.2.2 KNN. In this bucketing approach, the offline training phase is skipped and the buckets are determined at run-time. Namely, for each running prefix trace, its k nearest neighbors are selected from the historical prefix traces and a classifier is trained (at run-time) based on these k neighbors. This means that the number of buckets (and classifiers) is not fixed, but grows with each executed event at run-time.

The KNN method for predictive process monitoring was proposed by Maggi et al. [25]. Namely, they calculate the similarities between prefix traces using string-edit distance on the control flow. All instances that exceed a specified similarity threshold are considered as neighbors of the running trace. If the number of neighbors found is less than 30, the top 30 similar neighbors are selected regardless of the similarity threshold.

4.2.3 State. In state-based approaches, a process model is derived from the event log. Then, relevant *states* (or decision points) are determined from the process model and one classifier is

trained for each such state. At run-time, the current state of the running case is determined, and the respective classifier is used to make a prediction for the running case.

Given an event log, Lakshmanan et al. [21] construct a so-called *activity graph* where there is one node per possible activity (event class) in the log, and there is a directed edge from node a_i to a_j iff a_j has occurred immediately after a_i in at least one trace. This type of graph is also known as the *Directly-Follows Graph* (DFG) of an event log [41]. We observe that the DFG is the state-transition system obtained by mapping each trace prefix in the log to a state corresponding to the last activity appearing in the trace prefix (and hence the state of a running case is fully determined by its last activity). Alternative methods for constructing state abstractions are identified in [42] (e.g., set-based, multiset-based and sequence-based state abstractions), but these have not been used for predictive process monitoring, and they are likely not to be suitable since they generate a very large number of states, which would lead to very large number of buckets. Most of these buckets would be too small to train a separate classifier.

In Lakshmanan et al. [21], the edges in the DFG are annotated with transition probabilities. The transition probability from node a_i to a_j captures how often after performing activity a_i, a_j is performed next. We observe that this DFG annotated with transition probabilities is a first order Markov chain. For our purposes however, the transition probabilities are not necessary, as we aim to make a prediction for any running case regardless of its frequency. Therefore, in the rest of this paper, we will use the DFG without transition probabilities.

Lakshmanan et al. [21] build one classifier per *decision point* — i.e., per state in the model where the execution splits into multiple alternative branches. Given that in our problem setting, we need to be able to make a prediction for a running trace after each event, a natural extension to their approach is to build one classifier for every state in the process model.

4.2.4 Clustering. The cluster-based bucketer relaxes the requirement of a direct transition between the buckets of two subsequent prefixes. Conversely, the buckets (clusters) are determined by applying a clustering algorithm on the encoded prefix traces. This results in a number of clusters that do not exhibit any transitional structure. In other words, the buckets of $prefix(\sigma, l)$ and $prefix(\sigma, l+1)$ are determined independently from each other. Both of these prefixes might be assigned to the same cluster or different ones.

One classifier is trained per each resulting cluster, considering only the historical prefix traces that fall into that particular cluster. At run-time, the cluster of the running case is determined based on its similarity to each of the existing clusters and the respective classifier is applied.

A clustering-based approach is proposed by di Francescomarino et al. [10]. They experiment with two clustering methods, DBScan (with string-edit distance) and model-based clustering (with Euclidean distance on the frequencies of performed activities), while neither achieves constantly superior performance over the other. Another clustering-based method is introduced by Verenich et al. [45]. In their approach, the prefixes are encoded using index-based encoding (see 4.3.4) using both control flow and data payload, and then either hierarchical agglomerative clustering (HAC) or k-medoids clustering is applied. According to their results, k-medoids clustering consistently outperforms HAC.

4.2.5 Prefix length. In this approach, each bucket contains only the partial traces of a specific length. For example, one bucket contains traces where only the first event has been executed, another bucket contains those where first and second event have been executed, and so on. One classifier is built for each possible prefix length. The prefix length based bucketing was proposed by Leontjeva et al. [22]. Also, Verenich et al. [45] bucket the prefixes according to prefix length before applying a clustering method.

4.2.6 Domain knowledge. While the bucketing methods described so far can detect buckets through an automatic procedure, it is possible to define a bucketing function that is based on manually constructed rules. In such an approach, the input from a domain expert is needed. The resulting buckets can, for instance, refer to *context categories* [18] or *execution stages* [4, 36].

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

4.3 Sequence encoding

In order to train a classifier, all prefix traces in the same bucket need to be represented as fixed length feature vectors. The main challenge here comes from the fact that with each executed event, additional information about the case becomes available, while each trace in a bucket (independent of the number of executed events) should still be represented with the same number of features. This can be achieved by applying a trace abstraction technique [42], for example, considering only the last m events of a trace. However, choosing an appropriate abstraction is a difficult task, where one needs to balance the trade-off between the generality[2] and loss of information. After a trace abstraction is chosen, a set of feature extraction functions may be applied to each event data attribute of the abstracted trace. Therefore, a *sequence encoding* method can be thought of as a combination of a trace abstraction technique and a set of feature extraction functions for each data attribute.

In the following subsections we describe the sequence encoding methods that have been used in the existing predictive process monitoring approaches. As described in Section 2, a trace can contain any number of static case attributes and dynamic event attributes. Both the case and the event attributes can be of numeric, categorical, or textual type. As none of the compared methods deal with textual data, hereinafter we will focus on numeric and categorical attributes only.

4.3.1 Static. The encoding of case attributes is rather straightforward. As they remain the same throughout the whole case, they can simply be added to the feature vector "as is" without any loss of information. In order to represent all the information as a numeric vector, we assume the "as is" representation of a categorical attribute to be *one hot encoding*. This means that each value of a categorical attribute is transformed into a bitvector $(v_1, \cdots, v_n)$, where m is the number of possible levels of that attribute, $v_i = 1$ if the given value is equal to the ith level of the attribute, and $v_i = 0$ otherwise.

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

Using the last state abstraction, only one value (the last snapshot) of each data attribute is available. Therefore, no meaningful aggregation functions can be applied. Similarly to the static

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

encoding, the numeric attributes are added to the feature vector "as is", while one hot encoding is applied to each categorical attribute.

Last state encoding is the most common encoding technique, having been used in the KNN approach [25], state-based bucketing [21], as well as the clustering-based bucketing approach by Di Francescomarino et al. [10]. Furthermore, De Leoni et al. [8] mention the possibility of using the last and the previous (the last two) states.

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

In particular, the frequencies of performed activities (control flow) have been used in several existing works [10, 22]. Alternatively, boolean values have been used to express whether an activity has occurred in the trace. However, the frequency-based encoding has been shown to be superior to the boolean encoding [22]. For numerical attributes, De Leoni et al. [8] proposed using general statistics, such as the average, maximum, minimum, and sum.

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

Index-based encoding was proposed by Leontjeva et al. [22]. Additionally, in their work they combined the index-based encoding with HMM log-likelihood ratios. However, we decided not to experiment with HMMs in this study for mainly two reasons. Firstly, the HMMs did not consistently improve the basic index-based encoding in [22]. Secondly, rather than being an essential part of index-based encoding, HMMs can be thought of as an aggregation function that can be applied to each event attribute, similarly to taking frequencies or numeric averages. Therefore, HMMs are not exclusive to index-based encoding, but could also be used in conjunction with the aggregation encoding. Index-based encoding is also used in the approach of Verenich et al. [45].

Summary. An overview of the encoding methods can be seen in Table 4. Note that the static encoding extracts different type of data from the trace (case attributes) than the other three methods (event attributes). Therefore, for obtaining a complete representation for a trace, it is reasonable to concatenate the static encoding with one of the other three encodings. In our experiments, the static encoding is included in every method, e.g., the "last state" method in the experiments refers to the static encoding for case attributes concatenated with the last state encoding for event attributes.

4.4 Classification algorithm

The existing predictive process monitoring methods have experimented with different classification algorithms. The most popular choice has been decision tree (DT), which has obvious benefits in terms of the interpretability of the results. Another popular method has been random forest [3] (RF), which usually achieves better prediction accuracy than a single decision tree, but is harder to interpret. Additionally, Leontjeva et al. [22] experimented with support vector machines (SVM) and generalized boosted regression models (GBM), but found that their performance is inferior to RF.

Table 4. Encoding methods

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

Recently, gradient boosted trees [15] in conjunction with existing predictive process monitoring techniques have shown promising results, often outperforming RF [34, 38].

4.5 Discussion

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

The considered methods also differ in terms of the event log attributes that are used for making predictions. However, it has been shown [22] that including more information (i.e., combining control flow and data payload) can drastically increase the predictive power of the models. In order to provide a fair comparison of the different methods, it is preferable to provide the same set of attributes as input to all methods, and preferably the largest possible set of attributes. Accordingly, in the comparative evaluation below, we will encode traces using all the available case and event attributes (covering both control flow and data payload).

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

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

Figure 4 provides a taxonomy of the relevant methods based on these two perspectives. Note that although the taxonomy is based on 7 primary studies, it contains 11 different approaches. The reason for this is that while the primary approaches tend to mix different encoding schemes, for example, use aggregation encoding for control flow and last state encoding for data payload (see Table 3), the taxonomy is constructed in a modular way, so that each encoding method constitutes a separate approach. In order to provide a fair comparison of different encoding schemes, we have decided to evaluate each encoding separately, while the same encoding is applied to both control flow and data payload. Still, the different encodings (that are valid for a given bucketing method) can be easily combined, if necessary. Similarly, the taxonomy does not contain combinations of

several bucketing methods. An example of such "double bucket" approaches is the method by Verenich et al. [45], where the prefixes are first divided into buckets based on prefix length and, in turn, clustering is applied in each bucket. We believe that comparing the performance of each bucketing method separately (rather than as a combination) provides more insights about the benefits of each method. Furthermore, the double bucket approaches divide the prefixes into many small buckets, which often leads to situations where a classifier receives too little training instances to learn meaningful patterns.

We note that the taxonomy generalizes the state-of-the-art, in the sense that even if a valid pair of bucketing and encoding method has not been used in any existing approach in the literature, it is included in the taxonomy (e.g., the state-based bucketing approach with aggregation encoding). We also note that while the taxonomy covers the techniques proposed in the literature, all these techniques rely on applying a propositional classifier on an explicit vectorial representation of the traces. One could envisage alternative approaches that do not require an explicit feature vector as input. For instance, kernel-based SVMs have been used in the related setting of predicting the cycle time of a case [44]. Furthermore, one could envisage the use of data mining techniques to extract additional features from the traces (e.g., latent variables or frequent patterns). Although the taxonomy does not cover this aspect explicitly, applying such techniques is consistent with the taxonomy, since the derived features can be used in combination with any of the sequence encoding and bucketing approaches presented here. While most of the existing works on outcome-oriented predictive monitoring use the event/trace attributes "as-is" without an additional mining step, Leontjeva et al. used Hidden Markov Models for extracting additional features in combination with index-based encoding [22]. Similarly, Teinemaa et al. applied different natural language processing techniques to extract features from textual data [40]. Further on, although not yet applied to outcome-oriented predictive process monitoring tasks, different pattern mining techniques could be applied to extract useful patterns from the sequences, occurrences of which could then be used as features in the feature vectors of the traces. Such techniques have been used in the domain of early time series/sequence classification [16, 17, 19, 23, 47] and for predicting numerical measures (e.g., remaining time) for business processes [14].

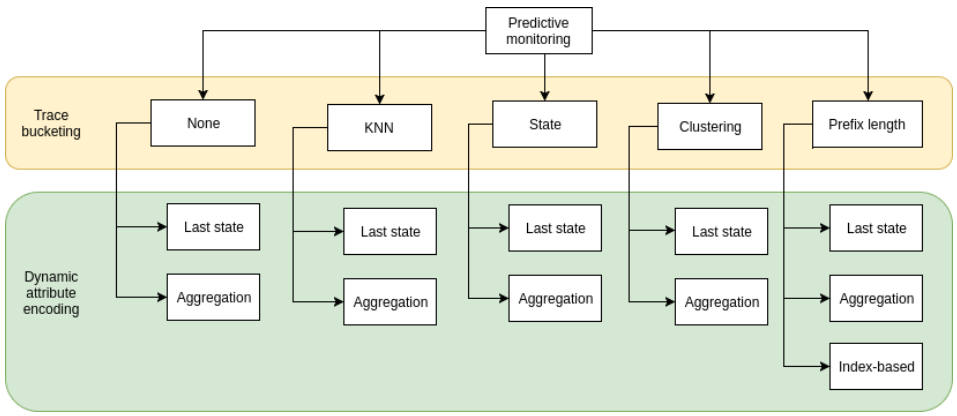


Fig. 4. Taxonomy of methods for predictive monitoring of business process outcome.

5 BENCHMARK

After conducting our survey, we proceeded with benchmarking the 11 approaches (shown in Figure 4) using different evaluation criteria (prediction accuracy, earliness and computation time), to address RQ3 (What is the relative performance of these methods?) – cf. Section 3.1.

To perform our benchmark, we implemented an open-source, tunable and extensible predictive process monitoring framework in Python.[3] All experiments were run using Python 3.6 and the scikit-learn library [31] on a single core of a Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz with 64GB of RAM.

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

5.1 Datasets

The benchmark is based on nine real-life event logs, out of which eight are publicly available and one is a private dataset. The public logs are accessible from the 4TU Centre for Research Data.[4] The private log *Insurance* originates from a claims handling process at an Australian insurance company. The criterion for selecting the public event logs for the evaluation was that the log must contain both case attributes (static) and event attributes (dynamic). Based on this, we discarded the logs from years 2013-2014. We also discarded the BPIC 2016 dataset because it is a click-dataset of a Web service, rather than an event log of a business process.

In case of some logs, we applied several labeling functions y. In other words, the outcome of a case is defined in several ways depending on the goals and needs of the process owner. Each such notion of the outcome constitutes a separate predictive process monitoring task with slightly different input datasets. In total, we formulated a total of 24 different outcome prediction tasks based on the nine original event logs. In the following paragraphs we describe the original logs, the applied labeling functions, and the resulting predictive monitoring tasks in more detail.

BPIC2011. This event log contains cases from the Gynaecology department of a Dutch Academic Hospital. Each case assembles the medical history of a given patient, where the applied procedures and treatments are recorded as activities. Similarly to previous work [10, 22], we use four different labeling functions based on LTL rules [32]. Specifically, we define the class label for a case σ according to whether an LTL rule φ is violated or satisfied by each trace σ.

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

Table 5 introduces the semantics of the LTL operators.

Table 5. LTL Operators Semantics

operator	*semantics*
Xφ	*φ has to hold in the next position of a path.*
Gφ	*φ has to hold always in the subsequent positions of a path.*
Fφ	*φ has to hold eventually (somewhere) in the subsequent positions of a path.*
φ Uψ	*φ has to hold in a path at least until ψ holds. ψ must hold in the current or in a future position.*

The four LTL rules used to formulate the four prediction tasks on the BPIC 2011 log are as follows:

[3]The code is available at https://github.com/irhete/predictive-monitoring-benchmark
[4]https://data.4tu.nl/repository/collection:event_logs_real

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

For example, the φ for *bpic2011_1* expresses the rule that at least one of the activities “tumor marker CA-19.9” or “ca-125 using meia” must happen eventually during a case. Evidently, the class label of a case becomes known and irreversible when one of these two events has been executed. In order to avoid bias introduced by this phenomenon during the evaluation phase, all the cases are cut exactly before either of these events happens. Similarly, the cases are cut before the occurrence of “histological examination-biopsies nno” in the *bpic2011_3* dataset and before “histological examination-big resectiep” in *bpic2011_4*. However, no cutting is performed in the *bpic2011_2* dataset, because the φ states that a “CEA-tumor marker using meia” event must *always* be followed by a “squamous cell carcinoma using eia” event sometime in the future. Therefore, even if one occurrence of “CEA-tumor marker using meia” has successfully been followed by a “squamous cell carcinoma using eia” (φ is satisfied), another occurrence of “CEA-tumor marker using meia” will cause the φ to be violated again and, thus, the class label is not irreversibly known until the case completes.

BPIC2015. This dataset assembles event logs from 5 Dutch municipalities, pertaining to the building permit application process. We treat the datasets from each municipality as separate event logs and apply a single labeling function to each one. Similarly to BPIC 2011, the labeling function is based on the satisfaction/violation of an LTL rule φ. The prediction tasks for each of the 5 municipalities are denoted as *bpic2015_i*, where $i = 1 \ldots 5$ indicates the number of the municipality. The LTL rule used in the labeling functions is as follows:

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

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

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

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

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

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

We created three different labelings for this log:

- *sepsis_1*: the patient returns to the emergency room within 28 days from the discharge,
- *sepsis_2*: the patient is (eventually) admitted to intensive care,

- *sepsis_3*: the patient is discharged from the hospital on the basis of something other than *Release A* (i.e., the most common release type).

BPIC2012. This dataset, originally published in relation to the Business Process Intelligence Challenge (BPIC) in 2012, contains the execution history of a loan application process in a Dutch financial institution. Each case in this log records the events related to a particular loan application. For classification purposes, we defined some labelings based on the final outcome of a case, i.e., whether the application is accepted, rejected, or canceled. Intuitively, this could be thought of as a multi-class classification problem. However, to remain consistent with previous work on outcome-oriented predictive process monitoring, we approach it as three separate binary classification tasks. In the experiments, these tasks are referred to as *bpic2012_1*, *bpic2012_2*, and *bpic2012_3*.

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

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

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

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

The resulting 24 datasets exhibit different characteristics which can be seen in Table 6. The smallest log is *production* which contains 220 cases, while the largest one is *traffic* with 129615 cases. The most heterogenous in terms of case length are the *bpic2011* labelled datasets, where the longest case consists of 1814 events. On the other hand, the most homogenous is the *traffic* log, where case length varies from 2 to 20 events. The class labels are the most imbalanced in the *hospital_2* dataset, where only 5% of cases are labeled as *positive* ones (class label = 1). Conversely, in *bpic2012_1*, *bpic2017_3*, and *traffic*, the classes are almost balanced. In terms of event classes, the most homogenous are the insurance datasets, with only 9 distinct activity classes. The most heterogenous are the *bpic2015* datasets, reaching 396 event classes in *bpic2015_2*. The datasets also differ in terms of the number of static and dynamic attributes. The *insurance* logs contain the most dynamic attributes (22), while the *sepsis* datasets contain the largest number of static attributes (24).

While most of the data attributes can be readily included in the train and test datasets, timestamps should be preprocessed in order to derive meaningful features. In our experiments, we use the following features extracted from the timestamp: month, weekday, hour, duration from the previous event in the given case, duration from the start of the case, and the position of the event in the case. Additionally, some recent works have shown that adding features extracted from collections of cases (inter-case features) increases the accuracy of the predictive models, particularly when predicting deadline violations [6, 38]. For example, waiting times are highly dependent on the number of ongoing cases of a process (the so-called "Work-In-Process"). In turn, waiting times may affect the outcome of a case, particularly if the outcome is defined with respect to a deadline or with respect to customer satisfaction. Accordingly, we extract an inter-case feature reflecting the number of cases that are "open" at the time of executing a given event. All the abovementioned features are used as numeric dynamic (event) attributes.

Table 6. Statistics of the datasets used in the experiments.

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

Not surprisingly, recent work along this latter direction has shown that adding features extracted from collections of cases (inter-case features) increases the accuracy of the predictive models, particularly when predicting deadline violations [6, 38].

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

5.2 Experimental set-up

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

5.2.1 Research questions and evaluation measures. In a predictive process monitoring use case, the quality of the predictions is typically measured with respect to two main dimensions based on the following desiderata: A good prediction should be *accurate* and it should be made in the *early* stages of the process. A prediction that is often inaccurate is a useless prediction, as it cannot be relied on when making decisions. Therefore, accuracy is, in a sense, the most important quality of a prediction. The earlier an accurate prediction is made, the more useful it is in practice, as it leaves more time to act upon the prediction. Based on this rationale, we formulate the first subquestion (RQ3.1) as follows: How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of accuracy and earliness of the predictions?

Different metrics can be used to measure the accuracy of predictions. Rather than returning a hard prediction (a binary number) on the expected case outcome, the classifiers usually output a

real-valued score, reflecting how likely it is that the case will end in one way or the other. A good classifier will give higher scores to cases that will end with a positive outcome, and lower values to those ending with a negative one. Based on this intuition, we use the *area under the ROC curve (AUC)* metric that expresses the probability that a given classifier will rank a positive case higher than a negative one. A major advantage of the AUC metric over the commonly used *accuracy* (the proportion of correctly classified instances) or *F-score* (the harmonic mean of precision and recall), is that the AUC remains unbiased even in case of a highly imbalanced distribution of class labels [2]. Furthermore, AUC is a threshold-independent measure, as it operates on the ranking of the scores rather than the binary class values. Still, relying on a single evaluation criterion may provide a biased viewpoint of the results; therefore, we report the F-scores (on the default threshold of 0.5) additionally to AUC.

From the literature, two different approaches emerge for measuring the earliness of the predictions. One way [22] is to evaluate the models separately for each prefix length. In each step, the prediction model is applied to a subset of prefixes of exactly the given length. The improvement of prediction accuracy as the prefix length increases provides an implicit notion of earliness. In particular, the smaller the prefix length when an acceptable level of accuracy is reached, the better the method in terms of earliness. If needed, earliness can be defined explicitly as a metric — the smallest prefix length where the model achieves a specified accuracy threshold. Another option is to keep monitoring each case until the classifier gives an outcome prediction with a sufficiently high confidence, and then we measure the earliness as the average prefix length when such a prediction is made [10, 40]. The latter approach is mostly relevant in failure prediction scenarios, when the purpose is to raise an alarm when the estimated risk becomes higher than a pre-specified threshold (for a survey of failure prediction models, see [35]). However, even when the predictions come with a high confidence score, they might not necessarily be accurate. In the benchmark, we employ the first approach to measuring earliness, evaluating the models on each prefix length, because it provides a more straightforward representation of earliness that is relevant for all business processes. Also, it assures that the "early predictions" have reached a suitable level of accuracy.

In order to be applicable in practice a prediction should be produced *efficiently*, i.e., the execution times should be suitable for a given application. To address this, we formulate the second subquestion (RQ3.2) as follows: How do the existing outcome-oriented predictive business process monitoring techniques compare in terms of execution times? When measuring the execution times of the methods, we distinguish the time taken in the offline and the online modes. The *offline time* is the total time needed to construct the classifier from the historic traces available in an event log. Namely, it includes the time for constructing the prefix log, bucketing and encoding the prefix traces, and training the classifier. Note that we do not add the time spent on model selection to the offline time measurements. The reason for this is that we consider the extent of hyperparameter optimization to be largely dependent on the requirements and the constraints of a given project. Specifically, if obtaining a final model with minimal amount of time is critical in a project, one can settle for a smaller number of iterations for hyperparameter optimization, while if the accuracy of the final model is of greater importance than the time for obtaining the model itself, more time can be spent on model selection. Still, a rough estimate of the total time needed for optimizing the hyperparameters can be obtained by multiplying the time taken for building the final model with the number of optimization rounds to be performed. In the online phase, it is essential that a prediction is produced almost instantaneously, as the predictions are usually needed in real time. Accordingly, we define the *online time* as the average time for processing one incoming event (incl. bucketing, encoding, and predicting based on this new event).

The execution times are affected by mainly two factors. Firstly, since each prefix of a trace constitutes one sample, the lengths of the traces have a direct effect on the number of (training) samples. It is natural that the more samples are used for training, the better accuracy the predictive monitoring system could yield. At the same time, using more samples increases the execution times of the system. In applications where the efficiency of the predictions is of critical importance, reducing the number of training samples can yield a reasonable tradeoff, bringing down the execution times to a suitable level, while accepting lower accuracy. One way to reduce the number of samples is gap-based filtering [10], where a prefix is added to the training set only after each g events in the trace. This leads us to the third subquestion (RQ3.3): To what extent does gap-based filtering improve the execution times of the predictions?

The second factor that affects the execution times is the number and the diversity of attributes that need to be processed. In particular, the number of unique values (levels) in the categorical attribute domains has a direct effect on the length of the feature vector constructed for each sample, since each level corresponds to a feature in the vector (this holds for one hot encoding, as well as using occurrences or frequencies). The dimensionality of the vector can be controlled by filtering of the levels, for instance, by using only the most frequent levels for each categorical attribute. However, such filtering may negatively impact the accuracy of the predictions. In the fourth subquestion (RQ3.4), we aim to answer the following: To what extent does filtering the levels of categorical attributes based on their frequencies improve the execution times of the predictions?

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

5.2.3 Classifier learning and bucketing parameters. We selected four classification algorithms for the experiments: random forest (RF), gradient boosted trees (XGBoost), logistic regression (logit), and support vector machines (SVM). We chose logistic regression because of its simplicity and wide application in various machine learning applications. SVM and RF have been used in existing outcome-oriented predictive monitoring studies, whereas RF has shown to outperform many other methods (such as decision trees) in both predictive monitoring scenarios [22] and in more general empirical studies [13]. We also included the XGBoost classifier which has recently gained attention and showed promising results when applied to business process data [34, 38]. Furthermore, a recent empirical study on the performance of classification algorithms across 165 datasets has shown that RF and boosted trees generally outperform other classifier learning techniques [28]. For the clustering-based bucketing approach (cf. Section 4.2.4), we use the k-means clustering algorithm, which is one of the most widely used clustering methods in general.

The classification algorithms as well as some of the bucketing methods (clustering and KNN), require one to specify a number of parameters. In order to achieve good performance with each of the techniques, we optimize the hyperparameters using the Tree-structured Parzen Estimator (TPE) algorithm [1], separately for each combination of a dataset, a bucketing method, and a sequence

encoding method. For each combination of parameter values (i.e., a configuration) we performed 3-fold cross validation within the whole set of prefix traces L^* extracted from the training set, and we selected the configuration that led to the highest mean AUC calculated across the three folds. In the case of the prefix length based bucketing method, an optimal configuration was chosen for each prefix length separately (i.e., for each combination of a dataset, a bucketing method, an encoding approach and a prefix length). Table 7 presents the bounds and the sampling distributions for each of the parameters, given as input to the optimizer. In the case of RF and XGBoost, we found via exploratory testing that the results are almost unaffected by the *number of estimators (i.e., trees)* trained per model. Therefore, we use a fixed value of $n\_estimators = 500$ throughout the experiments.

Table 7. Hyperparameters and distributions used in optimization via TPE.

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

Both k-means and KNN require us to map each trace prefix into a feature vector in order to compute the Euclidean distance between pairs of prefixes. To this end, we applied the aggregation encoding approach, meaning that we map each trace to a vector that tells us how many times each possible activity appears in the trace. In order to keep consistent with the original methods, we decided to use only the control flow information for the clustering and the determining of the nearest neighbors.

In the case of the state-based bucketing, we need to specify a function that maps each trace prefix to a state. To this end, we used the last-activity encoding, meaning that one state is defined per possible activity and a trace prefix is mapped to the state corresponding to the last activity in the prefix. Note that the number of buckets produced by this approach is equal to the number of unique activities in the dataset (see Table 6). The reason for this choice is because this approach leads to reasonably large buckets. We also experimented with the multiset state abstraction approach, but it led to too many buckets, some of small size, so that in general there were not enough samples per bucket to train a classifier with sufficient accuracy.

When using a state-based or a clustering-based bucketing method, it may happen that a given bucket contains too few trace prefixes to train a meaningful classifier. Accordingly, we set a minimum bucket size threshold. If the number of trace prefixes in a bucket is less than the threshold, we do not build a classifier for that bucket but instead, any trace prefix falling in that

bucket is mapped to the label (i.e., the outcome) that is predominant in that bucket, with a likelihood score equal to the ratio of trace prefixes in the bucket that have the predominant label. To be consistent with the choice of the parameter K in the KNN approach proposed in [25], we fixed the minimum bucket size threshold to 30. Similarly, when all of the training instances in a bucket belong to the same class, no classifier is trained for this bucket and, instead, the test instances falling to this bucket are simply assigned the same class (i.e., the assigned prediction score is either 0 or 1).

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

5.2.4 Filtering and feature encoding parameters. As discussed in Section 4.1, training a classifier over the entire prefix log L^* (all prefixes of all traces) can be time-consuming. Furthermore, we are only interested in making predictions for earlier events rather than making predictions towards the end of a trace. Additionally, we observe that the distributions of the lengths of the traces can be different within the classes corresponding to different outcomes (see Figures 15-16 in Appendix). When all instances of long prefixes belong to the same class, predicting the outcome for these (or longer) prefixes becomes trivial. Accordingly, during both the training and the evaluating phases, we vary the prefix length from 1 to the point where 90% of the minority class have finished (or until the end of the given trace, if it ends earlier than this point). For computational reasons, we set the upper limit of the prefix lengths to 40, except for the *bpic2017* datasets where we further reduced the limit to 20. We argue that setting a limit to the maximum prefix length is a reasonable design choice, as the aim of predictive process monitoring is to predict as early as possible and, therefore, we are more interested in predictions made for shorter prefixes. When answering RQ3.3, we additionally apply the gap-based filtering to the training set with $g \in \{3, 5\}$. For instance, in case of $g = 5$, only prefixes of lengths 1, 6, 11, 16, 21, 26, 31, and 36 are included in the training set.

In Section 4.3.3, we noted that the aggregation encoding requires us to specify an aggregation function for each event attribute. For activities and resource attributes we use the count (frequency) aggregation function (i.e., how many times a given activity has been executed, or how many activities has a given resource executed). The same principle is applied to any other event attribute of a categorical type. For each numeric event attribute, we include two numeric features in the feature vector: the mean and the standard deviation. Furthermore, to answer RQ3.4, we filter each of the categorical attribute domains by using only the top $\{10, 25, 50, 75, 90\}$ percent of the most frequent levels from each attribute.

For index-based encoding (Section 4.3.4), we focus on the basic index-encoding technique without the HMM extension proposed in [22]. The reason is that the results reported in [22] do not show that HMM provides any visible improvement, and instead this encoding adds complexity to the training phase.

5.3 Results: accuracy and earliness

Table 8 reports the overall AUC and F-score for each dataset and method using XGBoost, while Tables 13, 14 and 15 in the Appendix report the same results for RF, logit, and SVM. The overall metric values (AUR or F-score) are obtained by first calculating the scores separately for each prefix length (using only prefixes of a given length) and then by taking the weighted average of the obtained scores, where the weights are assigned according to the number of prefixes used for the calculation of a given score. This weighting assures that the overall metrics are influenced equally by each prefix in the evaluation set, instead of being biased towards longer prefixes (i.e., where many cases have already finished). The best-performing classifiers are XGBoost, which achieves the highest AUC in 15 out of 24 datasets and the highest F-score in 11 datasets, and RF, which achieves

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

In order to further assess the relative performance of the classifiers, we applied the Nemenyi test (as proposed in [9]) as a means for statistical comparison of classifiers over multiple datasets. In this setting, we compared the best AUC scores obtained by each classifier for a given dataset, i.e. we selected the best combination of bucketing and encoding technique for each dataset and classification algorithm. The resulting critical difference diagram (Figure 5), obtained using a 0.05 significance level, confirms that XGBoost is on average the best performing classifier, achieving an average rank of around 1.8. However, the difference between XGBoost, RF, and logit is not statistically significant (indicated by the horizontal line connecting these three classifiers). On the other hand, SVM performs significantly worse than XGBoost and RF. In the following we analyze the results obtained by XGBoost in detail.

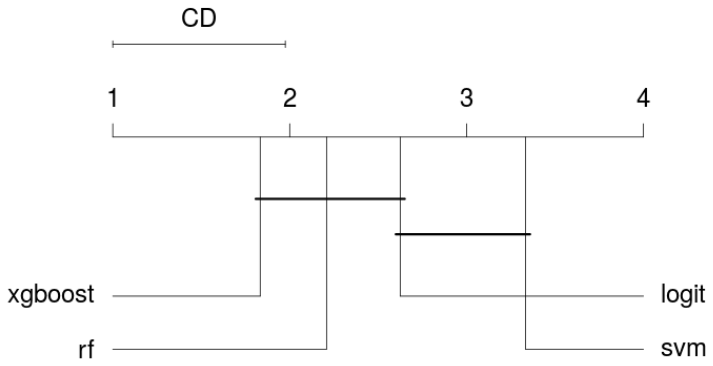


Fig. 5. Comparison of all classifiers against each other with the Nemenyi test. The classifiers are compared in terms of the best AUC achieved in each of the 24 datasets. Groups of classifiers that are not significantly different (at $p < .05$) are connected.

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

The critical difference diagram in Figure 6 shows that in terms of the average rank, prefix_agg slightly outperforms single_agg, while both are closely followed by cluster_agg and state_agg. Despite a larger gap in the average ranks, the differences from prefix_agg to prefix_index, cluster_laststate, state_laststate, and single_laststate are not statistically significant either. On the other hand, prefix_laststate, knn_laststate, and knn_agg are found to perform significantly worse than prefix_agg.

Table 8. Overall AUC (F-score) for **XGBoost**

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

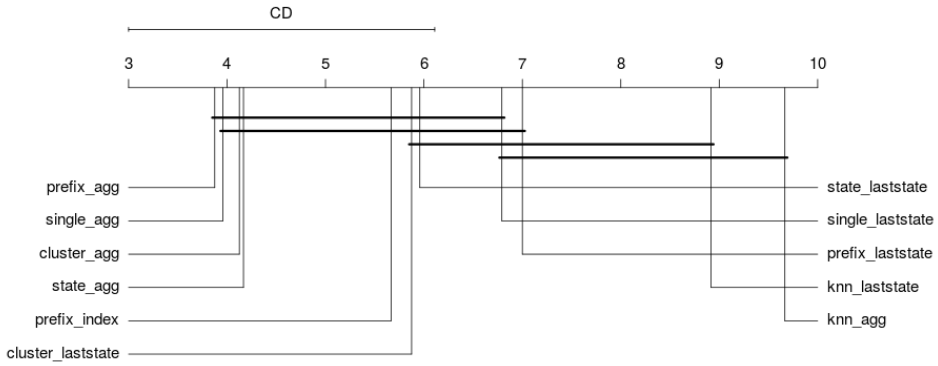


Fig. 6. Comparison of the bucketing/encoding combinations with the Nemenyi test. The methods are compared in terms of AUC achieved in each of the 24 datasets using the **XGBoost** classifier. Groups of methods that are not significantly different (at $p < .05$) are connected.

Figures 7 and 8 present the prediction accuracy in terms of AUC for the six best performing methods (according to Figure 6), evaluated over different prefix lengths[5]. Each evaluation point includes prefix traces of exactly the given length. In other words, traces that are altogether shorter than the required prefix are left out of the calculation. Therefore, the number of cases used for evaluation is monotonically decreasing when increasing prefix length. In most of the datasets, we see that starting from a specific prefix length the methods with aggregation encoding achieve perfect prediction accuracy ($AUC = 1$). It is natural that the prediction task becomes trivial when cases are close to completion, especially if the labeling function is related to the control flow or to the data payload present in the event log. However, there are a few exceptions from this rule, namely, in the *bpic2012* and *sepsis* datasets, the results seem to decline on larger prefix sizes. To investigate this phenomenon, we recalculated the AUC scores on the longer traces only, i.e., traces that have a length larger than or equal to the maximum considered trace length (see Figure 21 in Appendix). This analysis confirmed (with the exception of *sepsis_1*, which we discuss separately later in this section) that the phenomenon is caused by the fact that the datasets contain some short traces for which it appears to be easy to predict the outcome. These short traces are not included in the later evaluation points, as they have already finished by that time. Therefore, we are left with longer traces only, which appear to be more challenging for the classifier, dragging down the total AUC score on larger prefix lengths.

From the results presented above, we see that the choice of the bucketing method seems to have a smaller effect on the results than the sequence encoding. Namely, the best results are usually achieved using the aggregation encoding with either the single bucket, clustering, prefix length based, or state-based bucketing. In general these methods achieve very comparable results. Still, it appears that in event logs with many trace variants (relative to the total number of traces), such as *insurace*, *production*, *bpic2012*, and *bpic2015_4* (see Table 6) it may be preferred to use a

[5]For a comparison of all the twelve methods, see Figures 19 and 20 in Appendix.

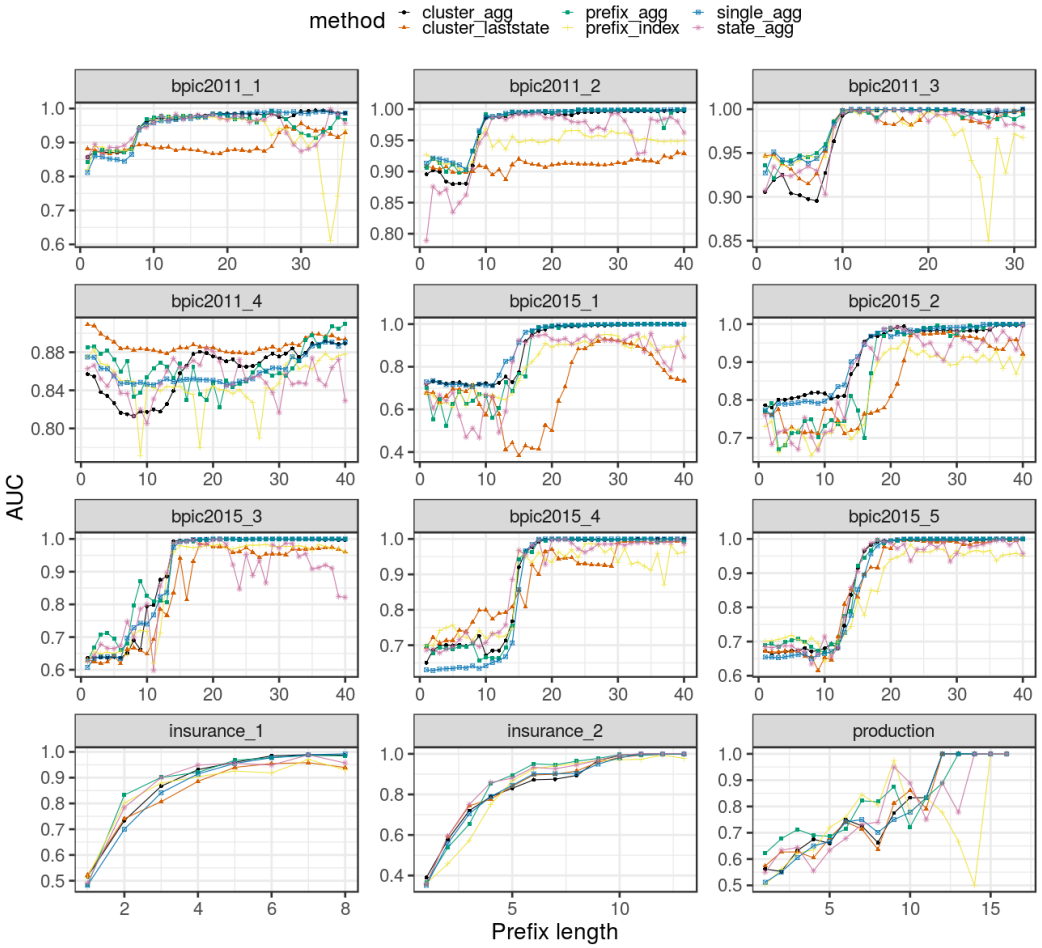


Fig. 7. AUC across different prefix lengths using **XGBoost**

multiclassifier instead of a single bucket. However, this only holds if each bucket receives enough data to learn relevant patterns. For instance, when the number of categorical attribute levels is very high (as in the *bpic2011*, *bpic2015*, and *hospital* datasets), a single classifier is usually able to produce better predictions. Similarly, when the classes are very imbalanced (*hospital*, *bpic2017_2*, *sepsis_2*), it is likely that some buckets receive too limited information about the minority class and, therefore, a single classifier is recommended over a multiclassifier. The effect of having too many buckets can easily be seen in case of state-based bucketing when the number of different event classes (and therefore, the number of buckets) is very large (see *bpic2011* and *bpic2015* in Figure 7 and the counts of training prefix traces in each bucket in Figures 17-18 in Appendix). As a result, each classifier receives a small number of traces for training, resulting in a very spiky performance across different prefix lengths. The same phenomenon can be seen in case of prefix_agg, which usually achieves very good performance, but at times can produce unexpectedly inaccurate results (like in the longer prefixes of *bpic2011_1* and *bpic2012_2*). On the contrary, single_agg and cluster_agg in general produce stable and reliable results on all datasets and across all prefix sizes. The optimal

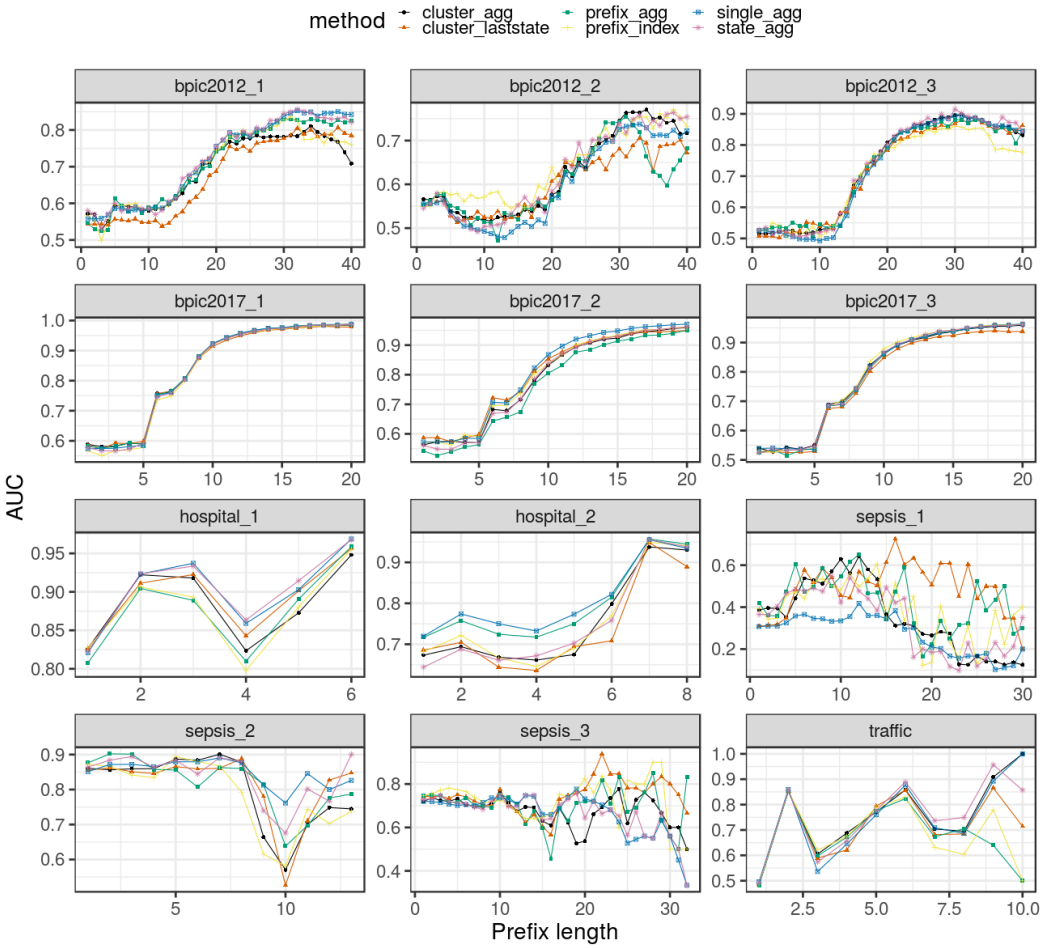


Fig. 8. AUC across different prefix lengths using **XGBoost** (continued)

number of clusters in case of cluster_agg with XGBoost was often found to be small, i.e. between 2-7 (see Table 11 in Appendix), which explains why these two methods behave similarly. In some cases where the optimized number of clusters was higher, e.g., *bpic2012_1* and *hospital_2*, the accuracy of cluster_agg drops compared to single_agg.

We can see from Figure 7 that in several cases (e.g., *bpic2011*, *bpic2015*, *bpic2012_1*, and *sepsis_2*), all the methods achieve a similar AUC on shorter prefixes, but then quickly grow apart as the size of the prefix increases. In particular, the aggregation encoding seems to be able to carry along relevant information from the earlier prefixes, while the last state encoding entails more limited information that is often insufficient for making accurate predictions. Comparing the overall AUC in Table 8, the last state encodings outperform the other methods in only three datasets. One such exceptional case is *bpic2011_4*, where single_laststate and cluster_laststate considerably outperform their aggregation encoding counterparts. A deeper investigation of this case revealed that this is due to overfitting in the presence of a concept drift in the dataset. In particular, the aggregation encodings yielded a more complex classifier (e.g. the optimized maximum tree depth is 15 in case of single_agg and

6 in case of single_laststate), memorizing the training data completely. However, a concept drift occurs in the relationship between some data attributes and the class labeling, which affects the aggregated features more than the "as-is" features (see Figure 22 in Appendix). Another exception is *sepsis_1*, where the best results are achieved by knn_laststate. In this dataset, all the methods consistently yield an AUC less than 0.5 (i.e. worse than random). Further investigation revealed that this phenomenon is also due to a concept drift in the dataset, which makes it impossible to learn useful patterns in case of a temporal train-test split. For instance, Figure 23 in Appendix illustrates that in the training set the values of *CRP* are larger in positive instances, while in the test set the *CRP* values are larger in negative instances. The third exceptional dataset is *insurance_2*, where state_laststate slightly outperforms other techniques. We did not find any peculiarities in this dataset that would explain this phenomenon, however, the differences in scores (between, e.g. state_laststate and state_agg) are much smaller compared to the previous two datasets.

We can also observe that the index-based encoding, although lossless, in general does not outperform the lossy encoding schemes, reaching the highest overall AUC only in 4 datasets: *bpic2012_2*, *bpic2017_3*, *sepsis_3*, and *traffic*. In these logs the number of levels in dynamic categorical attributes is not very high (see Table 6), which helps to keep the size of the feature vector in reasonable bounds. Still, even in these cases the difference in AUC compared to the other methods (such as prefix_agg) is marginal. In fact, in some datasets (e.g., *hospital_2* and *sepsis_2*) index-based encoding performs even worse than the last state encoding. This suggests that in the given datasets, the order of events is not as relevant for determining the final outcome of the case. Instead, combining the knowledge from all events performed so far provides much more signal. Alternatively, it may be that the order of events (i.e., the control-flow) does matter in some cases, but the classifiers considered in this study (including XGBoost) are not able to infer high-level control-flow features by themselves, which would explain why we see that even the simple aggregation-based methods outperform index-based encoding. This phenomenon deserves a separate in-depth study.

These observations, further supported by the fact that KNN does not appear among the top performing methods, lead to the conclusion that it is preferable to build few classifiers (or even just a single one), with a larger number of traces as input. XGBoost seems to be a classifier sophisticated enough to derive the "bucketing patterns" by itself when necessary. Another advantage of the single_agg method over cluster_agg is the simplicity of a single bucket. In fact, no additional pre-processing step for bucketing the prefix traces is needed. On the other hand, clustering (regardless of the clustering algorithm) comes with a set of parameters, such as the number of clusters in k-means, that need to be tuned for optimal performance. Therefore, the time and effort needed from the user of the system for setting up the prediction framework can be considerably higher in case of cluster_agg, which makes single_agg the overall preferred choice of method in terms of accuracy and earliness. This discussion concludes the answer to RQ3.1.

5.4 Results: time performance

The time measurements for all of the methods and classifiers, calculated as averages over 5 identical runs using the final (optimal) parameters, are presented in Tables 9 and 10 (XGBoost), Tables 16 and 17 (RF), Tables 18 and 19 (logit), and Tables 20 and 21 (SVM). In the offline phase, the fastest of the four classifiers is logit. The ordering of the others differs between the small (*production*, *bpic2011*, *bpic2015*, *insurance*, and *sepsis*) and the large (*bpic2017*, *traffic*, *hospital*) datasets. In the former group, the second fastest classifier is SVM, usually followed by RF and, then, XGBoost. Conversely, in the larger datasets, XGBoost appears to scale better than the others, while SVM tends to be the slowest of the three. In terms of online time, logit, SVM, and XGBoost yield comparable

performance, while RF is usually slower than the others. In the following, we will, again, analyse deeper the results obtained with the XGBoost classifier.

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

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

In general, all methods are able to process an event in less than 100 milliseconds during the online phase (the times in Tables 9 and 10 are in milliseconds per processed event in a partial trace). Exceptions are *hospital_1* and *hospital_2*, where processing an event takes around 0.3-0.4 seconds. The online execution times are very comparable across all the methods, except for KNN and prefix_index. While prefix_index often takes double the time of other methods, the patterns for KNN are less straightforward. Namely, in some datasets (*bpic2012*, *sepsis*, *production*, *insurance*, and *traffic*), the KNN approaches take considerably more time than the other techniques, which can be explained by the fact that these approaches train a classifier at runtime. However, somewhat surprisingly, in other datasets (*hospital* and *bpic2011* datasets) the KNN approaches yield the best execution times even at runtime. A possible explanation for this is that in cases where all the selected nearest neighbors are of the same class, no classifier is trained and the class of the neighbors is immediately returned as the prediction. However, note that the overall AUC in these cases is 7-21 percentage points lower than that of the best method (8). In the offline phase, the overhead of applying aggregation functions becomes more evident, with the last state encoding almost always outperforming the aggregation encoding methods by a few milliseconds. The fastest method in the online phase tends to be prefix_laststate, which outperforms the others in 17 out of 24 datasets. It is followed by knn_laststate, state_laststate, and single_laststate.

In terms of online execution times, the observed patterns are in line with those of other classifiers. However, there are some differences in the offline phase. Namely, in case of RF, the single classifiers perform relatively better as compared to bucketing methods. Furthermore, the difference between the encoding methods becomes more evident, with the last state encodings usually outperforming their aggregation encoding counterparts. The index-based encoding is still the slowest of the techniques. In case of logit, all the methods achieve comparable offline times, except for index-based encoding and the clustering based bucketings, which are slower than the others. In case of SVM, the single_laststate method tends to be much slower than other techniques. This discussion concludes the answer to RQ3.2.

5.5 Results: gap-based filtering

In order to investigate the effects of gap-based filtering on the execution times and the accuracy, we selected 4 methods based on their performance in the above subsections: single_agg, single_laststate,

Table 9. Execution times for **XGBoost**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	418.35 ± 0.56	69 ± 98	581.68 ± 1.09	62 ± 96	217.69 ± 1.38	71 ± 96
single_agg	317.18 ± 0.58	69 ± 99	342.3 ± 2.02	62 ± 97	271.33 ± 0.54	71 ± 97
knn_laststate	5.9 ± 0.31	**44 ± 65**	9.82 ± 0.66	**37 ± 59**	4.14 ± 0.06	**48 ± 72**
knn_agg	6.63 ± 0.12	52 ± 76	9.79 ± 0.44	46 ± 72	4.57 ± 0.06	61 ± 91
state_laststate	142.53 ± 0.31	52 ± 72	181.87 ± 0.92	48 ± 74	86.78 ± 0.52	53 ± 70
state_agg	183.67 ± 0.79	61 ± 84	169.98 ± 0.47	58 ± 90	119.84 ± 0.2	62 ± 82
cluster_laststate	211.88 ± 1.0	66 ± 112	592.92 ± 4.84	67 ± 112	93.83 ± 0.48	57 ± 98
cluster_agg	341.4 ± 1.89	70 ± 113	381.61 ± 1.87	62 ± 111	94.45 ± 0.71	72 ± 111
prefix_index	763.8 ± 20.33	114 ± 68	1405.87 ± 88.64	126 ± 62	428.6 ± 19.49	113 ± 69
prefix_laststate	290.64 ± 1.44	57 ± 86	264.71 ± 4.58	50 ± 81	108.84 ± 0.33	59 ± 84
prefix_agg	172.8 ± 12.28	56 ± 82	274.29 ± 9.08	53 ± 82	125.61 ± 7.42	58 ± 80
	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	418.65 ± 17.42	62 ± 97	263.56 ± 0.38	20 ± 30	134.87 ± 0.59	18 ± 29
single_agg	319.27 ± 10.75	62 ± 98	105.58 ± 0.39	22 ± 34	282.97 ± 8.14	20 ± 32
knn_laststate	8.99 ± 0.23	**42 ± 67**	8.44 ± 0.04	31 ± 51	11.46 ± 0.63	34 ± 59
knn_agg	9.47 ± 0.07	57 ± 86	9.45 ± 0.06	45 ± 72	11.95 ± 0.12	38 ± 66
state_laststate	156.45 ± 1.1	48 ± 73	25.2 ± 0.08	25 ± 44	30.28 ± 0.2	27 ± 45
state_agg	301.91 ± 7.54	58 ± 90	53.39 ± 0.34	29 ± 48	64.72 ± 0.08	31 ± 49
cluster_laststate	274.97 ± 8.03	69 ± 112	46.71 ± 0.58	37 ± 58	52.8 ± 1.6	36 ± 60
cluster_agg	252.81 ± 6.36	69 ± 112	62.25 ± 0.48	29 ± 46	135.33 ± 1.21	28 ± 45
prefix_index	794.59 ± 35.34	111 ± 59	396.03 ± 4.79	51 ± 10	442.17 ± 8.47	48 ± 13
prefix_laststate	344.73 ± 13.5	48 ± 80	62.17 ± 0.26	**7 ± 9**	45.27 ± 0.27	**6 ± 7**
prefix_agg	441.82 ± 1.01	63 ± 98	57.14 ± 0.16	10 ± 10	74.98 ± 1.04	8 ± 8
	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	150.3 ± 0.44	21 ± 33	67.21 ± 0.09	18 ± 27	83.78 ± 0.31	17 ± 26
single_agg	627.3 ± 3.48	21 ± 35	191.22 ± 0.21	19 ± 29	428.23 ± 1.24	17 ± 28
knn_laststate	18.9 ± 0.68	37 ± 61	7.38 ± 0.08	29 ± 50	17.22 ± 0.14	36 ± 60
knn_agg	19.57 ± 0.84	41 ± 69	7.5 ± 0.35	34 ± 58	19.67 ± 0.53	37 ± 87
state_laststate	45.06 ± 0.03	29 ± 49	18.5 ± 0.06	25 ± 42	37.97 ± 0.08	23 ± 38
state_agg	86.47 ± 0.06	33 ± 53	30.76 ± 0.07	29 ± 45	73.2 ± 0.46	26 ± 41
cluster_laststate	102.82 ± 0.26	35 ± 60	34.94 ± 0.19	34 ± 62	72.76 ± 0.26	35 ± 60
cluster_agg	181.85 ± 1.13	28 ± 45	61.06 ± 0.84	26 ± 40	74.79 ± 0.7	20 ± 34
prefix_index	2155.36 ± 80.83	55 ± 15	261.21 ± 0.5	41 ± 8	550.87 ± 4.23	48 ± 12
prefix_laststate	98.37 ± 0.7	**7 ± 9**	37.59 ± 0.06	**6 ± 7**	109.77 ± 0.1	**6 ± 8**
prefix_agg	108.91 ± 2.83	9 ± 9	40.83 ± 0.38	8 ± 8	92.61 ± 0.88	8 ± 9
	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	7.28 ± 0.12	25 ± 21	36.0 ± 0.1	38 ± 31	15.52 ± 0.09	33 ± 30
single_agg	4.3 ± 0.13	28 ± 25	12.81 ± 0.12	40 ± 33	78.17 ± 0.62	34 ± 32
knn_laststate	1.09 ± 0.01	51 ± 49	0.66 ± 0.01	49 ± 44	0.96 ± 0.01	48 ± 36
knn_agg	0.76 ± 0.0	62 ± 60	0.66 ± 0.0	57 ± 53	1.06 ± 0.01	69 ± 50
state_laststate	2.69 ± 0.11	**23 ± 19**	11.45 ± 0.1	**31 ± 22**	15.22 ± 0.06	**26 ± 21**
state_agg	4.37 ± 0.04	30 ± 26	17.67 ± 0.11	40 ± 31	30.62 ± 0.17	35 ± 30
cluster_laststate	6.94 ± 0.06	30 ± 30	32.75 ± 0.11	39 ± 37	42.42 ± 0.15	37 ± 34
cluster_agg	6.35 ± 0.07	35 ± 31	30.06 ± 0.2	52 ± 43	47.2 ± 0.14	45 ± 43
prefix_index	13.81 ± 0.04	49 ± 10	58.79 ± 0.58	90 ± 4	80.56 ± 0.12	91 ± 4
prefix_laststate	5.62 ± 0.35	28 ± 23	9.44 ± 0.03	**31 ± 21**	22.3 ± 0.07	**26 ± 21**
prefix_agg	5.89 ± 0.01	28 ± 23	17.17 ± 0.05	35 ± 24	24.66 ± 0.07	30 ± 24

Table 10. Execution times for **XGBoost** (continued)

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

prefix_index, and prefix_agg. The first three of these methods were shown to take the most time in the offline phase, i.e., they have the most potential to benefit from a filtering technique. Also, single_agg and prefix_agg achieved the highest overall AUC scores, which makes them the most attractive candidates to apply in practice. Furthermore, we selected 6 datasets which are representative in terms of their sizes (i.e., number of traces), consist of relatively long traces on average, and did not yield a very high accuracy very early in the trace.

Figures 9–11 plot the performance of the classifiers over different gap sizes, i.e. on the x-axis, $g = 1$ corresponds to no filtering (using prefixes obtained after every event), $g = 3$ to using prefixes obtained after every 3rd event, and $g = 5$ to prefixes after every 5th event. In Figure 9 we can see that using $g = 3$ yields an improvement of about 2-3 times in the offline execution times, while using $g = 5$, the improvement is usually around 3-4 times, as compared to no filtering ($g = 1$). For instance, in case of single_agg on the *bpic2017_2* dataset with $g = 5$, this means that the offline phase takes about 30 minutes instead of 2 hours. At the same time, the overall AUC remains at the same level, sometimes even increasing when a filtering is applied (Figure 10). On the other hand, the gap-based filtering only has a marginal (positive) effect on the online execution times, which usually remain on the same level as without filtering (Figure 11). This concludes the answer to RQ3.3.

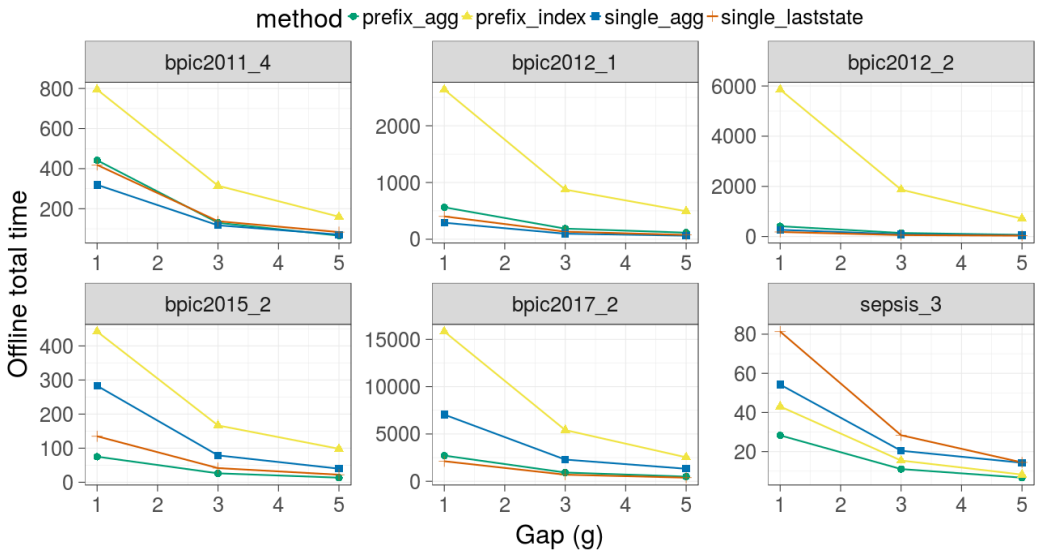


Fig. 9. Offline times across different gaps (XGBoost)

5.6 Results: categorical domain filtering

To answer RQ3.4, we proceed with the 4 methods as discussed in the previous subsection. To better investigate the effect of filtering the categorical attribute levels, we distinguish between the static and the dynamic categorical attributes. For investigating the effects of dynamic categorical domain filtering, we selected 9 datasets that contain a considerable number of levels in the dynamic categorical attributes.

Both the offline (Figure 12) and the online (Figure 14) execution times tend to increase linearly when the proportion of levels is increased. As expected, the prefix_index method benefits the most

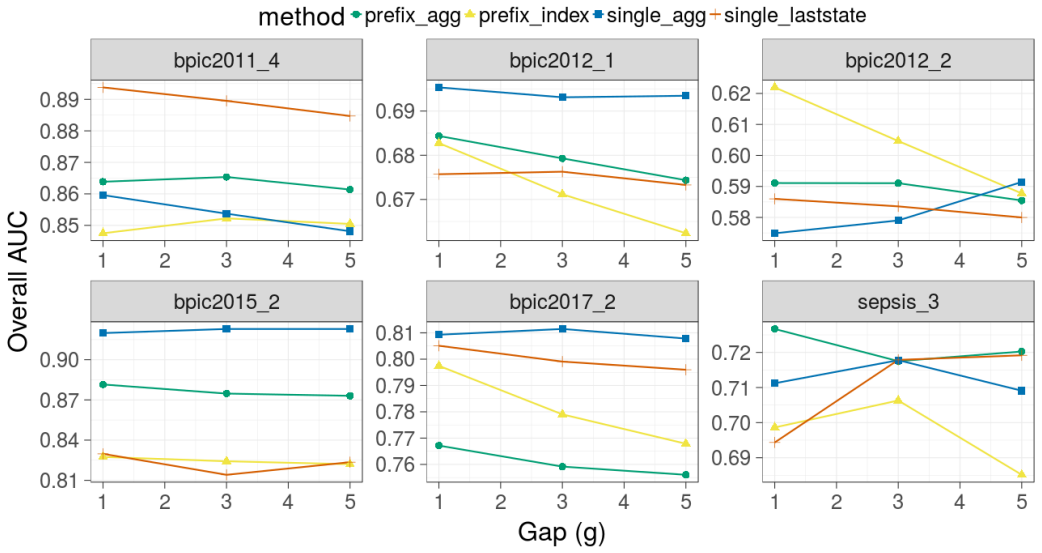


Fig. 10. AUC across different gaps (XGBoost)

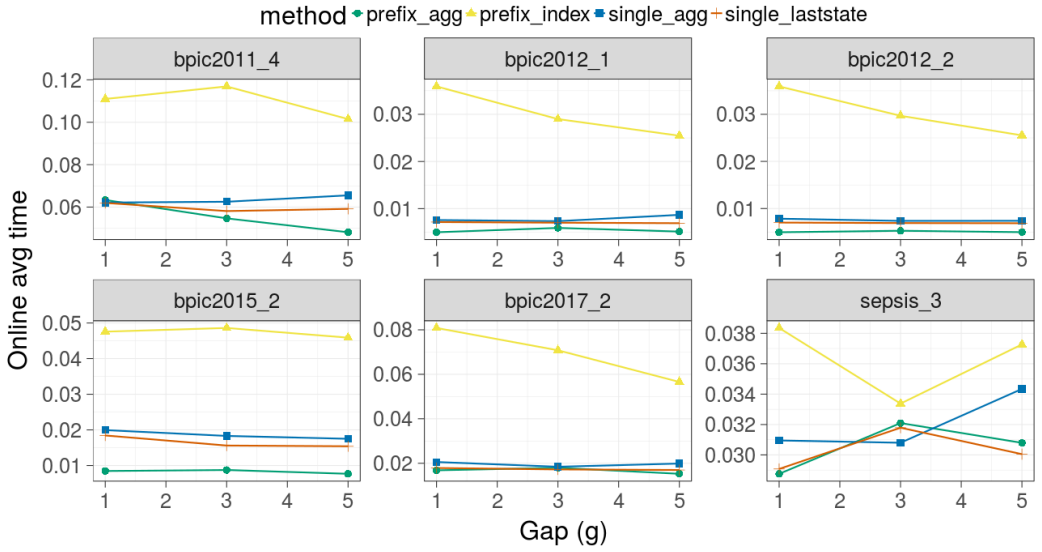


Fig. 11. Online times across different gaps (XGBoost)

from the filtering, since the size of the feature vector increases more rapidly than in the other methods when more levels are added (the vector contains one feature per level per event). Although the overall AUC is negatively affected by the filtering of levels (see Figure 13), reasonable tradeoffs can still be found. For instance, when using 50% of the levels in case of single_agg on the *hospital_2* dataset, the AUC is almost unaffected, while the training time has decreased by more than 30 minutes and the online execution times have decreased by a half.

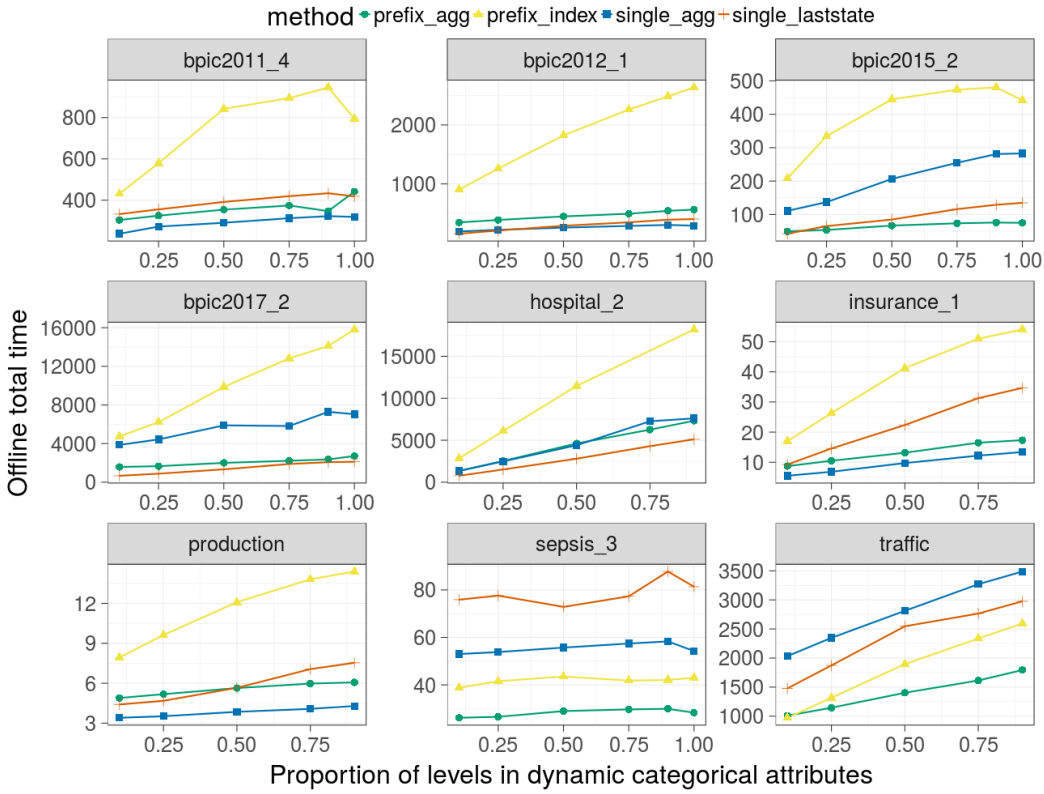


Fig. 12. Offline times across different filtering proportions of **dynamic** categorical attribute levels (XGBoost)

We performed similar experiments by filtering the static categorical attribute domains, selecting 6 datasets that contain a considerable number of levels in these attributes. However, the improvement in execution times were marginal compared to those obtained when using dynamic attribute filtering (see Figures 24-26 in Appendix). This is natural, since the static attributes have a smaller effect on the size of the feature vector (each level occurs in the vector only once). This concludes the answer to RQ3.4.

6 THREATS TO VALIDITY

One of the threats to the validity of this study relates to the potential selection bias in the literature review. To minimize this, we described our systematic literature review procedure on a level of detail that is sufficient to replicate the search. However, in time the search and ranking algorithms of the used academic database (Google Scholar) might be updated and return different results. Another potential source of bias is the subjectivity when applying inclusion and exclusion criteria, as well as when determining the primary and subsumed studies. In order to alleviate this issue, all the included papers were collected in a publicly available spreadsheet, together with decisions and reasons about excluding them from the study. Moreover, each paper was independently assessed against the inclusion and exclusion criteria by two authors, and inconsistencies were resolved with the mediation of a third author.

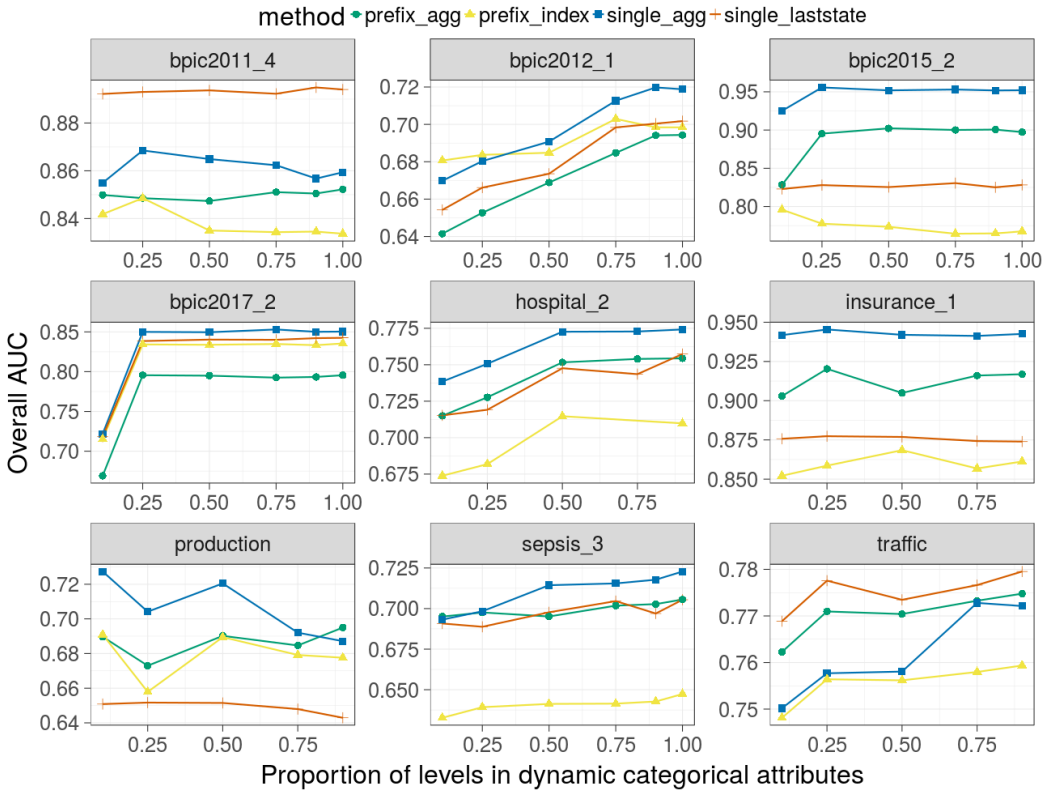


Fig. 13. AUC across different filtering proportions of **dynamic** categorical attribute levels (XGBoost)

Another threat to validity is related to the comprehensiveness of the conducted experiments. In particular, only one clustering method was tested, a single state abstraction was used when building the transition systems for state-based bucketing, and four classification algorithms were applied. It is possible that there exists, for example, a combination of an untested clustering technique and a classifier that outperforms the settings used in this study. Also, although the hyperparameters were optimized using a state-of-the-art hyperparameter optimization technique, it is possible that using more iterations for optimization or a different optimization algorithm, other parameter settings would be found that outperform the settings used in the current evaluation. Furthermore, the generalizability of the findings is to some extent limited by the fact that the experiments were performed on a limited number of prediction tasks (24), constructed from nine event logs. Although these are all real-life event logs from different application fields that exhibit different characteristics, it may be possible that the results would be different using other datasets or different log preprocessing techniques for the same datasets. In order to mitigate these threats, we built an open-source software framework which allows the full replication of the experiments, and made this tool publicly available. Moreover, additional datasets, as well as new sequence classification and encoding methods can be plugged in. So the framework can be used for future experiments. Also, the preprocessed datasets constructed from the three publicly available event logs are included together with the tool implementation in order to enhance the reproducibility of the experiments.

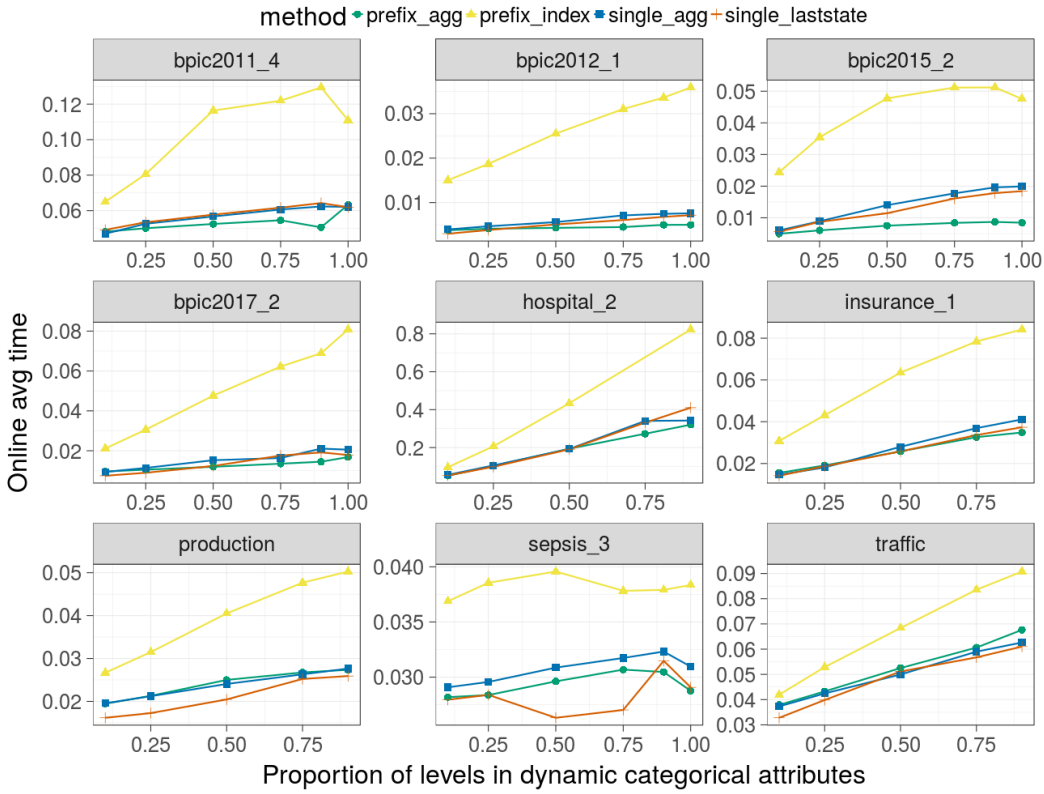


Fig. 14. Online times across different filtering proportions of **dynamic** categorical attribute levels (XGBoost)

7 CONCLUSION

This study provided a survey and comparative analysis and evaluation of existing outcome-oriented predictive business process monitoring techniques. The relevant existing studies were identified through a systematic literature review (SLR), which revealed 14 studies (some described across multiple papers) dealing with the problem of predicting case outcomes. Out of these, seven were considered to contain a distinct contribution (primary studies). Through further analysis of the primary studies, a taxonomy was proposed based on two main aspects, the trace bucketing approach and sequence encoding method employed. Combinations of these two aspects led to a total of 11 distinct methods.

The studies were characterized from different perspectives, resulting in a taxonomy of existing techniques. Finally, a comparative evaluation of the 11 identified techniques was performed using a unified experimental set-up and 24 predictive monitoring tasks constructed from 9 real-life event logs. To ensure a fair evaluation, all the selected techniques were implemented as a publicly available consolidated framework, which is designed to incorporate additional datasets and methods.

The results of the benchmark show that the most reliable and accurate results (in terms of AUC) are obtained using a lossy (aggregation) encoding of the sequence, e.g., the frequencies of performed activities rather than the ordered activities. One of the main benefits of this encoding is that it enables to represent all prefix traces, regardless of their length, in the same number of features. This

way, a single classifier can be trained over all of the prefix traces, allowing the classifier to derive meaningful patterns by itself. These results disprove the existing opinion in the literature about the superiority of a lossless encoding of the trace (index-based encoding) that requires prefixes to be divided into buckets according to their length, while multiple classifiers are trained on each such subset of prefixes.

The study also put into evidence the importance of checking for concept drifts when applying predictive monitoring methods. In the study, we found concept drifts in the data attributes extracted from two datasets, and in both cases, these drifts significantly affected the performance of all tested methods. This observation is aligned with previous studies in the field of process mining, which have shown that concept drifts are common in the control flow of business processes [24, 29, 30]. Techniques for automated detection and characterization of process control-flow drifts from event logs and event streams are available [24, 29, 30]. Researchers and practitioners using predictive monitoring methods should consider applying these detection methods, as well as standard statistical tests on the features extracted, to ensure that there is no drift present, which could affect the performance of the predictive models.

The study paves the way to several directions of future work. In Section 2 we noted that case and event attributes can be of categorical, numeric or textual type. The systematic review showed that existing methods are focused on handling categorical and numeric attributes, to the exclusion of textual ones. Recent work has shown how text mining techniques can be used to extend the index-based encoding approach of [40] in order to handle text attributes, however this latter work considered a reduced set of text mining techniques and has only been tested on two datasets of relatively small size and complexity.

Secondly, the methods identified in the survey are mainly focused on extracting features from one trace at a time (i.e., intra-case features), while only a single inter-case feature (the number of open cases) is included. However, due to the fact that the ongoing cases of a process share the same pool of resources, the outcome of a case may depend also on other aspects of the current state of the rest of ongoing cases in the process. Therefore, the accuracy of the models tested in this benchmark could be further improved by using a larger variety of inter-case features.

Lastly, as long-short term memory (LSTM) networks have recently gained attention in predicting remaining time and next activity of a running case of a business process [12, 39], another natural direction for future work is to study how LSTMs can be used for outcome prediction. In particular, could LSTMs automatically derive relevant features from collections of trace prefixes, and thus obviate the need for sophisticated feature engineering (aggregation functions), which has been so far the focus of predictive process monitoring research?

ACKNOWLEDGMENTS

This research is funded by the Australian Research Council (grant DP150103356), the Estonian Research Council (grant IUT20-55) and European Regional Development Fund (Dora Plus Program)

REFERENCES

[1] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Proc. of NIPS*. 2546–2554.

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

[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[4] Malu Castellanos, Norman Salazar, Fabio Casati, Umesh Dayal, and Ming-Chien Shan. 2005. Predictive business operations management. In *International Workshop on Databases in Networked Information Systems*. Springer, 1–14.

[5] Raffaele Conforti, Massimiliano De Leoni, Marcello La Rosa, and Wil MP Van Der Aalst. 2013. Supporting risk-informed decisions during business process execution. In *International Conference on Advanced Information Systems Engineering*.

Springer, 116–132.
[6] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil MP van der Aalst, and Arthur HM ter Hofstede. 2015. A recommendation system for predicting risks across multiple business process instances. *Decision Support Systems* 69 (2015), 1–19.
[7] Massimiliano De Leoni, Wil MP van der Aalst, and Marcus Dees. 2014. A general framework for correlating business process characteristics. In *International Conference on Business Process Management*. Springer, 250–266.
[8] Massimiliano de Leoni, Wil MP van der Aalst, and Marcus Dees. 2016. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems* 56 (2016), 235–257.
[9] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.
[10] Chiara Di Francescomarino, Marlon Dumas, Fabrizio M Maggi, and Irene Teinemaa. 2017. Clustering-based predictive process monitoring. *IEEE Transactions on Services Computing* (2017).
[11] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. 2018. *Fundamentals of Business Process Management* (2nd ed.). Springer.
[12] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. 2016. A Deep Learning Approach for Predicting Process Behaviour at Runtime. In *Proceedings of the Business Process Management Workshops*. Springer, 327–338.
[13] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res* 15, 1 (2014), 3133–3181.
[14] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. 2014. Mining predictive process models out of low-level multidimensional logs. In *International conference on advanced information systems engineering*. Springer, 533–547.
[15] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.
[16] Mohamed F Ghalwash and Zoran Obradovic. 2012. Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC bioinformatics* 13, 1 (2012), 195.
[17] Mohamed F Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. 2013. Extraction of interpretable multivariate patterns for early diagnostics. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 201–210.
[18] Johny Ghattas, Pnina Soffer, and Mor Peleg. 2014. Improving business process decision making based on past experience. *Decision Support Systems* 59 (2014), 93–107.
[19] Guoliang He, Yong Duan, Rong Peng, Xiaoyuan Jing, Tieyun Qian, and Lingling Wang. 2015. Early classification on multivariate time series. *Neurocomputing* 149 (2015), 777–787.
[20] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.
[21] Geetika T Lakshmanan, Songyun Duan, Paul T Keyser, Francisco Curbera, and Rania Khalaf. 2010. Predictive analytics for semi-structured case oriented business processes. In *International Conference on Business Process Management*. Springer, 640–651.
[22] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. 2015. Complex symbolic sequence encodings for predictive monitoring of business processes. In *International Conference on Business Process Management*. Springer, 297–313.
[23] Yu-Feng Lin, Hsuan-Hsu Chen, Vincent S Tseng, and Jian Pei. 2015. Reliable early classification on multivariate time series with numerical and categorical attributes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 199–211.
[24] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. 2017. Detecting Sudden and Gradual Drifts in Business Processes from Execution Traces. *IEEE Trans. Knowl. Data Eng.* 29, 10 (2017), 2140–2154.
[25] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. 2014. Predictive monitoring of business processes. In *International Conference on Advanced Information Systems Engineering*. Springer, 457–472.
[26] Andreas Metzger, Rod Franklin, and Yagil Engel. 2012. Predictive Monitoring of Heterogeneous Service-Oriented Business Networks: The Transport and Logistics Case. In *2012 Annual SRII Global Conference*. IEEE Computer Society, 313–322.
[27] Andreas Metzger, Philipp Leitner, Dragan Ivanovic, Eric Schmieders, Rod Franklin, Manuel Carro, Schahram Dustdar, and Klaus Pohl. 2015. Comparing and Combining Predictive Business Process Monitoring Techniques. *IEEE Trans. Systems, Man, and Cybernetics: Systems* 45, 2 (2015), 276–290.
[28] Randal S Olson, William La Cava, Zairah Mustahsan, Akshay Varik, and Jason H Moore. 2017. Data-driven advice for applying machine learning to bioinformatics problems. *arXiv preprint arXiv:1708.05070* (2017).
[29] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, and Arthur H. M. ter Hofstede. 2017. Characterizing Drift from Event Streams of Business Processes. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 210–228.

[30] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn F. van Dongen. 2016. Detecting Drift from Event Streams of Unpredictable Business Processes. In *Proceedings of the International Conference on Conceptual Modeling (ER) (LNCS)*. Springer, 330–346.
[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[32] Amir Pnueli. 1977. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE, 46–57.
[33] Andreas Rogge-Solti and Mathias Weske. 2013. Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays. In *International Conference on Service-Oriented Computing (ICSOC)*. Springer, 389–403.
[34] Andrii Rozumnyi. 2017. *A Dashboard-based Predictive Process Monitoring Engine*. Master's thesis. University of Tartu.
[35] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* 42, 3 (2010), 10.
[36] Bernd Schwegmann, Martin Matzner, and Christian Janiesch. 2013. A Method and Tool for Predictive Event-Driven Process Analytics.. In *Wirtschaftsinformatik*. Citeseer, 46.
[37] Bernd Schwegmann, Martin Matzner, and Christian Janiesch. 2013. preCEP: facilitating predictive event-driven process analytics. In *International Conference on Design Science Research in Information Systems*. Springer, 448–455.
[38] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maria Maggi. 2017. Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In *International Conference on Business Process Management*. Springer, 306–323.
[39] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. 2017. Predictive Business Process Monitoring with LSTM Neural Networks. In *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 477–492.
[40] Irene Teinemaa, Marlon Dumas, Fabrizio Maria Maggi, and Chiara Di Francescomarino. 2016. Predictive Business Process Monitoring with Structured and Unstructured Data. In *International Conference on Business Process Management*. Springer, 401–417.
[41] Wil MP van der Aalst. 2016. *Process mining: data science in action*. Springer.
[42] W MP Van Der Aalst, Vladimir Rubin, H MW Verbeek, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. 2010. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling* 9, 1 (2010), 87–111.
[43] Sjoerd Van Der Spoel, Maurice Van Keulen, and Chintan Amrit. 2012. Process prediction in noisy data sets: a case study in a dutch hospital. In *International Symposium on Data-Driven Process Discovery and Analysis*. Springer, 60–83.
[44] Boudewijn F van Dongen, Ronald A Crooy, and Wil MP van der Aalst. 2008. Cycle time prediction: When will this case finally be finished?. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 319–336.
[45] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Chiara Di Francescomarino. 2015. Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In *International Conference on Business Process Management*. Springer, 218–229.
[46] Zhengzheng Xing and Jian Pei. 2010. Exploring Disease Association from the NHANES Data: Data Mining, Pattern Summarization, and Visual Analytics. *IJDWM* 6, 3 (2010), 11–27. https://doi.org/10.4018/jdwm.2010070102
[47] Zhengzheng Xing, Jian Pei, Guozhu Dong, and Philip S Yu. 2008. Mining sequence classifiers for early prediction. In *Proceedings of the 2008 SIAM international conference on data mining*. SIAM, 644–655.

APPENDIX

This Appendix reports the following:

- The distributions of case lengths in different outcome classes (Figures 15-16);
- The optimal number of clusters (Table 11) and the optimal number of neighbors for KNN approaches (Table 12) found for each classifier;
- The distributions of bucket sizes for the different bucketing methods (Figures 17-18);
- The overall AUC and F-score values for RF (Table 13), logit (Table 14), and SVM (Table 15);
- The AUC scores across prefix lengths using XGBoost classifier and all of the compared methods (Figures 19-20);

- The AUC scores across prefix lengths, including long traces only, using the XGBoost classifier (Figure 21);
- Concept drift in the *bpic2011_4* log. (Figure 22);
- Concept drift in the *sepsis_1* log. (Figure 23);
- The execution times for RF (Tables 16-17), logit(Tables 18-19), and SVM(Tables 20-21);
- The offline (Figure 24) and online (Figure 25) execution times and the overall AUC scores (Figure 26) when filtering the static categorical attribute domain, using the XGBoost classifier.

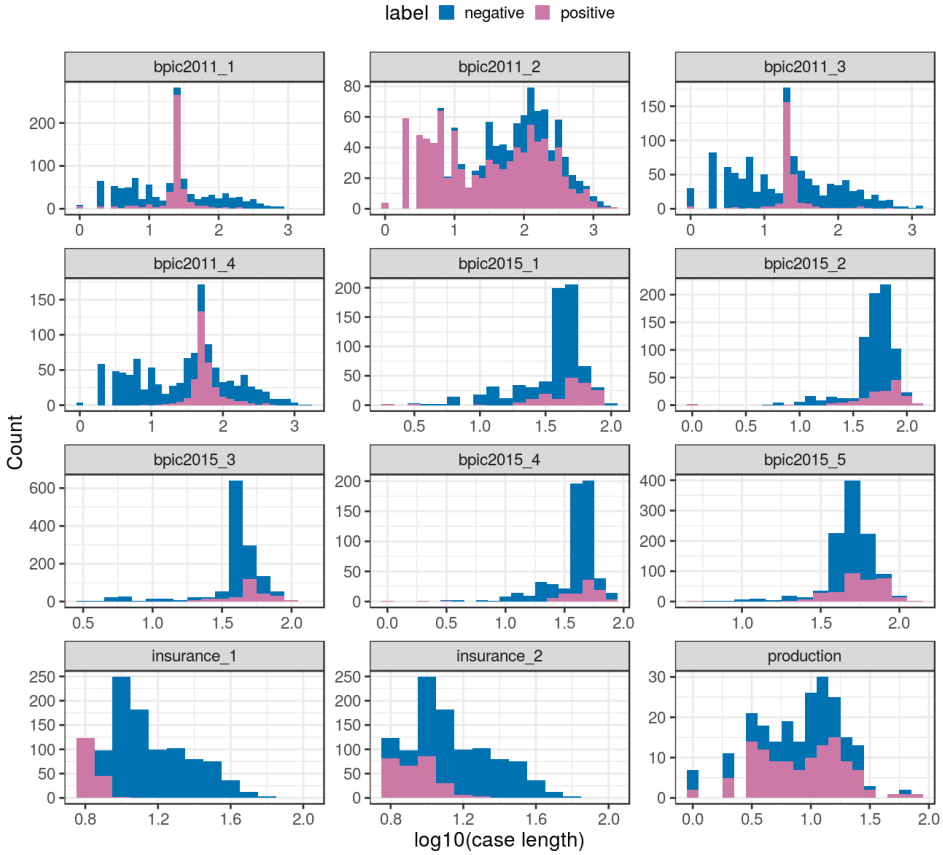


Fig. 15. Case length histograms for positive and negative classes

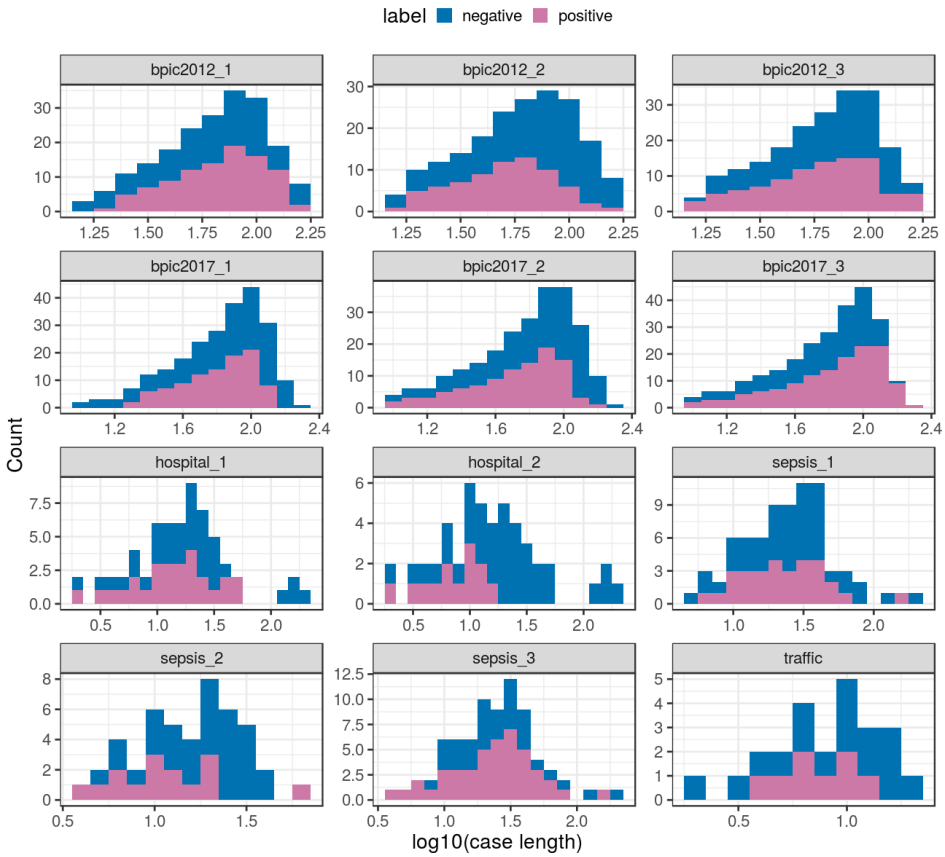


Fig. 16. Case length histograms for positive and negative classes (continued)

Table 11. Best number of clusters

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

Table 12. Best number of neighbors

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

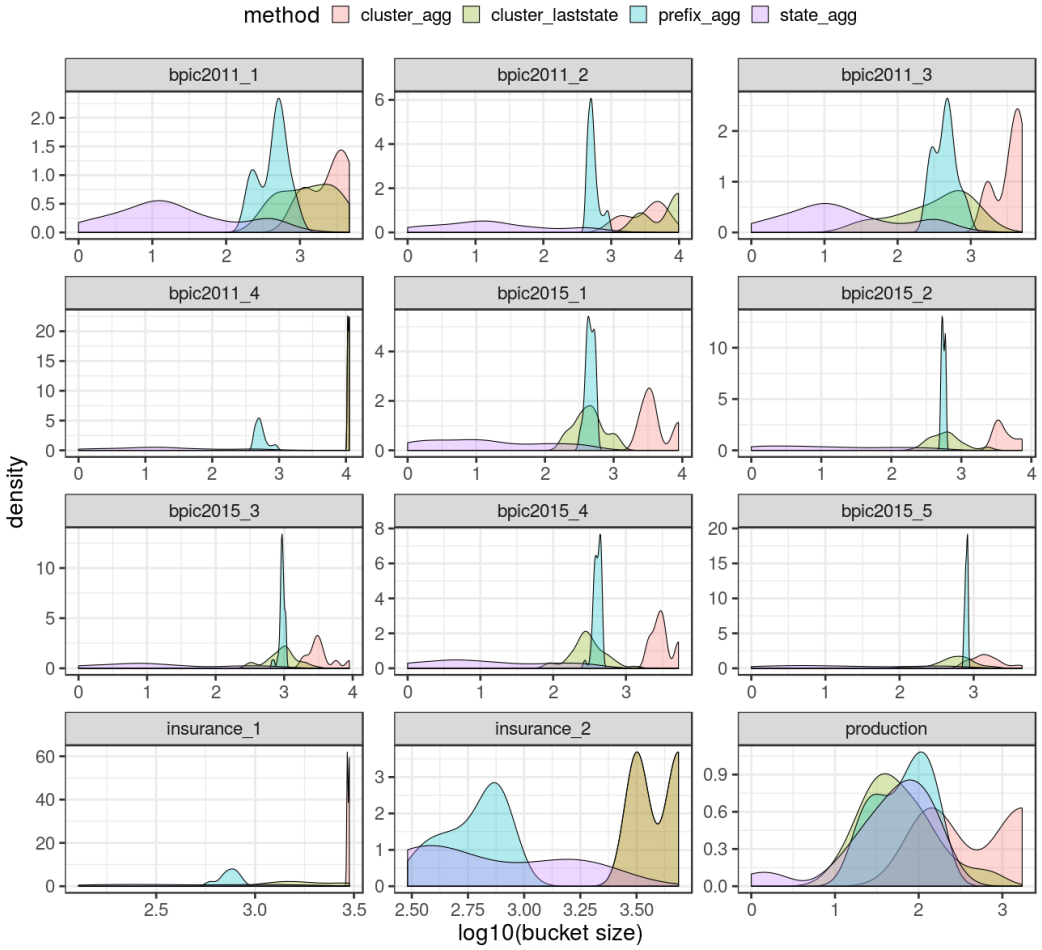


Fig. 17. Bucket size distributions

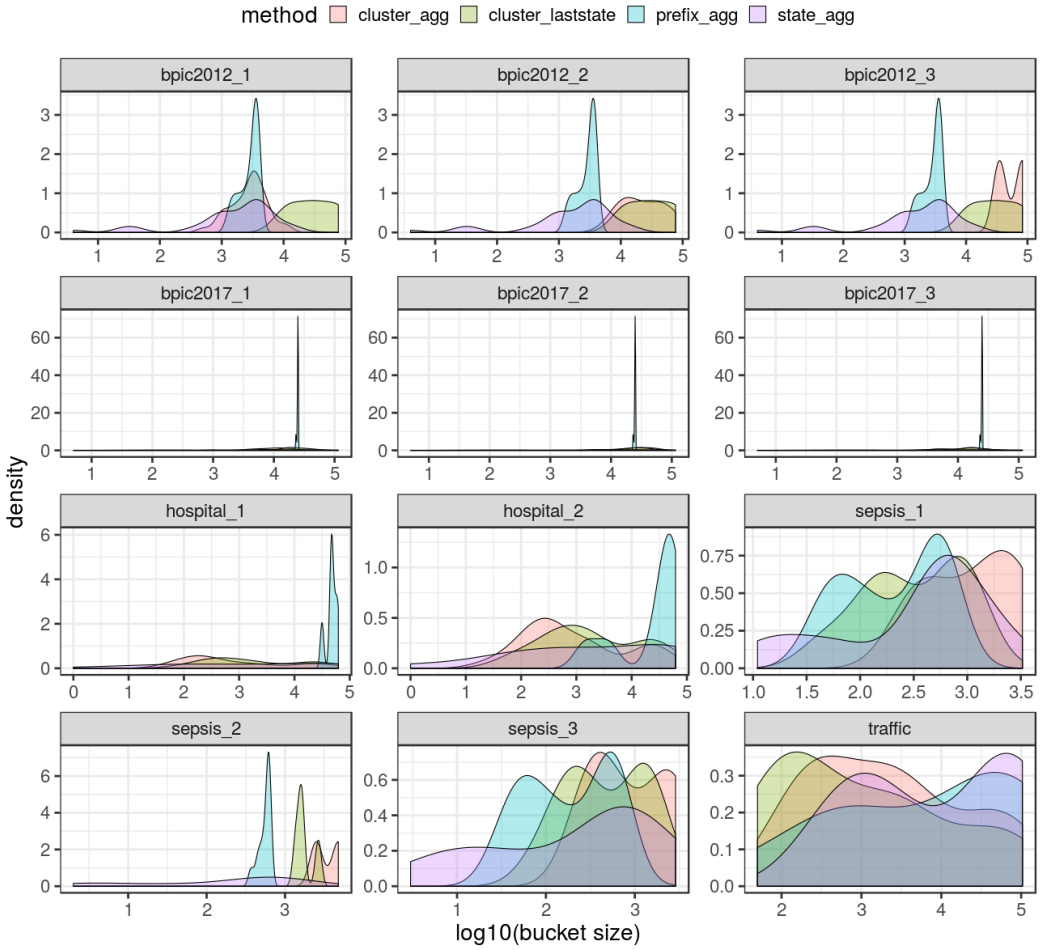


Fig. 18. Bucket size distributions (continued)

Table 13. Overall AUC (F-score) for **random forest**

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

Table 14. Overall AUC (F-score) for **logistic regression**

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

Table 15. Overall AUC (F-score) for **SVM**

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

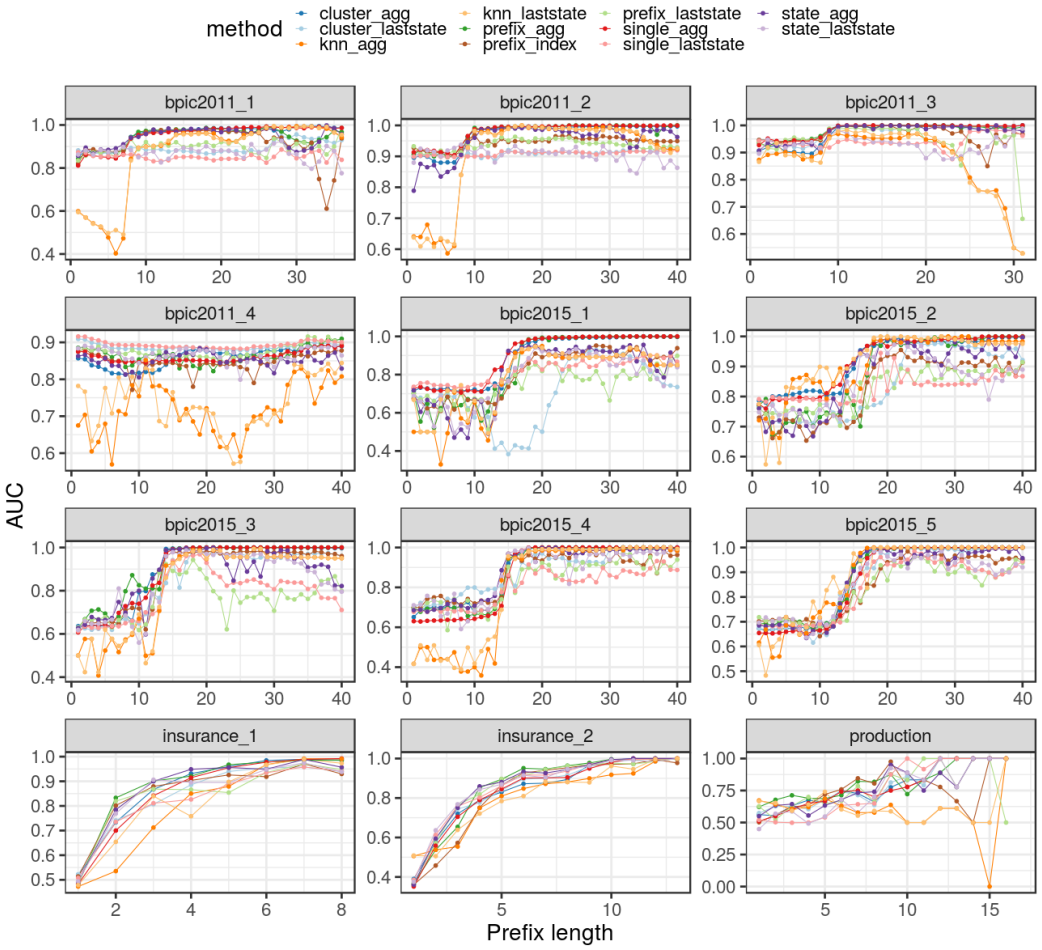


Fig. 19. AUC across prefix lengths using **XGBoost**, all methods

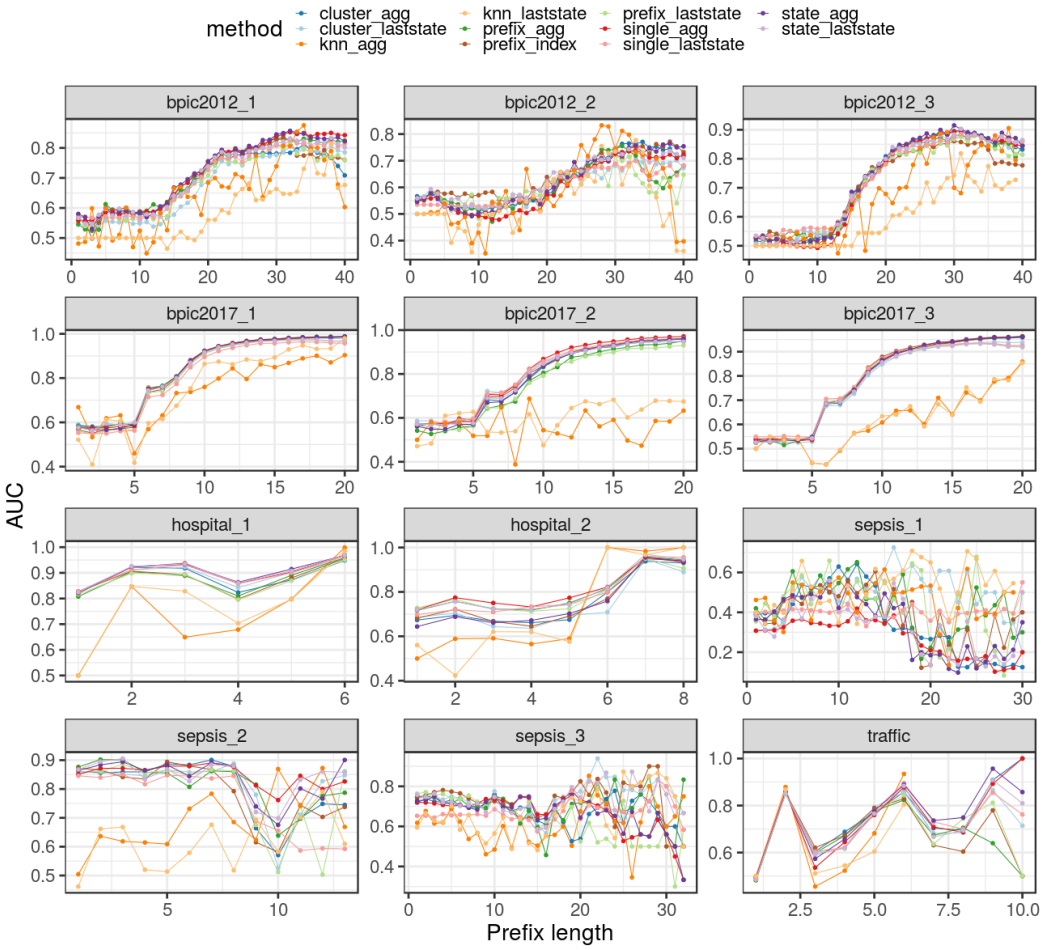


Fig. 20. AUC across prefix lengths using **XGBoost**, all methods (continued)

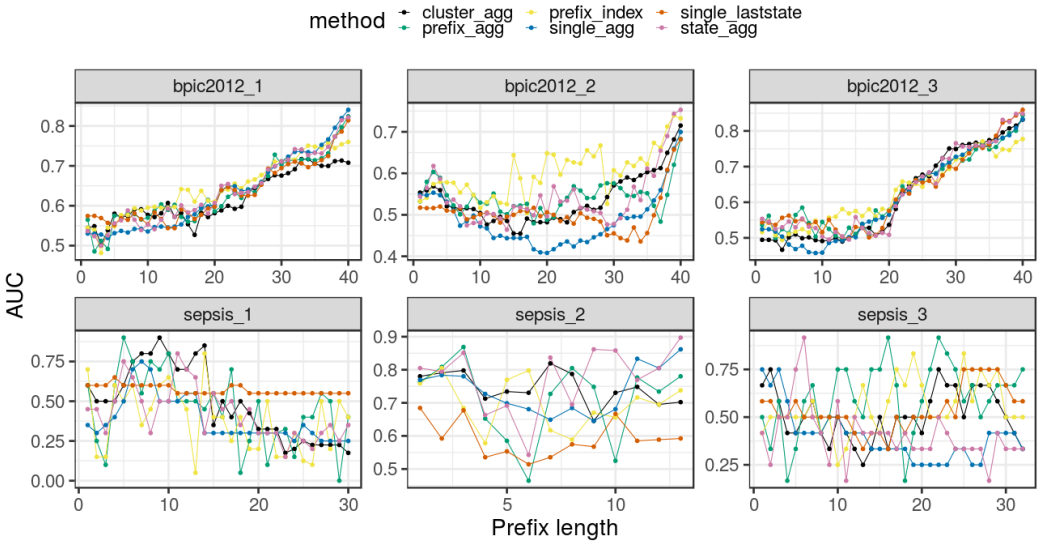
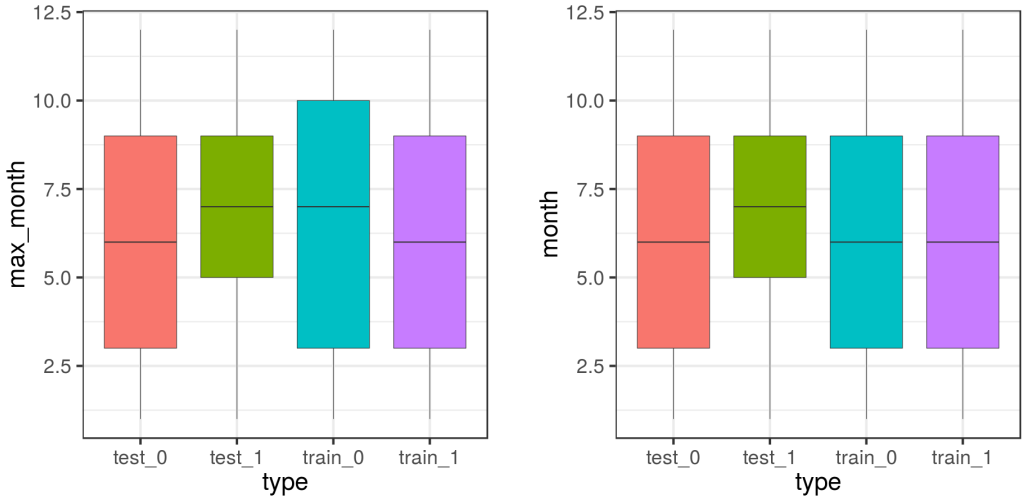


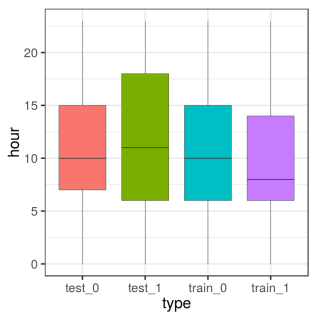
Fig. 21. AUC across prefix lengths using **XGBoost**, long traces only



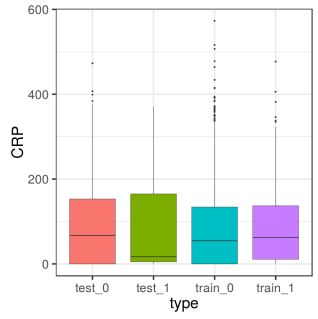
(a) Attribute = *max_month*. Significant differences in means between train_0–test_0 ($Z = 6.077$, $p < .001$) and train_1–test_1 ($Z = 7.972$, $p < .001$).

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

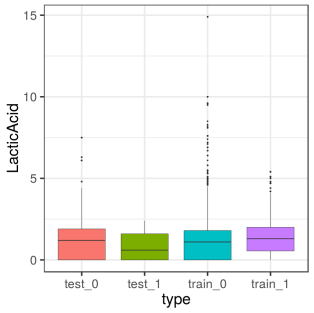
Fig. 22. Concept drift in the *bpic2011_4* log. The distributions of the variables are different across the two classes in the train and the test set. The drift becomes more evident in the *max_month* feature used by the aggregation encoding, while it is not so severe in the original *month* feature used by the last state encoding. Statistical significance of the differences is assessed using Wilcoxon signed-rank test.



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



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



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

Fig. 23. Concept drift in data attributes in *sepsis_1* log. The distributions of the variables are different across the two classes in the train and the test set. Statistical significance of the differences is assessed using Wilcoxon signed-rank test.

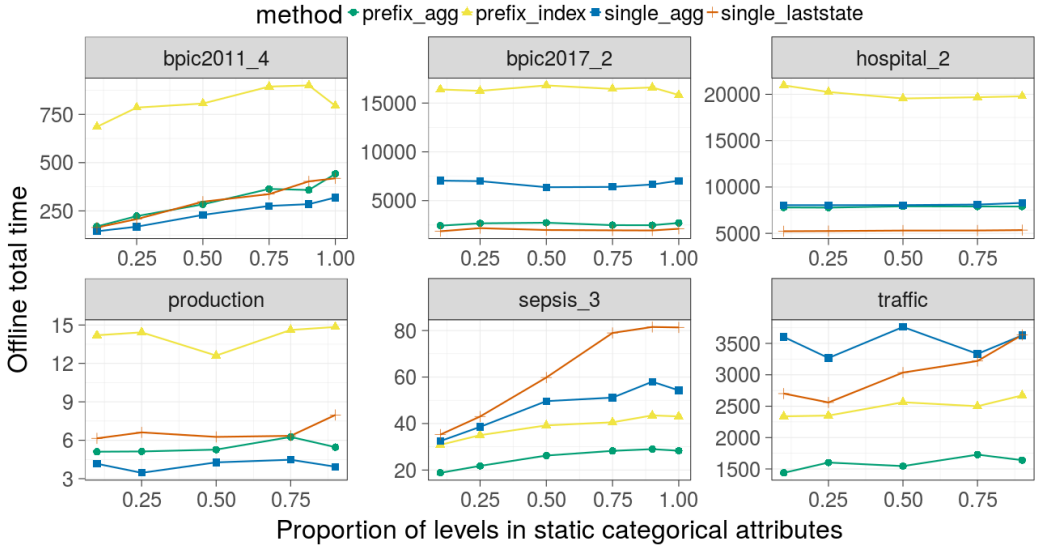


Fig. 24. Offline times across different filtering proportions of **static** categorical attribute levels (XGBoost)

Table 16. Execution times for **random forest**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	42.88 ± 0.78	76 ± 108	38.81 ± 0.27	67 ± 106	34.12 ± 0.17	78 ± 107
single_agg	51.92 ± 0.85	76 ± 109	369.75 ± 2.56	68 ± 107	57.62 ± 1.04	79 ± 109
knn_laststate	6.24 ± 0.24	168 ± 251	9.75 ± 0.44	143 ± 228	4.15 ± 0.09	169 ± 257
knn_agg	6.14 ± 0.12	176 ± 264	10.2 ± 0.35	151 ± 240	4.22 ± 0.22	190 ± 288
state_laststate	112.57 ± 0.39	**58 ± 80**	145.86 ± 0.49	**53 ± 82**	81.7 ± 0.19	**60 ± 79**
state_agg	265.71 ± 0.58	69 ± 95	226.93 ± 0.82	64 ± 100	153.58 ± 0.14	72 ± 95
cluster_laststate	35.88 ± 0.23	73 ± 122	73.3 ± 0.18	56 ± 105	42.61 ± 0.12	64 ± 114
cluster_agg	211.72 ± 0.86	76 ± 124	439.57 ± 0.68	73 ± 122	58.2 ± 0.1	80 ± 122
prefix_index	410.74 ± 5.03	126 ± 79	465.87 ± 10.13	127 ± 71	331.3 ± 12.36	121 ± 75
prefix_laststate	98.34 ± 0.22	66 ± 98	129.95 ± 0.5	57 ± 93	61.51 ± 0.07	69 ± 97
prefix_agg	173.76 ± 0.29	70 ± 101	189.64 ± 0.57	61 ± 95	122.13 ± 0.15	73 ± 100
	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	50.65 ± 0.21	68 ± 106	22.29 ± 0.54	26 ± 40	60.39 ± 0.53	24 ± 38
single_agg	135.39 ± 7.21	68 ± 108	102.42 ± 0.65	26 ± 41	86.86 ± 0.26	24 ± 40
knn_laststate	11.42 ± 2.19	151 ± 236	8.35 ± 0.48	126 ± 226	11.01 ± 0.16	116 ± 225
knn_agg	9.3 ± 0.2	152 ± 237	9.33 ± 0.13	138 ± 240	11.24 ± 0.54	115 ± 225
state_laststate	144.21 ± 4.36	**54 ± 83**	107.37 ± 0.18	31 ± 54	100.87 ± 0.27	33 ± 55
state_agg	227.05 ± 4.78	65 ± 100	135.72 ± 0.47	34 ± 57	132.69 ± 0.32	37 ± 58
cluster_laststate	428.27 ± 8.7	75 ± 121	40.05 ± 0.58	43 ± 69	55.96 ± 2.26	40 ± 66
cluster_agg	125.97 ± 0.29	**54 ± 87**	52.37 ± 0.83	30 ± 46	73.58 ± 2.67	32 ± 52
prefix_index	1121.44 ± 25.31	126 ± 71	504.84 ± 0.61	62 ± 12	1137.08 ± 2.07	60 ± 14
prefix_laststate	196.68 ± 0.02	57 ± 93	70.06 ± 0.24	**14 ± 19**	75.49 ± 0.13	**12 ± 16**
prefix_agg	306.88 ± 0.64	61 ± 95	97.72 ± 0.08	17 ± 21	159.75 ± 0.25	15 ± 18
	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	138.37 ± 2.95	26 ± 42	20.86 ± 0.6	24 ± 36	37.74 ± 0.9	22 ± 35
single_agg	95.39 ± 0.2	27 ± 43	70.51 ± 0.32	25 ± 38	105.67 ± 0.64	23 ± 36
knn_laststate	19.94 ± 0.64	113 ± 226	7.66 ± 0.02	120 ± 228	15.2 ± 0.33	115 ± 221
knn_agg	19.15 ± 0.68	124 ± 239	7.77 ± 0.35	127 ± 237	18.78 ± 0.24	122 ± 231
state_laststate	147.65 ± 0.1	35 ± 59	89.11 ± 0.16	31 ± 51	114.35 ± 0.4	29 ± 48
state_agg	192.76 ± 0.73	39 ± 62	93.86 ± 0.31	34 ± 54	157.16 ± 6.82	32 ± 52
cluster_laststate	121.76 ± 0.41	43 ± 73	41.47 ± 0.19	37 ± 72	72.07 ± 0.35	38 ± 66
cluster_agg	73.09 ± 0.4	31 ± 49	31.47 ± 0.27	35 ± 55	64.21 ± 0.58	31 ± 49
prefix_index	1531.4 ± 15.22	71 ± 17	176.56 ± 0.22	51 ± 10	366.0 ± 2.63	60 ± 13
prefix_laststate	100.27 ± 0.28	**13 ± 18**	62.28 ± 0.02	**12 ± 17**	99.77 ± 0.71	**12 ± 17**
prefix_agg	172.12 ± 4.21	16 ± 19	94.56 ± 0.16	15 ± 18	156.44 ± 3.28	14 ± 18
	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	2.99 ± 0.06	43 ± 37	12.51 ± 0.09	55 ± 44	9.65 ± 0.09	47 ± 42
single_agg	8.5 ± 0.05	47 ± 40	17.82 ± 0.12	57 ± 47	11.85 ± 0.08	48 ± 45
knn_laststate	0.89 ± 0.05	307 ± 338	0.61 ± 0.03	204 ± 206	0.89 ± 0.04	217 ± 173
knn_agg	1.04 ± 0.01	338 ± 333	0.65 ± 0.04	223 ± 228	0.94 ± 0.04	224 ± 178
state_laststate	17.34 ± 0.07	**41 ± 34**	14.36 ± 0.04	**48 ± 35**	16.63 ± 0.01	**40 ± 34**
state_agg	17.2 ± 0.21	47 ± 40	16.88 ± 0.12	57 ± 44	15.36 ± 0.18	49 ± 43
cluster_laststate	18.89 ± 0.26	45 ± 43	10.9 ± 0.1	50 ± 50	19.69 ± 0.06	48 ± 44
cluster_agg	15.18 ± 0.13	49 ± 47	24.89 ± 0.09	50 ± 49	31.68 ± 0.1	51 ± 48
prefix_index	24.93 ± 0.11	70 ± 27	27.69 ± 0.22	107 ± 10	28.91 ± 0.4	103 ± 11
prefix_laststate	18.32 ± 0.89	**41 ± 36**	15.16 ± 0.0	51 ± 37	22.93 ± 0.1	43 ± 37
prefix_agg	26.86 ± 0.08	57 ± 48	16.36 ± 0.03	**48 ± 35**	24.44 ± 0.38	**40 ± 33**

Table 17. Execution times for **random forest** (continued)

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

Table 18. Execution times for **logistic regression**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	8.14 ± 0.29	68 ± 97	11.09 ± 0.4	61 ± 96	5.55 ± 0.13	70 ± 96
single_agg	11.85 ± 0.39	69 ± 99	16.89 ± 0.17	62 ± 98	6.58 ± 0.05	71 ± 97
knn_laststate	5.69 ± 0.03	**29 ± 42**	9.86 ± 0.22	**23 ± 37**	4.12 ± 0.12	**29 ± 41**
knn_agg	5.93 ± 0.14	32 ± 46	9.69 ± 0.05	26 ± 42	4.28 ± 0.13	**29 ± 43**
state_laststate	8.64 ± 0.1	51 ± 70	11.97 ± 0.15	47 ± 73	6.57 ± 0.01	52 ± 68
state_agg	11.49 ± 0.05	66 ± 91	15.76 ± 0.59	57 ± 88	7.77 ± 0.01	61 ± 80
cluster_laststate	19.61 ± 0.1	56 ± 102	27.12 ± 0.79	53 ± 100	16.11 ± 0.07	55 ± 102
cluster_agg	19.1 ± 0.7	53 ± 76	27.15 ± 0.2	58 ± 109	13.86 ± 0.21	65 ± 107
prefix_index	28.51 ± 0.71	122 ± 71	44.66 ± 2.52	124 ± 65	18.37 ± 0.5	106 ± 62
prefix_laststate	7.52 ± 0.13	58 ± 87	10.34 ± 0.12	51 ± 83	5.32 ± 0.07	60 ± 85
prefix_agg	9.28 ± 0.11	61 ± 88	13.22 ± 0.14	54 ± 83	7.34 ± 0.08	64 ± 86
	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	11.13 ± 0.71	62 ± 97	7.97 ± 0.29	20 ± 30	7.04 ± 0.05	18 ± 29
single_agg	20.21 ± 2.74	62 ± 98	16.57 ± 0.41	22 ± 34	48.14 ± 0.66	20 ± 32
knn_laststate	8.55 ± 0.43	**26 ± 40**	8.98 ± 0.07	23 ± 37	10.59 ± 0.04	24 ± 39
knn_agg	8.56 ± 0.36	**26 ± 41**	8.16 ± 0.27	24 ± 39	11.9 ± 0.11	26 ± 45
state_laststate	11.71 ± 0.08	47 ± 72	8.52 ± 0.2	26 ± 45	9.65 ± 0.2	27 ± 46
state_agg	15.1 ± 0.04	55 ± 85	9.8 ± 0.36	29 ± 48	11.07 ± 0.08	31 ± 49
cluster_laststate	25.99 ± 0.08	54 ± 106	24.07 ± 0.07	22 ± 35	37.74 ± 0.67	36 ± 61
cluster_agg	26.49 ± 0.23	68 ± 112	19.51 ± 0.27	33 ± 52	27.18 ± 0.34	27 ± 45
prefix_index	41.41 ± 0.35	118 ± 60	27.99 ± 0.6	56 ± 12	38.47 ± 0.45	54 ± 15
prefix_laststate	10.81 ± 0.11	51 ± 82	7.09 ± 0.06	**7 ± 9**	7.67 ± 0.28	**6 ± 6**
prefix_agg	14.14 ± 0.08	54 ± 84	7.53 ± 0.03	8 ± 9	9.77 ± 0.07	7 ± 7
	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	38.41 ± 0.18	21 ± 33	6.57 ± 0.09	18 ± 27	18.67 ± 0.14	17 ± 26
single_agg	29.69 ± 0.72	21 ± 34	9.17 ± 0.24	18 ± 29	24.9 ± 0.24	17 ± 27
knn_laststate	19.6 ± 0.62	26 ± 42	7.6 ± 0.34	21 ± 35	14.0 ± 0.14	20 ± 33
knn_agg	18.5 ± 0.67	31 ± 49	7.76 ± 0.36	23 ± 38	14.56 ± 0.54	26 ± 42
state_laststate	14.56 ± 0.3	30 ± 50	7.17 ± 0.08	26 ± 42	12.4 ± 0.09	23 ± 39
state_agg	18.28 ± 0.07	33 ± 53	8.37 ± 0.09	29 ± 45	14.39 ± 0.3	26 ± 42
cluster_laststate	60.56 ± 1.47	36 ± 63	27.27 ± 0.18	34 ± 61	38.51 ± 0.45	18 ± 31
cluster_agg	44.96 ± 0.59	26 ± 41	26.35 ± 0.24	33 ± 60	48.48 ± 0.84	25 ± 40
prefix_index	65.17 ± 0.35	64 ± 19	25.3 ± 0.3	43 ± 8	46.49 ± 0.74	53 ± 13
prefix_laststate	12.02 ± 0.08	**7 ± 8**	6.37 ± 0.07	**6 ± 7**	11.19 ± 0.02	**6 ± 7**
prefix_agg	17.16 ± 0.1	8 ± 8	6.65 ± 0.04	7 ± 7	12.4 ± 0.13	7 ± 7
	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	0.89 ± 0.01	25 ± 21	1.41 ± 0.01	37 ± 30	2.88 ± 0.01	32 ± 29
single_agg	1.11 ± 0.13	28 ± 25	1.04 ± 0.05	39 ± 33	2.68 ± 0.09	34 ± 32
knn_laststate	0.87 ± 0.03	31 ± 28	0.62 ± 0.03	**20 ± 18**	0.92 ± 0.01	**24 ± 16**
knn_agg	0.93 ± 0.05	31 ± 31	0.63 ± 0.01	24 ± 21	0.89 ± 0.01	27 ± 18
state_laststate	1.47 ± 0.12	**23 ± 19**	1.29 ± 0.04	31 ± 22	2.43 ± 0.01	27 ± 21
state_agg	1.68 ± 0.06	30 ± 26	1.54 ± 0.03	43 ± 32	2.03 ± 0.02	37 ± 32
cluster_laststate	5.16 ± 0.08	28 ± 29	4.96 ± 0.08	32 ± 31	5.34 ± 0.13	34 ± 31
cluster_agg	5.72 ± 0.05	33 ± 34	5.83 ± 0.11	27 ± 28	8.5 ± 0.11	36 ± 34
prefix_index	3.03 ± 0.08	51 ± 10	2.89 ± 0.02	89 ± 5	3.86 ± 0.03	90 ± 4
prefix_laststate	1.15 ± 0.01	**23 ± 19**	1.26 ± 0.0	33 ± 23	2.33 ± 0.01	28 ± 22
prefix_agg	1.67 ± 0.0	34 ± 28	1.29 ± 0.01	32 ± 22	1.75 ± 0.01	27 ± 21

Table 19. Execution times for **logistic regression** (continued)

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	3.21 ± 0.15	27 ± 31	1.31 ± 0.03	**33 ± 33**	2.3 ± 0.04	29 ± 31
single_agg	21.44 ± 0.28	29 ± 33	1.48 ± 0.03	35 ± 35	2.54 ± 0.04	31 ± 33
knn_laststate	2.82 ± 0.06	28 ± 32	0.98 ± 0.04	35 ± 35	1.97 ± 0.05	31 ± 33
knn_agg	2.88 ± 0.11	31 ± 34	1.05 ± 0.06	40 ± 39	1.85 ± 0.06	34 ± 36
state_laststate	5.74 ± 0.17	29 ± 33	1.88 ± 0.03	36 ± 36	3.04 ± 0.02	31 ± 34
state_agg	22.54 ± 0.11	32 ± 36	2.1 ± 0.02	39 ± 39	3.18 ± 0.02	35 ± 37
cluster_laststate	9.51 ± 0.11	**26 ± 32**	5.52 ± 0.18	35 ± 34	6.52 ± 0.15	31 ± 34
cluster_agg	11.73 ± 0.04	27 ± 33	4.98 ± 0.09	38 ± 38	7.58 ± 0.03	30 ± 34
prefix_index	16.74 ± 0.07	39 ± 26	3.14 ± 0.02	45 ± 28	6.1 ± 0.02	40 ± 26
prefix_laststate	4.8 ± 0.07	27 ± 30	2.1 ± 0.07	**33 ± 32**	3.25 ± 0.03	**28 ± 30**
prefix_agg	8.01 ± 0.07	28 ± 32	2.27 ± 0.05	35 ± 34	3.64 ± 0.06	30 ± 32
	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	30.51 ± 0.38	7 ± 10	29.15 ± 0.79	7 ± 10	30.31 ± 1.76	7 ± 10
single_agg	31.1 ± 0.27	8 ± 12	60.4 ± 3.11	8 ± 12	74.28 ± 0.31	8 ± 12
knn_laststate	30.24 ± 0.38	55 ± 62	28.85 ± 0.08	27 ± 30	27.84 ± 0.83	74 ± 82
knn_agg	29.06 ± 0.46	350 ± 374	28.32 ± 0.08	310 ± 333	31.05 ± 2.0	192 ± 208
state_laststate	24.21 ± 0.29	8 ± 10	24.41 ± 0.1	8 ± 10	23.68 ± 0.37	8 ± 10
state_agg	29.84 ± 0.39	10 ± 12	28.8 ± 1.11	9 ± 11	30.25 ± 0.17	9 ± 12
cluster_laststate	38.2 ± 0.11	10 ± 15	37.28 ± 0.57	9 ± 13	39.4 ± 0.72	9 ± 13
cluster_agg	108.75 ± 1.51	10 ± 17	64.86 ± 2.16	9 ± 14	79.46 ± 6.91	9 ± 14
prefix_index	67.45 ± 2.49	36 ± 12	65.61 ± 1.1	34 ± 11	62.86 ± 1.03	36 ± 12
prefix_laststate	26.0 ± 0.87	**4 ± 3**	25.84 ± 0.39	**4 ± 3**	23.42 ± 0.3	**4 ± 3**
prefix_agg	38.74 ± 0.91	5 ± 5	42.84 ± 0.41	5 ± 5	47.5 ± 0.28	5 ± 5
	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	88.86 ± 1.74	19 ± 23	155.13 ± 3.04	19 ± 23	107.82 ± 2.12	18 ± 21
single_agg	213.25 ± 4.18	21 ± 26	143.2 ± 2.81	21 ± 27	209.74 ± 4.12	19 ± 23
knn_laststate	137.49 ± 2.7	1575 ± 1484	129.35 ± 2.54	1518 ± 1426	135.54 ± 2.66	1567 ± 1473
knn_agg	127.0 ± 2.49	1474 ± 1387	129.12 ± 2.53	1524 ± 1432	121.37 ± 2.38	1413 ± 1326
state_laststate	92.96 ± 1.82	19 ± 20	74.44 ± 1.46	18 ± 24	84.76 ± 1.66	17 ± 20
state_agg	472.57 ± 9.27	20 ± 22	244.76 ± 4.8	22 ± 24	114.88 ± 2.25	19 ± 22
cluster_laststate	154.87 ± 3.04	21 ± 25	172.13 ± 3.38	**15 ± 18**	104.51 ± 2.05	16 ± 19
cluster_agg	368.51 ± 7.23	19 ± 24	431.13 ± 8.46	17 ± 21	163.87 ± 3.22	16 ± 20
prefix_index	539.24 ± 10.58	72 ± 9	512.71 ± 10.06	72 ± 9	340.34 ± 6.68	72 ± 9
prefix_laststate	95.98 ± 0.95	**15 ± 16**	112.1 ± 5.26	**15 ± 16**	75.28 ± 0.25	**14 ± 15**
prefix_agg	281.04 ± 0.84	**15 ± 17**	551.86 ± 26.98	17 ± 20	325.93 ± 0.41	16 ± 17
	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	95.22 ± 0.54	**62 ± 34**	171.88 ± 3.37	381 ± 250	105.77 ± 2.08	395 ± 259
single_agg	273.55 ± 1.81	66 ± 37	361.41 ± 7.09	405 ± 267	269.08 ± 5.28	397 ± 262
knn_laststate	384.89 ± 5.26	75 ± 42	108.0 ± 0.13	**43 ± 24**	361.86 ± 7.1	**88 ± 64**
knn_agg	361.35 ± 8.72	81 ± 51	110.14 ± 1.45	78 ± 54	352.76 ± 6.92	94 ± 65
state_laststate	68.11 ± 0.53	66 ± 37	274.97 ± 5.4	349 ± 287	144.61 ± 2.84	334 ± 282
state_agg	151.29 ± 4.43	75 ± 42	274.89 ± 5.39	368 ± 230	308.15 ± 6.05	402 ± 249
cluster_laststate	93.99 ± 6.84	66 ± 39	225.33 ± 4.42	330 ± 284	124.98 ± 2.45	280 ± 244
cluster_agg	111.74 ± 2.5	68 ± 39	259.91 ± 5.1	386 ± 264	681.42 ± 13.37	369 ± 246
prefix_index	331.76 ± 1.14	89 ± 11	263.88 ± 5.18	964 ± 112	229.74 ± 4.51	848 ± 100
prefix_laststate	80.76 ± 0.95	**62 ± 34**	122.85 ± 1.04	363 ± 262	112.29 ± 0.47	313 ± 223
prefix_agg	186.94 ± 2.77	63 ± 35	166.22 ± 5.34	394 ± 232	173.75 ± 0.52	353 ± 204

Table 20. Execution times for **SVM**

	bpic2011_1		**bpic2011_2**		**bpic2011_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	503.86 ± 1.75	70 ± 100	457.58 ± 4.1	64 ± 100	173.03 ± 1.95	73 ± 99
single_agg	40.4 ± 0.35	69 ± 100	381.13 ± 0.35	63 ± 100	156.46 ± 0.44	73 ± 100
knn_laststate	6.0 ± 0.22	**29 ± 42**	9.79 ± 0.44	**27 ± 42**	4.26 ± 0.23	**29 ± 41**
knn_agg	5.91 ± 0.26	**29 ± 42**	9.98 ± 0.28	28 ± 44	4.25 ± 0.25	31 ± 44
state_laststate	12.44 ± 0.07	52 ± 72	19.77 ± 0.13	49 ± 75	9.52 ± 0.04	53 ± 70
state_agg	16.44 ± 0.08	61 ± 84	21.67 ± 0.04	58 ± 90	10.44 ± 0.08	63 ± 83
cluster_laststate	35.72 ± 0.22	62 ± 110	44.74 ± 0.14	45 ± 75	18.37 ± 0.28	48 ± 76
cluster_agg	30.12 ± 0.1	57 ± 99	46.53 ± 0.29	56 ± 106	21.49 ± 0.11	50 ± 74
prefix_index	39.94 ± 0.49	123 ± 72	64.98 ± 3.13	125 ± 65	25.33 ± 0.46	107 ± 62
prefix_laststate	16.31 ± 0.17	60 ± 90	23.17 ± 0.1	52 ± 84	11.99 ± 0.04	62 ± 87
prefix_agg	17.82 ± 0.08	62 ± 89	24.89 ± 0.04	54 ± 84	11.54 ± 0.1	65 ± 88
	bpic2011_4		**bpic2015_1**		**bpic2015_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	549.47 ± 23.94	65 ± 102	86.12 ± 0.31	21 ± 33	111.39 ± 0.15	20 ± 32
single_agg	130.1 ± 15.04	63 ± 99	68.34 ± 6.06	22 ± 34	88.97 ± 1.6	21 ± 33
knn_laststate	8.93 ± 0.43	**23 ± 35**	8.32 ± 0.43	23 ± 37	11.31 ± 0.37	23 ± 39
knn_agg	8.84 ± 0.41	26 ± 40	8.4 ± 0.33	21 ± 38	11.4 ± 0.34	18 ± 28
state_laststate	18.98 ± 0.03	48 ± 74	8.87 ± 0.11	26 ± 45	9.74 ± 0.07	28 ± 46
state_agg	25.72 ± 0.83	58 ± 89	10.33 ± 0.21	28 ± 47	11.49 ± 0.15	31 ± 48
cluster_laststate	46.62 ± 0.44	51 ± 103	22.02 ± 0.1	22 ± 34	29.91 ± 0.11	25 ± 41
cluster_agg	43.22 ± 0.12	49 ± 92	25.47 ± 0.13	25 ± 39	33.13 ± 0.22	25 ± 40
prefix_index	60.85 ± 0.68	119 ± 60	34.55 ± 0.4	52 ± 12	41.09 ± 0.26	52 ± 14
prefix_laststate	21.82 ± 0.04	52 ± 85	7.45 ± 0.05	**7 ± 9**	9.34 ± 0.1	**6 ± 6**
prefix_agg	22.69 ± 0.19	54 ± 85	9.16 ± 0.12	9 ± 10	10.71 ± 0.04	8 ± 7
	bpic2015_3		**bpic2015_4**		**bpic2015_5**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	363.33 ± 16.46	21 ± 32	45.01 ± 0.31	17 ± 27	567.21 ± 11.07	16 ± 25
single_agg	178.69 ± 12.55	21 ± 34	28.26 ± 0.59	18 ± 28	144.11 ± 1.39	17 ± 27
knn_laststate	18.82 ± 0.62	21 ± 37	7.6 ± 0.05	20 ± 35	14.34 ± 0.32	19 ± 32
knn_agg	19.6 ± 0.81	26 ± 45	7.07 ± 0.11	23 ± 38	14.37 ± 0.04	19 ± 34
state_laststate	15.56 ± 0.21	30 ± 50	7.3 ± 0.05	26 ± 43	13.02 ± 0.16	23 ± 39
state_agg	19.63 ± 0.09	34 ± 54	9.29 ± 0.07	29 ± 46	16.39 ± 0.11	27 ± 43
cluster_laststate	51.08 ± 0.33	25 ± 39	20.86 ± 0.1	21 ± 33	43.79 ± 0.08	21 ± 35
cluster_agg	52.3 ± 1.32	25 ± 40	19.61 ± 0.3	24 ± 37	140.02 ± 1.08	25 ± 41
prefix_index	77.94 ± 0.32	61 ± 18	27.1 ± 0.09	42 ± 8	66.81 ± 1.85	50 ± 12
prefix_laststate	16.14 ± 0.1	**7 ± 8**	6.4 ± 0.12	**6 ± 7**	13.36 ± 0.18	**6 ± 7**
prefix_agg	22.12 ± 0.07	9 ± 9	9.14 ± 0.03	10 ± 10	22.26 ± 0.07	10 ± 10
	production		**insurance_1**		**insurance_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	1.46 ± 0.05	26 ± 22	4.42 ± 0.05	38 ± 30	8.06 ± 0.05	33 ± 30
single_agg	1.48 ± 0.05	28 ± 24	3.63 ± 0.05	39 ± 33	7.16 ± 0.09	34 ± 32
knn_laststate	0.88 ± 0.05	26 ± 25	0.63 ± 0.02	**19 ± 17**	0.9 ± 0.02	**22 ± 15**
knn_agg	0.93 ± 0.05	33 ± 31	0.64 ± 0.01	22 ± 44	1.02 ± 0.07	27 ± 18
state_laststate	1.32 ± 0.06	**22 ± 19**	1.64 ± 0.03	31 ± 22	2.37 ± 0.01	27 ± 21
state_agg	1.67 ± 0.1	30 ± 26	1.78 ± 0.03	43 ± 33	3.01 ± 0.02	37 ± 32
cluster_laststate	4.84 ± 0.14	31 ± 29	5.48 ± 0.09	24 ± 25	6.56 ± 0.07	27 ± 26
cluster_agg	5.04 ± 0.13	37 ± 34	5.37 ± 0.1	39 ± 37	6.83 ± 0.03	30 ± 26
prefix_index	3.5 ± 0.01	61 ± 12	2.85 ± 0.03	89 ± 4	5.98 ± 0.04	89 ± 4
prefix_laststate	1.17 ± 0.01	23 ± 19	1.8 ± 0.0	33 ± 23	2.45 ± 0.01	28 ± 22
prefix_agg	1.56 ± 0.18	31 ± 26	1.55 ± 0.23	32 ± 23	2.12 ± 0.02	26 ± 21

Table 21. Execution times for **SVM** (continued)

	sepsis_1		**sepsis_2**		**sepsis_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	62.48 ± 0.54	28 ± 32	6.46 ± 0.02	34 ± 34	12.88 ± 0.09	30 ± 32
single_agg	15.52 ± 0.22	29 ± 33	5.81 ± 0.06	36 ± 36	14.88 ± 0.08	31 ± 34
knn_laststate	3.14 ± 0.07	29 ± 33	1.06 ± 0.01	38 ± 37	1.91 ± 0.06	31 ± 33
knn_agg	2.72 ± 0.12	30 ± 34	1.02 ± 0.02	37 ± 37	1.82 ± 0.03	33 ± 35
state_laststate	5.14 ± 0.09	29 ± 33	2.08 ± 0.02	35 ± 36	4.23 ± 0.02	31 ± 34
state_agg	9.5 ± 0.16	33 ± 37	2.61 ± 0.03	40 ± 40	5.08 ± 0.02	35 ± 38
cluster_laststate	10.03 ± 0.09	**26 ± 32**	6.02 ± 0.11	34 ± 34	7.7 ± 0.07	**29 ± 32**
cluster_agg	10.37 ± 0.09	28 ± 34	6.84 ± 0.03	**33 ± 35**	7.95 ± 0.06	31 ± 34
prefix_index	10.45 ± 0.03	36 ± 24	3.16 ± 0.01	41 ± 26	6.67 ± 0.05	38 ± 25
prefix_laststate	6.63 ± 0.02	28 ± 31	2.49 ± 0.03	34 ± 33	4.78 ± 0.04	30 ± 32
prefix_agg	6.61 ± 0.09	30 ± 33	2.85 ± 0.04	36 ± 35	4.96 ± 0.02	31 ± 34
	bpic2012_1		**bpic2012_2**		**bpic2012_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	12067.81 ± 399.76	10 ± 14	14557.12 ± 780.58	8 ± 11	2746.74 ± 133.03	9 ± 13
single_agg	3143.14 ± 24.75	10 ± 16	1336.26 ± 5.35	9 ± 14	3940.96 ± 508.12	10 ± 15
knn_laststate	28.92 ± 0.47	279 ± 299	31.47 ± 0.55	254 ± 274	30.7 ± 2.47	174 ± 188
knn_agg	33.21 ± 1.1	258 ± 277	29.0 ± 0.93	297 ± 319	31.11 ± 0.58	171 ± 189
state_laststate	108.84 ± 0.29	8 ± 10	156.77 ± 2.77	8 ± 10	128.66 ± 0.22	8 ± 10
state_agg	168.48 ± 1.15	10 ± 12	88.35 ± 1.29	10 ± 12	191.19 ± 3.22	10 ± 13
cluster_laststate	414.04 ± 3.83	9 ± 14	139.3 ± 0.17	8 ± 13	181.63 ± 2.76	8 ± 12
cluster_agg	186.63 ± 2.21	8 ± 9	2409.21 ± 27.71	11 ± 18	273.16 ± 6.91	10 ± 15
prefix_index	538.77 ± 6.5	37 ± 12	420.91 ± 13.0	34 ± 11	344.04 ± 8.18	34 ± 11
prefix_laststate	77.87 ± 0.94	**4 ± 3**	211.0 ± 9.02	**4 ± 3**	100.44 ± 2.24	**5 ± 4**
prefix_agg	100.43 ± 0.25	5 ± 5	288.14 ± 4.56	5 ± 5	66.23 ± 0.75	**5 ± 4**
	bpic2017_1		**bpic2017_2**		**bpic2017_3**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	86469.6 ± 1696.97	38 ± 46	91457.8 ± 1923.76	42 ± 56	64770.83 ± 1271.13	32 ± 39
single_agg	89619.4 ± 1736.87	40 ± 47	23877.21 ± 468.59	30 ± 38	68407.03 ± 1323.42	35 ± 42
knn_laststate	117.67 ± 2.31	118 ± 108	133.43 ± 2.62	1515 ± 1424	119.49 ± 2.35	1375 ± 1292
knn_agg	122.36 ± 2.4	1408 ± 1332	118.44 ± 2.32	1412 ± 1329	139.45 ± 2.74	1587 ± 1497
state_laststate	18343.13 ± 359.98	23 ± 26	1905.91 ± 37.4	20 ± 24	37822.2 ± 742.26	17 ± 23
state_agg	10257.06 ± 201.29	25 ± 28	58203.39 ± 1142.24	23 ± 26	10121.8 ± 198.64	24 ± 26
cluster_laststate	8744.81 ± 171.62	17 ± 22	7006.73 ± 137.51	17 ± 21	3026.05 ± 59.39	18 ± 22
cluster_agg	74691.13 ± 1465.81	22 ± 27	8903.74 ± 174.74	23 ± 28	5264.25 ± 103.31	18 ± 21
prefix_index	17933.49 ± 351.94	76 ± 9	39670.6 ± 778.54	87 ± 10	24417.21 ± 479.19	89 ± 11
prefix_laststate	16596.88 ± 1364.46	**16 ± 17**	3553.0 ± 3.84	**14 ± 15**	7667.05 ± 53.16	**15 ± 16**
prefix_agg	21941.34 ± 37.13	**16 ± 18**	5395.36 ± 587.31	16 ± 18	4509.12 ± 372.24	18 ± 20
	traffic		**hospital_1**		**hospital_2**	
method	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)	offline_total (s)	online_avg (ms)
single_laststate	52218.27 ± 1024.78	103 ± 57	163070.85 ± 3200.27	487 ± 319	100195.84 ± 1966.34	482 ± 311
single_agg	58867.16 ± 1155.27	114 ± 62	65398.88 ± 1283.45	436 ± 281	291765.77 ± 5725.9	545 ± 353
knn_laststate	394.55 ± 7.74	69 ± 40	109.75 ± 1.32	81 ± 55	312.82 ± 6.14	**62 ± 49**
knn_agg	482.25 ± 9.46	96 ± 54	102.69 ± 2.02	**42 ± 23**	321.04 ± 6.3	85 ± 68
state_laststate	31579.01 ± 1040.68	78 ± 48	33549.22 ± 658.4	320 ± 268	110178.73 ± 2162.26	365 ± 307
state_agg	71510.15 ± 1403.39	87 ± 53	31458.64 ± 617.38	359 ± 222	18274.1 ± 358.63	385 ± 236
cluster_laststate	7466.95 ± 149.95	72 ± 46	23090.77 ± 453.16	313 ± 292	19699.97 ± 386.61	285 ± 255
cluster_agg	97876.03 ± 1920.82	**45 ± 40**	46246.86 ± 907.59	324 ± 239	22553.25 ± 442.61	325 ± 242
prefix_index	28902.06 ± 145.18	125 ± 20	53322.05 ± 1046.45	1093 ± 132	82296.52 ± 1615.07	1013 ± 214
prefix_laststate	12316.56 ± 14.1	72 ± 43	39203.83 ± 1030.35	360 ± 255	76378.26 ± 118.46	344 ± 244
prefix_agg	24735.14 ± 425.56	75 ± 45	52526.59 ± 857.39	393 ± 234	57426.99 ± 22.74	364 ± 215

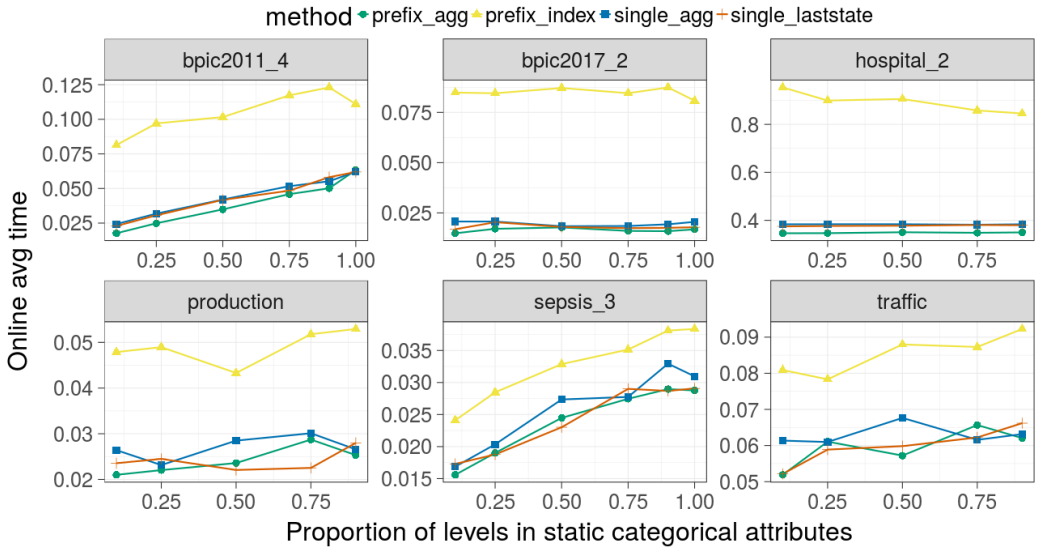


Fig. 25. Online times across different filtering proportions of **static** categorical attribute levels (XGBoost)

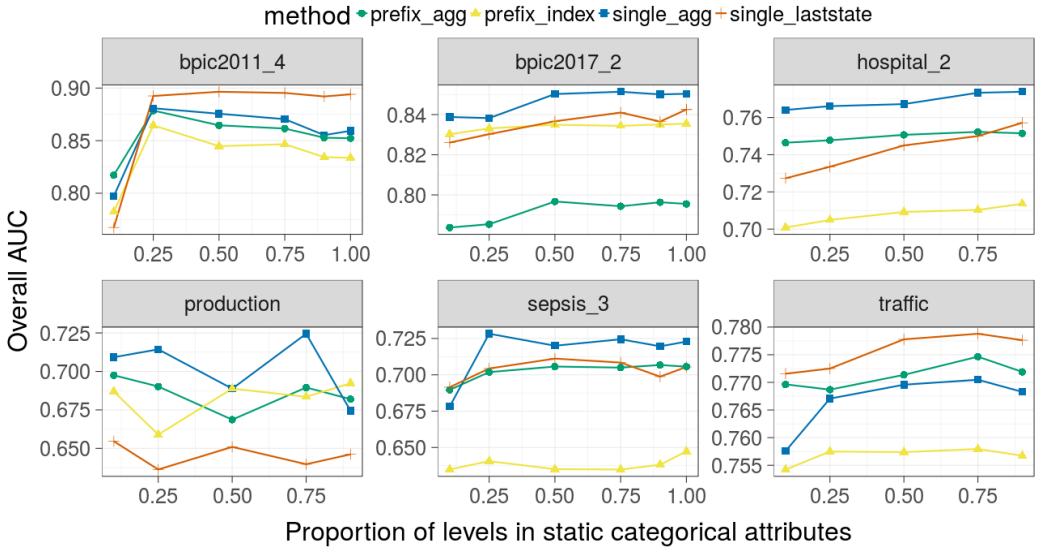


Fig. 26. AUC across different filtering proportions of **static** categorical attribute levels (XGBoost)