Machine Learning (2022) 111:2011–2036
https://doi.org/10.1007/s10994-021-06099-z

Stream-based active learning for sliding windows under the influence of verification latency

Tuan Pham[1] · **Daniel Kottke**[1] · **Georg Krempl**[2] · **Bernhard Sick**[1]

Received: 31 January 2021 / Revised: 28 July 2021 / Accepted: 8 October 2021 /
Published online: 18 November 2021

Abstract

Stream-based active learning (AL) strategies minimize the labeling effort by querying labels that improve the classifier's performance the most. So far, these strategies neglect the fact that an oracle or expert requires time to provide a queried label. We show that existing AL methods deteriorate or even fail under the influence of such verification latency. The problem with these methods is that they estimate a label's utility on the currently available labeled data. However, when this label would arrive, some of the current data may have gotten outdated and new labels have arrived. In this article, we propose to simulate the available data at the time when the label would arrive. Therefore, our method Forgetting and Simulating (FS) forgets outdated information and simulates the delayed labels to get more realistic utility estimates. We assume to know the label's arrival date a priori and the classifier's training data to be bounded by a sliding window. Our extensive experiments show that FS improves stream-based AL strategies in settings with both, constant and variable verification latency.

Keywords Classification · Active learning · Evolving data streams · Concept drift · Verification latency · Label delay

Editors: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann.

✉ Tuan Pham
tuan.pham@uni-kassel.de

Daniel Kottke
daniel.kottke@uni-kassel.de

Georg Krempl
g.m.krempl@uu.nl

Bernhard Sick
bsick@uni-kassel.de

[1] Universität Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany

[2] Utrecht University, Princetonplein 5, 3584 CC Utrecht, The Netherlands

1 Introduction

This article addresses data stream classification in non-stationary environments, where instances appear successively and are initially unlabeled.

To learn a classifier, we need labels for at least some of these instances. Therefore, we can select some instances to be passed to an oracle for labeling, e.g., a human expert or a computationally intensive simulation. Such a label acquisition induces some sort of cost, which we (for now) assume to be equal across all instances. Hence, we define a labeling budget as the fraction of instances that can be labeled, e.g., 8% of instances in the stream. The rest remains unlabeled. Algorithms from the field of stream-based active learning (AL) aim to maximize the classifier's performance under the given budget restrictions by selecting only the most informative instances for labeling. Similar to Zliobaite et al. (2014), we follow the common assumption that the AL strategy needs to decide immediately at the moment an instance arrives whether or not to acquire its label.

Imagine the following example: We want to train a classifier on a stream of posts from social media to find out if they mention a company's name positively or negatively. Using stream-based AL methods, we only select the most useful posts for labeling. However, when we select an instance for labeling, it is impossible for the expert to provide the label immediately as time is needed to read the post first. Sometimes the expert might even be occupied with other work, such as previous labeling requests, which delays the appearance of the label further. So far, all existing AL strategies assume that labels are immediately available. This is unrealistic and problematic as new instances might appear during an ongoing labeling process. This means that subsequent active learning decisions, i.e., whether to request these new instances' labels as well, might need to be taken while previous label requests are still being processed. This problem occurs even more often when we consider fast data streams or applications with high labeling costs, as these are often correlated with annotation time. Furthermore, even relaxing the commonly formulated requirement to decide a label request immediately upon an instance's arrival would not help: If we were to allow labels to be requested later, it would simply mean that the processing of these labels by the oracle would then be delayed as well. As a consequence, we would delay subsequent label requests even more, and therefore accumulate an increasing backlog in the labeling process.

While this problem has not been considered in active learning literature so far, it has been studied in the context of *passive* learning on non-stationary data streams. Denoted as *label delay* by Kuncheva (2008) and *verification latency* by Marrs et al. (2010), several approaches for adapting classifiers when labels arrive with delay have been proposed. Nevertheless, none of them addresses the effect this problem has on the *active selection* of labels, in particular on estimating the utility of requested labels that arrive with delay.

In this article, we therefore examine the influence of verification latency for stream-based AL algorithms and propose strategies to solve the accompanying problems. To understand the effects of verification latency, we handle the non-stationarity in data streams (due to concept drift or shift) using blind adaptation in the form of a sliding window with length w. The sliding window restricts the training data for the classifier to instances that appear within the window $[t-w;t]$. Although advanced change detection methods can react faster to change (Gama et al. 2014), they might induce an unintentional bias and is, therefore, subject to future work. The advantage of blind adaptation is its simplicity and predictability which helps the general understanding of the effects for active learning.

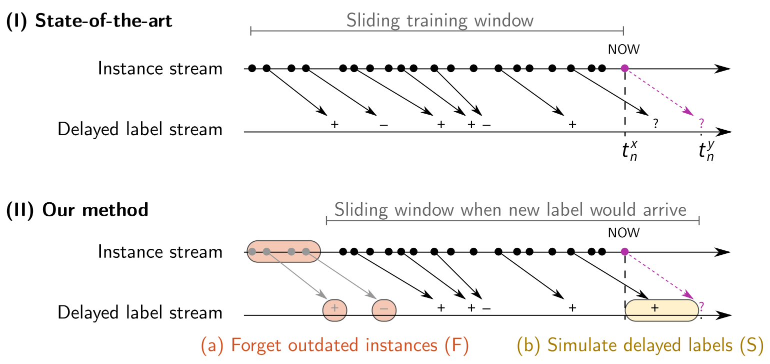


Fig. 1 We propose to assess the utility of acquiring a label at the time when the label would arrive. Therefore, we need to **a** forget outdated instances and **b** simulate labels that are not yet available but will be then (Color figure online)

To visualize our main ideas, we compare the current approach and our solutions in Fig. 1.

State-of-the-art stream-based AL methods quantify the utility of the current instance x_n (violet) at time t_n^x based on the available information that is contained in the sliding window of the current classifier (gray interval). Thereby, the following two problems occur: **(a)** If the new instance x_n (violet) is selected for acquisition, it could not be added before its label y_n becomes available, which is at time t_n^y. Nevertheless, in the state-of-the-art approach, the utility is quantified using the information from the current sliding window at time t_n^x. Thereby, one would consider older labels that would already have been removed when the new instance-label-pair would appear (at time t_n^y). This is a problem. For example, if x_n is similar to one of the old instances, the AL strategy sees no need to acquire x_n, as long as the old instances are still included for utility estimation. Nevertheless, their information will have been lost at t_n^y, and y_n would not have been requested to replace them. Hence, we propose to forget the soon-to-be outdated instances when quantifying an instance's utility (see Fig. 1II-a). **(b)** State-of-the-art algorithms tend to acquire labels from similar instances under the influence of verification latency. Imagine that now, in contrast to (a), the instance with the delayed label (black '?') is very similar to the new instance x_n (violet). If we ignore the information about delayed labels, we will also have a high utility for x_n. Thus, it is likely that redundant labels are acquired. In order to address this, we propose to simulate the delayed labels and include them when quantifying the utilities (see Fig. 1II-b). The proposed methods are integrated into an active learning python framework, called scikit-activeml[1] (Kottke et al. 2021a).

In this article, we answer the following research question:

"How must we change AL methods such that they still work for tasks with verification latency?"

To this end, we make the following contributions, which we evaluate within the given hypotheses:

[1] https://github.com/scikit-activeml/scikit-activeml.

- We show that verification latency impairs the performance of traditional AL methods (see Hypothesis 1).
- We propose two general wrapper strategies for stream-based AL algorithms that implement the idea of forgetting obsolete data (F) and simulating upcoming labels (S).
- We evaluate our strategies on multiple synthetic and real datasets for various constant (see Hypothesis 2) but also variable label delays (see Hypothesis 4) and show their effectiveness.
- We show that combining forgetting (F) and simulating (S) performs better compared to using only one wrapper F or S (see Hypothesis 3).

The remainder of this article is structured as follows: We start with a detailed description of related work. Then, we describe our approaches and evaluate them based on the four hypotheses. Finally, we conclude our work and motivate future work in the field of stream-based AL with verification latency.

2 Related work

The problem addressed by the approach in this publication has relations to several branches of research in literature: First, to data stream mining in general. Second, to change detection, to change and drift mining approaches for handling drift in presence of verification latency, and to active machine learning for handling costly labels in particular. We will first review these branches separately, before discussing the state-of-the-art at their intersection.

2.1 Data stream mining

As noted in Babcock et al. (2002), in contrast to conventional (pool-based) data mining, the characteristic of *data stream mining* is that instances therein arrive sequentially over time, either individually (i.e., are processed *instance-wise*) or in *batches*. Thus, the number of instances is potentially unbounded and grows continuously, from only a single or a few instances at the beginning. The first main challenge in data stream mining is therefore keeping the computational time and space complexities of processing an additional instance constant, i.e., independent of the number of previously processed instances. This is typically done by discarding or archiving already processed instances. Complementary to the challenge of complexity, a second major and common challenge is non-stationarity, as data generating distributions change over time. This phenomenon, denoted as *concept drift* in Schlimmer and Granger (1986), as *concept shift* in Klinkenberg and Renz (1998), or as *population drift* in Kelly et al. (1999), requires *adaptation* techniques. An exhaustive review of such techniques is given for example in Gama et al. (2014). The taxonomy proposed therein distinguishes techniques by three criteria: The first distinction is based on the learning mode of an approach, which might always retrain a new model from *scratch*, or might *incrementally* keep updating the same model. The second distinction is whether the adaptation is initiated in an *informed* manner, i.e., upon the detection of change, or so-called *blindly*, i.e., by adapting continuously. The third distinction is whether a *single* model or an *ensemble* is maintained and managed. Following this taxonomy, our approaches proposed below are characterized as incremental streaming approaches that manage a single model. As will be explained in the next subsection, due to verification latency we rely on continuous blind adaptation, achieved by forgetting the oldest instances.

2.2 Verification latency

The vast majority of data stream mining approaches assume that the actual label of an instance becomes available before proceeding to the classification of the next instance. Kuncheva (2008) and Marrs et al. (2010) were among the first publications to challenge this assumption. Kuncheva (2008) coined the term *label delay*, and Marrs et al. (2010) the term *verification latency*, respectively, for scenarios where the actual label to verify a prediction becomes available only with considerable delay. A first approach addressing this challenge, although for static streams, was *instance-based learning* proposed in Kuncheva and Sánchez (2008). This nearest neighbor-based, error-driven approach incorporates instances into a reference set if they are misclassified. However, the delay in labeling leads to a delay in identifying misclassifications as well. Thus, the authors propose conditional labeling, where for a yet unlabeled instance the labels of its two nearest labeled neighbors in the reference set are compared against each other, and a match indicates a correct prediction. The authors discuss different strategies to handle delayed labeling in the reference set, ranging from ignoring to using unlabeled data. They report inconclusive results when comparing these strategies. Subsequently, the first approaches to address *verification latency* in *non-stationary, evolving* streams were proposed by Krempl and Hofer (2011) and Krempl (2011): Since no recent, labeled data is available, such approaches require assumptions about the type of drift present in the data stream. Formalizing these assumptions allows formulating *drift models* that describe the transitions between distributions over time. Similarly to viewing the problem as an *unsupervised domain adaptation* task between chronologically ordered source and target domains, these drift models allow to anticipate or extrapolate distributional changes or classification model adaptations. In Krempl and Hofer (2011), this is done by using a mixture of labeled Gaussian clusters and tracking their movement within the feature space. In contrast, the *APT*-algorithm proposed in Krempl (2011) is non-parametric, and maps labeled instances from the source domain to the target domain by formulating this as an assignment problem and solving it for minimizing the overall distance. In Hofer and Krempl (2013), an approach for updating the class prior probabilities from unlabeled data is proposed. Most recently, Krempl et al. (2019) propose a non-parametric approach for extrapolating density changes over time. Thus, it complements the previously proposed techniques for updating the class prior with techniques that enable to extrapolate the evolution of the class-conditional feature distributions.

Rather than explicitly modeling a drifting distribution, Dyer et al. (2014) propose a series of geometry-based approaches for *Comp*acted *O*bject *S*ample *E*xtraction (*COMPOSE*). These approaches maintain a compacted geometric representation of each class and adapt these representations by using unlabeled data in a semi-supervised fashion. Recent extensions of COMPOSE include a faster variant proposed by Umer (2017), and a variant for handling scenarios with class-imbalance proposed in Frederickson and Polikar (2018).

Recent publications have further developed the *taxonomy* of verification latency. Plasse and Adams (2016) propose a distinction based on the cause of verification latency. Therein, one type is *delay mechanisms* that cause latencies independently of the data. An example is a constant delay that occurs due to the physical movement of objects to a testing facility. Another type is *lag magnitudes* that are dependent on the data. For example, in a fraud detection scenario, the time needed to analyze cases might vary with their difficulty. The framework proposed in Plasse and Adams (2016) uses a weighting approach to limit the impact of verification latency. This weighting approach allows adjusting the influence of instances on the classifier according to the latency. Another categorization is provided by considering the extent of verification latency. Souza et al. (2018) use this to distinguish three different types of latencies. One extreme is *null latency*, which corresponds to no latency at all. The other is categorized as *extreme verification latency*, where latency is assumed to be infinitely high. Thus, after the initial training has been completed, no further labels will arrive in a data stream. Between these two extremes lies the so-called *intermediate latency*, which we will focus on in this article. Here, the verification latency is finite and may or may not be known beforehand.

Finally, aspects of *evaluation* under verification latency have also been studied. In Souza et al. (2018), an extension of the *Kappa Statistic* is proposed to overcome shortcomings of conventional stream mining evaluation techniques. In Grzenda et al. (2019), the monitoring of each instance's prediction over time during the verification latency is suggested. A comparative study of selected approaches for extreme verification latency has been published in Umer and Polikar (2020).

While the approaches discussed above address adaptivity, a related question is that of *change detection* in the classes' posterior distribution in presence of verification latency: Large latency results in the absence of recent labels, which impairs the use of supervised change detection. Following the early works of Kuncheva (2008) on unsupervised, ensemble-based feature change detection, and the study by Žliobaité (2010) on the possibility of unsupervised detection of posterior changes, unsupervised change detection has recently gained attention. An example is Hammoodi et al. (2016), which addresses the detection of drift in features for real-time feature selection. A similar approach is proposed by dos Reis et al. (2016). Therein, a comparison of the training data for the classifier against the most recent data using an incremental version of the Kolmogorov-Smirnov test is proposed. This determines for each feature separately whether a change has happened. Finally, the problem of fault detection under verification latency has been addressed in Razavi-Far et al. (2019). Nevertheless, as for adaptation itself, using detected changes in unlabeled data to infer drift of the posterior distribution again requires making strong assumptions about the mechanism underlying the distributional changes. We acknowledge that this might have potential in some applications, and thus might be worth exploring in the future. For the sake of the versatility of the approaches in this paper, however, we avoid such assumptions, and instead use labels as they arrive for continuous (i.e., blind) adaptation. Concept changes are not the only changes that might occur within a data stream. Depending on the application, changes in the number of features might occur. So far, this has been addressed for pool-based AL (Pham et al. 2020). For this article, we focus on concept changes, however, changes in number of features in an active learning scenarios with verification latency might be worth investigating in the future.

2.3 Active learning in data streams

In the conventional, *passive* supervised machine learning setting, an algorithm processes all labeled instances that are present in a training set or that arrive in a data stream. In contrast, *active* learning approaches aim to *actively* develop and test new hypotheses (Settles 2012, pages 3–4). Typically, this is done by *interacting* with their environment (i.e., a data providing *oracle*) and thereby controlling the acquisition of new data, as described in Cohn (1993). For example, an active classifier might use the feature vectors given for all instances to determine the one instance that promises to be the most insightful, to query its label from an oracle, and to add it to its training set. Several techniques have been proposed in literature, as surveyed for example in Settles (2012) or Kumar and Gupta (2020). Most prominent is *Uncertainty Sampling*, proposed by Lewis and Gale (1994), where instances closest to the decision boundary or with the highest uncertainty in their posterior estimates are selected. Another is *Version Space Partitioning* proposed by Cohn (2010), where the instance with a maximal disagreement between hypotheses in the current version space is selected, or *Query by Committee*, where the instance with maximal disagreement within an ensemble of classifiers is selected. In *Loss Minimization*, a classifier's performance is directly optimized, such as in *Expected Error Reduction* by Roy and McCallum (2001). Related to the former is another decision-theoretic strategy, *Probabilistic Active Learning* by Krempl et al. (2015b), that computes the expected gain in classification performance from labeling additional instances.

From the active learning scenarios considered in the literature, our work focuses on so-called *stream-based selective sampling* as reviewed in Settles (2012) and Zliobaite et al. (2014): herein, instances arrive sequentially on a data stream, and whether an instance's label is acquired has to be decided ad hoc and irrevocably, since instances are forgotten and not accessible after processing. In particular in *non-stationary* data streams, this poses specific challenges: In a stationary setting, as a classification model converges further towards the optimal decision boundary, focusing on *exploitation* by sampling instances closer to the expected decision boundary is considered beneficial for an active learning strategy, as discussed in Bondu et al. (2010) and Loy et al. (2012). In contrast, if drift might occur at any time and anywhere, then even a previously already optimal classification model might have become obsolete. To address this, continuous *exploration* is required. To this end, Zliobaite et al. (2014) propose to continuously and randomly sample a few instances, such that there are representatives of all relevant areas of the feature space. Their *Variable Uncertainty* approach adjusts only a threshold parameter, such that the most uncertain instances over time are selected. While this approach does not use randomization for exploration, it is extended to *Random Variable Uncertainty*, by multiplying the adjusted threshold parameter with a factor that is randomly sampled from a normal distribution with mean 1. Finally, for the *Split* approach, they extend Variable Uncertainty further by combining it also with change detection, which is solely applied on the randomly sampled instances. Similar to Zliobaite et al. (2014), the density-based active learning (*DBAL*) approach proposed in Ienco et al. (2014) also uses uncertainty sampling with randomization for exploration but combines it with density weighting. In addition to the spatial aspect of ensuring both, the exploitation of already available labels, and the exploration of the whole feature space, Kottke et al. (2015) point out the need for *budget management*, i.e., to also consider the temporal aspect of when to acquire labels on evolving data streams. For the latter, consider an unlabeled instance at a promising spatial location. Whether to acquire such an instance *now* depends on the lifetime of this and similar instances. That is, on how long an instance will contribute significantly to improving the model's performance, and whether an instance at an even more promising spatial location might arrive later, such that saving budget now would pay off then. To achieve this, adaptive filtering techniques are proposed in Kottke et al. (2015). Therein, Probabilistic Active Learning is used to calculate the spatial utility of instances within a sliding window. Then, a *Balanced Incremental Quantile Filter* (*BIQF*) is used to select the instances with the highest spatio-temporal utility within the sliding window, such that a given labeling budget is never exceeded.

In addition to techniques for (temporal) budget management, there exist several approaches for streams where processing instances in *chunks* is possible. Examples are Zhu et al. (2007), which uses *Query by Committee*, or clustering-based approaches such as in Ienco et al. (2013) and Krempl et al. (2015a). Nevertheless, since we aim for an approach that is capable of instance-wise processing, we will not elaborate on these chunk-wise processing approaches further, and rather focus on instance-wise processing approaches with budget management.

2.4 Active learning in evolving data streams in presence of verification latency

Summarizing, several data stream mining approaches have addressed either the problem of verification latency or the problem of actively selecting labels. However, no approach has yet addressed the combination of actively selecting labels under verification latency. Nevertheless, the detrimental impact of verification latency on some of the stream-based active learning approaches, such as variable randomized uncertainty proposed in Zliobaite et al. (2014), has been empirically evaluated in Parreira and Prati (2019). Their results indicate that the informativeness of instances becomes more uncertain as latency increases. In their conclusion, they call for future work that develops active learning approaches that consider the effect of delayed labels. Our paper addresses this by contributing a first active learning approach for verification latency.

3 Handling verification latency in stream-based active learning

We assume to have a data stream as an ordered list of 4-tuple (t_i^x, x_i, t_i^y, y_i) with $i \in \{1, \dots, N\}, N > 0$ (Eq. (1)). Each data point consists of an instance x_i, its timestamp t_i^x, a label y_i that would be returned by the oracle, and the timestamp that label would arrive at t_i^y when queried. Thus, $t_i^y - t_i^x$ represents the verification latency for x_i.

$$S = \left\langle (t_i^x, x_i, t_i^y, y_i) | i \in \mathbb{N}_{>0} \right\rangle \tag{1}$$

$$x_i \in \mathbb{R}^D \qquad y_i \in Y = \{1, 2, \dots, C\} \tag{2}$$

$$t_i^x, t_i^y \in \mathbb{R}_{\geq 0} \tag{3}$$

$$t_j^x < t_k^x \text{ for } j < k \in \mathbb{N}_{>0} \quad \wedge \quad t_i^x < t_i^y \text{ for } i \in \mathbb{N}_{>0} \tag{4}$$

To our knowledge, we are the first to provide methods to handle verification latency in an AL scenario. Additionally, the stream is assumed to be infinitely long and changes over time, i.e., abrupt and gradual concept change may occur. Hence, storing all data is infeasible. Instead, we use a sliding window $\mathcal{W}_n$ at time t_n^x that allows us to reduce the amount of stored data and deal with concept change by forgetting old instances. $\mathcal{W}_n$ has a constant length of $w \in \mathbb{R}_{>0}$, defined as a time span.

$$\mathcal{W}_n = [t_n^x - w; t_n^x) \tag{5}$$

We model the AL process by introducing an acquisition vector Q_n, that stores a boolean value a_i for each x_i with $i \in \{1, \dots, n-1\}$ and $t_i^x \in \mathcal{W}_n$, whether the AL strategy queried the label y_i at t_i^x.

$$Q_n = \left\langle a_i | \forall i < n \text{ with } t_i^x \in \mathcal{W}_n \right\rangle \tag{6}$$

$$a_i \in \{0, 1\} \tag{7}$$

Thus, if a_i is 1, the AL strategy decides at time t_i^x to query the label y_i for instance x_i. Hence, the label will be accessible at time t_i^y. If a_i is 0, the label will not be queried. Thus, y_i will remain unavailable even after t_i^y.

Considering the sliding window $\mathcal{W}_n$ and the acquisition vector Q_n at t_n^x, a valid training instance x_i with its label y_i at time t_n^x has to be present within $\mathcal{W}_n$, i.e., $t_i^x, t_i^y \in \mathcal{W}_n$, and y_i has been queried in the past, i.e., a_i is 1, with $1 \leq i < n$. We exclude n as its label y_n is not available yet. The classification model's training data L_n is a set of triplets containing the data point, its label, and the corresponding training weight, which ranges from 0 to 1 and will be used by the classifier. We consider the training weight, as it will be a central part of one of the proposed methods below. For simplicity, we assume this to be set to 1.

$$L_n = \left\langle (x_i, y_i, 1) | \forall i < n \text{ with } a_i = 1 \wedge t_i^x, t_i^y \in \mathcal{W}_n \right\rangle. \tag{8}$$

3.1 The general AL cycle

For each instance x_n, the AL strategy has to decide whether to query the instance or not. As we focus on instance-wise processed evolving data streams, the AL strategies we address come with budget management. We further focus on AL strategies that assess the utility u_n for an instance x_n. Based on the utility, the budget management assesses, whether the label query fits into the budget. We propose to standardize the AL strategies by encapsulating them in classes. Given an AL strategy *al*, *al*.UTILITY assesses the utility an instance's label would have. Furthermore, *al*.QUERY decides based on this utility whether the label should be queried. We introduce GETUTILITY, that wraps around *al*.UTILITY, that will be used to handle verification latency. For traditional AL strategies, GETUTILITY is defined as shown in Algorithm 1.

Algorithm 1 Utility Without Verification Latency

```
1: function GETUTILITY($al, (t_n^x, x_n, t_n^y), f, L_n$)
2:     return $al$.UTILITY($x_n, f, L_n$)
3: end function
```

As underlying active stream-based sampling techniques we use the uncertainty sampling-based *Var-Uncertainty* and *Split* approaches proposed by Zliobaite et al. (2014), and the *BIQF*-approach proposed by Kottke et al. (2015), with its *Balanced Incremental Quantile Filter* and *Probabilistic Active Learning* components. Thus, in the appendix, we also provide the pseudocodes for computing the utility function based on *Variable Uncertainty* in Algorithm 8, and of the corresponding query function in Algorithm 9. Likewise, the pseudocodes for the utility function based on Split in Algorithm 10, as well as its query function in Algorithm 11, and the pseudocodes for computing the utility value using Probabilistic Active Learning in Algorithm 12 and for its corresponding query function in Algorithm 13 are given in the appendix.

Algorithm 2 AL Framework

```
1: $f \leftarrow$ initialize classifier
2: $al \leftarrow$ initialize AL strategy
3: for all $n \in \{1, \ldots\}$ do
4:     $(t^x_n, x_n, t^y_n) \leftarrow$ retrieve from data stream $S$ (Eq. (1))
5:     $L_n \leftarrow$ training dataset at time $t_n$ for acquisitions $A$ (Eq. (8))
6:     $u_n \leftarrow$ getUtility$(al, (t^x_n, x_n, t^y_n), f, L_n)$
7:     $a_n \leftarrow al.$query$(u_n)$
8:     if $a_n = 1$ then
9:         ask oracle for label of $x_n$ (label $y_n$ will be provided at $t^y_n$)
10:    end if
11: end for
```

The framework we work in is given in Algorithm 2. First, the classifier f that will be trained by the AL strategy and the AL strategy al itself are initialized (line 1 and 2). The triplet (t^x_n, x_n, t^y_n) is acquired from the data stream (line 4). Based on the current time t^x_n, the training data L_n needs to be adjusted, by moving the sliding window and incorporating labels that are newly available, i.e., labels y_i with $t^y_i \in [t^x_{n-1}, t^x_n)$ and $a_i = 1$ (line 5). The utility of x_n is then assessed using al (line 6) and based on the utility, it is decided whether to query y_n (line 7). If the label is queried, i.e., $a_n = 1$, then x_n is given to the oracle, so that the label is provided at t^y_n (line 8 to 10).

3.2 Forgetting obsolete data (F)

For active learning, it is essential to assess the classifier's uncertainty as accurately as possible. In case of verification latency, the current classifier (f^{L_n}) may provide different predictions from the classifier at time t^y_n. We denote the sliding window that would be used for training at t^y_n as $\mathcal{D}_n$.

$$\mathcal{D}_n = [t^y_n - w; t^y_n) \tag{9}$$

Due to incomplete information about $\mathcal{D}_n$, i.e., the incoming instances and labels after t^x_n are still unknown, we will estimate the information within $\mathcal{D}_n$ as good as possible. One problem is that L_n includes data that is not included within $\mathcal{D}_n$ (see the red area in Fig. 2). This needs to be considered in AL. Using L_n for estimating the uncertainty might underestimate the classifier's uncertainty in areas with data that will be obsolete soon.

Thus, to obtain more accurate uncertainty estimations, we propose Forgetting Old Data (F). Forgetting mechanisms are a standard technique in data stream mining (Gama et al. 2014), which we hereby introduce to AL scenarios with verification latency. F discards data that would be unavailable by t^y_n. We denote the ordered list that contains these data as O_n.

$$O_n = \left\langle (x_i, y_i, 1) | \forall i < n \text{ with } a_i = 1 \wedge t^x_i, t^y_i \in \mathcal{W}_n \setminus \mathcal{D}_n \right\rangle \tag{10}$$

To be more precise, only data points (t^x_i, x_i, t^y_i, y_i) with $t^x_i, t^y_i \in \mathcal{D}_n$ will be used to infer a classifier that will be used to assess the utility u_n at t^y_n. In Fig. 2, state-of-the-art is compared to F. The state-of-the-art strategies include all data in $\mathcal{W}_n$ for the utility assessment. Hence, x_2 and x_3 are included for that example as well. However, those instances will be

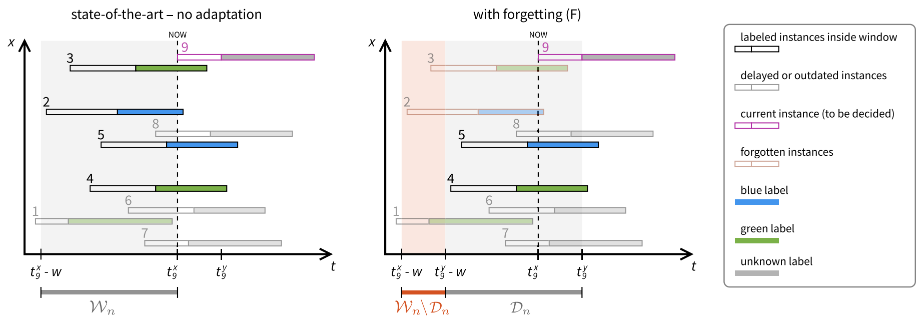

state-of-the-art – no adaptation
with forgetting (F)
NOW
x
t
t_9^x - w
t_9^x
t_9^y
t_9^y - w
$\mathcal{W}_n$
$\mathcal{W}_n \setminus \mathcal{D}_n$
$\mathcal{D}_n$
labeled instances inside window
delayed or outdated instances
current instance (to be decided)
forgotten instances
blue label
green label
unknown label

Fig. 2 The plots show the data within the sliding window that is used for assessing the acquisition of label y_9 at time t_9^x. The left side shows state-of-the-art strategies. These use all data that is used to train the classifier. The right side shows the difference when combining stream-based AL strategies with the proposed forgetting wrapper (F). As highlighted in the red area, instances that will be unavailable at t_9^y, are excluded as those will not be within the sliding window anymore (Color figure online)

unavailable to the classifier training, once y_9 arrives which will lead to inaccurate utility assessments. When using F, x_2 and x_3 will be excluded as both instances would be contained in O_n (see Eq. (10)).

Algorithm 3 GETUTILITY using F

```
1: function GETUTILITYF($al, (t_n^x, x_n, t_n^y), f, L$)
2:     $O_n \leftarrow$ find obsolete data (Eq. (10))
3:     return GETUTILITY($al, (t_n^x, x_n, t_n^y), f, L \setminus O_n$)
4: end function
```

Algorithm 3 shows the corresponding pseudocode for F. First, O_n, the labeled data not within $\mathcal{D}_n$, has to be identified (line 2). Next, all data in O_n is omitted from L_n (line 3). Additionally, the GETUTILITY function is defined in a recursive manner (line 3) so that it can also be easily combined with the methods proposed in the following two sections.

3.3 Simulating incoming labels with bagging (S.B)

F by itself only handles data that will be unavailable in the future ($\mathcal{D}_n$), by forgetting them. Hence, labels y_i that have been queried but are not yet available ($t_n^x \le t_i^y < t_n^y \wedge a_i = 1$) are still ignored by the active learning strategy. Not knowing that the labels of x_6 and x_7 in Fig. 3 have already been queried, the AL strategy will probably select instance x_9, although more labels will be available in that region soon. By simulating y_6 and y_7 as in Fig. 3, the AL strategy would be aware of the incoming labels and react accordingly, e.g., not selecting x_9. We denote $\mathcal{D}_n^+$ as the time span of $\mathcal{D}_n$ that is not covered by $\mathcal{W}_n$. Hence, it covers the data that will be observed in the future within $\mathcal{D}_n$.

$$\mathcal{D}_n^+ = \mathcal{D}_n \setminus \mathcal{W}_n = [t_n^x; t_n^y) \tag{11}$$

To incorporate still unavailable but already queried labels, we propose Simulating Incoming Labels with Bagging (S.B) to estimate the utilities for labels in the future. We propose to estimate the utility of y_n by employing a bagging approach by averaging K utility

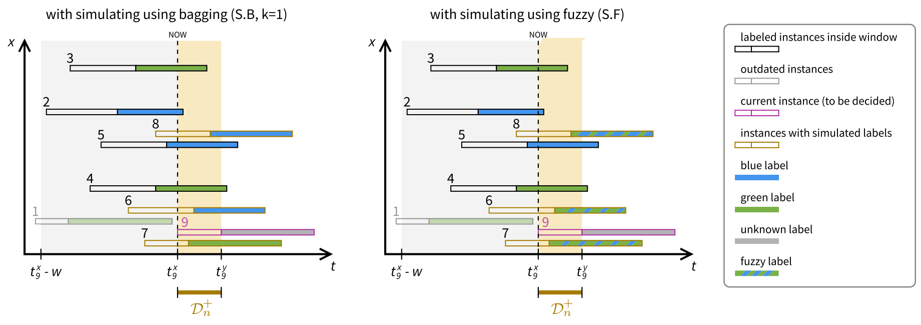

with simulating using bagging (S.B, k=1)
with simulating using fuzzy (S.F)
NOW
x
t
t_9^x - w
t_9^x
t_9^y
$\mathcal{D}_n^+$
1
2
3
4
5
6
7
8
9
labeled instances inside window
outdated instances
current instance (to be decided)
instances with simulated labels
blue label
green label
unknown label
fuzzy label

Fig. 3 The plots show the data that will be used to assess the utility for querying the label y_9 at time t_9^x. The left side shows the behavior of S.B with $k = 1$. The brown contour shows the still unlabeled instances whose label will arrive between t_9^x and t_9^y. Each of these instances is assigned a randomly sampled label (see Eq. (12)) in each iteration. The right side shows the behavior of S.F. Contrary to S.B, S.F assigns a fuzzy label (see Eq. (15)) to the instances whose label will arrive between t_9^x and t_9^y (Color figure online)

estimations. For each utility estimation, S.B samples a set of labels $B_{n,k}$ with $k \in \{1, \dots, K\}$ for data with $t_i^y \in \mathcal{D}_n^+$ and $A_i = 1$. The labels $\hat{y}_i$ for each sample are distributed according to the categorical distribution (Murphy 2012) with parameter $p^{L_n}(\cdot|x_i)$.

$$B_{n,k} = \left\langle (x_i, \hat{y}_i, 1) | \forall i < n \text{ with } a_i = 1 \wedge t_i^x \in \mathcal{W}_n \wedge t_i^y \in \mathcal{D}_n^+ \right\rangle \\ \text{with } \hat{y}_i \sim \mathrm{Cat}(p^{L_n}(\cdot|x_i)) \tag{12}$$

As the label of y_i is not yet available, we need to estimate the probability for each class to sample $\hat{y}_i$ from the categorical distribution. Therefore, we use all available instance-label-pairs (L_n) and a Bayesian approach to regularize the estimated probabilities using a conjugate prior α (Murphy 2012). Similarly to Chapelle (2005), we calculate the probabilities using a kernel function $k(\cdot, \cdot)$ with $k(x, x) = 1$.

$$p^{L_n}(y|x) = \frac{\left(\sum_{(x',y',w') \in L_n} \mathbb{1}_{y'=y} \cdot w' \cdot k(x_i, x') \right) + \alpha_y}{\left(\sum_{(x',y',w') \in L_n} w' \cdot k(x, x') \right) + \sum_{i=1}^{C} \alpha_i} \tag{13}$$

Choosing a uniform prior ($\alpha = (1, \dots, 1)$), this estimate provides equal probabilities for all classes if there are no labels near x. Contrary, the regularizing effect of the prior diminishes with increasing label counts. This idea of using Bayesian probability estimation has also been used in the area of AL in Kottke et al. (2021b) and Chapelle (2005). The performance of the simulations in S and subsequently of the active learning depend on the correctness of the probability estimation. In some applications, other estimation methods might be more suitable.

We sketch S.B on the left side of Fig. 3. This approach focuses on labels that will arrive within $\mathcal{D}_n^+$, i.e., y_6, y_7, and y_8. S.B simulates those labels by randomly sampling the labels as shown in Eq. (12). Hence, the AL strategies will be aware of their own queries, even though, the true labels y_6, y_7, and y_8 are still unknown.

Algorithm 4 GETUTILITY USING S.B

```
1: function GETUTILITYSB($al, (t^x_n, x_n, t^y_n), f, L$)
2:     $u \leftarrow 0$
3:     for all $k \in \{1, \ldots, K\}$ do
4:         $B_{n,k} \leftarrow$ sampling unknown labels (Eq. (12))
5:         $u \leftarrow u + \frac{1}{K}$GETUTILITY($al, (t^x_n, x_n, t^y_n), f, L \cup B_{n,k}$)
6:     end for
7:     return $u$
8: end function
```

S.B uses $L \cup B_{n,k}$ to estimate the training data available for the classifier at t^y_n. Algorithm 4 shows the pseudocode for S.B. First, a variable u is initialized with 0 to store the resulting utility (line 2). Next, the instances x_i with $t^y_i \in \mathcal{D}^+_n$ and $a_i = 1$ are used to construct $B_{n,k}$ (line 4). $L \cup B_{n,k}$ is then used to estimate the utility (line 5). This is repeated K-times (line 3 to 6) to obtain the average utility.

3.4 Simulating incoming labels with fuzzy labeling (S.F)

The main idea of Simulating Incoming Labels with Fuzzy Labeling (S.F) is similar to S.B. Queried labels y_i with $t^y_i \in \mathcal{D}^+_n$, and $a_i = 1$ should be included in the classifier for which the utility is assessed. Instead of randomly sampling likely labeling realizations like S.B does multiple times, S.F exploits the classifiers' ability to assign training weights to the training instances (see Fig. 3). S.F relies on $p^{L_n}\left(\cdot|x_i\right)$ to estimate the labels as accurately as possible. As each x_i may belong to one of the classes ($y' \in \{1, \ldots, C\}$), x_i is added C-times to the training data, i.e., once for each class, with the respective probability $p^{L_n}\left(y'|x_i\right)$ as a training weight.

$$I_n = \left\langle i | \forall i < n \text{ with } a_i = 1 \wedge t^x_i \in \mathcal{W}_n \wedge t^y_i \in \mathcal{D}^+_n \right\rangle \tag{14}$$

$$F_n = \left\langle (x_i, \hat{y}, p_{i,\hat{y}}) | \forall (i, \hat{y}) \in (I_n \times Y) \text{ with } p_{i,\hat{y}} = p^{L_n}\left(\hat{y}|x_i\right) \right\rangle \tag{15}$$

The right plot in Fig. 3 shows S.F. The labels that would have been simulated with S.B, are now simulated using S.F. Hence, y_6, y_7, and y_8 are each assigned a fuzzy label. Thus, S.F makes the AL strategies aware of its own queries even though the queried labels are still unavailable.

Algorithm 5 GETUTILITY using S.F

```
1: function GETUTILITYSF($al, (t^x_n, x_n, t^y_n), f, L$)
2:     $F_n \leftarrow$ create fuzzy label set (Eq. (15))
3:     return GETUTILITY($al, (t^x_n, x_n, t^y_n), f, L \cup F_n$)
4: end function
```

S.F uses $L \cup B_{n,k}$ to estimate the training data available for the classifier at t^y_n. Algorithm 5 sketches S.F. First, F_n is constructed (line 2). Using $L \cup F_n$ as training data, the utility is estimated (line 3).

4 Experimental evaluation

In this section, we first discuss the experimental design. Afterward, we present five different hypotheses. For each, a short summary of the findings will be followed by a detailed description of the experiments and a discussion of their results. The experiment framework is available online[2].

4.1 Design of experiments

We perform all experiments within the framework sketched in Algorithm 2. For evaluation, we use prequential evaluation over all instances of the data stream. Prequential evaluation (Gama et al. 2009) describes an evaluation procedure, where the label of a new instance is first predicted for evaluation purposes. Afterward, the instance is processed further, e.g., by the AL strategy that queries labels.

To assess the robustness of the selection strategies to small changes, we introduce some randomness to the data stream. Independently from using the whole stream for prequential evaluation, we only pass 80% of the data (randomly chosen) to the active learning strategy. We repeat this 10 times.

We evaluate our proposed method FS (forgetting and simulating) and its variants using 12 different datasets. D0–D2 are synthetically generated datasets where the instances are distributed within Gaussian centroids that are assigned to a class. We use three different datasets to reflect varying difficulty. Each of those datasets consists of 4000 instances with two features and ten centroids per class. However, the number of classes differs. D0 only has two classes, while D1 and D2 have 3 and 4 classes, respectively. An assumption of data streams is that they may contain changes. We introduce such changes with the method proposed by Shaker and Hüllermeier (2013). Hence, we generate two separate static datasets with 2000 instances each using different random seeds. After that, we merge them with the method mentioned above. D3–D7 are derived from benchmark datasets. Table 1 lists the names and properties of these datasets. Similar to D0–D3, we construct a data stream that includes a concept drift. To do so, we select a random feature. We then divide the static dataset into two subsets using the median of the randomly selected feature. Afterward, this feature is discarded, and the two subsets are shuffled and then merged using the same method as used for D0–D3. The parameter Shaker and Hüllermeier (2013) denote as w is set to 50 for controlling the rate of change. This rate of change leads to a window of approximately 500 instances, where the gradual change happens. Lastly, D8–D11 are benchmark data streams that are listed in Table 1 as well. For all data streams we use each instance's index within the data stream as its timestamp, i.e., $t_i^x = i$ for x_i with x_0 being the first instance. Hence, the training window and the verification latency does not need to be rescaled individually for each dataset.

For our method FS, we choose a uniform prior for the probability estimation which is described by a parameter of $\alpha = (1, \dots, 1)^T$ for a Dirichlet distribution. To perform the bagging in S.B, we set the number K to 3. For the classifier, we use the Parzen Window Classifier as described in Chapelle (2005) using a Gaussian kernel and the mean-bandwidth heuristic as proposed in Chaudhuri et al. (2017). The classifier is trained using a sliding window spanning the last 500 instances, i.e., $w = 500$. The verification latency in

[2] https://github.com/tpham93/al-for-sw-verification-latency.

Table 1 Properties for the data streams used in the following experiments

Dataset id	Dataset name	Instances	Features	Classes
D0	Synthetic 0	4000	2	2
D1	Synthetic 1	4000	2	3
D2	Synthetic 2	4000	2	4
D3	Phoneme (static)	5404	5 (+1 split)	2
D4	Mfeat-morphological	2000	6 (+1 split)	10
D5	Segment	2310	19 (+1 split)	7
D6	Kin8nm	8192	8 (+1 split)	2
D7	Space_ga	3107	6 (+1 split)	2
D8	Electricity	45312	9	2
D9	Luxembourg	1901	31	2
D10	NOAA weather	18159	8	2
D11	Rialto	82250	27	10

D0–D3 are synthetically generated data streams. D3–D7 are data streams derived from static benchmark datasets. There, a feature is excluded to derive a data stream that contains a concept shift (note the "+1 split" in the features column). D8–D11 are benchmark data streams

our experiments is assumed to be constant for most of the experiments. The verification latency we test are 50, 100, 150, 200, 250, and 300 instances. However, we also investigate the influence of variable latency in the experiments for Hypothesis 4 where the latency is modeled by a uniform distribution $\mathcal{U}(0, 300)$. We test three different budgets, namely 2%, 8%, and 32% of instances within the stream. As a selection of AL strategies, we use Var-Uncertainty (Var_Uncer), and Split (Split) as proposed in Zliobaite et al. (2014). Additionally, we test Probabilistic Active Learning with the Balanced Incremental Quantile Filter as proposed in Kottke et al. (2015). We test the three proposed individual wrappers F, S.B, S.F, and the two variations of SF, namely FS.B and FS.F, which are F combined with S.B and S.F, respectively. As a baseline approach, we use Random Selection (Rand) that samples instances randomly according to the provided budget. The pseudocode on how the traditional AL strategies are integrated into our framework is listed in the appendix.

All tested active learning strategies optimize the accuracy. Hence, we evaluate all tested strategies using the average accuracy across the whole data stream. The achievable accuracy might differ across datasets. Hence, we aggregate performances across multiple datasets by using the average rank over all repetitions.

The lower the assigned rank, the better the strategy's performance. The best performing strategy is assigned rank 1. In the case of ties, the average of the ranks is assigned to the strategies, e.g., two strategies that are tied as best performing strategies are assigned the rank 1.5. These ranks are then averaged over the ten repetitions to obtain the average ranking for the respective experiment configuration.

4.2 Effect of verification latency in stream-based AL

In this section, we investigate the effect of verification latency on the performance of traditional AL strategies.

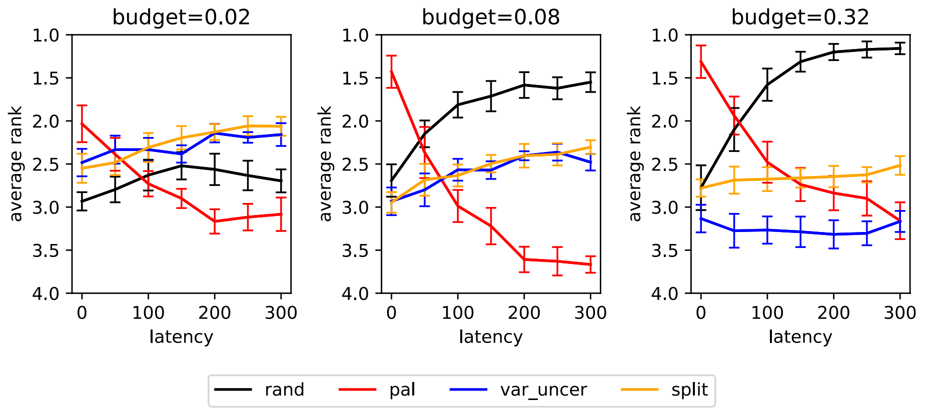


Fig. 4 The average rank and its standard error for the denoted AL strategies and budgets over all tested verification latencies (Color figure online)

Hypothesis 1 Existing AL strategies perform poorly under verification latency.

Findings: AL strategies that perform better than Random Selection in the state-of-the-art (without any verification latency) are likely to perform worse under the influence of verification latency without any adaptation.

Detailed Description: We compare the selected AL strategies, Var_Uncer, Split, PAL, and Rand for different verification latencies. Figure 4 shows the average rank over all datasets for each budget separately. The plots show that Rand performs the best with high latency compared to the AL strategies.

The results of Var_Uncer and Split show that they only perform better than rand for a budget of 2%. However, Split performs marginally better than Var_Uncer for 2% and 8% budget. For 32%, Split performs consistently better than Var_Uncer. The performance of PAL decreases the most when incorporating verification latency. While it has the best average rank for no verification latency, it performs worst with high latency. The results suggest, that the performance of traditional AL strategies decreases when compared to selecting instances randomly. Hence, AL provides less or no benefit, when this effect is not mitigated. A reason for this observation might be, that the utility assessment assumes that the label will be available immediately, which is not the case. Labels that will be obsolete are considered, while recently queried but unavailable labels are ignored.

Furthermore, AL strategies that perform well when no verification latency is present, seem to be impacted heavily under verification latency, as shown by PAL. In the detailed discussion for Hypothesis 2, we show why this phenomenon occurs.

4.3 FS improves label selection under verification latency

In this section, we investigate how FS can improve the performance of AL strategies under the influence of verification latency. Additionally, we investigate which combinations of the proposed wrappers for FS perform best.

Hypothesis 2 Combining traditional AL strategies with FS improves their performance.

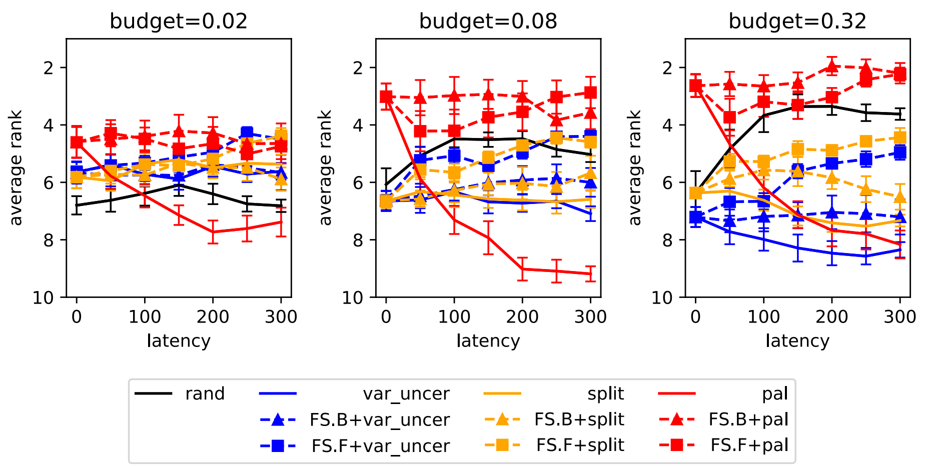


Fig. 5 The average rank and its standard error respective to the present verification latency for three different budgets (Color figure online)

Findings: The AL strategies perform better when combined with FS. This improvement is especially visible for Var_Uncer and PAL, but less for Split due to the randomization within Split. FS.B performs slightly better than FS.F when combined with PAL. For Var_Uncer and Split, FS.F performs better.

Detailed Description: Figure 5 extends Fig. 4 with additional strategies. Additionally to the former strategies, Fig. 5 includes those strategies wrapped in FS.B and FS.F. All in all, there are 10 strategies. Hence, the ranks now range from 1 to 10. The strategies wrapped in FS perform better than the traditional strategies. FS.B and FS.F are marked with triangles and rectangles, respectively. For PAL, FS.B performs better than FS.F. We assume that having discrete labeling instead of a fuzzy label influences the classifier's decision boundary more. However, Var_Uncer and Split perform better when combined with FS.F. We assume that F promotes exploration as information is removed and the classifier will be less certain. The simulation approaches S.B and S.F, increase the certainty of the classifier by increasing the number of training instances in the region of simulated labels.

Figure 6 shows the average ranks for a fixed verification latency of 200 and a budget of 8% for all datasets individually. The variance of the ranks over all data sets is high. The improvements for PAL are very high, except for D9, where all strategies seem to perform equally. The improvement for Var_Uncer and Split is also noticeable, except for D7 and D9. The average ranks over all datasets suggest that the traditional AL strategies perform better when combined with FS.

The performance of PAL without any adaptation performs the worst under verification latency, however, PAL combined with FS performs the best. To find an explanation for this improvement of PAL, Fig. 7 shows its behavior with no latency and a latency of 200 timesteps for a 1-dimensional dataset. Additionally, the latency of 200 timesteps while wrapped in FS.B is depicted for comparison as well. Each line represents the lifetime of an instance at location x and time t. The color represents the class of these instances. As for a latency of 200, we do not know the label for the first 200 timesteps. Hence, the lines are plotted in black for the first 200 timesteps.

The black lines mark queried labels, which are still unknown due to the verification latency. The colored lines mark queried and available labels. The queries for PAL

	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	average rank
rand	4.10	4.25	3.50	4.40	4.50	4.40	5.85	6.60	4.65	4.00	4.50	3.00	4.48
var_uncer	7.20	6.60	7.70	6.60	6.60	7.90	7.30	5.45	5.70	4.45	8.20	7.00	6.73
FS.B+var_uncer	6.60	6.85	6.60	4.70	8.40	5.30	3.60	7.85	3.30	4.85	4.70	8.30	5.92
FS.F+var_uncer	5.30	4.75	5.40	4.30	4.90	4.60	6.50	3.25	6.20	4.35	5.00	4.70	4.94
split	7.10	8.00	7.10	5.90	7.85	7.60	5.90	6.05	4.20	4.65	8.20	6.90	6.62
FS.B+split	6.80	7.05	7.50	4.85	8.30	6.70	3.10	8.40	2.45	5.40	4.00	7.80	6.03
FS.F+split	5.30	4.70	4.40	4.50	4.80	4.60	4.75	3.95	5.60	4.75	4.90	4.30	4.71
pal	9.60	9.80	9.80	9.70	5.45	9.90	9.00	7.00	9.80	8.20	10.00	10.00	9.02
FS.B+pal	1.60	1.70	1.80	3.15	2.40	1.30	4.50	3.20	5.90	7.25	2.30	1.00	3.01
FS.F+pal	1.40	1.30	1.20	6.90	1.80	2.70	4.50	3.25	7.20	7.10	3.20	2.00	3.55

Fig. 6 The average rank of the tested AL strategies for each dataset for a budget of 8% and a verification latency of 200 timesteps (Color figure online)

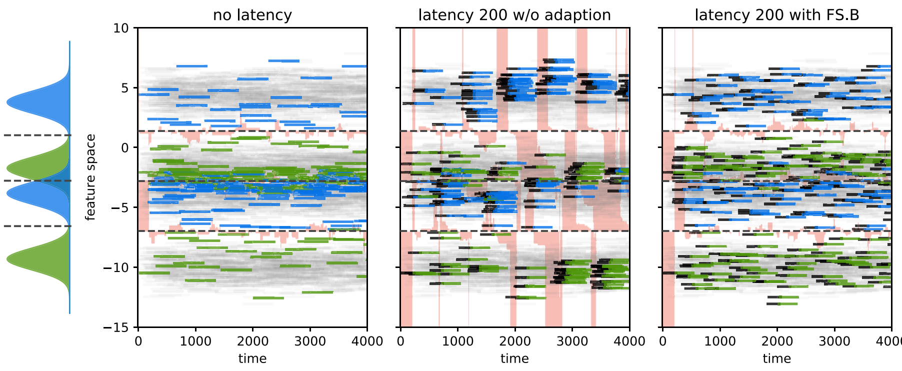


Fig. 7 These plots show the acquired instances and its labels over time for PAL. The y-axis represents the 1-dimensional feature space while the x-axis shows the time. The black lines from t_i^x to t_i^y show that an instance x_i has been acquired, while the following colored line from t_i^x to $t_i^x + w$ denotes the available class label. The red area depicts the area where the trained classifier deviates from the optimal decision boundary (Color figure online)

without adaptation and a latency of 200 appear to be in short bursts, where PAL focuses on different aspects of the feature space at a time.

The traditional PAL starts to query labels after the knowledge of a specific region is pushed out of the sliding window. The verification latency leads to PAL being unaware of its past acquisitions that happened in the last 200 timesteps. Hence, PAL queries instances until the first label arrives in that region after 200 timesteps. As this happens, PAL will forget another region, which it compensates in the same fashion. Contrary, when PAL is combined with FS, the forgetting of obsolete instances allows PAL to acquire new labels earlier. Thus, they will be available in time, right after the old labels will be forgotten. Additionally, by simulating the incoming labels, PAL knows that the recently queried labels will be available later.

Hypothesis 3 The combined wrappers FS.F and FS.B perform better than F, S.F, and S.B alone.

Findings: The results show that Var_Uncer and Split perform best when wrapped with FS.F in most cases. Contrary, PAL performs best when combined with FS.B.

Detailed Description: For this hypothesis, we combined the strategies not only with FS but also F, S.B, and S.F alone, to see which kind of adaptation is more important. Figure 8 shows the average rank of each strategy compared to its combinations with F, S.B, S.F, FS.B, and FS.F. These ranks are averaged over all datasets and plotted separately for each budget. We don't use the same color for the same base AL strategies, as that would reduce the distinguishability of the individual curves. Strategies using the F wrapper are plotted as dashed lines, while the curves for S.B and S.F are marked with triangle and square, respectively. Hence, the strategies combined with FS.B have a dashed curve with triangle markers. All strategies wrapped with FS.B and FS.F perform consistently better or equal to the non-wrapped strategies consistently. As Split is a randomized extension to Var_Uncer, both show similar behavior. The improvement provided for both is higher in most cases compared to F and S.F alone, except for very high latencies (i.e., 250 and 300) where F performs better. Split and Var_Uncer perform better with F, as it increases exploration by discarding data. The results of FS.F are comparable to F as S.F does not change the utility much. Fuzzy labels only influence the classifier's predictions marginally and the increase in number of labels is not considered by Split and Var_Uncer. The performance of PAL is improved by using F, S.F and S.B. By using F, PAL is able to account for data that will be unavailable soon. The simulation approaches S.F and S.B increases the number of labels for the areas, where labels are simulated. This decreases the utility for subsequent assessed labels near the simulated labels. With increasing budget, the difference between ranks becomes more visible, because the estimates can be very unreliable for small budgets due to the small number of labeled training instances. As the number of labels increases, the estimates become more robust, so the positive impact of forgetting and simulating becomes more noticeable.

4.4 Handling of variable verification latency

In this section, we investigate the effect of variable verification latency instead of constant verification latency as tested above.

Hypothesis 4 FS is not restricted to constant verification latencies but is able to mitigate the effect of variable latencies.

Findings: Using FS, AL strategies are aware of the knowledge missing in the future, as long as the delay until the labels become available is known. Hence, FS improves the performance even when having variable verification latency instead of constant verification latency.

Detailed Description: We investigate this hypothesis by assigning each instance an individual verification latency. This latency is distributed according to $\mathcal{U}(0, 300)$ but each instance's latency stays constant across all repetitions. Figure 9 shows the results for a budget of 8%, similarly to Fig. 6. The variable verification latency leads to a worse

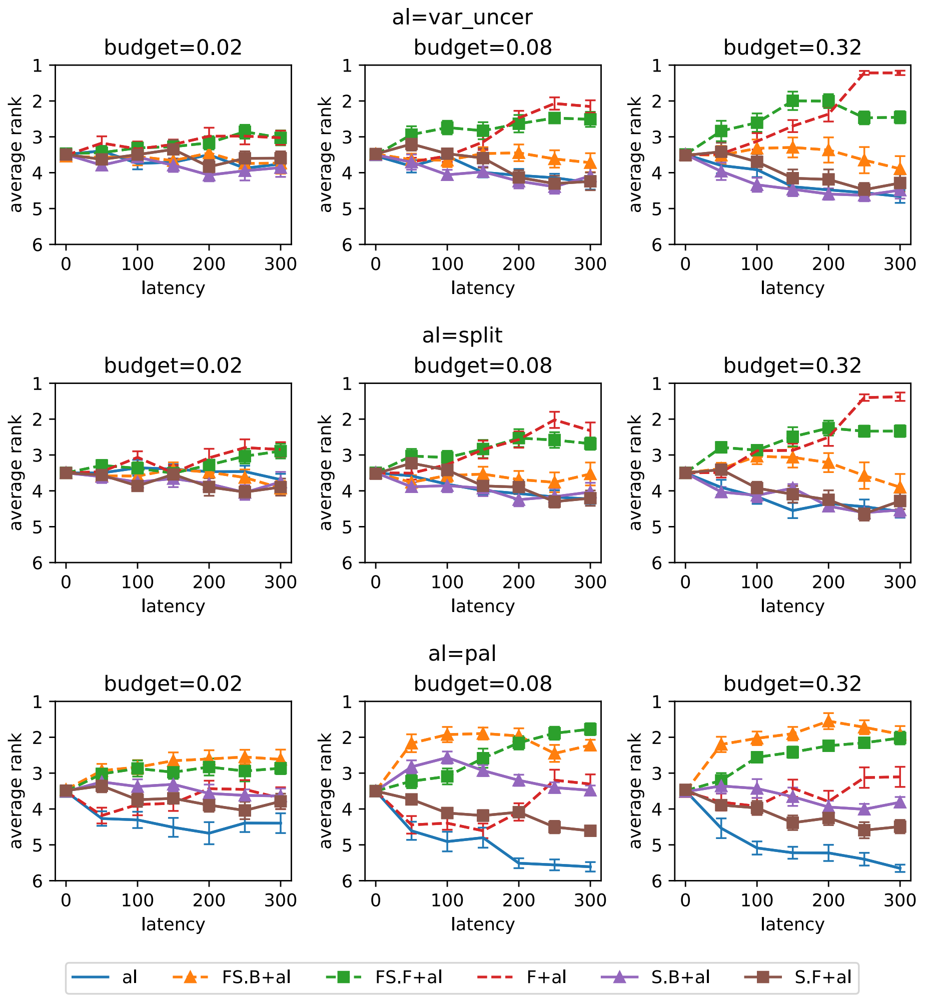
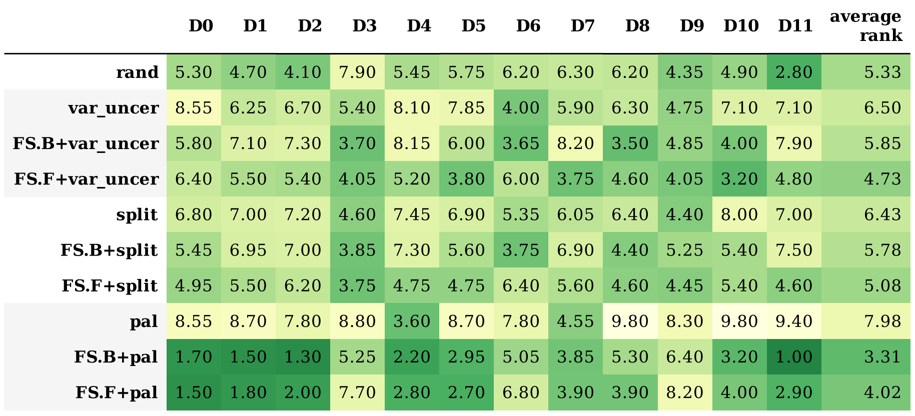


Fig. 8 The average rank and its standard error respective to the present verification latency for three different budgets. The plots are split among the base AL strategies Var_Uncer, Split and PAL with all combination for the wrapper strategies (Color figure online)

performing Random Selection when compared to the constant verification latency. The average rank over all datasets, suggests that the wrappers improve the performance of all AL strategies. When using Var_Uncer and Split in combination with FS.F, their performances are improved so that they perform slightly better than random. PAL benefits the most from the proposed approaches as both variants of the adapted PAL outperform all other strategies. The results for this experiment match the observations of the previously conducted experiments.



	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	average rank
rand	5.30	4.70	4.10	7.90	5.45	5.75	6.20	6.30	6.20	4.35	4.90	2.80	5.33
var_uncer	8.55	6.25	6.70	5.40	8.10	7.85	4.00	5.90	6.30	4.75	7.10	7.10	6.50
FS.B+var_uncer	5.80	7.10	7.30	3.70	8.15	6.00	3.65	8.20	3.50	4.85	4.00	7.90	5.85
FS.F+var_uncer	6.40	5.50	5.40	4.05	5.20	3.80	6.00	3.75	4.60	4.05	3.20	4.80	4.73
split	6.80	7.00	7.20	4.60	7.45	6.90	5.35	6.05	6.40	4.40	8.00	7.00	6.43
FS.B+split	5.45	6.95	7.00	3.85	7.30	5.60	3.75	6.90	4.40	5.25	5.40	7.50	5.78
FS.F+split	4.95	5.50	6.20	3.75	4.75	4.75	6.40	5.60	4.60	4.45	5.40	4.60	5.08
pal	8.55	8.70	7.80	8.80	3.60	8.70	7.80	4.55	9.80	8.30	9.80	9.40	7.98
FS.B+pal	1.70	1.50	1.30	5.25	2.20	2.95	5.05	3.85	5.30	6.40	3.20	1.00	3.31
FS.F+pal	1.50	1.80	2.00	7.70	2.80	2.70	6.80	3.90	3.90	8.20	4.00	2.90	4.02

Fig. 9 The performances for experiments with variable verification latencies with a budget of 8% (Color figure online)

5 Conclusion

In this article, we covered the problem of active learning under verification latency and proposed a general adaptation strategy for AL algorithms to overcome the corresponding challenges. We evaluated our work based on four hypotheses that show that verification latency decreases traditional active learning performance. We also showed that this decrease is mitigated by assessing the label's utility differently compared to traditional AL strategies. Hence, the assessment should be based on data that is available by the time the label is accessible. We demonstrated that by forgetting obsolete data and simulating incoming labels using Forgetting and Simulating (FS), traditional AL strategies assess a label's utility more accurately. We also showed that FS works under variable verification latency. In summary, this work provides a framework to adapt stream-based AL approaches to scenarios with verification latency, which allows their use in a broader range of applications.

Further research may extend the tested datasets to other benchmark or real-world datasets. In settings with variable verification latency, the lifetime of a label can be incorporated into the utility assessment. The current F approach is reactionary in nature, as it only reconsiders a region after obsolete labels are discarded. It may be preferable to query new labels before old labels become obsolete. This proactive behavior can be achieved by forgetting labels earlier. For this article, we researched the behavior using a Parzen Window Classifier. However, technically the approach is not restricted to that type of classifier. For instance, semi-supervised classifiers are a fitting choice and would complement the active learning component well. So far we assumed that our classifier is operating in a sliding window, but recent state-of-the-art methods use change detection algorithms to forget outdated data. This will be addressed in future work. However, as we do not know when these instances might be forgotten due to change detection algorithms, we need estimates for the most likely change point. Furthermore, FS may be combined with a verification latency-aware classifier. These classifiers aim to improve their estimates by tracking change in a data stream, for example. Finally, even though we assume that each label's arrival date is known, FS might be used in settings where the individual verification latency is not known apriori but can be estimated.

Appendix A pseudocodes

In this section, we show how the tested AL strategies can be incorporated into the framework that we specify in Algorithm 2. The methods proposed in Zliobaite et al. (2014), i.e., Var-Uncertainty and Split, have to be restructured to separate the utility assessment and the budget management. For PAL with BIQF, there is no restructuring necessary, as it separates both already.

Random Selection The implementation for Random Selection is quite simple. Its utility assessment consists of sampling a random variable uniformly distributed between 0 and 1 (line 2 and 3 in Algorithm 6). The query function simply compares this random sample against the targeted budget to determine whether to sample the instance (line 2 to 6 in Algorithm 7).

Algorithm 6 Utility function for Random Selection

```
1: function utilityRandom($x, f, L$)
2:     $u_{\mathrm{rand}} \sim U[0, 1]$
3:     return $u_{\mathrm{rand}}$
4: end function
```

Algorithm 7 Query function for Random Selection

```
1: function queryRandom($u$)
2:     if $u \leq B$ then
3:         return 1
4:     else
5:         return 0
6:     end if
7: end function
```

Var-Uncertainty For our experiments, the utility assessment for Var-Uncertainty, consists solely of the certainty assessment for each instance x (line 2 in Algorithm 8). The query function for Var-Uncertainty is depicted in Algorithm 11. Each of the methods proposed in Zliobaite et al. (2014) compare the utility against a threshold. In the case of Var-uncertainty, the threshold is adapted over time (line 4 and 7). Additionally, there is a check, whether the targeted budget is exhausted already (line 2).

Algorithm 8 Utility function for Var-Uncertainty

```
1: function utilityVarUncertainty($x, f, L$)
2:     return $\max_{y' \in \{1,\ldots,C\}} f^L(y'|x)$
3: end function
```

Algorithm 9 Query function for Var-Uncertainty

```
function queryVarUncertainty($u$)
    if $\hat{b} < B$ then
        if $u \leq \theta$ then
            $\theta \leftarrow \theta \cdot (1 - s)$
            return 1
        else
            $\theta \leftarrow \theta \cdot (1 + s)$
            return 0
        end if
    end if
    return 0
end function
```

Split Split is structured very similarly to Var-Uncertainty, as the only difference is, that Random Selection is used sometimes to increase the exploration. Hence, the utility assessment in Algorithm 10 is equivalent to Algorithm 8. Algorithm 11 shows the query function for Split. As in Algorithm 9, Split checks whether the budget is exhausted already (line 2). Then, it is checked whether to use Random Selection or Var-Uncertainty (line 3) and the results of those strategies are returned (line 4, 5, and 7).

Algorithm 10 Utility function for Split

```
1: function utilitySplit($x, f, L$)
2:     return $\max_{y' \in \{1,\ldots,C\}} f^L(y'|x)$
3: end function
```

Algorithm 11 Query function for Split

```
1:  function querySplit($u$)
2:      if $\hat{b} < B$ then
3:          if $u' < v$ for $u' \sim U[0,1]$ then
4:              $u_{\text{rand}} \sim U[0,1]$
5:              return queryRandom($u_{\text{rand}}$)
6:          else
7:              return queryVarUncertainty($u$)
8:          end if
9:      end if
10:     return 0
11: end function
```

PAL PAL for datastreams as proposed in Kottke et al. (2015), splits the approach between a spatial utility and temporal utility assessment. For the spatial utility assessment, we use McPAL, to be able to handle datasets with more than two classes. The spatial utility assessment is done entirely in the utility function, i.e., Algorithm 12. To decide whether the spatial utility, is sufficient to acquire the instance's label, Kottke et al. use BIQF. Hence, BIQF assesses the temporal utility based on the spatial utility. This is done in the query function, i.e., Algorithm 13.

Algorithm 12 Utility function for PAL

```
1: function UTILITYPAL($x, f, L$)
2:     return McPAL($x, f, L$)
3: end function
```

Algorithm 13 Query function for PAL

```
1: function QUERYPAL($u$)
2:     return BIQF($u$)
3: end function
```

Acknowledgements We would like to thank our colleagues from the Intelligent Embedded Systems group, in particular Lukas Rauch, Yujiang He, David Meier, and Alexander Benz. Furthermore, we thank the anonymous reviewers for their helpful comments and suggestions.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and issues in data stream systems. In ACM SIGMOD-SIGACT-SIGART, ACM, New York, NY, USA, PODS 02, pp 1–16.

Bondu, A., Lemaire, V., & Boulle, M. (2010). Exploration vs. exploitation in active learning: A bayesian approach. In IJCNN, IEEE, pp 1–7.

Chapelle, O. (2005). Active learning for parzen window classifier. In AISTATS, Max–Planck–Gesellschaft, pp. 49–56.

Chaudhuri, A., Kakde, D., Sadek, C., Gonzalez, L., & Kong, S. (2017). The mean and median criteria for kernel bandwidth selection for support vector data description. In ICDM Workshops, pp. 842–849.

Cohn, D. A. (1993). Neural network exploration using optimal experiment design. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *NIPS* (pp. 679–686). Burlington: Morgan Kaufmann.

Cohn, D. (2010). Active learning. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 10–14). New York: Springer.

dos Reis, D. M., Flach, P., Matwin, S., & Batista, G. (2016). Fast unsupervised online drift detection using incremental Kolmogorov–Smirnov test. In SIGKDD, ACM, New York, NY, USA, KDD 16, pp. 1545–1554.

Dyer, K. B., Capo, R., & Polikar, R. (2014). Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *TNNLS, 25*(1), 12–26.

Frederickson, C., & Polikar, R. (2018). Resampling techniques for learning under extreme verification latency with class imbalance. In IJCNN, IEEE, pp. 1–8.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In SIGKDD, Association for Computing Machinery, pp. 329–338.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *CSUR, 46*(4), 1–44.

Grzenda, M., Gomes, H. M., & Bifet, A. (2019). Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery*.

Hammoodi, M., Stahl, F., & Tennant, M. (2016). Towards online concept drift detection with feature selection for data stream classification. In ECAI, Frontiers in Artificial Intelligence and Applications, vol 285, pp. 1549–1550.

Hofer, V., & Krempl, G. (2013). Drift mining in data: A framework for addressing drift in classification. *CSDA, 57*(1), 377–391.

Ienco, D., Bifet, A., Zliobaite, I., & Pfahringer, B. (2013). Clustering based active learning for evolving data streams. In J. Fürnkranz, E. Hüllermeier, & T. Higuchi (Eds.), *Discovery Science*. Lecture Notes in Artificial Intelligence, (Vol. 8140, pp. 79–93). Springer.

Ienco, D., Pfahringer, B., & Zliobaitė, I. (2014). High density-focused uncertainty sampling for active learning over evolving stream data. In SIGKDD BigMine, pp. 133–148.

Kelly, M. G., Hand, D. J., & Adams, N. M. (1999). The impact of changing populations on classifier performance. In SIGKDD, pp. 367–371.

Klinkenberg, R., & Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drifts. In Workshop Notes of the AAAI-98/ICML-98 workshop Learning for Text Categorization, AAAI Press, pp. 33–40.

Kottke, D., Herde, M., Minh, T. P., Benz, A., Mergard, P., Roghman, A., Sandrock, C., & Sick, B. (2021a). scikit-activeml: A library and toolbox for active learning algorithms. Preprints, 2021030194.

Kottke, D., Krempl, G., & Spiliopoulou, M. (2015). Probabilistic active learning in datastreams. In É. Fromont, T. D. Bie, & M. van Leeuwen (Eds.), *IDA*. Lecture Notes in Computer Science, (Vol. 9385, pp. 145–157). Springer.

Kottke, D., Herde, M., Sandrock, C., Huseljic, D., Krempl, G., & Sick, B. (2021b). Toward optimal probabilistic active learning using a Bayesian approach. *Machine Learning, 110,* 1199–1231.

Krempl, G. (2011). The algorithm apt to classify in concurrence of latency and drift. In IDA, Springer, pp. 222–233.

Krempl, G., & Hofer, V. (2011). Classification in presence of drift and latency. In M. Spiliopoulou, H. Wang, D. Cook, J. Pei, W. Wang, O. Zaïane, & X. Wu (Eds.), *ICDM Workshops.* IEEE.

Krempl, G., Lang, D., & Hofer, V. (2019). Temporal density extrapolation using a dynamic basis approach. *Data Mining and Knowledge Discovery,33*(5), 1323–1356. Special Issue of the ECML/PKDD 2019 Journal Track.

Krempl, G., Ha, T. C., & Spiliopoulou, M. (2015a). Clustering-based optimised probabilistic active learning (COPAL). In N. Japkowicz & S. Matwin (Eds.), *Discovery Science* (Vol. 9356, pp. 101–115). New York: Springer.

Krempl, G., Kottke, D., & Lemaire, V. (2015b). Optimised probabilistic active learning (OPAL) for fast, non-myopic, cost-sensitive active classification. *Machine Learning, 100,* 2.

Kumar, P., & Gupta, A. (2020). Active learning query strategies for classification, regression, and clustering: A survey. *JCST, 35*(4), 913–945.

Kuncheva, L. I. (2008). Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In O. Okun & G. Valentini (Eds.), *SUEMA*. Studies in Computational Intelligence, (Vol. 245, pp. 5–10). Springer.

Kuncheva, L. I., & Sánchez, J. S. (2008). Nearest neighbour classifiers for streaming data with delayed labelling. In ICDM, pp. 869–874.

Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. In SIGIR, Springer, New York, NY, USA, SIGIR 94, pp. 3–12.

Loy, C. C., Hospedales, T. M., Xiang, T., & Gong, S. (2012). *Stream-based joint exploration-exploitation active learning*. In CVPR, IEEE pp. 1560–1567.

Marrs, G., Hickey, R., & Black, M. (2010). The impact of latency on online classification learning with concept drift. In Y. Bi & M. A. Williams (Eds.), *KSEM*. Lecture Notes in Computer Science, (Vol. 6291, pp. 459–469). Springer.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Cambridge: MIT Press.

Parreira, P., & Prati, R. (2019). *Aprendizagem ativa em fluxo de dados com latência intermediária*. In ENIAC, SBC, pp. 365–376

Pham, M. T., Kottke, D., Tsarenko, A., Gruhl, C., & Sick, B. (2020). Improving self-adaptation for multi-sensor activity recognition with active learning. In IJCNN.

Plasse, J., & Adams, N. (2016). Handling delayed labels in temporally evolving data streams. In IEEE Big-Data, pp. 2416–2424.

Razavi-Far, R., Hallaji, E., Saif, M., & Ditzler, G. (2019). A novelty detector and extreme verification latency model for nonstationary environments. *IEEE TIE, 66*(1), 561–570.

Roy, N., & McCallum, A. (2001). Toward optimal active learning through sampling estimation of error reduction. *ICML* (pp. 441–448). San Francisco, CA, USA: Morgan Kaufmann.

Schlimmer, J. C., & Granger, R. H. (1986). Beyond incremental processing: Tracking concept drift. In AAAI, pp. 502–507.

Settles, B. (2012). Active Learning. No. 18 in Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers.

Shaker, A., & Hüllermeier, E. (2013). Recovery analysis for adaptive learning from non-stationary data streams. *Advances in Intelligent Systems and Computing, 226,* 289–298.

Souza, V., Pinho, T., & Batista, G. (2018). Evaluating stream classifiers with delayed labels information. In BRACIS, pp. 408–413.

Umer, M. (2017). Learning extreme verification latency quickly with importance weighting: Fast compose and level_iw. PhD thesis, Rowan University.

Umer, M., & Polikar, R. (2020). Comparative analysis of extreme verification latency learning algorithms. arXiv:2011.14917.

Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2007). Active learning from data streams. In ICDM, IEEE Computer Society, Washington, DC, USA, ICDM 07, pp. 757–762.

Žliobaitė, I. (2010). Change with delayed labeling: When is it detectable? In ICDM Workshops, pp. 843–850.

Zliobaite, I., Bifet, A., Pfahringer, B., & Holmes, G. (2014). Active learning with drifting streaming data. *TNNLS, 25,* 27–39.