



# Delayed labelling evaluation for data streams

Maciej Grzenda<sup>1</sup> · Heitor Murilo Gomes<sup>2,3</sup> · Albert Bifet<sup>2,3</sup>

Received: 20 November 2018 / Accepted: 5 September 2019 / Published online: 16 November 2019  
© The Author(s) 2019

## Abstract

A large portion of the stream mining studies on classification rely on the availability of true labels immediately after making predictions. This approach is well exemplified by the test-then-train evaluation, where predictions immediately precede true label arrival. However, in many real scenarios, labels arrive with non-negligible latency. This raises the question of how to evaluate classifiers trained in such circumstances. This question is of particular importance when stream mining models are expected to refine their predictions between acquiring instance data and receiving its true label. In this work, we propose a novel evaluation methodology for data streams when verification latency takes place, namely continuous re-evaluation. It is applied to reference data streams and it is used to differentiate between stream mining techniques in terms of their ability to refine predictions based on newly arriving instances. Our study points out, discusses and shows empirically the importance of considering the delay of instance labels when evaluating classifiers for data streams.

**Keywords** Stream mining · Delayed labels · Evaluation procedures · Classification

---

Responsible editor: Po-Ling Loh, Evimaria Terzi, Antti Ukkonen, and Karsten Borgwardt.

---

✉ Maciej Grzenda  
M.Grzenda@mini.pw.edu.pl

Heitor Murilo Gomes  
heitor.gomes@waikato.ac.nz

Albert Bifet  
albert.bifet@waikato.ac.nz

<sup>1</sup> Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland

<sup>2</sup> LTCI, Télécom Paris, IP-Paris, Paris, France

<sup>3</sup> Department of Computer Science, University of Waikato, Hamilton, New Zealand

## 1 Introduction

The evaluation of stream mining algorithms in many cases relies on the test-then-train or prequential approach (Gama and Rodrigues 2009), i.e., first an unlabelled instance is used to generate model prediction; next, the model is provided with the label of the instance to trigger possible model updates. This approach, while clear and uniform over all stream instances, is not applicable when delayed labels are observed. In this case, the time period between using the unlabelled instance data as a model input and receiving the true label is non-negligible.

Since prequential evaluation does not match all stream mining settings, verification latency (Ditzler et al. 2015) and stream mining methods focusing the delayed label setting have been investigated (Kuncheva and Sánchez 2008; Masud et al. 2011; Souza et al. 2015). Many of these works assume that the prediction for an instance of interest is performed when the instance data arrives for the first time. However, in problems such as delay prediction in transportation systems an extension of this approach can be proposed aiming at iterative re-consideration of predictions already made. As an example, when the objective is to predict whether a plane will arrive in time, the prediction is made at departure time. However, such initial prediction can be refined in the period preceding planned arrival based on the most recent situation at the destination airport. More formally, the on-line learning model that reflects the knowledge of the process arising from the most recent labelled instances may be polled to get a new prediction made for the data of the instance of interest. Such extension aiming at generating additional predictions can be applied to both classification and regression tasks.

The objective of such re-evaluation proposed in this study is to verify whether a model can improve its prediction for the same instance data when polled for further predictions before the true label arrives. The evaluation of such iterative predictions provides additional insight into the way different stream mining techniques exploit newly arriving labelled instances. We expect that some of the models may respond during the entire period with the same predicted class for the same input instance, while other models are going to update their predictions, either because they were given access to other instances' labels or in response to concept drift detection. In fact, the evaluation process that we propose reveals the varied ability of the techniques to refine their predictions under delayed label scenarios. This is achieved through both numerical indicators and visual analysis that we outline in this work. A reference task that illustrates our approach is the delay prediction performed with airlines data. The objective is to investigate the ability of streaming classification models to refine their delay prediction after flight departure and before its real arrival. Other model evaluation methodologies rely on the first prediction only, while our extension considers intermediary predictions. We claim that our approach yields model performance assessment better matching use cases where streaming classification models are used, i.e., where the evolution of the models is desirable and predictions provided by these models may evolve in turn.

In summary, the main contributions of this work are the following:

- We point out the need for a specific evaluation for delayed labelled streams;

- We propose a novel evaluation method to be applied when verification latency takes place, and provide an open source implementation of it<sup>1</sup> that can be used by the community;
- We discuss the impact of delayed labels using synthetic and real world data streams of flights’ delays and others;
- We present empirical experiments highlighting our evaluation procedures and the traditional test-then-train procedure;
- Finally, we discuss possible extensions to this work as well as an experiment with a regression problem.

This work is organized as follows: we present in Sect. 2 relevant related work, and in Sect. 3 the problem statement and its motivation. Sect. 4 presents the new methodology proposed, and Sect. 5 shows a comprehensive evaluation, which is followed by Sect. 6 discussing the role of hyper-parameters of the evaluation procedure, alongside considerations about the computational resources. Sect. 7 presents possible extensions to our methodology with special attention on how it can be applied to regression. Finally, the conclusions are presented in Sect. 8.

## 2 Related work

There are at least two major aspects to take into account when evaluating data stream mining algorithms. The first aspect concerns “how” past predictions will influence the current model learning performance estimation. The second aspect must take into account “when” the ground truth values will become available for estimating the learning performance.

The first aspect was thoroughly studied and there are already well established methods to provide robust evaluations that consider the time when predictions were obtained. These evaluation methods include: periodic holdout, test-then-train, prequential (Gama and Rodrigues 2009) and cross-validated prequential (Bifet et al. 2015).

**Periodic holdout evaluation** interleaves training and testing using predefined windows, such that one window of instances is used for training, the next window for testing and so on. **Test-then-train evaluation** uses each instance first for testing and then right after for training. **Prequential evaluation** is similar to test-then-train except that prequential includes a fading factor (or windowed evaluation) (Gama and Rodrigues 2009) to ‘forget’ old predictions performance. Finally, in **cross-validated prequential evaluation** models are trained and tested in parallel on different folds of data to better estimate its performance.

The second aspect, which concerns “when” labels are made available has not yet been thoroughly explored. Until recently the majority of works assumed that instances appear sequentially in fixed intervals, and that an instance’s ground truth is available before the next instance arrives. This problem setting can be identified as the ‘immedi-

---

<sup>1</sup> Available at <https://github.com/mgrzenda/ContinuousReEvaluation>. The implementation is an extension of the Massive Online Analysis (MOA) framework.

ate labelling setting' (Gomes et al. 2017), and many algorithms (Domingos and Hulten 2000; Hulten et al. 2001) were developed focusing this setting.

Formally, we can define a data stream as a sequence of tuples  $\mathcal{S}_1, \mathcal{S}_2, \dots$  which includes two types of tuples i.e.

$$\mathcal{S}_a = \begin{cases} \{(\mathbf{x}_k, ?)\} & \text{if no true label is available yet} \\ \{(\mathbf{x}_k, y_k)\} & \text{when a true label becomes available} \end{cases}$$

Furthermore,  $\mathbf{x}_k$  denotes a feature vector  $\mathbf{x}_k \in R^n$  and  $y_k$  a discrete class label (Ditzler et al. 2015). Under the aforementioned assumption that an instance class label is available before the next instance arrives, the stream can be defined as a sequence of tuples  $\{(\mathbf{x}_k, ?)\}, \{(\mathbf{x}_k, y_k)\}, \{(\mathbf{x}_{k+1}, ?)\}, \{(\mathbf{x}_{k+1}, y_{k+1})\}, \dots$ . Since the true class label  $y_k$  of a given instance  $\{(\mathbf{x}_k, ?)\}$  is available before the next instance feature vector  $\{(\mathbf{x}_{k+1}, ?)\}$  arrives,  $\{(\mathbf{x}_k, y_k)\}$  can be used to update the learner performance and its internal model without any delay.

The existence of non-negligible delay between the instance data generation and the labels has been discussed on several works (Kuncheva and Sánchez 2008; Zliobaite 2010; Abdulsalam et al. 2010; Plasse and Adams 2016; Gomes et al. 2017), which we briefly survey below.

Abdulsalam et al. (2010) discuss a streaming scenario where class labelling is intermittent, such that blocks of labelled instances are interleaved with blocks of unlabelled instances. Furthermore, the authors propose an algorithm that combines a streaming random forest implementation with several components to adapt to concept drifts and cope with multi-class problems. More importantly, the algorithm is designed to decide whether the current model can be used for predictions, based on a threshold, in situations where the number of labelled records is not sufficient to build or update it.

Gomes et al. (2017) present an approach to evaluate the classification performance assuming a problem setting where the labels are not readily available. It was identified as 'delayed labelling setting' and its definition was quite simple: the learner is evaluated based on its capacity to correctly predict an instance, which ground truth will only be available a fixed number of time units latter. In Gomes et al. (2017), authors show that useful information can be obtained when comparing the results for both the immediate and the delayed labelling settings, using the same algorithms and datasets.

Plasse and Adams (2016) introduce a taxonomy to determine the delay mechanism and magnitude, present real world applications where delayed labels occur (e.g. credit scoring), notation for the delayed setting and how the set of delayed labels can be used to pre-update the classifier. However, in Plasse and Adams (2016) no specific evaluation procedure was introduced when verification latency occurs. Kuncheva and Sánchez (2008) present experiments focusing on different approaches to exploit delayed labelled instances using IB2 and IB3, however, the evaluation of the results followed the traditional holdout approach (i.e. 90% for training and 10% for testing).

Zliobaite (2010) presents an analytical view of the conditions that must be met to allow concept drift detection in a delayed labelled setting. Three types of concept drifts are analytically studied and two of them also empirically evaluated. Similar to

Kuncheva and Sánchez (2008) the experiments focusing on classification performance did not further explore the verification latency of the data.

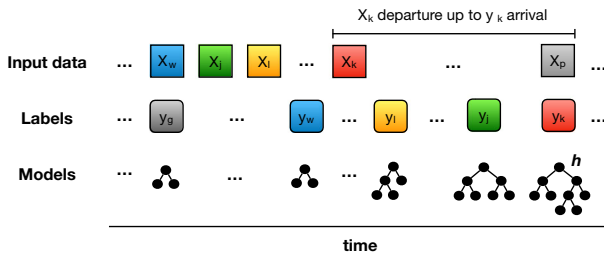
In the current work we present a novel evaluation method that assumes labels will be presented to the model with verification latency, i.e. delay. The goal is to better estimate the learner capability to refine predictions by successively polling it for predictions before the true label arrives. This approach degenerates to test-then-train when there is no verification latency, but it is different from the delayed evaluation presented in Gomes et al. (2017) when delay exists, as in our approach there is no fixed amount of time units. Also, in the current work we take into account the intermediary predictions (re-evaluation), while in Gomes et al. (2017) only the prediction obtained at the time the instance data was first presented was used to update the learner performance.

### 3 Problem statement and justification

As discussed in Sect. 2, many stream classification studies assume that  $y_k$  is available after  $\{(\mathbf{x}_k, ?)\}$  and immediately before the next unlabelled instance arrives. However, when labels are available with some delay, between  $\{(\mathbf{x}_k, ?)\}$  and  $\{(\mathbf{x}_k, y_k)\}$ , other labelled and unlabelled instances may arrive in the stream. Let us use the term *time* to refer to the moment that an instance becomes available in the stream. Without the loss of generality, let us assume that time is represented by a numeric timestamp for real data streams and by an instance index for synthetic streams for which time notion has not been defined. In line with these assumptions, let  $t(\mathcal{S}_a)$  denote the time of observing instance  $\mathcal{S}_a$  in the stream. Moreover,  $t(\mathcal{S}_a) \leq t(\mathcal{S}_b)$ ,  $a < b$ . The unlabelled instance  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}$  appears in the stream before its labelled counterpart  $\{(\mathbf{x}_k, y_k)\}$ .  $\mathcal{S}_a$  can be used to obtain prediction  $\hat{y}_k^{t(\mathcal{S}_a)}$  from a model  $h_a$  available at this stage of the stream processing. Let  $\tilde{t}(\mathcal{S}_a)$  denote the time that the label of  $\mathcal{S}_a$  arrived, and  $\Delta t_k = \tilde{t}(\mathcal{S}_a) - t(\mathcal{S}_a)$ ,  $\Delta t_k \geq 0$  the latency for  $\{(\mathbf{x}_k, y_k)\}$  to arrive in the stream. At this stage, the loss between prediction  $\hat{y}_k^{t(\mathcal{S}_a)}$  and  $y_k$  can be computed.

Importantly, the presence of delayed labels means that the stream potentially includes sequences  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}, \dots, \mathcal{S}_b = \{(\mathbf{x}_j, y_j)\}, \dots, \mathcal{S}_c = \{(\mathbf{x}_k, y_k)\}$ ,  $a < b < c$ . Hence, between an unlabelled instance  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}$  and its labelled version  $\mathcal{S}_c = \{(\mathbf{x}_k, y_k)\}$  other (possibly many) labelled instances  $\{(\mathbf{x}_j, y_j)\}$  arrive. As a consequence, it is possible that the models differ, i.e.,  $h_a \neq h_b \neq h_c$ . In particular, perhaps  $h_a(\mathbf{x}_k) \neq h_c(\mathbf{x}_k)$ , i.e., the prediction made for the same input data  $\mathbf{x}_k$  changes with time as model  $h_a$  changes and becomes model  $h_c$ . Such *evolving predictions* are possible when model  $h$  changes between the time of generating prediction  $t(\mathcal{S}_a)$  and the time of obtaining true label  $\tilde{t}(\mathcal{S}_a)$ . Changes in the model are expected especially when concept drift occurs, i.e., the posterior probability  $p_t(y|\mathbf{x})$  shifts. However, such changes are also expected in stationary conditions, for instance when a tree-based model is gradually extended by adding new branches, which refines the tree predictions.

Let us illustrate the aforementioned problem setting with the flight delay prediction task. Let the objective be to predict flight delay at a destination airport for individual flights. Hence, flight features such as planned duration, destination, time of the day or weather conditions at departure time can be placed in  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}$  to obtain delay



**Fig. 1** Illustration of the flight problem. From the time  $x_k$  is generated up to the time its corresponding label  $y_k$  is available the tree model  $h$  changes as new labelled instances (e.g.,  $y_i, y_j$ ) are presented to it

prediction  $h_a(x_k) = \hat{y}_k^{t(S_a)}$ . Such prediction is made at the time of planned departure. However, during the flight the prediction model is likely to evolve taking into account recently observed delays of other flights. Moreover, the predictive features, such as weather conditions may change. Hence, requests for new predictions from an evolving model  $h$  are justified. We also show this example graphically on Fig. 1.

The aforementioned changes to model  $h$  and possible evolution of instance data enable new, and possibly more accurate predictions provided by  $h$  in the period preceding the arrival of the delayed label. Not only, can initial prediction  $\hat{y}_k^{t_0}$  be acquired at time  $t_0 = t(S_a)$ , but also a sequence of predictions  $\hat{y}_k^{t_1}, \hat{y}_k^{t_2}, \dots; t_i \leq \tilde{t}(S_a)$  can be considered. Let us use the term *continuous re-evaluation* to refer to the process of obtaining and evaluating such predictions. Moreover, let  $\check{y}_k^t$  stand for the most recent prediction, i.e. the most recent refinement of the initial prediction for  $\{(x_k, ?)\}$  available at time  $t$ .

In an ideal case, we expect the model to provide accurate predictions immediately, or shortly, after the original instance data  $x : S_a = \{(x, ?)\}$  was used to obtain the initial prediction. Obviously, such model would be preferred to a model that provides accurate predictions only shortly before true label arrives. In the case of flight data, this would mean a preference for the models capable of producing more accurate predictions shortly after the flight departure.

Therefore, we believe there is a need for both continuous re-evaluation and new evaluation procedures taking into account possible multiple requests made to evolving models to generate predictions for instance data between the time of knowing the instance data and the time of receiving its ground truth. Such evaluation procedures should extend existing procedures applied under both immediate and delayed label settings. Moreover, they should help differentiate models and techniques in terms of the ability of providing accurate response based on (if possible) limited volume of labelled instances  $S_k; k > a$  following individual  $S_a$  instances, for which no labels have arrived yet. Furthermore, such extended evaluation should foster the development of novel stream classification techniques by providing additional insight into the process of model evolution.

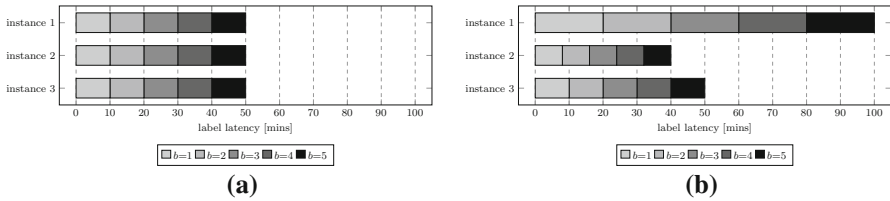


Fig. 2 Mapping predictions into bins,  $B = 5$  under fixed (a) and varied (b) label latency

### 4 New delayed evaluation methodology

The question arises as of how frequently to get new predictions for an instance of interest awaiting its true label. Intuitively, we expect the unlabelled instance to be used as an input for a prediction model more frequently when: (a) the period between getting instance data for the first time and receiving delayed label is long, or (b) multiple new labelled instances  $S_m$  arrive before the ground truth label  $y_k$  becomes available. This is because numerous  $S_m$  instances may cause the evolution of the model  $h$  during this period. Hence, let us propose continuous re-evaluation to be parameterised with the number  $K$  of labelled instances  $\{(x_i, y_i)\}$  that have to appear in the stream between two consecutive  $\hat{y}_k^{t_j}, \hat{y}_k^{t_{j+1}}$  predictions made for the instance  $\{(x_k, ?)\}$ . In other words, a new prediction for  $\{(x_k, ?)\}$  will be made every  $K$  new labelled instances until true label  $y_k$  arrives.

The periods of waiting for the labels can be decomposed into subperiods to analyse the changes in the performance measures before the labels arrive. In case all labels arrive with the same latency, predictions made within same-length subperiods can be analysed, as illustrated in Fig. 2a for  $B = 5$  subperiods. In the analysed case of  $\Delta t_i = 50$  in Fig. 2a, every subperiod would share the same length of  $\frac{\Delta t_i}{B} = 10$  [mins].

Let us refer to every subperiod as a *bin*. Importantly, for some data streams true labels arrive with varied latency. Still, bins of varied lengths can be identified in this case, as shown in Fig. 2b. Furthermore, performance measures can be calculated for the predictions made in each of the periods of individual bins, separately. Hence, by defining a number of bins, we can analyse how the values of performance measures change between making initial predictions and receiving true labels also in the case of varied label latencies. The label latency is always known when the true label arrives. Consequently, the periods of individual bins can be calculated and performance measures for individual bins can be updated, in turn. Before this happens all predictions made for the instance have to be buffered. Once the true label arrives, individual buffered predictions can be mapped to corresponding bins and the evaluation of performance measures for every bin can be made.

The proposed continuous re-evaluation algorithm is summarised in Algorithm 1. First, the algorithm checks whether an unlabelled instance, or a true label arrived. In the former case of an instance awaiting its true label, the instance is placed in a buffer  $L$  and the first prediction is made for it and placed in bin  $b = 0$ . When a true label arrives, the prediction immediately preceding model updates is placed in a bin  $B + 1$ . Next, performance measures are updated based on all the buffered predictions

**Input:**  $S_1, S_2, \dots$  - data stream,  $K$  - the number of labelled instances triggering new prediction,  $B$  - the number of bins,  $d(\{(\hat{y}_i, t_i)\}) \rightarrow Y$  - function converting possibly many predictions made during a certain period into one prediction from the set  $Y$  denoting the set of possible values of  $y_k$

**Data:**  $L$  - list of tuples  $(\{(x_k, ?)\}, l)$ , where  $\{(x_k, ?)\}$  is an instance for which true label has not arrived yet and  $l$  is the number of other labelled instances received since  $t(\{(x_k, ?)\})$ ;  $P(k)$  - the list of prediction tuples made for  $S_i = \{(x_k, ?)\}$ , the list contains tuples  $(\hat{y}, t, b)$ , where  $\hat{y}$  denotes prediction,  $t$  - the time of generating it and  $b$  - the bin it belongs to, respectively, while  $b = -1$  denotes a placeholder bin i.e. a bin containing periodical predictions until true label  $y_k$  arrives i.e. until these predictions can be shuffled into proper bins  $b = 1, \dots, B$  associated with subperiods between  $t(S_i)$  and  $\tilde{t}(S_a)$ ;  $E(b), b = 0, \dots, B + 1$  - evaluators for individual bins calculating performance measures, e.g. accuracy,  $h_i$  - the prediction model trained with labelled instances  $S_j; j = 1, \dots, i - 1, C(i), C_L(i)$  - the number of calculated predictions following initial prediction, after processing  $i$  instances for all and labelled instances, respectively

```

begin
   $h_1 = \phi; C(0) = 0; C_L(i) = 0;$ 
  for  $i = 1, \dots$  do
     $C(i) = C(i-1);$ 
    /* New unlabelled instance arrived */
    if  $S_i = \{(x_k, ?)\}$  then
       $L.add(\{(x_k, ?)\}, l = 0);$ 
      /* obtain first time prediction, add it to bin  $b=0$  and
      publish */
       $P(k).add(h_i(x_k), t(S_i), b = 0);$ 
    else
      /*  $S_i = \{(x_k, y_k)\}$ , i.e. true label arrived */
      /* obtain test-then-train prediction, add it to bin  $b = B + 1$ 
      and publish */
       $P(k).add(h_i(x_k), t(S_i), b = B + 1);$ 
       $E(b = 0, \dots, B + 1) = \text{updatePerformanceMeasures}(P(k), y_k, t(S_i));$ 
       $C_L(i) += \text{card}(P(k) : P(k).b = -1); L.remove(k);$ 
      for  $m \in L$  do
        if  $m.l \% K = 0$  then
          /* obtain periodical prediction, add it to
          placeholder bin  $b = -1$  and publish */
           $\hat{y}_k^{t(S_i)} := h_i(m.x);$ 
           $P(m).add(\hat{y}_k^{t(S_i)}, t(S_i), b = -1);$ 
           $C(i) = C(i) + 1;$ 
        end
         $m.l = m.l + 1;$ 
      end
       $h_{i+1} = \text{train}(h_i, \{(x_k, y_k)\});$ 
    end
  end
end

```

**Algorithm 1:** Continuous re-evaluation



made for the instance. Furthermore, for every other instance  $m \in L$  still awaiting its true label, the number of other labelled instances received since  $t(m)$  is checked. Every  $K$  labelled instances a new prediction is made by the most recent model. In this way possible updates to the model can be reflected in refined predictions, which are immediately published and can be used in scenarios such as on-line monitoring of industrial processes. Such periodic predictions are all placed in a placeholder bin ( $b = -1$ ) to be shuffled into proper bins once true label latency and bin length is known. Last, but not least the model  $h_i$  is updated.

The way performance measures are updated is particularly important. This is achieved by `updatePerformanceMeasures` method, and takes as an input all prediction tuples  $P(k)$  obtained for the instance of interest, the true label  $y_k$  and the time this true label arrived  $t(\mathcal{S}_i)$ . The way performance measures can be calculated based on prediction tuples is proposed below.

### 4.1 Performance indicators

Let us propose *binned performance* to be defined as an average performance of the models available in  $b$ th bin between attaining instance data and receiving its true label, where  $b = 0, \dots, B + 1$  stands for bin index. We propose that every labelled instance contributes to binned performance with the same weight. The primary objective for binned performance is to enable visualization of performance evolution under evolving predictions. The performance can be defined through various indicators, including, but not limited to accuracy and kappa statistic. The first bin, denoted by  $b = 0$  stands for the model performance observed at the time the instance data becomes available for the first time. The last bin, denoted by  $b = B + 1$  stands for test-then-train performance. All the remaining bins are used to aggregate performance indicators of evolving predictions for individual subperiods of the periods between  $t(\mathcal{S}_i)$  and  $\tilde{t}(\mathcal{S}_i)$ . The way such aggregation is performed for individual instances and the entire data stream is what we propose below.

Once true label becomes available, individual predictions can be linked to the appropriate bins. Let us observe that for  $1 \leq b \leq B$ , the sequence of predictions  $U(\mathcal{S}_i, b)$  available for instance  $\mathcal{S}_i = \{(\mathbf{x}_k, ?)\}$  and period  $b$  includes all predictions for the period of this bin. More precisely this means the prediction available at the beginning of bin period, predictions made during this period and the prediction available at the end of it. Formal definition of  $U(\mathcal{S}_i, b)$  is provided in ‘‘Appendix 1’’.

In the case of flight data, these predictions would contain possibly changing predicted flight status displayed at the beginning of and during bin period. Hence, the set of predictions made for  $\mathcal{S}_i$  and linked to bin  $b$  is as follows:

$$R(\mathcal{S}_i, b) = \begin{cases} \{(y_i, t_i) \in P(k) : b_i = 0\} & b = 0 \\ \{(y_i, t_i) \in U(\mathcal{S}_i, b)\} & b \in \{1, \dots, B\} \\ \{(y_i, t_i) \in P(k) : b_i = B + 1\} & b = B + 1 \end{cases}$$

It follows from Algorithm 1 that  $R(\mathcal{S}_i, b)$  contains exactly one element for  $b \in \{0, B + 1\}$  i.e. first time prediction and test-then-train prediction, respectively. In the case of

periodic predictions, i.e.,  $b \in \{1, \dots, B\}$ ,  $\text{card}(R(\mathcal{S}_i, b)) > 1$ . The question arises how to calculate performance indicators for individual bins, when it is possible that multiple, possibly different predictions exist for one bin period and one instance. We propose to handle such possibly diverse predictions in the same way the output of probabilistic classifier, taking the form of a vector of probabilities, is mapped to a single predicted class. Hence, we propose to determine a single *averaged prediction*  $d(U(\mathcal{S}_i, b))$ . The way the averaged prediction is selected is defined by  $d()$  function, which is a parameter of Algorithm 1. In the case of classification, we propose to use a function selecting dominating prediction i.e. the prediction displayed during the largest part of the bin period as an averaged prediction. Such prediction is defined as follows:  $d_D(U(\mathcal{S}_i, b)) = \arg \max_c \sum_{j=2}^{\text{card}(U(\mathcal{S}_i, b))} (t_j - t_{j-1}) : y_{j-1} = c$ . In other words, we propose to determine the prediction that was announced for the longest period of time during bin subperiod, i.e. the prediction with the highest probability. In the case of airlines use case, the prediction would be the predicted delay status that was displayed during the longest total period of time during bin period. It is worth noting here that frequently multiple identical predictions contained in  $R(\mathcal{S}_i, b)$  will be replaced with a single one.

Hence, for every instance and bin combination, one predicted label  $\hat{y}_k$  is used to calculate performance measures. For every bin  $b \in \{0, \dots, B + 1\}$ ,  $V(\mathcal{S}_i, b)$  denotes the pair  $\{(y_k, \hat{y}_k)\}$  of true label and predicted label used to calculate performance measures. In the case of bins  $b \in \{0, B + 1\}$  a predicted label  $\hat{y}_k$  is the first time and test-then-train prediction for  $\mathcal{S}_i$  instance, respectively. In the case of remaining bins i.e. the bins representing periodic predictions, the predicted label  $\hat{y}_k$ , which is used to calculate performance measures is the averaged prediction for  $\mathcal{S}_i$  instance in  $b - th$  subperiod. Formal definition of  $V(\mathcal{S}_i, b)$  is provided in ‘‘Appendix 1’’.

Furthermore, let us also propose aggregate loss function  $\lambda$  over a single bin  $b$  to be applied to a sequence of tuples  $V(\mathcal{S}_i, b), \dots, V(\mathcal{S}_j, b), \dots$  gradually produced for individual labelled instances. In particular,  $\forall i, j \tilde{t}(\mathcal{S}_i) \leq \tilde{t}(\mathcal{S}_j)$ . Moreover, every labelled instance contributes to performance evaluation of every bin. Since performance evaluation is not affected by varied number of predictions in  $b$ th sub-period for individual instances, performance evolution is due to changes in the models, not due to the varied sets of instances and their predictions used to calculate performance for different bins. Furthermore, for  $B = 0$ , the formula resolves to performance assessment taking into account only predictions made at the time of receiving unlabelled instance data for the first time ( $b=0$ ) and at the time preceding receiving true label ( $b = B + 1$ ).

Finally, let us propose aggregate loss function to be defined as a function of bin index:  $\Lambda(T, b) = \lambda(V(\mathcal{S}_i, b), \dots, V(\mathcal{S}_j, b), \dots : \mathcal{S}_i, \mathcal{S}_j \in \Omega(T))$ , where  $\Omega(T)$  denotes a sequence of all labelled instances  $\{(\mathbf{x}_k, y_k)\}$  that arrived until time  $T$ , i.e.  $t(\{(\mathbf{x}_k, y_k)\}) \leq T$ , and  $b \in \{0, \dots, B + 1\}$ . The function provides numerical assessment of the performance of evolving classification models as a function of discretised proportion of time that elapsed between getting instance data and receiving their true labels.

It is important to note that every  $V(\mathcal{S}_i, b)$  takes the form of one pair  $\{(y_k, \hat{y}_k)\}$ . Hence, for every bin index, performance assessment performed by  $\Lambda(T, b)$  is based on the same sequence of true labels  $y_k$  matched with  $\hat{y}_k$  predicted labels, which depend

on  $b$  value. The predicted labels  $\hat{y}_k$  are: first time predictions (for  $b = 0$ ), averaged predictions (for  $b \in \{1, \dots, B\}$  i.e. for individual subperiods between first time prediction and true label arrival); and test-then-train predictions (for  $b = B + 1$ ). Hence, this approach is uniform for all these three cases. In particular,  $\Lambda(T, 0)$  provides aggregate loss value for the first time predictions,  $\Lambda(T, B + 1)$  yields performance assessment calculated for test-then-train mode, and  $\Lambda(T, b)$ ,  $b \in \{0, \dots, B\}$  provides aggregate loss value for the predictions available in individual subperiods of the periods preceding true label arrivals. Hence, the aggregate loss function  $\Lambda(T, b)$  can be considered the generalisation of previously existing performance indicators used for delayed label setting.

To calculate  $\Lambda(T, b)$ , different loss functions can be applied. Without loss of generality, a zero-one loss function where the cost of misclassification  $\xi(y_a, \hat{y}_a) = 0$  if  $\hat{y}_a = y_a$  and 1 otherwise will be used throughout the rest of this study. In this case, 
$$\Lambda(T, b) = \frac{\sum_{S_i \in \Omega(T)} \xi(y_k, \hat{y}_k) : (y_k, \hat{y}_k) = V(S_i, b)}{\text{card}(\Omega(T))}.$$

## 5 Experimental analysis

In order to analyse the proposed performance indicators, we have used three synthetic and three real streaming datasets. Furthermore, we have selected stream classification methods illustrating both methods developed to address unique requirements of stream mining, and methods successfully adapted to stream processing needs. The first group of stream classification methods featured in the experiments was Hoeffding tree (HT) (Domingos and Hulten 2000), Hoeffding Adaptive Tree (HAT) (Bifet and Gavaldà 2009), and recently proposed Adaptive Random Forest (ARF) (Gomes et al. 2017). Other methods included k Nearest Neighbours (kNN) and Naive Bayes (NB). Finally, the baseline models No Change (NC) and Majority Class (MC) were also used. Hence, methods typically used for comparison purposes in stream mining studies and representing different groups of classification methods i.e., tree models, distance-based models and probabilistic models, but also model ensembles were used to analyse the proposed performance indicators.

HT is a traditional stream classification algorithm based on the Hoeffding Bound. HT serves as the base for many stream classifiers, and ensemble methods, as it defines how to grow trees incrementally. HAT is an extension of HT in which the drift detection algorithm ADaptive WINdow (ADWIN) (Bifet and Gavaldà 2007) is used for signalling changes, which triggers updates to the tree model. Adaptive Random Forest is an adaptation of the original Random Forest (Breiman 2001) algorithm to streaming data with the addition of drift detectors and a weighting function based on models prediction performance. NB is based on the Bayes' theorem and it is called 'naive' as it assumes independence of the features. NB is straightforward to train incrementally since it does not add structure to the model. kNN is the classic lazy learning approach adapted to stream mining via a windowing approach, i.e. only the latest instances (stored in the window) are used for predictions. NC always predicts the next label as the latest ground-truth it had access to. This baseline model outperforms many classifiers when the data stream exhibit temporal dependencies (Zliobaite et al. 2015).

Finally, the MC baseline simply predicts the majority class. In our experiments, the algorithms that are most likely to adapt faster are HAT and ARF, since both include strategies for detecting and adapting to concept drifts.

We briefly discuss each dataset below and present the results, per dataset, in the following subsections.

## 5.1 Reference data

*Hyperplane* We start with a synthetic generator often used to assess classification algorithms (Gomes et al. 2017; Masud et al. 2011) simulating incremental concept drifts. This generator is based on the hyperplane generator (Hulten et al. 2001). A hyperplane is a flat,  $n - 1$  dimensional subset of that space that divides it into two disconnected parts. It is possible to change a hyperplane orientation and position by slightly changing its relative size of the weights  $w_i$ . This generator can be used to simulate time-changing concepts, by varying the values of its weights as the stream progresses. Formally, a hyperplane in a  $n$ -dimensional space is the set of points  $x$  that satisfy Eq. 1 where  $x_i$  is the  $i^{\text{th}}$  coordinate of  $x$ .

$$\sum_{i=1}^n w_i x_i = w_0 \quad (1)$$

In binary classification, instances where  $\sum_{i=1}^n w_i x_i \geq w_0$  are labelled positive, and instances for which  $\sum_{i=1}^n w_i x_i < w_0$  are labelled negative. This generator can be used to simulate incremental concept drifts, by varying the values of its weights as the stream progresses (Bifet et al. 2011b).

The Hyperplane data represents the case of streams missing the notion of time. Therefore, first the time of every instance is set to instance index in this dataset. Next, labels from original instances  $\mathcal{S}_a$  are removed and labelled instances  $\mathcal{S}_b$  are added at  $t(\mathcal{S}_b) = t(\mathcal{S}_a) + 100$ . Hence, for this data stream we set the latency of true label to a fixed value. As a consequence the stream includes pairs  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}, \dots, \mathcal{S}_b = \{(\mathbf{x}_k, y_k)\}$ .

*LED* The LED data set simulates both abrupt and gradual drifts based on the LED generator, early introduced in Breiman (2017). This generator yields instances with 24 boolean features, 17 of which are irrelevant. The remaining 7 features corresponds to each segment of a seven-segment LED display. The goal is to predict the digit displayed on the LED display, where each feature has a 10% chance of being inverted. To simulate drifts in this data set the relevant features are swapped with irrelevant features. Concretely, we parametrize 3 abrupt drifts each with an amplitude of 3.75k up to 7.5k instances and centered at the 3.75k, 7.5k and 15k instance, respectively. The first drift swaps 3 features, the second drift swaps 5 features, and the last one 7 features. Furthermore, the delay for LED was set to 1000 instances.

*Agrawal* The synthetic generator introduced in Agrawal et al. (1993), often referred to as AGRAWAL, produces data streams with six nominal and three continuous attributes. As commented in Bifet et al. (2011a), the synthetic data describes hypothetical loan

applications. There are ten different functions<sup>2</sup> that map instances into two different classes. It is possible to simulate concept drift by changing the function. For example, the first function assign instances to class A according to age:  $age < 40 \vee age \leq 65$ ; while the seventh function uses the salary, commission and loan attributes to assign instances to class A:  $(0.67 \times (salary + commission) - 0.2 \times loan - 20000) > 0$ . A perturbation factor is used to add noise to the data. This factor changes the original value of an attribute by adding a deviation value to it, which is defined according to a uniform random distribution. In our experiments with AGRAWAL, we simulate 4 gradual concept drifts. The size of the window of change for each drift was set to 400; the amount of instances per concept was set to 2000; the functions (concepts) vary from 1 to 5; the total amount of instances was set to 10, 000; and the delay was fixed to 1000.

*Airlines* The airlines dataset comes from Data Expo 09<sup>3</sup> and contains flight departure and arrival records for all commercial flights within the USA. A part of this dataset constrained to flights arriving to Atlanta International Airport (ATL) has been used. Only departure and arrival events for flights not cancelled and not diverted were included in our analysis. The objective is to predict whether a flight will arrive to ATL before time, on time or delayed based on instance data describing the flight and including features such as distance, planned departure and arrival time, origin airport and carrier. We assume that the instance data becomes available at planned departure time. Moreover, delayed label becomes available at true arrival time. Hence, the time between obtaining the instance data  $t(\mathcal{S}_i)$  and the time of receiving its true label  $\tilde{t}(\mathcal{S}_i) = t(\mathcal{S}_i) + \Delta t_i$  is the time between planned departure and true arrival. Not surprisingly, the latency individual labels arrive with strongly varies. Approximately two hours being the most frequent value, but some  $\Delta t_i$  values exceeding 13 hours. Before true label of an instance arrives evolving predictions for the instance are expected to be made. We expect temporal dependencies to occur in the data.

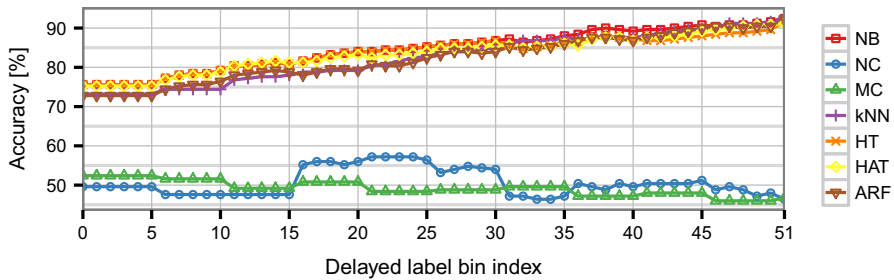
*Electricity* The Electricity dataset, made available by J. Gama<sup>4</sup>, contains class labels annotating energy price changes (UP or DOWN). Further details on Electricity data can be found *inter alia* in Bifet et al. (2013). Based on the raw Electricity data, we develop a stream of instances reflecting frequent practice in power systems of performing 24-hour-ahead prediction (Kuo et al. 2014; Srinivasan et al. 1995). More precisely, we create out of the raw instances including features such as energy prices and demand, a stream of instances each containing these features for 24 time periods of day  $d$ . The objective is to predict the price change (UP or DOWN) during  $i$ th period of day  $d + 2$  i.e. at least 24 hours ahead. Hence, we assume that after the midnight starting day  $d + 1$ , all instances containing day  $d$  data become available. Next, the true label which arrives on the day  $d + 2$  after  $i \times 30$  minutes contains actual price change observed on the  $d + 2$  day in the  $[(i - 1) \times 30, i \times 30]$  period,  $i = 1, \dots, 48$ . Therefore, the delay that every true label arrives with varies approximately between one and two days.

*Forest cover type* The forest cover type dataset (Blackard and Dean 1999) contains 581, 383 instances each of them representing one of 7 different forest cover types. The

<sup>2</sup> See Bifet et al. (2011a) for the complete list of functions.

<sup>3</sup> <http://stat-computing.org/dataexpo/2009/>.

<sup>4</sup> [http://www.inescporto.pt/~jgama/ales/ales\\_5.html](http://www.inescporto.pt/~jgama/ales/ales_5.html).



**Fig. 3** Prediction accuracy for individual bins (hyperplane data)

cover types are  $30 \times 30$ -meter cells obtained from the US Resource Information System (RIS), which are represented by 10 numeric and 44 binary features. This dataset does not contain a “natural” delay, therefore we simulated the delay using 1000 instances.

## 5.2 Experimental results

The primary objective of this section is to investigate the insights provided by the evaluation measures proposed in this study. All experiments were performed in the Massive Online Analysis (MOA) framework (Bifet et al. 2011b). For each dataset, the results were produced by running Algorithm 1 varying the learning algorithm. Table 1 provides an overview of the results and characteristics of the datasets, including the number of labelled instances  $|\Delta\Omega(T)|$  after which binned performance is reported. In all experiments,  $K = 10$  was used to ensure a balance between the computational cost of polling models for refined predictions and additional insight arising from these predictions. Furthermore,  $B = 50$  was applied to divide the periods between receiving individual unlabelled instances and their delayed labels into 50 sub-periods and produce smooth performance trends. How  $K$  and  $B$  values can be set, and their impact on computational cost and binned performance visualization will be discussed in Sect. 6.

### 5.2.1 Hyperplane data

Figure 3 depicts the binned accuracy  $\mathcal{A}_P(T, b)$  for hyperplane data. It is worth noting that Fig. 3 shows the results of the evaluation performed in the initial stage of stream processing when the scale of concept drift is relatively large. Furthermore,  $\mathcal{A}_P(T, b)$  values for individual bins are provided in Table 1.

It follows from Fig. 3 and Table 1 that the most accurate initial predictions are provided by NB, HT, HAT, and ARF. However, the accuracy of additional predictions made periodically by NB, HT, HAT, kNN and ARF while waiting for the true label grows over time, while the baseline models NC and MC maintain more stable results. This improvement in accuracy is due to exploiting new labelled instances that reflect incremental concept drift and provide a basis to move the decision border. The question arises whether such almost linear accuracy improvement from  $\mathcal{A}_P(T, 0)$ , i.e., initial

**Table 1** Performance evaluation for different bins

Data $ \Omega(T)  \times \dim(x_i)$	$A_P(T, b)$	NB	NC	MC	kNN	HT	HAT	ARF
Hyperplane 250 x 2	$b=0$	<b>75.60</b>	49.60	52.40	73.20	<b>75.60</b>	75.20	72.80
	$b=45$	<b>90.80</b>	51.20	48.00	89.60	88.00	89.20	90.00
	$b=B + 1$	92.40	46.40	46.80	<b>92.80</b>	91.60	91.20	92.40
Airlines 6227 x 10	$b=0$	47.53	39.68	<b>48.45</b>	44.66	45.59	45.25	47.47
	$b=45$	<b>48.92</b>	40.13	48.48	47.63	47.13	46.76	48.34
	$b=B + 1$	49.24	42.52	48.58	<b>50.33</b>	47.34	46.92	48.76
Electricity 44204 x 149	$b=0$	53.44	49.78	57.51	<b>64.36</b>	56.93	59.54	57.00
	$b=45$	53.33	55.33	57.51	<b>59.68</b>	56.13	59.37	54.42
	$b=B + 1$	54.34	<b>85.37</b>	57.56	71.58	61.59	64.99	68.27
Cover type 24500 x 54	$b=0$	<b>61.09</b>	34.20	31.16	52.75	53.37	53.51	51.12
	$b=45$	68.36	43.28	38.13	<b>71.02</b>	65.38	67.24	69.42
	$b=B + 1$	70.92	70.51	39.62	79.94	70.85	73.09	<b>80.87</b>
Agrawal 10000 x 9	$b=0$	56.47	53.18	50.11	54.39	<b>56.80</b>	56.07	53.29
	$b=45$	60.51	51.28	52.11	59.55	64.60	<b>65.36</b>	57.79
	$b=B + 1$	61.07	51.95	52.37	60.35	66.42	<b>66.64</b>	58.04
LED 20000 x 24	$b=0$	53.74	9.75	10.58	56.66	<b>59.56</b>	59.13	56.17
	$b=45$	58.89	10.59	10.66	61.77	<b>65.53</b>	65.35	61.59
	$b=B + 1$	59.61	9.45	10.65	62.10	<b>66.27</b>	65.98	62.13

Results for the best method for each bin and data set are shown in bold

prediction accuracy to test-the-train accuracy  $A_P(T, B + 1)$  takes place also for other data sets.

### 5.2.2 Airlines data

The Airlines dataset depicts a realistic scenario where the labels are presented with varied label latency, thus before discussing the results we elucidate the implications of such configuration. As a consequence of major differences among  $\Delta t_i$  values, the set of instances still waiting for their delayed labels after  $\Delta t$  is  $\Delta t$ -dependent. Hence, it remains impossible to analyse the accuracy of individual stream classification techniques over all instances as a function of time that elapsed since  $t(S_i)$ . In the analysed case, time-based aggregate accuracy could be developed only for the period of 19 mins, when all instances are awaiting their true labels. Once the analysis is extended beyond this point, the set of instances over which any aggregate performance indicators would need to be calculated would vary for individual points on time axis. As a consequence, the aggregate performance changes could be caused by the changes in instance set. As an example, class distribution can be different for instances with major label latencies compared to short latency instances. Hence, under major differences of label latency times, the accuracy defined as a function of  $\Delta t$  can be misleading. The accuracy trends observed in such cases may be caused by the changes in  $\{S_i :$

$\Delta t_i > \Delta t\}$  rather than the ability of stream classification method to model the process of interest.

Therefore, let us analyse binned accuracy  $\Lambda_P(T, b)$  for the airlines data set. Unlike in time-based accuracy, in binned accuracy, the predictions made for all instances contribute to every point in the accuracy trend plot. Figure 4 depicts binned accuracy  $\Lambda_P(T, b)$  for the data set and a selection of stream classification methods previously analysed in the case of hyperplane data.

Bin 51 is reserved for the prediction made in test-then-train mode, i.e., immediately before getting the true label. It follows from the figure that kNN, followed by NB and ARF, provide the highest accuracy then. However, in the context of flight delays, the closer to the label arrival the less relevant the prediction is, i.e., if we accurately predict that a flight will be delayed minutes before it lands it is less relevant than a model with an average performance at the time the flight departs.

It can be observed that kNN, HT and HAT algorithms offer similar accuracy of predictions made when instance data becomes available, i.e., for  $b=0$ . However, when predictions preceding arrival of a true label are made, they can take into account most recent labels of other instances. Moreover, some algorithms, such as kNN, which operates on a sliding window of recent instances can exploit temporal dependencies in the data (Zliobaite et al. 2015). In the case of airlines data, this means that some techniques may identify periods of major delays and tune models to predict delays more accurately. In particular, NB, kNN, and ARF, clearly benefit from the evolution of the models based on most recent data. This is unlike MC, which yields the same accuracy irrespective of whether prediction is made at the time of departure or shortly before receiving true label. At the same time, Fig. 4 extends the comparison of test-then-train accuracy by showing that HT, kNN and NB largely differ in their ability to provide accuracy gains in the period between getting instance data and receiving its true label.

### 5.2.3 Electricity data

Let us investigate the results of similar analysis for Electricity data. Figure 5 shows the accuracy  $\Lambda_P(T, b)$  of the same set of methods on the electricity instances, described above. Test-then-train results show that NC classifier yields largest accuracy, which shows major temporal dependence in the data successfully exploited by NC which simply predicts the next label to be the same as the last true label received. However,

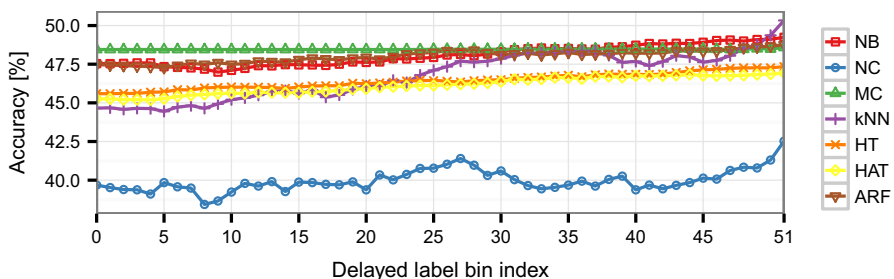


Fig. 4 Prediction accuracy for individual bins (airlines data)



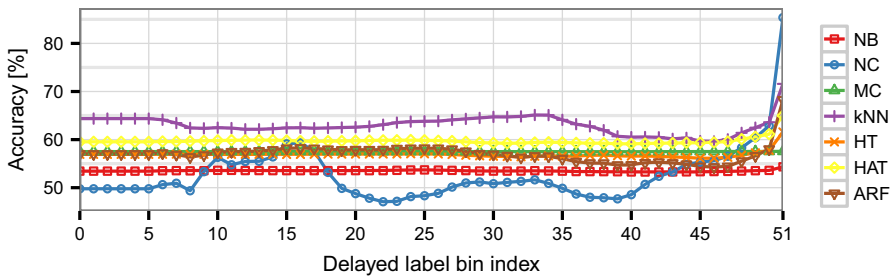


Fig. 5 Prediction accuracy for individual bins (electricity data)

once the prediction under delayed label setting is an objective, it should be performed within the period preceding true label arrival, not immediately before receiving it. The binned accuracy trends shown in Fig. 5 reveal that it is kNN method that for  $b = 0$  yields superior accuracy i.e. the accuracy higher than in the case of the remaining methods. However, when predictions made in the periods  $[t(S_i), \tilde{t}(S_i)]$  are considered, kNN accuracy varies. In particular, it is lower than the accuracy of HAT during some bins. This is caused by periodical changes in the electricity data that reflect time of the day and week. Hence, a sliding window of recent instances that kNN relies on, may contain instances reflecting similar conditions as the conditions for which prediction is performed or not. Continuous re-evaluation answers the question whether additional labelled data can be used by the learner to improve on its initial predictions or not. kNN is an example of a technique that can both benefit from newer labelled instances for some stream mining tasks (hyperplane data) or not (electricity data at some bins) depending on to what extent these instances contribute to valid reasoning for the instance of interest.

Finally, let us investigate which of the methods yields highest accuracy models. It follows from Table 1 that the accuracy of the models and which of them offers the best performance largely varies. Importantly, the best model in terms of its ability to correctly predict the value of the feature of interest at the first time of receiving it ( $b = 0$ ) is not the same as the one consuming additional data possibly reflecting concept drift and applied at  $b = B + 1 = 51$ . Interestingly, it is the value of binned accuracy in the intermediate period, here shown for  $b = 45$  that reveals whether a modelling technique provides improved prediction capabilities within non-negligibly long period before true label becomes available or not.

Figure 6 presents the prediction accuracy throughout the execution for  $b = 0$ ,  $b = 25$ ,  $b = 45$  and  $b = 51$ , in particular, this plot serves the purpose of observing some of the results depicted in Fig. 5 over time for kNN and HAT. We can observe in Fig. 6 that the accuracy of both algorithms is at its best when observing them at  $b = 51$ .

### 5.2.4 Forest cover type data

Figure 7 presents the results of the continuous evaluation for forest cover type. At time of the instance data generation, the most accurate model is NB, which is an interesting

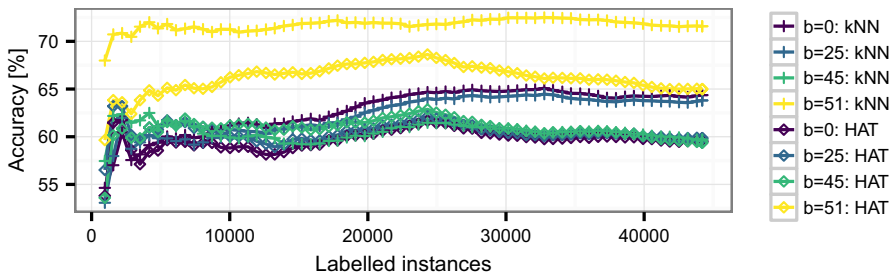


Fig. 6 Prediction accuracy at different stages of stream processing (electricity data)

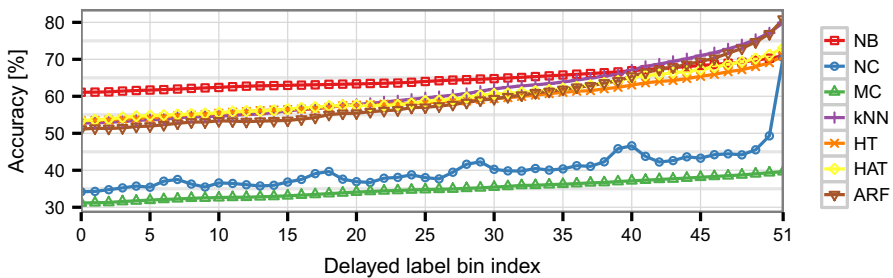


Fig. 7 Prediction accuracy for individual bins (forest cover type data)

result, especially as we observe that the more complex models, such as HAT and ARF, can only outperform NB once we start considering the performance closer to bin 40. In other words, these results indicate that a simple NB model would be the most appropriate model if we are only interested in the quality of the first prediction. Furthermore, ARF and kNN only obtain reasonably good results for  $b = B + 1$ , i.e., their accuracies rose as the bin index increase. In ARF, this might be due to how the algorithm relies on having access to labelled instances immediately as its feature drift detection method is most useful when the feedback (true labels) is immediately available.

### 5.2.5 LED data

In Fig. 8 we can observe how the synthetic stream LED combined with latency make the evolution of the models reasonably simple. Besides the baselines NC and MC, every model shows an increase in accuracy as we focus the evaluation closer to bin  $b = B + 1$ . This evolution is not trivial in other synthetic streams, such as AGRAWAL. It is interesting to observe how continuous assessment reveal this ‘simple’ evolution characteristic in LED, something that would not be observable just evaluating at the first prediction and test-then-train.

### 5.2.6 Agrawal data

Figure 9 shows how accuracy changes under periodic re-evaluation for Agrawal data stream for which labels arrive with fixed latency of 500 instances. It can be observed

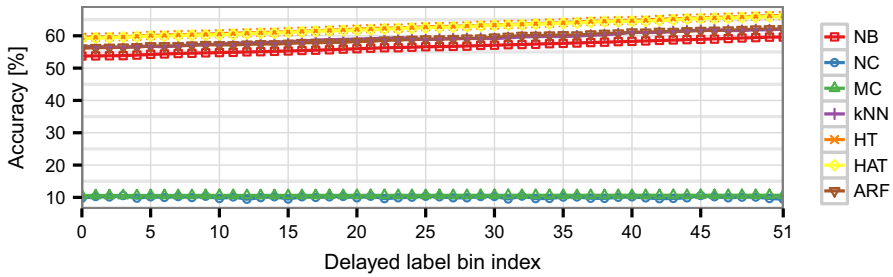


Fig. 8 Prediction accuracy for individual bins (LED data)

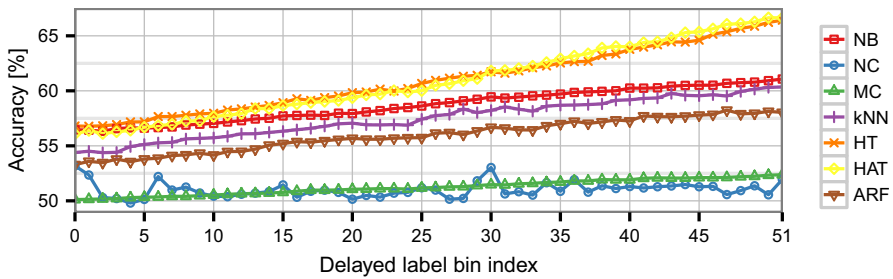


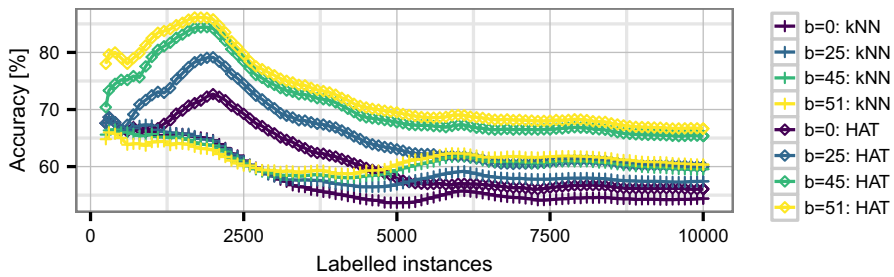
Fig. 9 Prediction accuracy for individual bins (Agrawal data)

that additional predictions made before true label arrives provide substantially increasing accuracy. This is in spite of the complexity of the stream. Hence, additional predictions can yield performance gains even under complicated streaming scenarios, involving multiple concept drift periods. In particular,  $\Delta_P(T, 45)$  provided by HT and HAT shows that the predictions available in the period preceding the arrival of true labels are significantly more accurate than the best initial predictions. While the use of continuous re-evaluation is not limited to improving initial predictions, it may yield benefits also of this category even for complex concept drifting streams.

In Fig. 10 we can observe how HAT obtains the best performance in comparison to itself and kNN when we the predictions taken into account are those closer the arrival of the true class label (i.e.,  $b = 45$  and  $b = 51$ ). However, we can also observe that kNN starts improving in comparison to itself, e.g., in  $b = 45$ , while HAT only decreases in accuracy after the first drift around instance 2000. This shows how complex accuracy evolution for concept drifting data streams can be.

### 5.2.7 Discussion

In summary, in these experiments we observe two main aspects of continuous re-evaluation. First, by evaluating intermediary predictions between the time of instance being presented and its label arrival we can better observe the model classification performance in these intermediary stages. Consistently,  $b = B + 1$  (i.e. evaluating only the latest prediction before the label arrivals) depicts optimistic accuracy results comparing the same learner across  $b = 0$  and  $b = 45$  in all experiments, conversely



**Fig. 10** Prediction accuracy at different stages of stream processing (Agrawal data)

$b = 0$  (i.e. evaluating only with the first prediction made) can be too pessimistic. The intermediary analysis given by  $b = 45$  gives a clearer view of how the models perform when pooled several times for predictions. Second, the experiments confirm that additional predictions made with newer models may gradually provide higher accuracy than initial predictions. However, the newer model may also turn out to be less appropriate than the one existing at the time of initial predictions, which is shown in the electricity case. Both airlines and electricity data sets show that no linear trend in accuracy change between initial  $b = 0$  and test-then-train  $b = B + 1$  predictions can be assumed. This shows that periodic predictions and their continuous evaluation, i.e. both components of continuous re-evaluation are needed. In this way, the method providing potential accuracy improvement can be determined.

It is also interesting to note that a method operating with a sliding window i.e. kNN typically provides higher accuracy shortly before true label arrives e.g. at  $b = 45$  subperiod than at the time of making first prediction. However, Electricity data shows that in case major changes such as periodic changes occur in the data, the accuracy at  $b = 45$  can also be substantially lower than at the time of first time prediction. In other words, the content of instance buffer, which kNN relies on, may be less suitable to derive true classification border e.g. at bin  $b = 45$  i.e. shortly before true label arrival than the content of the instance buffer available at the time of first prediction. This shows that attempts to predict which group of stream mining methods is likely to provide better predictions at different times between first time prediction and true label arrival can be successful to some extent only.

These results show that additional insight into the way newer labelled instances influence a model is provided. This may suggest adaptive techniques of selecting instances for instance-based classifiers, going beyond sliding window technique, which directly follows from the experiments with electricity data. In this way, continuous re-evaluation can be used to guide the development or tuning of classification methods. Hence, also under scenarios where only initial predictions are used to trigger immediate actions, e.g. to block suspicious network connections in intrusion detection systems, the methods we propose can be used.

Furthermore, experiments with hyperplane data show potential accuracy improvements, which occur in spite of limited number of labelled instances causing model updates. This shows that continuous re-evaluation can be used to reveal for which industrial use cases, initial predictions of events of interest, such as device failures,

can be refined based on newly arriving labelled data. This includes the scenarios when machine learning is used to predict whether a device will fail during the next maintenance period. More precisely, during this period, as defined in Algorithm 1 new predictions are periodically published, which may change these previously made for a device of interest. These new predictions can be based on the knowledge arising from newly available labelled instances including failures of other devices in this period. Since in some industrial settings labelled data is very limited, the ability to exploit the benefits of on-line learning by updating first predictions with more accurate predictions based on newly available labelled data is of particular interest.

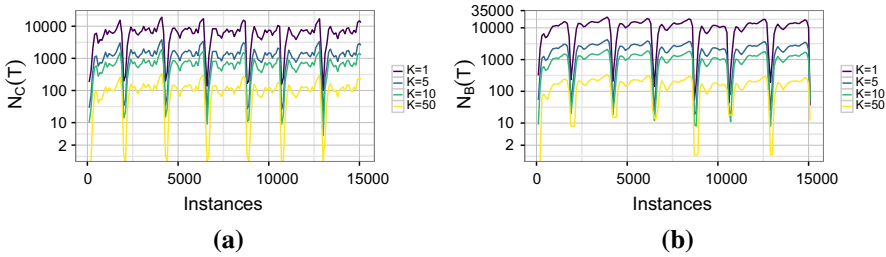
Let us note that as defined in Sect. 3 we assume that we rely on the processing on time feature, which is represented by a numeric timestamp (for real data streams) or by an instance index (for synthetic streams). In particular, bin periods are calculated based on the proportion of time between receiving instance data and receiving its true label. Hence, continuous re-evaluation can be used for data streams, including synthetic data streams, but also for time series data. In the latter case, any time series data that among its features include a timestamp and instance identifier can be used with continuous re-evaluation. Importantly, assigning timestamps and unique identifiers is a recommended industry practice for large volume realtime data systems (Marz and Warren 2015). Out of the data used in this study, Electricity and Airlines data illustrate the use of time series data with the evaluation method we propose and additional insight into the performance of evolving models this method provides.

Last but not least, results for diverse data streams including airlines, electricity and cover type show that during the period of waiting for true label, there can be no single model that provides most accurate predictions in all bins. This shows that hybrid methods, including the methods that switch between the use of different modelling techniques such as kNN and Hoeffding trees can yield the best performance when entire periods of waiting for true labels are concerned.

## 6 Analysis of hyper-parameters

### 6.1 Computational cost and memory consumption

Continuous re-evaluation relies on additional predictions, the frequency of which depends on the value of hyper-parameter  $K$ . Hence, the question arises how frequently these predictions should be made. First of all, let us observe that under online learning paradigm, every labelled instance is used to trigger potential model updates. For some models, such as instance-based classifiers, such a change occurs every time new labelled instance is used to update the model. Importantly, changes to a model may occur because of concept drift or may be caused by incremental updates aiming at improving the approximation of true decision borders in stationary cases. In general, when no assumptions regarding the stationarity of the process are made and different online learning methods are considered, generating new predictions every time new true label arrives i.e. setting  $K = 1$  could be considered. However, this could cause major computational overhead.



**Fig. 11** The number of predictions made **(a)** and the number of predictions in memory buffer awaiting their true labels **(b)**

Before we analyse the impact of  $K$  on the computational complexity of re-evaluation, let us note that we will focus on the predictions made for  $b \in \{1, \dots, B\}$ . This is because initial predictions ( $b = 0$ ) and test-then-train predictions ( $b = B + 1$ ) are made under delayed labelling scenario, irrespective of whether continuous re-evaluation is used or not. Furthermore, there is always exactly one prediction of each of these two types for any labelled instance.

To analyse the number of predictions made, let us define the number of calculated predictions over a sliding window of 100 instances  $N_C(T) = \sum_{i=T-100}^T C(i)$  while  $C(i)$  is calculated in Algorithm 1. Furthermore, let  $L(T)$  denote all instances stored in memory buffer and awaiting their true labels at time  $T$ , and let  $N_B(T) = \sum_{k \in L(T)} \text{card}(\{P(k) : P(k).b = -1\})$  denote the number of additional predictions made for these instances. Hence,  $N_C(T)$  shows the number of extra predictions triggered by continuous re-evaluation, whereas  $N_B(T)$  reveals memory allocation needs of the method.

Figure 11 shows both resource use indicators for airlines data. It follows from the figure that the number of predictions stored is closely related to the number of predictions made. Both indicators drop significantly at some periods, which is because of a significantly lower number of overnight flights, which reduces substantially  $\text{card}(L(T))$ . Since  $N_C(T)$  and  $N_B(T)$  are influenced by  $\text{card}(L(T))$ , we propose  $N_R(T) = \frac{N_C(T)}{|L(T)|}$  i.e. the average number of additional predictions per a labelled instance until time  $T$  to be analysed. Figure 12 shows how  $N_R(T)$  changes depending on  $K$  value. It follows from the figure that  $N_R(T)$  is stable. For  $K = 10$  on average approx. 10 additional predictions are made for every labelled instance. It is important to note that the number of predictions made for labelled instance  $\{(\mathbf{x}_k, y_k)\}$  is no larger than  $\frac{\text{card}(\{S_b = \{(\mathbf{x}_j, y_j) : t(\{(\mathbf{x}_k, ?)\}) \leq t(S_b) \leq t(\{(\mathbf{x}_k, y_k)\})\})}{K}$ . Hence, by calculating the average number of labelled instances that arrive between initial and test-then-train prediction is made, we can estimate the number of additional predictions arising from  $K$  value under consideration. In this way and through the plots such as Fig. 12,  $K$  value can be set. In the analysed case of the airlines data  $K = 10$  on average provides one new prediction for approx. every 10% of the period of waiting for a true label, which can be considered optimal. Importantly, all  $N_R(T)$  plots are very stable in spite of major volatility of the number of instances arriving at different times of the day.

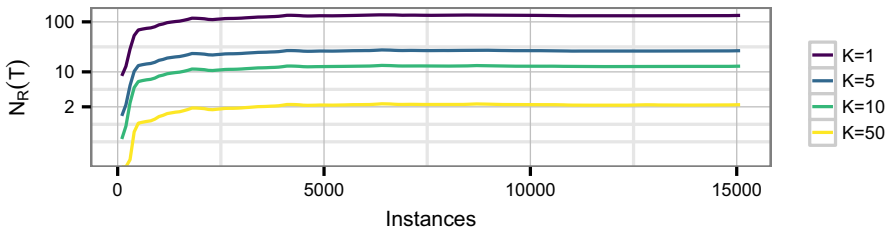


Fig. 12 Computational cost. The average number of re-predictions per a labelled instance since the beginning of the stream

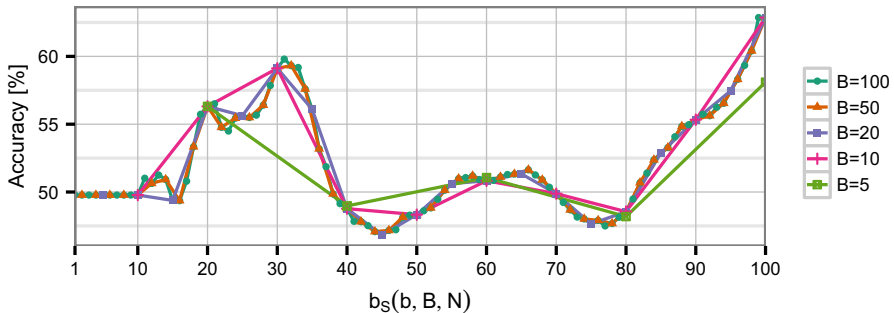


Fig. 13 The impact of the number of bins  $B$  on the calculation of performance indicators. Electricity data

### 6.2 Performance aggregation settings

The value of  $B$  parameter controls the number of bins, which are used to aggregate model performance. It is important to note that how many predictions are made and when is not driven by  $B$  value. The role of  $B$  is to control the calculation of performance indicators on the top of the same predictions and true labels.

To compare how different  $B$  values influence the aggregation process, let us define standardised bin index as  $b_S(b, B, N) = b \frac{N}{B}$ . In this way, performance indicators such as accuracy or kappa attained for individual  $b = 1, \dots, B$  can be referred to the same range  $[1, N]$  of  $x$  axis, irrespective of  $B$  value. Figure 13 shows the role of various  $B$  settings by presenting the accuracy  $\Delta_P(T, b)$  for electricity data set. We selected results for NC method to be shown in Fig. 13 as these results are particularly variable. The figure includes data for  $b = 1, \dots, B$  and skips the data of remaining two bins, since bins  $b \in \{0, B + 1\}$  contain the same values (for the same data stream and stream mining method), irrespective of  $B$  value. As noted before, individual  $[b, \Delta_P(T, b)]$  points were mapped to  $[b_S(b, B, N), \Delta_P(T, b)]$ . Moreover,  $N = 100$  was applied. Not surprisingly, the higher  $B$  value, the more smooth accuracy trend is revealed as it relies on a larger number of accuracy points, each representing binned performance.  $B < 20$  yields only coarse approximation of the performance evolution of a model during the periods between individual unlabelled instances and their true labels arrive.

Furthermore, only a minor difference between  $B = 50$  and  $B = 100$  can be observed in Fig. 13. This confirms that for continuous re-evaluation  $B \in [50, 100]$  is justified, as any  $B$  in this period provides precise estimation of the trends in the per-

formance indicator changes. Moreover, under  $B = 50$ , every bin represents 2% of the length of the periods  $[t(\{\mathbf{x}_k, ?\}), t(\{\mathbf{x}_k, y_k\})]$ , which provides precise estimation of performance changes during such short sub-periods. This confirms that  $B \in [50, 100]$  can be applied to different stream mining settings.

## 7 Extensions of the base continuous evaluation

Let us observe that continuous evaluation defined in Algorithm 1 and discussed above can be extended to other related stream mining tasks, some of which are outlined below.

### 7.1 Regression tasks

#### 7.1.1 The applicability of continuous evaluation

Continuous evaluation is applicable both to classification and regression tasks. In both cases, delayed labels may result in verification latency. Hence, the accuracy of initial predictions made at the time of observing an instance for the first time may differ from the accuracy of test-then-train prediction also for regression tasks. Similarly, the accuracy of predictions made in the periods preceding arrival of individual delayed labels may change. Therefore, continuous evaluation can be used to reveal accuracy evolution also for these regression tasks for which verification latency occurs.

The only change, which has to be applied to adapt Algorithm 1 to regression tasks is to make it match continuous rather than discrete true and predicted labels. Furthermore, other than  $d_D()$  functions can be used to obtain averaged prediction. In the case of regression tasks, we propose the use of weighted average to convert possibly many predictions made during bin period into a single averaged prediction  $d_A(U(S_i, b))$ . More formally,  $d_A(U(S_i, b)) = \sum_{j=2}^{|U(S_i, b)|} \frac{y_{j-1} \times (t_j - t_{j-1})}{t_{|U(S_i, b)|} - t_1}$ ,  $b = 1, \dots, B$ . In this way predictions displayed over longer periods of time have a larger impact on averaged prediction.

#### 7.1.2 Sample results

As in the case of classification, all experiments described in this subsection were performed with MOA framework (Bifet et al. 2011b). The results reported below were produced by running Algorithm 1 every time with a different learning algorithm. Bike sharing data was used for all experiments.

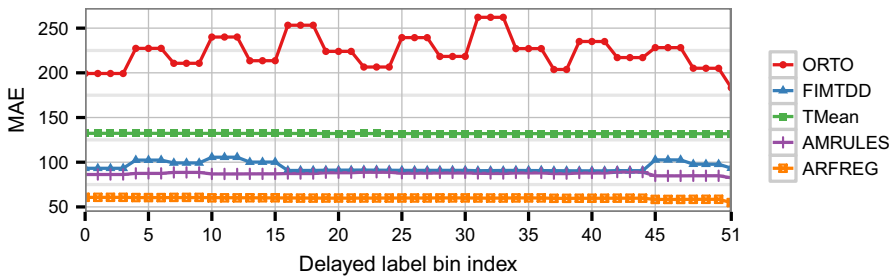
*Bike sharing data* The bike dataset (Fanaee-T and Gama 2013) includes 2 years (2011 and 2012) worth of a bike-sharing service from Washington D.C., USA. The goal is to predict how many bikes in total will be rented in the next hour using weather and temporal data (e.g., time of the day, the day of the week and so forth). Some features from the original data were removed to avoid data leakage (e.g., “registered” and “casual”), as their sum converges to the target variable (i.e., “cnt”). The delay for



**Table 2** Performance evaluation for different bins: regression task

Data $ \Omega(T)  \times dim(x_t)$	$A_p(T, b)$	ORTO	FIMTDD	TMean	AMRules	ARFREG
Bike sharing 17379 x 12	$b=0$	199.27	93.21	132.30	86.27	<b>60.69</b>
	$b=45$	228.20	102.58	131.81	84.75	<b>58.37</b>
	$b=B + 1$	183.22	93.35	131.80	82.55	<b>54.85</b>

Results for the best method for each bin are shown in bold



**Fig. 14** Mean absolute error for individual bins (bike sharing data)

bike sharing was set to one week, however, since there is no timestamp available in the data we used a delay of 168 instances, which corresponds to 24 hours  $\times$  7 days.

Table 2 provides a summary of bike sharing data used for the experiments including the number of labelled instances  $|\Omega(T)|$  after which binned performance is reported below. This is combined with Mean Absolute Error (MAE) of the models developed with each of the regression methods.

The regression methods used in the experiments include decision trees, rule-based models, a random forest model, and a baseline model of Target Mean (TMean). The first regression tree used is the Fast and Incremental Model Trees (FIMTDD) (Ikonovska et al. 2011a) algorithm, which grows the tree model similarly to the Hoeffding Trees (Domingos and Hulten 2000) algorithm. FIMTDD also includes a change detection scheme that periodically signals and adapts subbranches of the tree where significant variance increases are observed. The second tree algorithm used is ORTO (Ikonovska et al. 2011b), which also grows trees incrementally as instances arrive. Yet, ORTO also includes the notion of “option” nodes, which allow an instance to follow all the branches available in a tree node. The Adaptive Model Rules (AMRules) (Almeida et al. 2013) was designed to learn both ordered and unordered rule sets from a data stream. AMRules associates each of its rules with a Page-Hinkley drift detector to make it possible to detect and adapt to concept drifts by pruning the rule set. Finally, Adaptive Random Forest for Regression (ARFReg) (Gomes et al. 2018) is an ensemble method based on the ARF (Gomes et al. 2017) algorithm for classification tasks. The main difference is that the base learner used in ARFReg is based on the FIMTDD algorithm instead of Hoeffding Trees. As a consequence, when using ARFReg, one can choose between using the external drift detection methods, as in ARF, or relying on the internal drift detection methods within FIMTDD. Experiments in Gomes et al. (2018) indicate that the latter is a more sustainable approach.

Figure 14 presents the results per bin index for the Bike sharing dataset, where we can observe that most of the learners produced better performance than the baseline TMean, except for ORTO. Interestingly, there is not much variation in MAE as the bin index varies, e.g., ARFReg outperforms all other methods for every bin. Moreover, MAE for ARFREG is lower for test-then-train predictions ( $b = B + 1$ ) than for first time predictions ( $b = 0$ ), which is in line with previously observed trend of the higher accuracy of predictions made shortly before true label arrival.

## 7.2 Evolving instances

Let us observe that even though we assume  $\mathbf{x}_k$  in Algorithm 1 to be fixed, it may be defined as  $\mathbf{x}_k(t)$  i.e. it may change with time. In the case of the reference flight delay prediction task we already assumed that  $\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}$  could include weather conditions at departure time. It is natural to observe that every time new prediction is made for  $\mathcal{S}_a$ , it could use  $\mathbf{x}_k(t)$  as an input i.e. evolving instance data rather than static data available at the time of making first prediction  $t(\mathcal{S}_a)$  for this instance. In the case of flight prediction task, this could mean using weather conditions at the time of making periodical prediction rather than weather conditions at departure time as a part of input data for the classifier. Hence, in Algorithm 1 it is enough to use  $\mathbf{x}_k(t)$  as an input for  $h_i()$  model when periodic and test-then-train predictions are made to consider evolving values of input features rather than constant  $\mathbf{x}_k$  data.

Importantly, the changes in instance data could also cause changes in predictions. Hence, a sequence of predictions made for an instance  $\hat{y}_k^{t_1}, \hat{y}_k^{t_2}, \dots; t_i \leq \tilde{t}(\mathcal{S}_a)$  could contain different predictions not only because of the changes of  $h$  model, but also because of the changes in instance data  $\mathbf{x}_k$ .

To sum up, by replacing  $\mathbf{x}_k$  with  $\mathbf{x}_k(t)$ , we can apply continuous re-evaluation to these classification tasks in which instance data evolves between making initial prediction and receiving true label.

## 7.3 Weighted binned performance

In some classification and regression tasks, it may be important to create a ranking of different methods in terms of their ability to create and evolve models of high accuracy. In such cases, based on individual performance indicators comprising on the vectors of  $[\Lambda(T, 0), \Lambda(T, 1), \dots, \Lambda(T, B + 1)]$ , summary performance  $\Psi(T) \in \mathbb{R}$  can be developed. Furthermore, in some use cases, predictions made in different bins can be of different importance. As an example, predictions made in test-then-train mode may be considered not important in the evaluation or less important than initial predictions.

Let us propose to apply weight vector  $\mathbf{w}$  to aggregate performance indicators calculated for individual bins. Furthermore, let us propose summary performance  $\Psi(T)$  to be a function of  $[\Lambda(T, 0), \Lambda(T, 1), \Lambda(T, B + 1)]$  providing one summary assessment of the performance of stream mining method attained for individual bins.

In the simplest case, weighted average can be used to calculate summary performance. More precisely, let us propose  $\Psi_{\mathbf{w}}(T) = \frac{\sum_{b=0}^{B+1} \Lambda(T, b) \times \mathbf{w}_b}{\sum_{b=0}^{B+1} \mathbf{w}_b}$ . Let us observe that by applying  $\mathbf{w} = [1, 0, \dots, 0]$  summary performance will be equal to the performance observed for first time predictions. Furthermore,  $\mathbf{w} = [0, \dots, 0, 1]$  yields test-then-train performance,  $\mathbf{w} = [1, 0, \dots, 0, 1]$  provides summary performance equal to the average of first time and test-then-train method performance. Furthermore,  $\mathbf{w} = [1, 1, \dots, 1]$  gives equal weights to all binned performance indicators  $\Lambda(T, b), b = 0, \dots, B + 1$ . Importantly,  $\mathbf{w}$  vector can be used to make summary performance take into account only some of the predictions e.g. only the predictions made in the second half of the period between making initial predictions and true label arrival. In this way, unique needs of individual settings can be considered.

Moreover, growing including exponential weights can be assigned to predictions made shortly before true label arrives. In this way methods providing particularly high performance indicators then can receive higher summary performance values.

## 8 Conclusions

Many applications of stream mining require non-negligible latency that the true labels arrive with to be taken into account. This results in the development of novel stream mining techniques. In this study we concentrate on the evaluation of these techniques both to select the best technique for a problem of interest and to foster the development of new techniques through increased understanding of the impact of model evolution on its accuracy in the period separating instance arrival and delayed label arrival.

Experiments were conducted with seven data sets widely used in research community, which revealed that the investigation of first time and test-then-train performance only, while important may not be sufficient to reveal the evolution of model prediction performance. The evaluation procedure proposed in this study reveals the evolution of performance and shows how the ranking of most accurate methods changes depending on varied abilities of the techniques to exploit temporal dependencies and new evidence arriving before true label becomes available. Furthermore, our study reveals that some of the evolving models when polled for further predictions before ultimate true label arrives can gradually provide more accurate prediction for the same instance data. This provides basis for further development of methods that can refine previously made predictions. One limitation of our current evaluation method is that it does not account for evolving class labels, i.e. once the true label arrives, performance indicators are calculated and possible newer versions of this label are not considered. This aspect can also be addressed when developing new stream mining methods and extending continuous re-evaluation.

For future works, one of the objectives is the use of the proposed approach to analyse the evaluation metrics evolution under evolving instances. In fact, the airlines data used in this study provides an immediate use case for it by providing the data on departure delays, which can be used to update initial schedule-based instance data and in this way contribute to the prediction performance. Furthermore, it would be interesting to use this evaluation procedure to assess the performance of semi-supervised techniques,

as it can be used to verify how intermediary predictions are affected by leveraging unlabelled instances. Moreover, hybrid methods exploiting the fact that no single online learning method may yield the best performance during entire period before true label arrival, can be developed.

**Acknowledgements** During the first stage of this research, Maciej Grzenda was both with Warsaw University of Technology and Orange Polska.

## Compliance with Ethical Standards

**Conflict of interest** The authors declare that they have no conflict of interest.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## A Formal definitions

As observed in Sect. 4, once true label becomes available, individual predictions can be linked to the appropriate bins. For every bin  $1 \leq b \leq B$  the sequence of predictions  $U(\mathcal{S}_i, b)$  available for this bin  $\mathcal{S}_i = \{(\mathbf{x}_k, ?)\}$  and period  $b$  is as follows:

$$U(\mathcal{S}_i, b) = \left( \hat{y}_k^{t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b-1}{B}}, t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b-1}{B} \right), \\ \left\{ (y_i, t_i) \in P(k) : b_i = -1 \wedge t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b-1}{B} \leq t_i < t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b}{B} \right\}, \\ \left( \hat{y}_k^{t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b}{B}}, t(\{(\mathbf{x}_k, ?)\}) + \Delta t_k \frac{b}{B} \right)$$

Furthermore,  $V(\mathcal{S}_i, b)$  denoting the pair of true and predicted label for  $\mathcal{S}_i = \{(\mathbf{x}_k, ?)\}$ , which is used to calculate performance measures is defined as follows:

$$V(\mathcal{S}_i, b) = \begin{cases} \{(y_k, \hat{y}_k) : \hat{y}_k \in R(\{(\mathbf{x}_k, ?)\}, b), y = b - 1\} & b \in \{0, B + 1\} \\ \{(y_k, \hat{y}_k) : \hat{y}_k = d(U(\{(\mathbf{x}_k, ?)\}, b))\} & b \in \{1, \dots, B\} \end{cases}$$

## References

- Abdulsalam H, Skillicorn DB, Martin P (2010) Classification using streaming random forests. *IEEE Trans Knowl Data Eng* 23(1):22–36
- Agrawal R, Imieliński T, Swami A (1993) Database mining: a performance perspective. *IEEE Trans Knowl Data Eng* 5(6):914–925
- Almeida E, Ferreira C, Gama J (2013) Adaptive model rules from data streams. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 480–492

- Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 7th SIAM international conference on data mining, April 26–28, 2007, Minneapolis, Minnesota, USA. Society for Industrial and Applied Mathematics SIAM, pp 443–448
- Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: International symposium on intelligent data analysis. Springer, pp 249–260. [https://doi.org/10.1007/978-3-642-03915-7\\_22](https://doi.org/10.1007/978-3-642-03915-7_22)
- Bifet A, Holmes G, Kirkby R, Pfahringer B (2011a) MOA data stream mining—a practical approach. Centre for Open Software Innovation COSI
- Bifet A, Holmes G, Pfahringer B, Read J, Kranen P, Kremer H, Jansen T, Seidl T (2011b) MOA: a real-time analytics open source framework. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 617–620
- Bifet A, Read J, Žliobaitė I, Pfahringer B, Holmes G (2013) Pitfalls in benchmarking data stream classification and how to avoid them. Joint European conference on machine learning and knowledge discovery in databases. Springer, Berlin, pp 465–479
- Bifet A, de Francisci Morales G, Read J, Holmes G, Pfahringer B (2015) Efficient online evaluation of big data stream classifiers. In: 21st ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 59–68
- Blackard JA, Dean DJ (1999) Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Comput Electron Agric* 24(3):131–151
- Breiman L (2001) Random forests. *Mach Learn* 45(1):5–32
- Breiman L (2017) Classification and regression trees. Routledge, New York
- Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. *IEEE Comput Intell Mag* 10(4):12–25
- Domingos P, Hulten G (2000) Mining high-speed data streams. In: 6th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80
- Fanaee-T H, Gama J (2013) Event labeling combining ensemble detectors and background knowledge. *Progr Artif Intell*. <http://dx.doi.org/10.1007/s13748-013-0040-3>
- Gama J, Rodrigues P (2009) Issues in evaluation of stream learning algorithms. In: 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 329–338
- Gomes HM, Bifet A, Read J, Barddal JP, Enembreck F, Pfahringer B, Holmes G, Abdessalem T (2017) Adaptive random forests for evolving data stream classification. *Mach Learn* 106(9):1469–1495. <https://doi.org/10.1007/s10994-017-5642-8>
- Gomes HM, Barddal JP, Boiko LE, Bifet A (2018) Adaptive random forests for data stream regression. In: Proceedings of the 26th European symposium on artificial neural networks (ESANN). pp 267–272
- Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 97–106
- Ikonomovska E, Gama J, Džeroski S (2011a) Learning model trees from evolving data streams. *Data Min Knowl Discov* 23(1):128–168
- Ikonomovska E, Gama J, Zenko B, Džeroski S (2011b) Speeding-up Hoeffding-based regression trees with options. In: International conference on machine learning. Omnipress, pp 537–544
- Kuncheva LI, Sánchez JS (2008) Nearest neighbour classifiers for streaming data with delayed labelling. In: IEEE international conference on data mining. IEEE, pp 869–874
- Kuo Ss, Lee Cm, Ko Cn (2014) Hybrid learning algorithm based neural networks for short-term load forecasting. In: International conference on fuzzy theory and its applications. IEEE, pp 105–110
- Marz N, Warren J (2015) Big data: principles and best practices of scalable realtime data systems. Manning Publications Co., Greenwich
- Masud M, Gao J, Khan L, Han J, Thuraisingham BM (2011) Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE TKDE* 23(6):859–874
- Plasse J, Adams N (2016) Handling delayed labels in temporally evolving data streams. In: 2016 IEEE International Conference on Big Data (Big Data). IEEE, pp 2416–2424
- Souza VMA, Silva DF, Batista GEAPA, Gama J (2015) Classification of evolving data streams with infinitely delayed labels. In: IEEE International Conference on Machine Learning and Applications. IEEE, pp 214–219
- Srinivasan D, Chang CS, Liew AC (1995) Demand forecasting using fuzzy neural computation, with special emphasis on weekend and public holiday forecasting. *IEEE Trans Power Syst* 10(4):1897–1903
- Žliobaitė I (2010) Change with delayed labeling: When is it detectable? In: IEEE International conference on data mining workshops. IEEE, pp 843–850

---

Žliobaitė I, Bifet A, Read J, Pfahringer B, Holmes G (2015) Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach Learn* 98(3):455–482

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.