








# Towards best practices for online predictive process monitoring

Stijn Kas<sup>1</sup> , Ivar Siccama<sup>2</sup> , Peter van der Putten<sup>2</sup> , Georg Kremp<sup>1</sup>   
, Ruben Post<sup>1</sup> , Sebastiaan Wiewel<sup>1</sup> , and Hajo Reijers<sup>1</sup> 

<sup>1</sup> Utrecht University, Utrecht, the Netherlands

<sup>2</sup> Pegasystems, Amsterdam, The Netherlands

**Abstract.** State-of-the-art research on predictive process monitoring (PPM) is predominantly based on 'offline' models, which are trained ahead of time and can be used during the run-time of a process. In reality, process data is streaming data that arrives sequentially over time. For practical implementation this means that the data needs to be collected over a longer time period before a model can be used, which won't be able to adapt to newer information. To this end, this work describes a pipeline approach applicable on streaming data, integrating various components from traditional PPM to facilitate online learning: traces are encoded and labeled in real-time to create a generic end-to-end pipeline allowing for experimentation with different configurations. Experimental results with this pipeline approach show a need for proper encoder and algorithm selection which is supported by this pipeline.

**Keywords:** predictive process monitoring · online learning · stream analytics · concept drift

## 1 Introduction

Predictive process monitoring (PPM) aims to predict the final business outcome of a process while the process is still ongoing [24]. A business outcome, the main deliverable of a process, delivers value for the end-customer. Improving a business outcome can lead to shortening the process or improving the quality of service; but can also bring value to the business itself through improved *efficiency*, *effectiveness* or *flexibility* of the process [14]. By predicting these outcomes in real-time, action can be taken during individual process execution, influencing the process execution to ultimately *optimize* the outcome. For this optimization task various different approaches have been described under the term *prescriptive* process monitoring, such as alarms, interventions, or next best action recommendations [26, 29].

The vast majority of process monitoring approaches use historical event logs to first train a machine learning model, then re-use this model to predict the outcome for a new ongoing case. These are typically defined as two separate phases, one for *offline* training and one for *online* deployment [18, 24]. However,

if the underlying data distribution changes over time, the model might perform poorly due to the differences between data at training and deployment time. These distribution changes are known as *concept drift*, which have important particularities in for business processes [4]. While traditionally concept drift is seen as external factors influencing the data distribution (COVID-19 being a particularly influential example), processes are also subject to change in the use of the process itself by its users, such as an increase of certain behavior or a change in desired business outcomes. Additionally, business processes naturally evolve and degrade in effectiveness and efficiency over time in a concept known as *entropy* [7, 8]. From this perspective, it is desirable to continuously improve a process over time, which in turn changes the data distribution at each change. Lastly, a self-reinforcing effect can be found when more algorithms are used throughout a process creating an even more dynamic environment. If a model cannot adapt to these changes, any change in process design immediately decreases the performance and usefulness of a model. From this perspective it is naive to assume a steady-state process throughout its lifetime, and it would seem worthwhile to instead adapt the application of predictive process monitoring to an ever-evolving business process. For this, we look at the domain of *online learning*.

In an online learning or stream analytics setting, a machine learning model is deployed which can be continuously adapted, allowing the model to remain up-to-date, removing the need to develop any models up front before making use of predictions. In the case of (*instance-*)*incremental* learning, each individual new data instance is used to update the weights of the model. Thus far, [8] and [17] have explored the use of online learning in the context of predictive process monitoring in experimental settings. These works focused on exploring the effect of concept drift on specific classifiers within a business process, using synthetic logs and two datasets from the BPI Challenge [28]. While these approaches demonstrate the possibility of using incremental models to counteract the effects of concept drift on predictive process monitoring, they fail to give detailed insight into practical challenges and possible implementation, rather focusing on evaluation of a single approach. Thus, this work discusses various approaches for implementing online predictive process monitoring, and presents experimental results when applying them to a real-life dataset.

The remainder of this paper is structured as follows: In section 2, background and related work is discussed for input into the setup, which is described in section 3. With this setup, the results are achieved which are described in section 4, and discussed in section 5. Lastly, section 6 concludes and summarizes the entire work.

## 2 Background

This section covers three main notions: the domain of predictive process monitoring from a BPM and process mining point of view, the components of a PPM pipeline approach, and the domain of online learning and streaming analytics.

**Table 1.** Encoding method, adapted from on [22]

Encoding	Trace abstraction	Numeric	Categorical
Last State	Last event or attribute	As is	One-hot
Aggregation	All events, unordered	Min, max, mean, sum, std	Frequencies or occurrences
Index	All events, ordered	As is for each index	One-hot for each index

## 2.1 Business processes

As part of any business process, various activities are performed in a specific order to come to some deliverable for an end-customer. In other words, a process leads to some *business outcome*, which a process is designed around. While these activities are performed, information systems can be used to support the activities and, as a byproduct, record data about these activities and the data associated with each activity, collectively referred to as an *event*. In this context, we adhere to the definition of an event as a tuple  $(a, c, t, (d_1), \dots, (d_m))$  consisting of an activity  $a$ , a case id  $c$ , a timestamp  $t$  and  $\geq 0$  other data attributes  $d$ . Case id  $c$  denotes a single instance of a process, such as the request of a single customer within a process. This instance, or case, is uniquely identified by its case id. When following the events corresponding to just a single case through the data points collected by the information system, we find a *trace* of this case. More formally, if the universe of all events is denoted by  $\varepsilon$ , a *trace* is a non-empty sequence  $\sigma = [e_1, \dots, e_n]$  of events such that  $\forall i \in [1..n], e_i \in \varepsilon$  and  $\forall i, j \in [1..n] e_i.c = e_j.c$  [24]. In other words, all events in a trace refer to the same case. Various traces can be combined and collected in an *event log*. When data arrives in real-time, a trace can be *incomplete*: not all data has arrived, only the first  $k$  activities. Then, the trace has a *prefix length* of  $k$ .

By nature of an event log, a single row represents a single event within a trace. This preserves the sequential nature of information arriving when activities are performed and allows for sparse data storage as data attributes only need to be stored for their corresponding activity within a case [13]. This allows for a wide range of analyses since it is a lossless form of data storage, but also means that a single row does not include information about previous events in the trace. When the object of interest is not the individual events (like it may be in process mining) but instead the trace (and thus the case) as a whole, the sparseness of the format means that a row does not include much information about previous events in the trace. Thus, to analyze an entire case at once and not just the events within them, there is a need to *encode* information about previous events and the attributes within the trace as features. Note that there is a slight mismatch in terminology between data mining literature and PPM literature: the concept of *encoding* is known in the data mining literature as *aggregation*. Additionally, we make a distinction between *attributes* of a trace and *features* which are encoded attributes. Table 1 describes the main encoding methods used in state-of-the-art literature on PPM.

The *last state* encoder encodes the last known value for each attribute (which is not missing). With the sparse nature of an event log this is not the same as the current value for each feature: a value is only stored once for an activity, thus without an encoder a lot of missing values would be found. An *aggregation* encoder applies a statistical function to a numeric value, such as the min, max or mean of the entire attribute over all activities within the case. The *index* encoder incorporates the ordering of an attribute, and generates a feature for each  $n$ th value for a certain attribute within that case.

## 2.2 Online learning

Predictive process monitoring traditionally analyzes historical event logs to simulate real-time predictions by training a model on a train set from an event log and then predicting on traces from a test set. In this context, PPM is algorithm-agnostic and all machine learning models can be used [22]. Recent works have also explored use of AutoML to automatically do model selection and hyperparameter tuning on process datasets [15]. However, when applying PPM in a practical setting, the data is not an historical event log but instead a data stream with continuously incoming attributes. Such streaming data might exhibit concept drift over time, resulting in an outdated model. To handle this change, we look at the domain of stream analytics or *online learning*, where a model is continuously *updated*.

Incrementally learning algorithms, or *adaptive models*, were designed to handle data that was too large to fit into memory by showing the data to the model in batches [2]. This leads to one of the properties of adaptive models: they only need a 'single pass' of the data to learn, so once the model has seen the data there is no need to store it for learning purposes. Literature distinguishes five categories of incremental online learning classifiers: *frequency-based*, *tree-based*, *ensemble-based*, *neighborhood-based*, and *neural network-based* [1–3, 5, 10]. Naive Bayes, the most common frequency-based algorithm, knows the *independence assumption*, assuming all features are independent of each other given the class, though in practice it can still perform fairly well even if this assumption does not hold [21]. Naive Bayes can be used in tree-based algorithms like Hoeffding Trees as well by having Naive Bayes leaves predict the probabilities for predictions within a leaf node.

## 2.3 Delayed learning

In a process, activities are executed one after the other until the case is closed. In predictive process monitoring, an *outcome* can be chosen to predict with machine learning, and before the outcome is known a prediction of that outcome can be made at each activity. However, once a prediction is made the label is not known instantly, rather once the case outcome is known. This problem is known as *delayed labeling* or verification latency, and is an emerging research subject [10–12, 16]. In particular, recent works focus on how to measure and evaluate the performance of predictors in a delayed labeling setting since, for

instance, predictions early on in a sequence are more valuable than predictions later on in a sequence, which should be reflected in evaluation metrics [11]. This idea of *earliness* has also been included in PPM literature in the context of *temporal stability*, where two metrics were seen as important: how *early* a prediction is made within a case and how *stable* a prediction remains within that case [23].

In data stream mining, the delay might not necessarily be known a priori, and might even be infinite [9,16]. A related important characteristic of predictive process monitoring applications is that the moment a case’s outcome gets known might differ between processes and even between instances of the same process. In the ‘offline’ learning PPM benchmark study by [22], this is resolved after the fact by computing the label for each case and adding that label to each event in the trace. Thereby, the classifier is trained on each individual set of predictors before the outcome of a case is known. In other words, at each point where the model makes a prediction, those same predictors are used to train the model. In a streaming setting this naturally does not work quite as easily since the label is not yet available for each event.

## 2.4 Related work

Not much research has looked into predictive process monitoring in an online learning setting. This section aims to discuss the three works that have, their conclusions and shortcomings.

First experiments with incremental techniques in PPM were made in [17]. Therein, the effect of concept drift in synthetic event logs is evaluated in experiments with three different algorithms: a simple Hoeffding Tree, Naive Bayes and a perceptron-based classifier. Major differences between these classifiers were found, with the perceptron performing worst overall, and Hoeffding Trees able to adapt to concept drift more quickly. They experimented with different encoders, focusing on different aspects of the data: the control flow and the order of their activities or the data aspect which corresponds to the case attributes. Their evaluation was solely based on synthetic datasets. They used temporal performance metrics (mean accuracy) of the classifier improving over time, but did not evaluate the performance of the classifier at different points within a case.

This was extended in [8] by experiments on real and synthetic datasets. Therein, a clustering-based update logic is used, by adapting the concept of *bucketing*: using different models for different *kinds* of traces. One such implementation is to cluster the traces based on their attributes and use different classifiers for each cluster. They also experimented on two real datasets apart from the synthetic logs. They did not use incremental models but rather batch-updates: the model was updated three times during the overall event log duration. They used the F1 score and accuracy as measures, also not evaluating the performance at different points within a case. They only compared a Hoeffding Tree and an Adaptive Hoeffding Tree as the used classifiers.

The most recent contribution towards online PPM is [20], using LSTM neural network models to predict the next activity. Since the prediction target for

next activity prediction and the pre-processing and encoding methods for neural networks are much different from the general PPM case, their results are not further evaluated.

## 2.5 Contribution

This work aims to contribute to online predictive process monitoring by evaluating the effectiveness of a general online PPM framework compared to the state-of-the-art PPM benchmark study by [22]. We define a conceptual model of a general PPM pipeline and its different components such as encoding, algorithms and train set generation, and the interaction thereof. Lastly, we aim to evaluate the results from an experimental setup with different configurations which is made available on GitHub.

## 3 Illustration

Figure 1 shows a basic timeline of steps in a streaming PPM framework over time. Here, an activity is denoted  $a$ , timestamps are denoted  $t$  and the encoded trace of activity  $a$  and its data attributes is denoted  $E(a)$ . As events are executed, an outcome within the case can be predicted for each event. For this, the current event trace is encoded after which the model is able to make a prediction.

After a prediction it generally takes an additional number of steps until the outcome is known. During these steps, initially missing values can be filled and attributes updated. Intuitively, the closer to the outcome, the better the prediction will be as there is more input data available, but the sooner the prediction the more useful. In that sense, there is a trade-off between the earliness of a prediction versus the quality, as described in more detail in [23]. Once the desired outcome is known, in fig. 1 at activity  $a_6$ , the model is trained using the encoded trace of the case thus far,  $E(a_5)$ . Once the outcome is known and the model is trained, there is no need to predict or train since the known outcome can be retrieved. This generic definition of the outcome allows for intermediate outcomes, which appear somewhere in the case, or final outcomes, which denote the end of a case. Also, by rolling back in the trace, the state at previous instances of the case can be reconstructed, which can be used to augment the training set with additional instances and make it more representative (see fig. 5). For example, less attribute values may be known at earlier stages, or certain value may always be overwritten after a certain stage.

In the context of online PPM, there are two general actions with a new event: either the model predicts the cases' outcome, or the outcome is known and the model is trained to include the now known outcome. Additionally, if the model has already been trained on a trace and thus the outcome is already known, no actions need to be taken. This could be instantiated using, for example, an API call, where the returning value would always be an outcome and the model is trained in the background if possible. This concept is visualized in fig. 2 from the view of listening for new events to be recorded from an event stream.

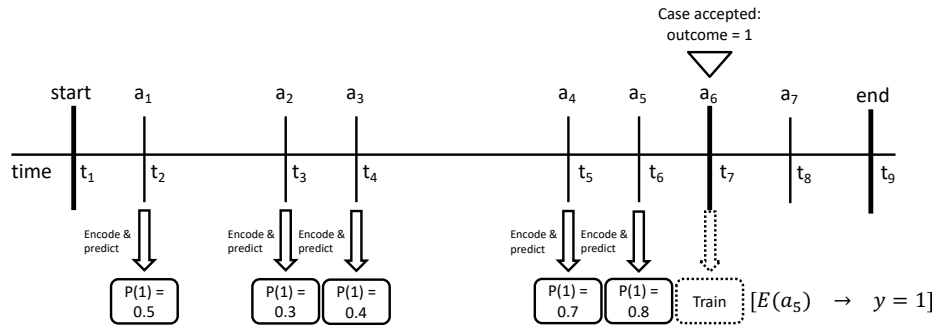


Fig. 1. PPM timeline

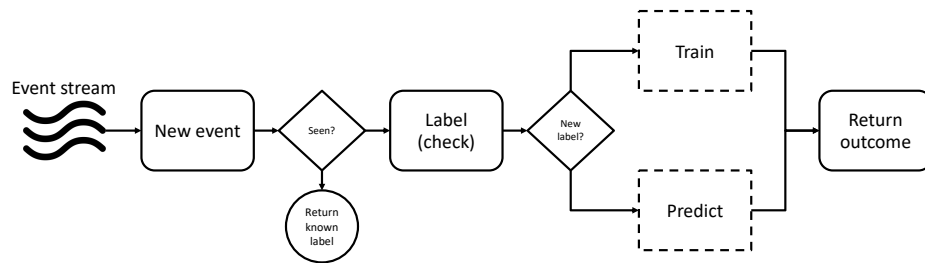


Fig. 2. Event stream

To train the model, a sequence of steps visualized in fig. 3 is executed. First, the label is computed using some labeling function. Next, the encoding of the current trace is retrieved. To avoid leaking variables, these encodings are not updated but just retrieved from memory. With the encoded trace retrieved, the model can be trained after which the case id can be added to a list of 'seen' cases so the model does not need to be trained on the same case again if more data is recorded after the outcome. Lastly, since the model has already been trained on the case, the encoders are no longer necessary and can be removed from memory.

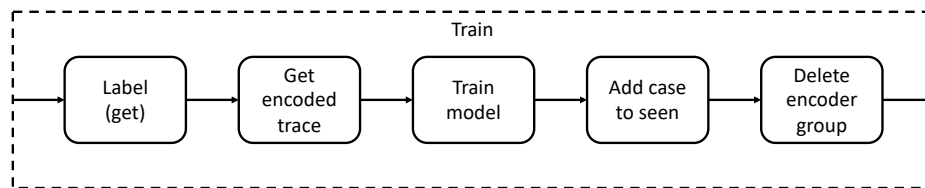


Fig. 3. Train phase

To predict the outcome of a case, the current trace is to be encoded using some subset of encoders. The encoded trace can then be used to make a

prediction about the cases' outcome, which is stored to evaluate and calculate prediction metrics.

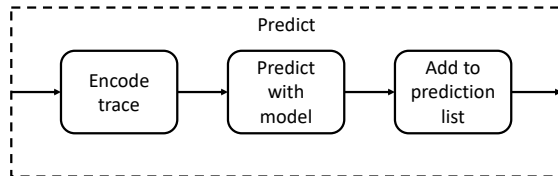


Fig. 4. Predict phase

## 4 Setup

Since [8] and [17] focused mainly on 1) a comparison of different streaming algorithms and 2) the effects of concept drift on PPM, this research acknowledges these results and instead focuses on *how* to implement these algorithms supporting concept drift in an algorithm-agnostic learning pipeline approach. Ideally, utilizing a proper pipeline approach transforms a PPM problem into a regular incremental learning approach, where feature construction in the form of encoding and labeling are the main problems to cover. The pipeline should cover the situation that there is typically a delay on capturing feedback for the outcome.

For the setup, online learning Python library River was used [19]. River allows for most of the required components for an online PPM pipeline, such as incremental streaming algorithms, advanced pipeline features, real-time preprocessing, transformers, and evaluation functions. However, some PPM-specific aspects are not typically covered in traditional data mining libraries and are thus not found in the River library so they had to be custom-built. These components, along with all the code used for this work, can be found on our GitHub page<sup>3</sup>.

### 4.1 Encoding and data preparation

As described in table 1, different encodings for a given trace can be generated to represent a trace as one tabular row. Some of these are readily available as simple statistic functions, such as *min*, *max*, *mean*, *count* or *variance*. For PPM, other non-traditional encoders are utilized as well, such as *index* encoders, *last state* encoders or more complex encoders to generate multiple features from a single attribute such as a timestamp. Additionally, while literature did not mention

<sup>3</sup> <https://github.com/ICPM-submission/Online-Predictive-Process-Monitoring>



including a *first state* encoder, it records the first non-NA value for a given attribute. An implementation of these encoders as River-compatible *transformers* can be found on the Github repo under StreamPPM/custom\_components.

Most PPM literature evaluates encoders by using an encoder on one or multiple attributes. However, it is also possible to apply encoders the other way around: by applying one or more encoders to a single attribute. For instance, it may be of interest to consider the last resource who worked on a case but also the total number of resources who worked on the same case. This leaves a significant search space of encoders: the number of encoders  $\times$  the number of attributes. In principle, encoders could also be parameterized, such as by a time window size for an aggregator, expanding the search space even more. Therefore, a selection of encoders should be made to reduce the complexity. In the current setup, this is done by creating an Excel template whereby each encoder is mapped to each attribute and their compatibility is given (*aggregation* encoders only apply to numerical variables, for example). These encoder objects can then be generated in the pipeline setup.

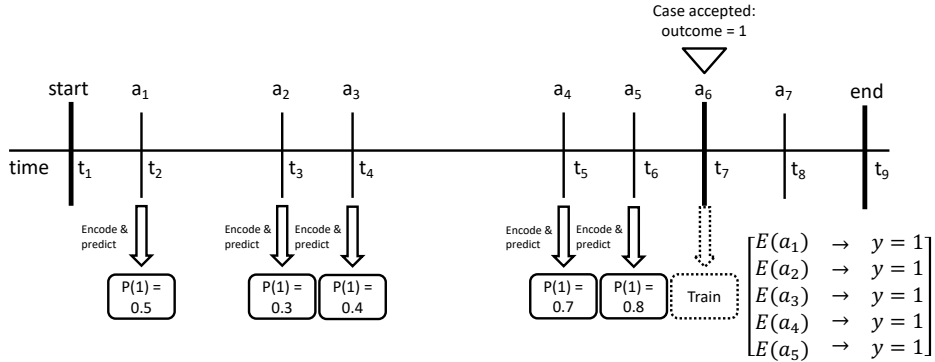
For data pre-processing the proper variable types are given for a dataset, the timestamps are parsed, categorical variables are one-hot encoded by default and numerical variables are scaled. Note that the type conversion and timestamp parsing are executed *before* encoding, while one-hot encoding and scaling are applied *after* encoding. Additionally, features based on timestamps can be generated as also generated in the benchmark dataset by [22]. These additional features are referred to as *time features* in the remainder of this work. Lastly, if value occurrence labeling is used, the feature in which the label is eventually found should be discarded when training the model as to avoid any leaking predictors.

## 4.2 Labeling function

Processes are typically not created with a prediction purpose, thus there is no clear 'label' attribute. To have a  $y$  for the model to train on, a labeling function is needed. This has been implemented as an abstract Python class with two functions: *check* whether the label is in the current data instance and *get* the label if it is in the current instance. Currently this labeling function only supports *value occurrence* as a label, which evaluates the *activity* column and has some pre-determined outcome activities, based on which the final label is determined to be either positive or negative. For instance: a loan request can be *accepted*, *cancelled* or *denied*. The labeling function evaluates whether the current activity is one that either accepts, cancels or denies the case and if it is, determines the label (positive if the activity is *accepted*, negative if otherwise).

## 4.3 Evaluation

Evaluation of PPM predictions is not trivial since multiple predictions are made within a single case. If one metric such as AUC were used over all of the predictions, it would be naturally skewed towards longer cases where more observations



**Fig. 5.** Timeline with feature rollback

are made. Therefore we use an evaluation method also used in [22] which computes the AUC over all cases for a given *prefix length*. Thus, a prediction is made for each activity in each case up until the label is known, and afterwards the AUC is calculated for each prefix length. AUC, or Area Under the Curve, is an appropriate metric because it remains unbiased even on an imbalanced distribution of class labels, and is threshold-independent. AUC scores are generally between 0.5 and 1, where an AUC of 0.5 is purely random and an AUC of 1 is a perfect classifier which is always correct.

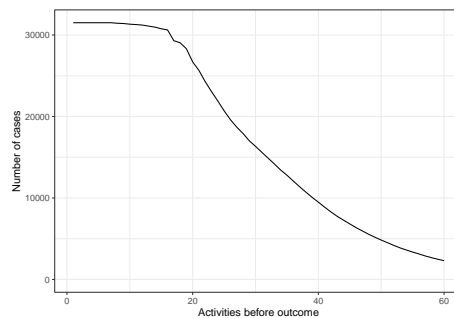
#### 4.4 Rollback

In offline predictive process monitoring, such as the approach by [25], a label is determined once it is known and then added as a feature over all of the activities in the trace. In a streaming setting this is not possible because there is no way to tell the label before seeing it. By encoding and waiting until we see the label in the data somewhere we can train the model, but we only train the model with data that is fully encoded, based on a fully completed trace. However, since encoders incrementally grow with newer data, at some point every state of the encoded process is stored. By storing each version to the encoded variables, we can keep track of 'versions' of the variables. At training time, by simply iterating through every version of our variables and training the model on each version, the model is also trained on incomplete traces in an attempt to improve the generalizability of the model when it sees incomplete traces in the prediction phase. This does not change the made predictions and thus the test set, but only adds incomplete training samples to the train set of the classifier.

## 5 Results

This section describes the results from applying the streaming PPM pipeline to a real dataset. For this, the BPIC dataset from 2017 was used for its number

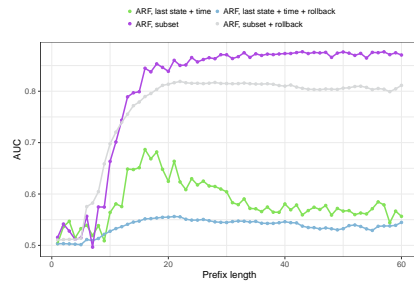
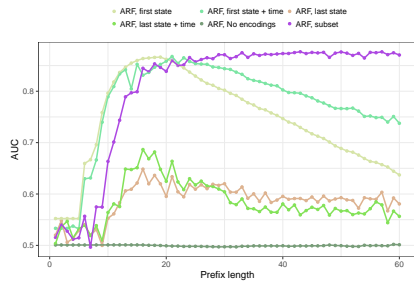
of cases and richness of attributes [27]. Additionally, since the BPIC17 dataset was used in the benchmark study by [22], the results can be well compared. As a prediction target, the loan offers are considered, where a positive label equals the activity "O\_Accepted" in the activity column, and a negative label containing "O\_Cancelled" or "O\_Denied". As preprocessing, categorical variables were one-hot encoded, and numerical variables scaled using River's built-in functions. To validate the learning pattern and the performance of the algorithm, Adaptive Random Forest (ARF) was applied with various encoding methods to establish benchmark performance and a comparison to that of [22]. For reference, fig. 6 shows the aggregation of the number of activities before the outcome is known for each case. From this it leads that the majority of cases have fewer than 30 activities until their outcome is known.



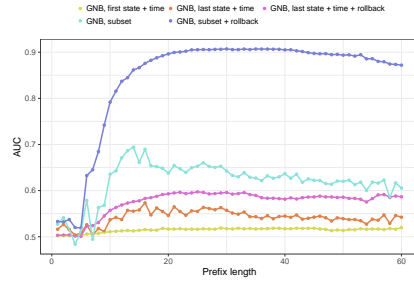
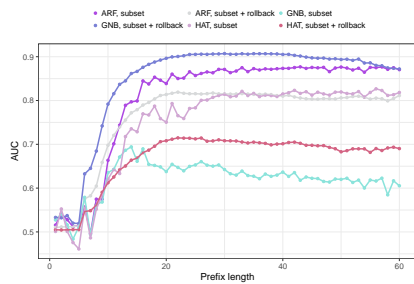
**Fig. 6.** Most cases have fewer than 30 activities before the outcome is known

From fig. 7, it is clear that some form of encoding is necessary. A simple baseline that iterates over the data and only uses the attributes corresponding to the current event gives a consistent AUC of 0.5 (guessing based on last event only). A better approach is to use the latest available (non-missing) data. This corresponds to *ARF, last state*, which iterates over the data and remembers the last known value for each attribute within the current trace. However, in our experiments with ARF, this improves AUC, but not beyond 0.65. Only using the first known value of each attribute yields a better prediction performance over nearly all prefix lengths than only considering the last known value for each attribute, especially early on in a process. 'Encoder subset' is a subset of encoders combining multiple encoders such as first and last state, aligning with state-of-the-art approaches in PPM literature. This encoder combination seems to predict better over longer cases than first state and last state individually, but performs slightly worse over shorter cases than the first state encoder.

Figure 8 introduces the concept of feature rollback with the ARF classifier to both the custom encoder subset and the last state encoder. Rollback for a first state encoder would only introduce missing values, thus is not included. From these results, feature rollback does not seem to improve performance for ARF



**Fig. 7.** Performance for different encoders **Fig. 8.** ARF performance with and without rollback



**Fig. 9.** Performance of various algorithms with encoder subset **Fig. 10.** Performance for different encoders with Gaussian Naive Bayes

with encoder subsets or only using the last state encoder. The performance of the feature rollback predictions is more stable and consistent over different prefix lengths.

Figure 9 shows the performance of three different algorithms: ARF, Hoeffding Adaptive Tree (HAT) classifier and Gaussian Naive Bayes (GNB), with and without rollback. As the results show, without rollback, Random Forest has the best performance, Hoeffding Adaptive Tree performs slightly worse and Naive Bayes performs worse than that. With feature rollback however, Naive Bayes scores better than every other classifier for all prefix lengths. Again, the performance is slightly more stable over the different prefix lengths. There is no clear single reason why Naive Bayes stands out, but in general it is known that at times it can perform surprisingly well on real world learning problems, even if the conditional independence assumption is violated [6, 21]. Since GNB suddenly performs much better than all other classifiers, fig. 10 again evaluates the performance for different encoders, this time using GNB.

As shown in fig. 10, the first state encoder seems to perform quite poorly while the last state encoder performs slightly better, with feature rollback improving

performance. The encoder subset with feature rollback performs significantly better than the encoder subset without rollback.

## 6 Discussion

The learning curve of the results from the online PPM approach seem to match the results of [22] quite well: on the same dataset, they found an uptick in prediction performance after around 5 events, after which it slowly increased towards higher performance. While their total AUC value was slightly higher, this could be due to slight differences in the used algorithms (XGBoost vs ARF). From this, it seems that the online learning approach is able to learn the same patterns as the benchmark study, with the added benefits of using adaptive models.

Because all of the preprocessing, encoding and labeling is performed in real-time, when simulating an online setting by iterating through a dataset, the time complexity is quite high. Adding feature rollback increases the time complexity by a factor of two to three. Depending on the configuration of the pipeline we regularly achieved speeds of between 400 and 1600 rows per second on a 2017 MacBook Pro with a 2,8 GHz Quad-Core Intel Core i7 and 16 GB of 2133MHz LPDDR3 RAM.

While the results show a similar learning curve to the benchmark study, further experiments with different encoders and algorithms lead to interesting results. For one, it is counter-intuitive that the first known value for a given value can have a higher predictive value than the last, most up-to-date value. While this effect does not always hold, it does show the need to experiment with the different encoders for a specific situation. The likely cause of this phenomenon is that the first part of a case has a different data distribution from the last. In that case, the first state encoder can be expected to perform better in small prefix sizes because the model is also trained on data from early on in the process, while the last state encoder is trained on the final state of a process, with the possibility of variables being overwritten in the meanwhile.

Other influential variables are algorithm choice and feature rollback: the combination of which can differ quite a lot, with last state encoders actually performing better with Naive Bayes and rollback being quite effective using Naive Bayes but under-performing when using ARF. A possible explanation of this effect is that ARF is able to better generalize the data to different variables by its tree structure, while Naive Bayes does not consider the relations between variables. By applying feature rollback the model is more smoothed out and does not need to make this generalization itself. These results do not give definitive proof that one is better than the other, but does show one should not make assumptions about which encoder, algorithm pair to use and whether feature rollback is effective.

## 7 Limitations & Future Work

This section describes the limitations of the current approach and several opportunities for directions of future research.

- The results show that the performance is quite dependent on the choice of encoders. Future work could focus on selecting applicable encoders from the search space of encoders for each attribute. Advancements in the field of Meta-learning could be applied here to learn patterns in the datasets which lean to certain encoders.
- An explanation of the relatively poor performance of the last state encoder are differences in the data distributions between the beginning of a case versus later on. Literature utilizes the concept of *bucketing* to deal with these different distributions by using a different model for different stages of the process, i.e. one for the beginning and a different model for the end of the case. It would be an interesting addition to an online predictive process monitoring pipeline to evaluate the effectiveness of different bucketing techniques.
- The current pipeline assumes that the label does not change after the model is trained on the case, and ignores all data after the label is known. In some applications, multiple labels may be known (i.e., first rejected but later accepted). Incorporating this effect could be an interesting addition for future research towards online PPM.
- By nature of a process, attributes arrive sequentially in a certain order. This makes the model quite dependent on the current attributes and the order thereof. To move more towards prescriptive process monitoring, future work could focus on process design improvements based on predictive features in regards to the outcome. Features with a highly predictive value could be moved forward in the process as to improve the early predictive power of machine learning models.
- Decisions in a process seldom depend on just a single predicted KPI, and background constraints and knowledge also need to be taken into account. Decision logic can be used to combine models for different KPIs with business rules, weightings and policies for more sophisticated real time decisioning.

## 8 Conclusion

Our results show that an online predictive process monitoring pipeline can achieve similar results as the benchmark 'offline' approach achieves. Furthermore, we show that different combinations of encoders, algorithms and feature construction can lead to unexpected results. To still achieve good results, it is therefore necessary to experiment with these different components in different scenarios. We provide an online PPM pipeline which allows for these experiments in a simple way using Python.

## References

1. Antonakis, A.C., Sfakianakis, M.E.: Assessing naïve Bayes as a method for screening credit applicants. *Journal of Applied Statistics* (2009). <https://doi.org/10.1080/02664760802554263>
2. Bahri, M., Bifet, A., Gama, J., Gomes, H.M., Maniu, S.: Data stream analysis: Foundations, major tasks and tools (2021). <https://doi.org/10.1002/widm.1405>
3. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: *Proceedings of the 7th SIAM International Conference on Data Mining* (2007). <https://doi.org/10.1137/1.9781611972771.42>
4. Carmona, J., Gavaldà, R.: Online techniques for dealing with concept drift in process mining. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2012). [https://doi.org/10.1007/978-3-642-34156-4\\_10](https://doi.org/10.1007/978-3-642-34156-4_10)
5. Domingos, P., Hulten, G.: Mining high-speed data streams. In: *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2000). <https://doi.org/10.1145/347090.347107>
6. Domingos, P., Pazzani, M.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning* **29**(2), 103–130 (1997). <https://doi.org/10.1023/A:1007413511361>, <https://doi.org/10.1023/A:1007413511361>
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). <https://doi.org/10.1007/978-3-642-33143-5>, <http://link.springer.com/10.1007/978-3-642-33143-5>
8. Francescomarino, C.D., Ghidini, C., Maggi, F.M., Rizzi, W., Persia, C.D.: Incremental predictive process monitoring: How to deal with the variability of real environments (2018)
9. Frederickson, C., Polikar, R.: Resampling Techniques for Learning Under Extreme Verification Latency with Class Imbalance. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. pp. 1–8. IEEE (2018)
10. Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfharinger, B., Holmes, G., Abdessalem, T.: Adaptive random forests for evolving data stream classification. *Machine Learning* (2017). <https://doi.org/10.1007/s10994-017-5642-8>
11. Grzenda, M., Gomes, H.M., Bifet, A.: Performance measures for evolving predictions under delayed labelling classification. In: *Proceedings of the International Joint Conference on Neural Networks* (2020). <https://doi.org/10.1109/IJCNN48605.2020.9207256>
12. Hofer, V., Kreml, G.: Drift mining in data: A framework for addressing drift in classification. *Computational Statistics and Data Analysis* **57**(1), 377–391 (2013)
13. IEEE: IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. IEEE Std 1849-2016 pp. 1–50 (2016). <https://doi.org/10.1109/IEEESTD.2016.7740858>
14. Karimi, J., Somers, T.M., Bhattacharjee, A.: The impact of ERP implementation on business process outcomes: A factor-based study. *Journal of Management Information Systems* (2007). <https://doi.org/10.2753/MIS0742-122240103>
15. Kas, S., Post, R., Wiewel, S.: Automated Machine Learning in a Process Mining Context. *International Conference for Process Mining (ICPM) 2020* p. 25 (2020), [https://icpmconference.org/2020/wp-content/uploads/sites/4/2020/10/ICPM\\_2020\\_paper\\_44.pdf](https://icpmconference.org/2020/wp-content/uploads/sites/4/2020/10/ICPM_2020_paper_44.pdf)

16. Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., Stefanowski, J.: Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter* (2014). <https://doi.org/10.1145/2674026.2674028>
17. Maisenbacher, M., Weidlich, M.: Handling Concept Drift in Predictive Process Monitoring. In: *Proceedings - 2017 IEEE 14th International Conference on Services Computing, SCC 2017* (2017). <https://doi.org/10.1109/SCC.2017.10>
18. Marquez-Chamorro, A.E., Resinas, M., Ruiz-Cortes, A.: Predictive monitoring of business processes: A survey. *IEEE Transactions on Services Computing* (2018). <https://doi.org/10.1109/TSC.2017.2772256>
19. Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., Bifet, A.: *River: machine learning for streaming data in Python* (2020)
20. Pauwels, S., Calders, T.: Incremental Predictive Process Monitoring: The Next Activity Case. In: *Business Process Management*. Springer International Publishing (2021), [https://www.researchgate.net/publication/352508887\\_Incremental\\_Predictive\\_Process\\_Monitoring\\_The\\_Next\\_Activity\\_Case](https://www.researchgate.net/publication/352508887_Incremental_Predictive_Process_Monitoring_The_Next_Activity_Case)
21. van der Putten, P., van Someren, M.: A Bias-Variance Analysis of a Real World Learning Problem: The CoIL Challenge 2000. *Machine Learning* **57**(1/2), 177–195 (oct 2004). <https://doi.org/10.1023/B:MACH.0000035476.95130.99>, <http://link.springer.com/10.1023/B:MACH.0000035476.95130.99>
22. Teinemaa, I.: Predictive and prescriptive monitoring of business process outcomes. In: *CEUR Workshop Proceedings* (2019)
23. Teinemaa, I., Dumas, M., Leontjeva, A., Maggi, F.M.: Temporal stability in predictive process monitoring. *Data Mining and Knowledge Discovery* **32**(5), 1306–1338 (sep 2018). <https://doi.org/10.1007/s10618-018-0575-9>, <http://link.springer.com/10.1007/s10618-018-0575-9>
24. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark (2019). <https://doi.org/10.1145/3301300>
25. Teinemaa, I., Tax, N., de Leoni, M., Dumas, M., Maggi, F.M.: Alarm-based prescriptive process monitoring. In: *Lecture Notes in Business Information Processing* (2018). [https://doi.org/10.1007/978-3-319-98651-7\\_6](https://doi.org/10.1007/978-3-319-98651-7_6)
26. Teinemaa, I., Tax, N., de Leoni, M., Dumas, M., Maggi, F.M.: Foundations of Prescriptive Process Monitoring. *CoRR* **abs/1803.0** (2018), <http://arxiv.org/abs/1803.08706>
27. Van Dongen, B.F.: BPI Challenge 2017 (2017). <https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b>, [https://data.4tu.nl/articles/dataset/BPI\\_Challenge\\_2017/12696884](https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884)
28. Van Dongen, B.F.: BPI Challenges (2021), <https://www.tf-pm.org/competitions-awards/bpi-challenge>
29. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: *Lecture Notes in Business Information Processing* (2020). [https://doi.org/10.1007/978-3-030-58638-6\\_12](https://doi.org/10.1007/978-3-030-58638-6_12)