

# Prequential AUC: properties of the area under the ROC curve for data streams with concept drift

Dariusz Brzezinski<sup>1</sup>  · Jerzy Stefanowski<sup>1</sup>

Received: 27 April 2016 / Revised: 15 November 2016 / Accepted: 2 January 2017 /

Published online: 16 January 2017

© The Author(s) 2017. This article is published with open access at Springerlink.com

**Abstract** Modern data-driven systems often require classifiers capable of dealing with streaming imbalanced data and concept changes. The assessment of learning algorithms in such scenarios is still a challenge, as existing online evaluation measures focus on efficiency, but are susceptible to class ratio changes over time. In case of static data, the area under the receiver operating characteristics curve, or simply AUC, is a popular measure for evaluating classifiers both on balanced and imbalanced class distributions. However, the characteristics of AUC calculated on time-changing data streams have not been studied. This paper analyzes the properties of our recent proposal, an incremental algorithm that uses a sorted tree structure with a sliding window to compute AUC with forgetting. The resulting evaluation measure, called prequential AUC, is studied in terms of: visualization over time, processing speed, differences compared to AUC calculated on blocks of examples, and consistency with AUC calculated traditionally. Simulation results show that the proposed measure is statistically consistent with AUC computed traditionally on streams without drift and comparably fast to existing evaluation procedures. Finally, experiments on real-world and synthetic data showcase characteristic properties of prequential AUC compared to classification accuracy, G-mean, Kappa, Kappa M, and recall when used to evaluate classifiers on imbalanced streams with various difficulty factors.

**Keywords** AUC ROC · Concept drift · Data streams · Classifier evaluation

---

**Electronic supplementary material** The online version of this article (doi:[10.1007/s10115-017-1022-8](https://doi.org/10.1007/s10115-017-1022-8)) contains supplementary material, which is available to authorized users.

---

✉ Dariusz Brzezinski  
dariusz.brzezinski@cs.put.poznan.pl

<sup>1</sup> Institute of Computing Science, Poznan University of Technology, Ul. Piotrowo 2,  
60-965 Poznan, Poland

## 1 Introduction

In many data mining applications, e.g., in sensor networks, banking, energy management, or telecommunication, the need for processing rapid data streams is becoming more and more common [50]. Such demands have led to the development of classification algorithms that are capable of processing instances one by one, while using limited memory and time. Furthermore, due to the non-stationary characteristics of streaming data, classifiers are often additionally required to react to *concept drifts*, i.e., changes in definitions of target classes over time [20]. To fulfill these requirements, several data stream classification algorithms have been proposed in recent years; for their review see, e.g., [7, 14, 20].

An important issue when learning classifiers from streaming environments is the way classifiers are evaluated. Traditionally, the predictive performance of a static classifier is measured on a separately curated set of testing examples [31]. However, batch techniques are not applied in streaming scenarios, where simpler and less costly incremental procedures are used, such as interleaving testing with training [33]. Moreover, since data streams can evolve over time, predictive abilities of classifiers are usually calculated with forgetting, i.e., sequentially on the most recent examples [21]. As a result, stream classifiers are mostly evaluated using the least computationally demanding measures, such as accuracy or error rate.

Nevertheless, simple evaluation measures are not sufficient for assessing classifiers when data streams are affected by additional complexity factors. In particular, this problem concerns *class imbalance*, i.e., situations when one of the target classes is represented by much less instances than other classes [27]. Class imbalance is an obstacle even for learning from static data, as classifiers are biased toward the majority classes and tend to misclassify minority class examples [10]. In such cases, simple performance metrics, such as accuracy, give overly optimistic estimates of classifier performance [41]. That is why, for static imbalanced data researchers have proposed several more relevant measures such as precision/recall, sensitivity/specificity, G-mean, and, in particular, the area under the ROC curve [26, 27].

The *area under the ROC curve*, or simply AUC, summarizes the relationship between the true and false positive rate of a binary classifier, for different decision thresholds [18]. Several authors have shown that AUC is more preferable for classifier evaluation than total accuracy [29, 30, 41], making it one of the most popular metrics for static imbalanced data. However, in order to calculate AUC one needs to sort a given dataset and iterate through each example. This means that AUC cannot be directly computed on large data streams, as this would require scanning through the entire stream after each example. That is why, the use of AUC for data streams has been limited only to estimations on periodical holdout sets [28, 44] or entire streams [13, 36], making it either potentially biased or computationally infeasible for practical applications.

To address these limitations, in an earlier paper [9] we introduced a new approach for calculating AUC. We presented an efficient algorithm, which incorporates a sorted tree structure with a sliding window as a forgetting mechanism, making it both computationally feasible and appropriate for concept-drifting streams. As a result, we have introduced a new evaluation measure, called *prequential AUC*, suitable for assessing classifiers on evolving data streams.

To the best of our knowledge, prequential AUC is the first proposal for efficiently calculating AUC on drifting data streams. Nonetheless, the properties of this new evaluation measure have not been thoroughly investigated in our previous work [9]. Thus, the aim of this paper is to carry out a detailed study of basic properties of prequential AUC, which in our view include the following issues.

First, we will analyze whether prequential AUC leads to consistent results with traditional AUC calculated on stationary data streams. For this purpose, we will use the definitions of consistency and discriminancy proposed by Huang and Ling [30]. Although the computation of traditional batch AUC is infeasible for real streams, it nevertheless serves as the best AUC estimate for data without concept changes. Moreover, the proposed measure will be compared with earlier attempts at using AUC to assess stream classifiers, in particular those based on block processing [28,44]. Additionally, we perform a series of experiments, which analyze the use of different AUC calculation procedures for visualizing concept changes over time and assess the processing speed of the proposed measure. Finally, we compare and characterize classifier evaluations made by AUC and other measures commonly used for imbalanced data: G-mean, recall, and the Kappa statistic. For this purpose, we design a series of synthetic datasets that simulate various difficulty factors, such as high class imbalance ratios, sudden and gradual ratio changes, minority class fragmentation (small disjuncts), appearing minority class subconcepts, and minority–majority class swaps.

In summary, the main contributions of this paper are as follows:

- Section 4.1 discusses the applicability of prequential AUC to performance visualization over time, for stationary and drifting streams;
- Section 4.2 analyzes the consistency and discriminancy of prequential AUC compared to AUC computed traditionally on an entire stationary dataset;
- Section 5.3 summarizes experiments performed to evaluate the speed of the proposed measure;
- Section 5.4 analyzes parameter sensitivity of prequential AUC;
- Section 5.5 compares prequential AUC to other class imbalance evaluation measures on a series of streams involving various difficulty factors.

Section 2 provides basic background for the conducted study and covers related work. To make the paper self-contained, in Sect. 3 we recall our algorithm [9] for computing AUC online with forgetting. Sections 5.1 and 5.2 summarize the experimental setup and used datasets. Section 6 concludes the paper and discusses potential lines of future research. All of the algorithms described in the paper are implemented in Java as part of the MOA framework [3] and are publicly available.

## 2 Related work

### 2.1 Stream classifier evaluation

Requirements posed by data streams make *processing time*, *memory usage*, *adaptability*, and *predictive performance* key classifier evaluation criteria [7,22]. In this paper, we focus only on that last criterion.

The predictive performance of stream classifiers is usually assessed using evaluation measures known from static supervised classification, such as accuracy or error rate. However, due to the size and speed of data streams, re-sampling techniques such as cross-validation are deemed too expensive and simpler error estimation procedures are used [33]. In particular, stream classifiers are evaluated either by using a holdout test set or by interleaving testing with training, one instance after another or using blocks of examples [20]. More recently, Gama et al. [21] advocated the use of prequential procedures for calculating performance measures in evolving data streams. The term *prequential* (blend of predictive and sequential) stems

from online learning [39] and is used in data stream mining literature to denote algorithms that base their functioning only on the most recent data, rather than the entire stream.

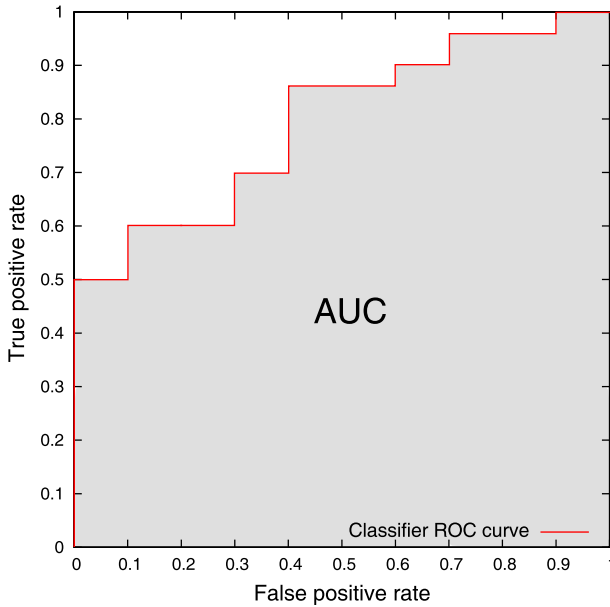
Prequential accuracy was the first measure for evaluating stream classifiers, which worked according to the aforementioned predictive-sequential procedure. The authors of [21] have shown that computing accuracy only using the most recent examples, instead of the entire stream, is more appropriate for continuous assessment and drift detection. However, it is worth noting that prequential accuracy inherits the weaknesses of traditional accuracy, that is, variance with respect to class distribution and promoting majority class predictions in imbalanced data. Recent works have also introduced the use of prequential recall [46], i.e., class-specific accuracy; however, such a measure must be recorded for each single class. A popular way of combining information about many imbalanced classes is the use of the geometric mean (G-mean) of all class-specific accuracies [35], which has also been recently recognized in online learning [47]. Finally, for imbalanced streams Bifet and Frank [2] proposed to prequentially calculate the Kappa statistic with a sliding window. The Kappa statistic ( $\kappa$ ) is an alternative to accuracy, that may be seen as the probability of outperforming a base classifier. Furthermore, this metric has been recently extended to take into account class skew ( $\kappa_m$ ) [4] and temporal dependence ( $\kappa_{per}$ ) [49].

## 2.2 Area under the ROC curve

The receiver operating characteristics (ROC) curve is a graphical plot that visualizes the relation between the true positive rate and the false positive rate for a classifier under varying decision thresholds [18]. It was first used in signal detection to represent the trade-off between hit and false alarm rates [15]; however, it has also been extensively applied in medical diagnosis [37] and, more recently, to evaluate machine learning algorithms [17,41]. Unlike single point metrics, the ROC curve compares classifier performance across the entire range of class distributions, offering an evaluation encompassing a wide range of operating conditions.

The area under the ROC curve, or AUC, is one of the most popular classifier evaluation metrics for static imbalanced data [31]. Its basic interpretation and calculation procedure can be explained by analyzing the way in which ROC curves are created. If we denote examples of two classes distinguished by a binary classifier as *positive* and *negative*, the ROC curve is created by plotting the proportion of positives correctly classified (*true positive rate*) against the proportion of negatives incorrectly classified (*false positive rate*). If a classifier outputs a score proportional to its belief that an instance belongs to the positive class, decreasing the classifier's decision threshold (i.e., the score above which an instance is deemed to belong to the positive class) will increase both true and false positive rates. Varying the decision threshold results in a piecewise linear curve, called the ROC curve, which is presented in Fig. 1. AUC summarizes the plotted relationship in a scalar metric by calculating the area under the ROC curve.

The popularity of AUC for static data comes from the fact that it is invariant to changes in class distribution. If the class distribution changes in a test set, but the underlying class-conditional distributions from which the data are drawn stay the same, the ROC curve will not change [48]. Moreover, for scoring classifiers it has a very useful statistical interpretation as the expectation that a randomly drawn positive example receives a higher score than a random negative example. Therefore, it can be used to assess machine learning algorithms which are used to rank cases, e.g., in credit scoring, customer targeting, or churn prediction. Furthermore, AUC is equivalent to the Wilcoxon–Mann–Whitney (WMW)  $U$  statistic test of ranks [24]. It is worth mentioning that this statistical interpretation led to the development of algorithms that are capable of computing AUC without building the ROC curve itself, by



**Fig. 1** Example ROC curve

counting the number of positive–negative example misorderings in the ranking produced by classifier scores [48]. Additionally, several researchers have shown that for static data AUC is a more preferable classifier evaluation measure than total accuracy[30].

Nonetheless, the use of AUC for evaluating classifier performance has been challenged in a publication by Hand et al. [23]. Therein, the authors derive a linear relationship between a classifier’s expected minimum loss and AUC under a classifier-dependent distribution over cost proportions. In other words, the authors have shown that the fact that two classifiers have equal AUC does not necessarily imply they have equal expected minimum loss. To amend this deficiency, Hand et al. [23] proposed an alternative for allowing fairer comparisons: the H-measure. However, this criticism concerns using AUC to assess classification performance, and does not affect its usefulness for evaluating ranking performance. Moreover, the relationship between AUC and expected loss is classifier-dependent only if one takes into account solely optimal thresholds [19]. This is especially of note in the context of data stream mining, as due to concept drift an optimal threshold for a part of the stream will be suboptimal after a change occurs. Therefore, if one anticipates concept or data distribution changes, it might be better to consider classification performance for all thresholds created by a classifier, which is exactly what AUC is measuring [19]. Despite the fact that it has been surrounded with some controversy, AUC still remains one of the most used measures under imbalanced domains; for a broader discussion and extensions of AUC see [6].

AUC has also been used for data streams, however, in a very limited way. Some researchers chose to calculate AUC using entire streams [13,36], while others used periodical holdout sets [28,44]. Nevertheless, it was noticed that periodical holdout sets may not fully capture the temporal dimension of the data [33], whereas evaluation using entire streams is neither feasible for large datasets nor suitable for drift detection. It is also worth mentioning that an algorithm for computing AUC incrementally has also been proposed [5], yet one which calculates AUC from all available examples and is not applicable to evolving data streams.

Although the cited works show that AUC is recognized as a measure which should be used to evaluate data stream classifiers, until now it has been computed the same way as for static data. In the following section, we present a simple and efficient algorithm for calculating AUC incrementally with forgetting, which we previously introduced in [9]. Later, we investigate the properties of the resulting evaluation measure with respect to classifiers for evolving imbalanced data streams.

### 3 Prequential AUC

As AUC is calculated on ranked examples, we will consider *scoring classifiers*, i.e., classifiers that for each predicted class label additionally return a numeric value (*score*) indicating the extent to which an instance is predicted to be positive or negative. Furthermore, we will limit our analysis to binary classification. It is worth mentioning that most classifiers can produce scores, and many of those that only predict class labels can be converted to scoring classifiers. For example, decision trees can produce class-membership probabilities by using Naive Bayes leaves or averaging predictions using bagging [40]. Similarly, rule-based classifiers can be modified to produce instance scores indicating the likelihood that an instance belongs to a given class [16].

We propose to compute AUC incrementally after each example using a special sorted structure combined with a sliding window forgetting mechanism. It is worth noting that, since the calculation of AUC requires sorting examples with respect to their classification scores, it cannot be computed on an entire stream or using fading factors without remembering the entire stream. Therefore, for AUC to be computationally feasible and applicable to evolving concepts, it must be calculated using a sliding window.

A sliding window of scores limits the analysis to the most recent data, but to calculate AUC, scores have to be sorted. To efficiently maintain a sorted set of scores, we propose to use the *red-black tree* data structure [1]. A red-black tree is a self-balancing binary search tree that is capable of adding and removing elements in logarithmic time while requiring minimal memory. With these two structures, we can efficiently calculate AUC sequentially on the most recent examples. Algorithm 1 lists the pseudo-code for calculating prequential AUC. Contrary to [9], here we present an extended version of the algorithm that deals with score ties.

For each incoming labeled example, the score assigned to this example by the classifier is inserted into the window (line 15) as well as the red-black tree (line 10), and if the window of examples has been exceeded, the oldest score is removed (lines 5 and 15). The red-black tree is sorted in descending order according to scores, in case of score ties positives before negatives, and in ascending order according to arrival time. This way, we maintain a structure that facilitates the calculation of AUC and ensures that the oldest score in the sliding window will be promptly found in the red-black tree. After the sliding window and tree have been updated, AUC is calculated by summing the number of positive examples occurring before each negative example (lines 18–28) and normalizing that value by all possible pairs  $pn$  (line 29), where  $p$  is the number of positives and  $n$  is the number of negatives in the window. This method of calculating AUC, proposed in [48], is equivalent to summing the area of trapezoids for each pair of sequential points on the ROC curve, but more suitable for our purposes, as it requires very little computation given a sorted collection of scores. We note that in line 26 we take into account score ties between positive and negative examples by reducing the increment of AUC.

**Algorithm 1** Prequential AUC

```

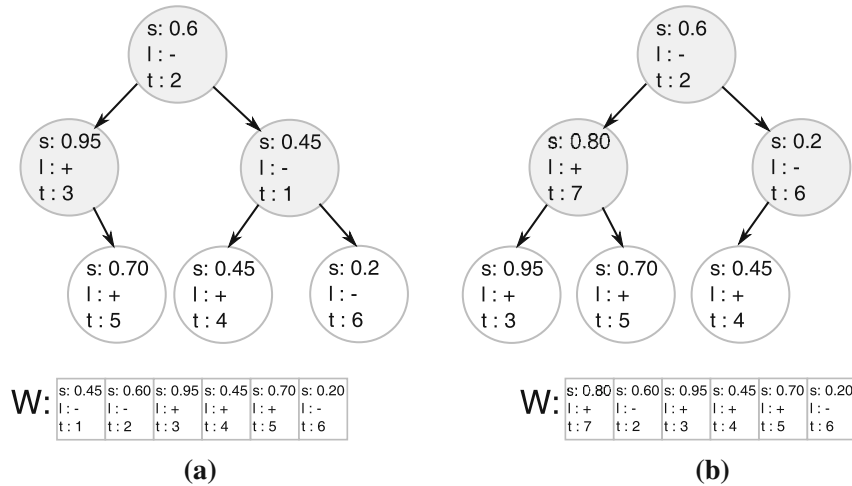
Input:  $S$ : stream of examples,  $d$ : window size
Output:  $\hat{\theta}$ : prequential AUC after each example
1:  $W \leftarrow \emptyset$ ;  $idx \leftarrow 0$ ; ▷ Sliding window/example number
2:  $n \leftarrow 0$ ;  $p \leftarrow 0$ ; ▷ Number of negatives/positives in the window
3: for all scored examples  $\mathbf{x}^t \in S$  do
4:   if  $idx \geq d$  then
5:      $scoreTree.remove(W[idx \bmod d])$ ; ▷ Remove oldest score
6:     if  $isPositive(W[idx \bmod d])$  then
7:        $p \leftarrow p - 1$ ;
8:     else
9:        $n \leftarrow n - 1$ ;
10:     $scoreTree.add(\mathbf{x}^t)$ ; ▷ Add new score
11:    if  $isPositive(\mathbf{x}^t)$  then
12:       $p \leftarrow p + 1$ ;
13:    else
14:       $n \leftarrow n + 1$ ;
15:     $W[idx \bmod d] \leftarrow \mathbf{x}^t$ ;
16:     $idx \leftarrow idx + 1$ ;
17:     $AUC \leftarrow 0$ ;  $c \leftarrow 0$ ;  $prevc \leftarrow 0$ ;  $lastPosScore \leftarrow \infty$ ; ▷ Calculate AUC [48]
18:    for all scored examples  $s \in scoreTree$  do
19:      if  $isPositive(s)$  then
20:        if  $s.value \neq lastPosScore$  then
21:           $prevc \leftarrow c$ ;
22:           $lastPosScore \leftarrow s.value$ 
23:           $c \leftarrow c + 1$ ;
24:        else
25:          if  $s.value = lastPosScore$  then ▷ Tied scores for positives and negatives
26:             $AUC \leftarrow AUC + (c + prevc)/2$ ;
27:          else
28:             $AUC \leftarrow AUC + c$ ;
29:     $\hat{\theta} \leftarrow \frac{AUC}{pn}$ ;

```

An example of using a sliding window and a red-black tree is presented in Fig. 2. Window  $W$  contains six examples, all of which are already inserted into the red-black tree. As mentioned earlier, examples in the tree are sorted (depth-first search wise) descending according to scores  $s$ , positives before negatives, and ascending according to arrival time  $t$ . When a new instance is scored by the classifier ( $t: 7, l: +, s: 0.80$ ), the oldest instance ( $t: 1$ ) is removed from the window and the tree. After the new scored example is inserted, AUC is calculated by traversing the tree in a depth-first search manner and counting labels as presented in lines 17–29 of Algorithm 1. In this example, the resulting AUC would be 0.875.

Although prequential AUC requires a sliding window and is not fully incremental (cannot be computed based solely on its previous value), in [9], we have shown that the presented algorithm requires  $O(1)$  time and memory per example and is, therefore, suitable for data stream processing.

Prequential AUC aims at extending the list of available evaluation measures, particularly for assessing classifiers and detecting drifts in streams with evolving class distributions. However, we must verify if this measure is suitable for visualizing performance changes over time. Furthermore, we are also interested in how averaged prequential AUC relates to AUC calculated periodically on blocks or once over the entire stream. Finally, we must evaluate the newly proposed measure on real-world and synthetic data to verify its processing speed and applicability to large streaming data with different types of drift and imbalance ratios. In



**Fig. 2** Red-black tree of examples from window  $W$ , where  $l$  is the example’s true label,  $s$  its assigned score, and  $t$  its timestamp ( $AUC_{(a)}$ : 0.833,  $AUC_{(b)}$ : 0.875). **a** Before adding a new instance. **b** After adding a new instance

the following sections, we examine the aforementioned characteristics of prequential AUC and present the main contributions of this study.

### 4 Properties of prequential AUC

We start the study of properties of prequential AUC by comparing it against other attempts at computing AUC on data streams. First, we will compare the differences in visualizations of classifier performance on stationary and drifting streams. In the second part of this section, we will examine the consistency of prequential and block AUC with batch calculations on stationary data.

#### 4.1 AUC visualizations over time

As it was mentioned in Sect. 2, there have already been attempts to use AUC as an evaluation measure for data stream classifiers. Some researchers [28,44] calculated AUC on periodical holdout sets, i.e., consecutive blocks of examples. Others [13,36], for experimental purposes, treated small data streams as a single batch of examples and calculated AUC traditionally. Furthermore, there has been a proposal of an algorithm for computing AUC incrementally, instance after instance [5]. With the proposed prequential estimation, this gives in total four ways of evaluating data stream classifiers using AUC: batch, block-based, incremental, and prequential.

Figures 3 and 4 present visualizations of the aforementioned four AUC calculation procedures. More precisely, both plots present the performance of a single Hoeffding Tree classifier [20] on a dataset with 20 k examples created using the RBF generator (the dataset will be discussed in more detail in Sect. 5.2). The first dataset contained no drifts, whereas in the second dataset a sudden drift was added after 10 k examples. Prequential and block AUC were calculated using a window of  $d = 1000$  examples (the impact of the  $d$  parameter will



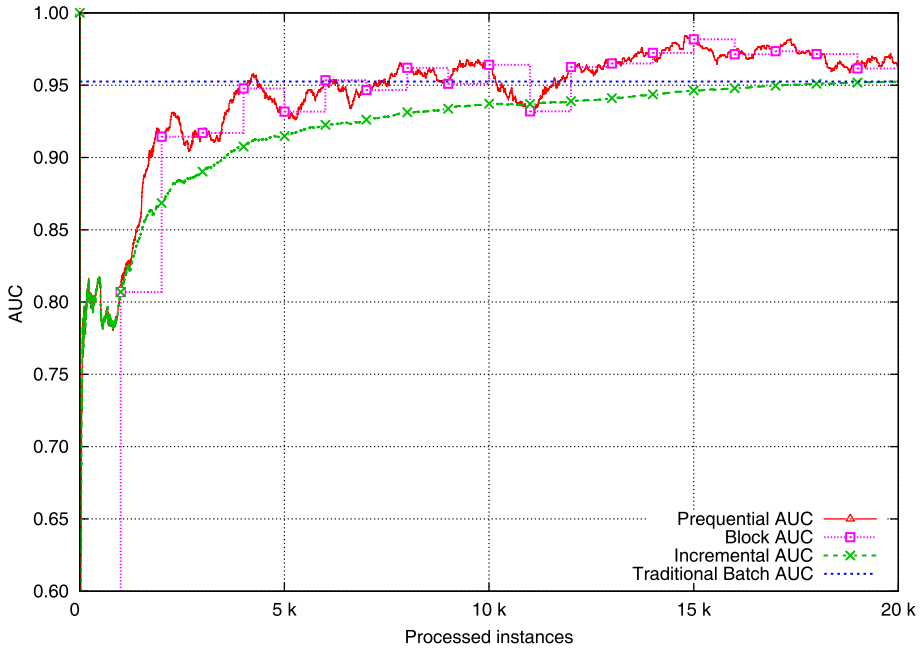


Fig. 3 Batch, incremental, block-based, and prequential AUC on a data stream with no drifts (RBF<sub>20k</sub>)

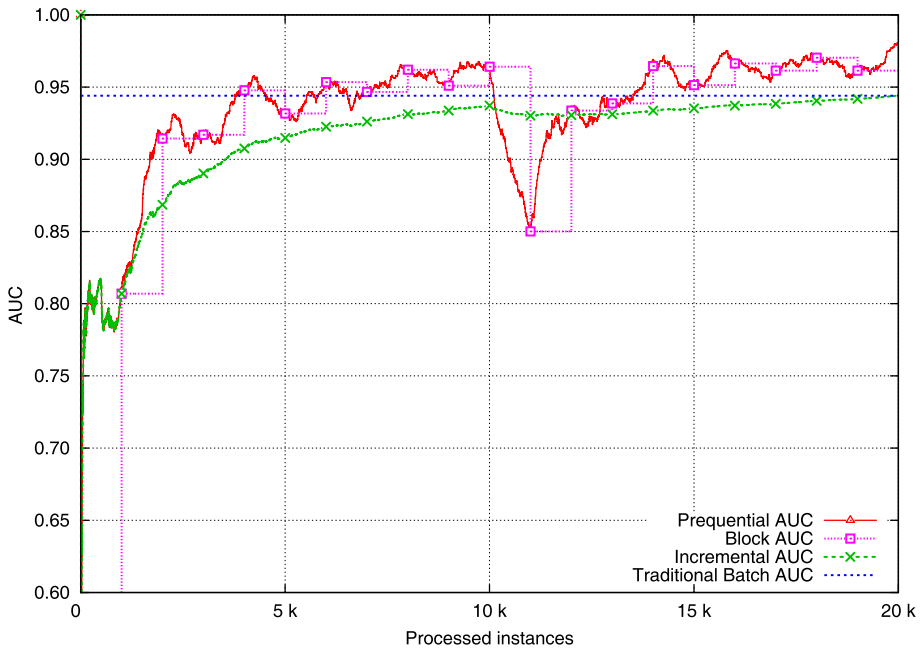


Fig. 4 Batch, incremental, block-based, and prequential AUC on a data stream with a sudden drift after 10k examples (RBF<sub>20kSD</sub>)

be analyzed in Sect. 5.4). It is worth noting that similar comparisons were previously done for incremental and prequential accuracy [21,33].

On the stream without any drifts, presented in Fig. 3, we can see that AUC calculated in blocks or prequentially is less pessimistic than AUC calculated incrementally. This is due to the fact that the true performance of an algorithm at a given point in time is obscured when calculated incrementally—algorithms are punished for early mistakes regardless of the level of performance they are eventually capable of, although this effect diminishes over time. This property has also been noticed by other researchers when visualizing classification accuracy over time [21,33]. Therefore, if one is interested in the performance of a classifier at a given moment in time, prequential and block AUC give less pessimistic estimates, with prequential calculation producing a much smoother curve.

Figure 4 presents the difference between all four AUC calculation methods in the presence of concept drift. As one can see, after a sudden drift occurring after 10 k examples, the change in performance is most visible when looking at prequential AUC over time. AUC calculated on blocks of examples also depicts this change, but delayed according to the block size. However, the most relevant observation is that AUC calculated incrementally is not capable of depicting drifts due to its long “memory” of predictions. For this reason, prequential evaluations should be favored over incremental and block-based assessment in drifting environments where class labels are available after each example [21].

In summary, on streams without drift prequential AUC provides a smoother and less pessimistic classifier evaluation than block-based and incremental calculations. Furthermore, prequential AUC is best at depicting sudden changes over time. It is also worth emphasizing that incremental and batch AUC calculations are presented here only for reference, as they do not fulfill computational requirements of data stream processing. Based on the presented analysis, we believe that similarly as prequential accuracy should be favored over batch, incremental, and block-based accuracy [21], prequential AUC should be preferred to batch, block, and incremental AUC when monitoring classifier performance online on drifting data streams.

## 4.2 Prequential AUC averaged over entire streams

### 4.2.1 Motivation

The above analysis shows that prequential AUC has several advantages when monitored over time, especially in environments with possible concept drifts. However, in many situations, particularly when comparing classifiers over multiple datasets, it is easier to examine simple numeric values rather than entire performance plots. In such cases, researchers are more interested in performance values averaged over entire streams.

As it was shown in Fig. 4, in the presence of concept drift, prequential AUC is the most appropriate calculation procedure for showcasing reactions to changes, even when averaged over the entire stream. However, if no drifts are expected, AUC calculated traditionally in a batch manner over all examples should give the best performance estimate. This observation stems from the fact that in a stationary stream all examples represent a stationary set of concepts, and therefore, all predictions can be simultaneously taken into account during evaluation. Although the computation of batch AUC is not feasible for large data streams, we are interested in how prequential and block calculations averaged over the entire stream compare to AUC calculated once using all predictions.

First, it is worth noticing that if we simultaneously take all examples into account, their order of appearance does not affect the final batch AUC estimation, as long as an exam-

**Table 1** Classifiers with the same batch AUC but different prequential and block AUC (for a window of  $d = 2$ )

$C_1$	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+
$t$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_2$	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+
$t$	1	3	5	7	9	11	13	15	2	4	6	8	10	12	14	16

**Table 2** An example in which one classifier has higher batch AUC but lower prequential and block AUC (for  $d = 2$ )

$C_3$	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-
$t$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_4$	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+
$t$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ple receives a certain score regardless of its position in the stream. This is contrary to AUC calculated prequentially or in blocks where the order of examples affects the final averaged performance values. Let us analyze two examples that demonstrate this issue. In each of the presented examples, the arrival time of each instance will be denoted by  $t$  and scores will be sorted from highest to lowest, left to right. We assume that classifiers are trained to output higher scores for positive examples and lower scores for negative examples.

Table 1 presents a stream where two classifiers ( $C_1, C_2$ ) give the highest score to negative instances (-) and the lowest score to positive instances (+). As a result, batch AUC calculated on the entire dataset at once will be 0.00, regardless of the arrival order of examples. However, if positive and negative instances are clearly separated (e.g., all negative instances appear before all positive, as in  $C_1$ ), averaged AUC calculated in blocks or prequentially with a window of  $d = 2$  instances will give an estimate of 1.00 and 0.93, respectively. Such high estimates are due to the fact that for most window positions all instances have the same class label. On the other hand, if the same examples arrive in a different order ( $C_2$ ), prequential or block AUC with the same window of  $d = 2$  when averaged over the stream will be 0.00 just as batch AUC.

Table 2 presents an additional issue. For  $C_3$  batch AUC is 0.57, averaged block AUC is 1.00, and averaged prequential AUC is 0.53. For Classifier 4, on the other hand, batch AUC is 0.00, block AUC 1.00, and prequential AUC 0.93. This shows that prequential AUC does not have to be necessarily higher than batch AUC. It is worth noting, that if the sequence of interleaved positives and negatives for  $C_4$  started with a negative example, block AUC would be 0.00.

The above examples show that depending on the calculation procedure, one can obtain AUC estimates that are far away from each other. This might be beneficial for concept-drifting streams, but not for stationary distributions. Since we cannot ensure equal prequential, block-based, and batch AUC estimates, we are interested in how different these estimates are on average. More importantly, we are interested in how these differences affect classifier evaluation. To answer these questions, we will use criteria for comparing evaluation measures proposed by Huang and Ling [30].

### 4.2.2 Theoretical background

When discussing two different measures  $f$  and  $g$  used for evaluating two learning algorithms A and B, we want  $f$  and  $g$  to be *consistent* with each other. That is, when  $f$  shows that algorithm A is better than B, then  $g$  will not say B is better than A [30]. Furthermore, if  $f$  is more *discriminating* than  $g$ , we expect to see cases where  $f$  can tell the difference between A and B but  $g$  cannot, but not vice versa. These intuitive descriptions of consistency and discriminancy were made precise by the following definitions [30].

**Definition 1** For two measures  $f, g$  and two classifier outputs  $a, b$  on a domain  $\Psi$ ,  $f$  and  $g$  are strictly *consistent* if there exists no  $a, b \in \Psi$ , such that  $f(a) > f(b)$  and  $g(a) < g(b)$ .

**Definition 2** For two measures  $f, g$  and two classifier outputs  $a, b$  on a domain  $\Psi$ ,  $f$  is strictly more *discriminating* than  $g$  if there exists  $a, b \in \Psi$  such that  $f(a) \neq f(b)$  and  $g(a) = g(b)$ , and there exists no  $a, b \in \Psi$ , such that  $g(a) \neq g(b)$  and  $f(a) = f(b)$ .

As we have already shown in Tables 1 and 2, counter examples on strict consistency and discriminancy do exist for average AUC calculated on an entire stream and using blocks or sliding windows. Therefore, it is impossible to prove strict consistency and discriminancy between batch and prequential or block AUC, based on Definitions 1 and 2. In this case, we have to rather consider the degree of being consistent and the degree of being more discriminating. This leads to the definitions of the *degree of consistency* and *degree of discriminancy* [30].

**Definition 3** For two measures  $f, g$  and two classifier outputs  $a, b$  on a domain  $\Psi$ , let  $R = \{(a, b) | a, b \in \Psi, f(a) > f(b), g(a) > g(b)\}$ ,  $S = \{(a, b) | a, b \in \Psi, f(a) > f(b), g(a) < g(b)\}$ . The *degree of consistency* of  $f$  and  $g$  is  $\mathbf{C}$  ( $0 \leq \mathbf{C} \leq 1$ ), where  $\mathbf{C} = \frac{|R|}{|R|+|S|}$ .

**Definition 4** For two measures  $f, g$  and two classifier outputs  $a, b$  on a domain  $\Psi$ , let  $P = \{(a, b) | a, b \in \Psi, f(a) > f(b), g(a) = g(b)\}$ ,  $Q = \{(a, b) | a, b \in \Psi, g(a) > g(b), f(a) = f(b)\}$ . The *degree of discriminancy* for  $f$  over  $g$  is  $\mathbf{D} = \frac{|P|}{|Q|}$ .

As it was suggested by Huang and Ling, two measures should agree on the majority of classifier evaluations to be comparable. Therefore, we require prequential AUC averaged over the entire stream to be  $\mathbf{C} > 0.5$  consistent with batch AUC. Furthermore, the degree of discriminancy  $\mathbf{D}$  shows how many times it is more likely that a given measure can tell the difference between two algorithms, when the other measure cannot. In the following paragraphs, we will say that measures  $f$  and  $g$  are statistically consistent if  $\mathbf{C} > 0.5$  and that measure  $f$  is more discriminating than a base measure  $g$  if  $\mathbf{D} < 1$  [30].

Since prequential and block AUC are dependent on the ranking of examples, their order in the stream, and the used window size  $d$ , it is difficult to formally prove whether they are statistically consistent ( $\mathbf{C} > 0.5$ ) and more discriminating ( $\mathbf{D} < 1$ ) than AUC calculated using the entire stream at once. Therefore, we will use simulations to verify statistical consistency with batch AUC and whether we can decide which calculation procedure is most discriminating. More importantly, empirical evaluations on artificial datasets will give us an insight into the practical degree of consistency and discriminancy for different class imbalance ratios and window sizes.

### 4.2.3 Simulations

We generate datasets with 4, 6, 8, and 10 examples, which could be ranked by two hypothetical classifiers. For each number of examples, we enumerate *all possible orderings* of examples

of all possible pairs of classifier-ranked lists with 50, 34, 14% minority class examples and calculate prequential (and block) AUC for all possible window sizes ( $1 < d < n$ ). For a dataset with  $n_p$  positive examples and  $n$  examples in total, there are  $\binom{n}{2} \cdot n!$  such non-repeating pairs of ranked lists with different example orderings. Due to such a large number of ordering/ranking possibilities, we were only able to exhaustively test datasets up to 10 instances. The three class imbalance ratios were chosen to show performance on a balanced dataset (50%), 1:2 imbalance ratio (34%), and the extreme case of only one positive instance regardless of the dataset size (14%).

We exhaustively compare all orderings of examples for all possible example rankings to verify the degree of consistency and discriminancy for different window sizes  $d$ . To obtain the degree of consistency, we count the number of pairs for which “ $AUC(a) < AUC(b)$  and  $pAUC(a) < pAUC(b)$ ” and the number of pairs for which “ $AUC(a) < AUC(b)$  and  $pAUC(a) > pAUC(b)$ ,” where  $AUC(\cdot)$  denotes batch-calculated AUC and  $pAUC(\cdot)$  prequential AUC averaged over the entire stream. To obtain the degree of discriminancy, we count the number of pairs which satisfy “ $AUC(a) < AUC(b)$  and  $pAUC(a) = pAUC(b)$ ” and the number of pairs which satisfy “ $AUC(a) = AUC(b)$  and  $pAUC(a) < pAUC(b)$ ”. Similar computations were done for block AUC, denoted as  $bAUC(\cdot)$ . Tables 3, 4, 5, 6, 7 and 8 shows the experiment results, with prequential AUC always shown on the left side and block AUC on the right side. In the table headers AUC, prequential AUC, and block-based AUC are denoted as  $A(\cdot)$ ,  $pA(\cdot)$ ,  $bA(\cdot)$ , respectively.

Regarding consistency (Tables 3, 4, 5), the results show that both block-based and prequential AUC have a high percentage of decisions consistent with batch AUC. More precisely, both estimations usually achieve a degree of consistency between 0.80 and 0.90, which is much larger than the required 0.50. However, it is also worth noticing that, for all class imbalance ratios and most window sizes, prequential AUC is more consistent with batch AUC than its block-calculated competitor. What is even more important is that this difference is most apparent for smaller window sizes, where prequential AUC usually has a 0.05 higher degree of consistency. Finally, it is worth noticing that larger windows persistently allow to achieve estimations that are more consistent with batch AUC.

In terms of the degree of discriminancy (Tables 6, 7, 8), the results vary. For very small window sizes, batch AUC seems to be better at differentiating rankings ( $D > 1$ ), whereas windows of sizes  $d > 3$  invert this relation ( $D < 1$ ). However, once again it is worth noticing that prequential AUC is usually more discriminant (has smaller  $D$ ) than block AUC regardless of the window size or class imbalance ratio. Moreover, higher class imbalance ratios appear to make the differentiation more difficult for prequential and block estimations, especially for smaller window sizes. This is understandable, as with such small datasets, for higher class imbalance ratios only a single positive example is available. If a window is too small compared to the class imbalance ratio, no positive examples are available in a window. This situation is especially visible in the datasets with 14% minority class examples, where for all dataset sizes  $n$  the number of positive examples is  $n_p = 1$ .

A direct conclusion can be drawn from this observation: Window sizes used for prequential (or block) AUC estimations should be large enough to always contain at least one positive example to ensure higher discriminancy. Nevertheless, apart from very small window sizes compared to the class distribution, prequential AUC is comparably discriminant with AUC calculated on the entire stream. Although not presented in this manuscript, the authors also analyzed the measures' indifference (ratio of cases where neither of the measures can differ-

**Table 3** Degree of consistency with batch-calculated AUC for balanced datasets (50% both classes)

$n$	$d$	$A(a) < A(b) \& pA(a) < pA(b)$	$A(a) < A(b) \& pA(a) > pA(b)$	$C$	$n$	$d$	$A(a) < A(b) \& bA(a) < bA(b)$	$A(a) < A(b) \& bA(a) > bA(b)$	$C$
(a) <i>Prequential AUC</i>									
4	2	226	30	0.883	4	2	208	56	0.788
4	3	276	0	1.000	4	3	236	28	0.894
6	2	78,558	14,206	0.847	6	2	67,230	17,428	0.794
6	3	95,086	11,688	0.891	6	3	87,150	14,164	0.860
6	4	106,566	10,098	0.913	6	4	92,930	19,356	0.828
6	5	109,152	3840	0.966	6	5	101,520	7,608	0.930
8	2	53,158,628	11,185,976	0.826	8	2	45,405,586	13,358,810	0.773
8	3	63,240,034	10,425,237	0.858	8	3	53,850,202	14,391,887	0.789
8	4	69,182,304	9,972,770	0.874	8	4	66,567,040	8,624,167	0.885
8	5	70,761,192	7,693,012	0.902	8	5	63,572,560	12,086,730	0.840
8	6	73,479,168	6,411,408	0.920	8	6	64,611,792	14,232,192	0.819
8	7	75,503,520	2,403,360	0.969	8	7	72,136,080	4,448,880	0.942
10	2	60,634,866,784	14,063,999,524	0.812	10	2	51,487,887,104	17,000,441,762	0.752
10	3	71,130,163,463	13,290,485,752	0.843	10	3	59,490,757,935	16,883,741,696	0.779
10	4	74,037,644,404	14,163,943,582	0.839	10	4	64,512,419,513	20,509,708,299	0.759
10	5	78,920,700,364	11,176,341,290	0.876	10	5	76,714,147,735	9,078,493,566	0.894
10	6	73,168,730,742	13,467,350,816	0.845	10	6	71,855,703,700	13,934,726,872	0.838
10	7	79,076,275,296	9,293,058,744	0.895	10	7	70,312,608,720	16,552,837,632	0.809
10	8	83,944,244,880	6,701,320,800	0.926	10	8	74,378,685,600	15,590,530,080	0.827
10	9	86,539,824,000	2,926,627,200	0.967	10	9	83,365,107,840	4,817,836,800	0.945
(b) <i>Block-based AUC</i>									

**Table 4** Degree of consistency with batch-calculated AUC for imbalanced datasets (34% minority class)

$n$	$d$	$A(a) < A(b)$ & $pA(a) < pA(b)$	$A(a) < A(b)$ & $pA(a) > pA(b)$	$C$	$n$	$d$	$A(a) < A(b)$ & $bA(a) < bA(b)$	$A(a) < A(b)$ & $bA(a) > bA(b)$	$C$
(a) Prequential AUC									
4	2	96	8	0.923	4	2	92	8	0.920
4	3	112	8	0.933	4	3	102	18	0.850
6	2	44,328	7430	0.856	6	2	39,222	9582	0.804
6	3	53,318	7312	0.879	6	3	49,482	7590	0.867
6	4	58,774	6904	0.895	6	4	52,302	11,468	0.820
6	5	62,064	3312	0.949	6	5	59,496	5352	0.917
8	2	34,250,730	6,980,006	0.831	8	2	28,945,260	8,463,258	0.774
8	3	40,699,497	6,793,142	0.857	8	3	34,682,961	8,761,157	0.798
8	4	44,193,331	6,982,899	0.864	8	4	43,361,987	6,740,645	0.865
8	5	45,463,956	5,426,998	0.893	8	5	41,705,650	7,623,636	0.845
8	6	47,516,640	4,326,864	0.917	8	6	42,382,608	8,985,336	0.825
8	7	49,281,120	2,127,600	0.959	8	7	47,743,920	3,418,560	0.933
10	2	14,056,095,136	2,991,723,204	0.825	10	2	12,195,033,276	3,413,021,306	0.781
10	3	16,506,930,025	3,058,609,668	0.844	10	3	13,792,398,281	3,675,831,232	0.790
10	4	17,397,851,565	3,458,536,944	0.834	10	4	15,903,298,910	4,113,647,610	0.794
10	5	17,857,902,489	3,077,583,820	0.853	10	5	17,306,025,784	3,102,200,565	0.848
10	6	17,368,267,282	3,208,774,634	0.844	10	6	17,469,134,532	2,947,901,726	0.856
10	7	18,541,658,208	2,198,965,392	0.894	10	7	16,864,120,896	3,466,868,304	0.829
10	8	19,469,178,720	1,653,883,920	0.922	10	8	17,293,451,760	3,656,527,920	0.825
10	9	20,338,174,080	834,825,600	0.961	10	9	19,822,360,320	1,216,010,880	0.942
(b) Block-based AUC									

**Table 5** Degree of consistency with batch-calculated AUC for imbalanced datasets (14% minority class)

<i>n</i>	<i>d</i>	$A(a) < A(b) \ \& \ pA(a) < pA(b)$	$A(a) < A(b) \ \& \ pA(a) > pA(b)$	<b>C</b>	<i>n</i>	<i>d</i>	$A(a) < A(b) \ \& \ bA(a) < bA(b)$	$A(a) < A(b) \ \& \ bA(a) > bA(b)$	<b>C</b>
<i>(a) Prequential AUC</i>					<i>(b) Block-based AUC</i>				
4	2	96	8	0.923	4	2	92	8	0.920
4	3	112	8	0.933	4	3	102	18	0.850
6	2	6888	840	0.891	6	2	6096	784	0.886
6	3	8068	1240	0.867	6	3	8152	644	0.927
6	4	8568	1220	0.875	6	4	8276	934	0.899
6	5	9264	816	0.919	6	5	8880	1200	0.881
8	2	701,568	96,768	0.879	8	2	571,680	71,040	0.889
8	3	841,168	145,632	0.852	8	3	796,032	90,528	0.898
8	4	868,680	164,584	0.841	8	4	914,944	64,888	0.93
8	5	891,960	164,384	0.844	8	5	923,656	77,832	0.922
8	6	936,144	139,056	0.871	8	6	940,368	99,312	0.904
8	7	1,006,560	82,080	0.925	8	7	982,800	105,840	0.903
10	2	99,792,000	14,636,160	0.872	10	2	81,250,560	11,632,320	0.875
10	3	121,113,792	21,683,232	0.848	10	3	105,151,848	16,303,848	0.866
10	4	125,448,672	25,481,808	0.831	10	4	126,349,656	12,907,656	0.907
10	5	127,000,512	27,198,096	0.824	10	5	139,374,336	8,251,152	0.944
10	6	128,755,248	27,134,520	0.826	10	6	142,754,112	12,808,368	0.918
10	7	132,687,744	24,559,776	0.844	10	7	141,324,576	10,824,720	0.929
10	8	139,400,640	19,555,920	0.877	10	8	142,572,240	12,827,520	0.917
10	9	148,861,440	10,805,760	0.932	10	9	146,603,520	13,063,680	0.918

entiate two distinct rankings [30]), which remained very small throughout all window sizes and class imbalance ratios.<sup>1</sup>

Apart from the number of ranking pairs where prequential or block estimations are consistent or more discriminating than batch AUC, one may be interested in the absolute difference between the values of these estimations. To verify these differences, we have calculated and plotted the values of  $pAUC(a) - AUC(a)$  and  $bAUC(a) - AUC(a)$  for the three analyzed class imbalance ratios. Due to space limitations, Figs. 5 and 6 present these differences only for the balanced datasets.<sup>2</sup>

The left-hand side of each figure presents a three-dimensional plot, where the *x*-axis denotes the difference between prequential (or block) AUC and batch AUC, the *y*-axis describes window sizes, and the *z*-axis shows the number of rankings for which a given difference was observed. The right-hand side of each figure shows a two-dimensional top view of the same plot. The left plots are intended to demonstrate the dominating difference values and their variation for each window size. It is also worth noticing that these plots usually demonstrate peaks around the 0.0 difference. The right plots, on the other hand, clearly show the range of possible differences for each window size.

<sup>1</sup> Results for the degree of indifference are in Appendix A in the supplementary materials.

<sup>2</sup> Plots for imbalanced datasets are in Appendix B in the supplementary materials.



**Table 6** Degree of discriminancy compared to batch-calculated AUC for balanced datasets (50% both classes)

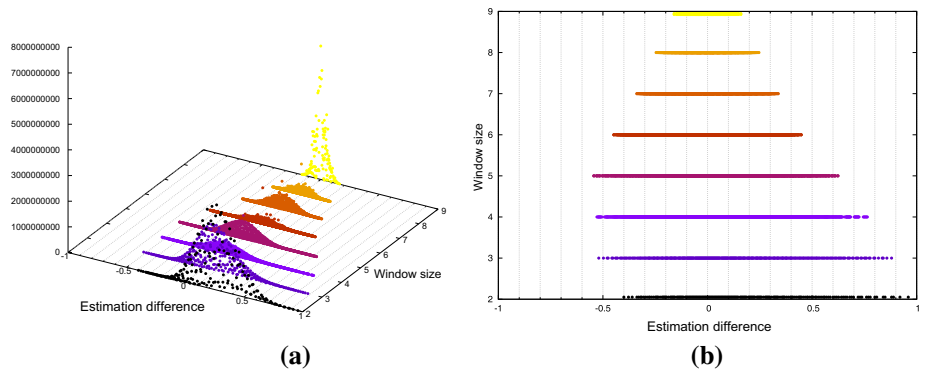
$n$	$d$	$A(a) < A(b)$ & $pA(a) = pA(b)$	$A(a) = A(b)$ & $pA(a) < pA(b)$	$D$	$n$	$d$	$A(a) < A(b)$ & $bA(a) = bA(b)$	$A(a) = A(b)$ & $bA(a) < bA(b)$	$D$
(a) Prequential AUC									
4	2	80	8	10.00	4	2	72	0	$\infty$
4	3	60	8	7.50	4	3	72	4	18.00
6	2	26,756	3500	7.64	6	2	34,862	2010	17.34
6	3	12,746	4044	3.15	6	3	18,206	2802	6.50
6	4	2856	4724	0.60	6	4	7234	4122	1.75
6	5	6528	4032	1.62	6	5	10,392	2712	3.83
8	2	16,214,756	2,353,182	6.89	8	2	21,794,964	1,827,868	11.92
8	3	6,894,089	2,738,339	2.52	8	3	12,317,271	2,152,120	5.72
8	4	1,304,298	2,996,764	0.44	8	4	5,268,165	2,258,632	2.33
8	5	2,104,792	2,870,944	0.73	8	5	4,899,706	2,262,634	2.17
8	6	356,712	2,891,136	0.12	8	6	1,403,304	2,475,192	0.57
8	7	2,652,480	2,349,360	1.13	8	7	3,974,400	1,751,040	2.27
10	2	16,474,070,244	2,325,714,636	7.08	10	2	22,684,607,686	1,764,461,488	12.86
10	3	6,751,652,794	2,692,735,085	2.51	10	3	14,797,802,378	2,108,350,183	7.02
10	4	1,135,333,025	2,782,776,394	0.41	10	4	4,314,793,199	2,648,700,613	1.63
10	5	1,523,949,099	2,693,165,272	0.57	10	5	5,828,349,452	2,133,672,486	2.73
10	6	205,901,522	2,464,416,666	0.08	10	6	1,051,552,508	2,577,780,700	0.41
10	7	837,730,392	2,688,388,944	0.31	10	7	2,341,618,080	2,381,208,000	0.98
10	8	163,838,880	2,513,982,960	0.07	10	8	840,188,880	2,230,413,840	0.38
10	9	1,707,148,800	2,113,695,360	0.81	10	9	2,990,655,360	1,607,114,880	1.86
(b) Block-based AUC									

**Table 7** Degree of discriminatory compared to batch-calculated AUC for imbalanced datasets (34% minority class)

$n$	$d$	$A(a) < A(b)$ & $pA(a) = pA(b)$	$A(a) = A(b)$ & $pA(a) < pA(b)$	$D$	$n$	$d$	$A(a) < A(b)$ & $bA(a) = bA(b)$	$A(a) = A(b)$ & $bA(a) < bA(b)$	$D$
<i>(a) Prequential AUC</i>									
4	2	40	0	$\infty$	4	2	44	0	$\infty$
4	3	24	0	$\infty$	4	3	24	0	$\infty$
6	2	15,922	1676	9.50	6	2	18,876	1050	17.98
6	3	7050	2080	3.39	6	3	10,608	1042	10.18
6	4	1988	2408	0.83	6	4	3896	2156	1.81
6	5	2304	2040	1.13	6	5	2832	1416	2.00
8	2	10,822,384	1,445,690	7.49	8	2	14,644,602	1,061,404	13.80
8	3	4,560,481	1,698,162	2.69	8	3	8,609,002	1,206,625	7.13
8	4	904,810	1,861,822	0.49	8	4	1,978,408	1,835,340	1.08
8	5	1,156,828	1,864,496	0.62	8	5	2,718,496	1,620,150	1.68
8	6	191,496	1,831,680	0.10	8	6	667,056	1,522,224	0.44
8	7	646,560	1,517,760	0.43	8	7	892,800	1,100,880	0.81
10	2	4,231,464,860	512,002,316	8.26	10	2	5,671,228,618	384,816,444	14.74
10	3	1,713,743,507	605,486,578	2.83	10	3	3,811,053,687	451,977,687	8.43
10	4	295,397,178	680,067,479	0.43	10	4	1,134,839,167	602,144,260	1.88
10	5	297,697,500	660,113,562	0.45	10	5	824,957,460	625,114,249	1.32
10	6	43,162,624	665,026,736	0.06	10	6	203,168,282	669,160,838	0.30
10	7	104,005,512	765,190,632	0.14	10	7	513,639,912	557,695,848	0.92
10	8	79,602,480	614,255,040	0.13	10	8	252,685,440	474,176,160	0.53
10	9	114,589,440	486,864,000	0.24	10	9	249,217,920	341,107,200	0.73
<i>(b) Block-based AUC</i>									

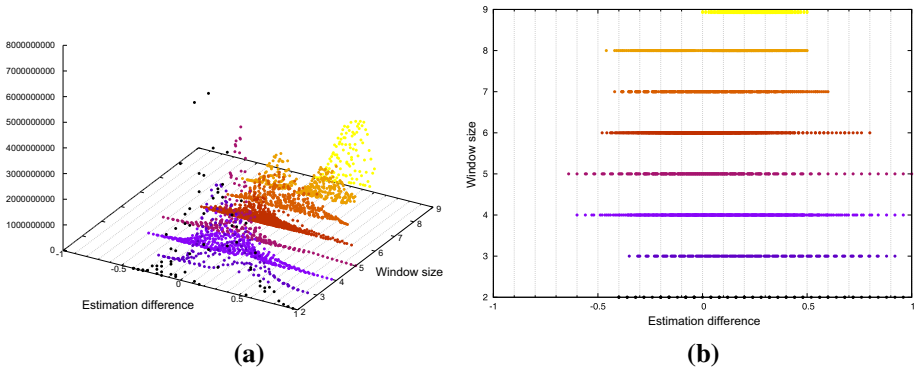
**Table 8** Degree of discriminatory compared to batch-calculated AUC for imbalanced datasets (14% minority class)

$n$	$d$	$A(a) < A(b) \ \& \ pA(a) = pA(b)$	$A(a) = A(b) \ \& \ pA(a) < pA(b)$	$\mathbf{D}$	$n$	$d$	$A(a) < A(b) \ \& \ bA(a) = bA(b)$	$A(a) = A(b) \ \& \ bA(a) < bA(b)$	$\mathbf{D}$
<i>(a) Prequential AUC</i>					<i>(b) Block-based AUC</i>				
4	2	40	0	$\infty$	4	2	44	0	$\infty$
4	3	24	0	$\infty$	4	3	24	0	$\infty$
6	2	3072	0	$\infty$	6	2	3920	0	$\infty$
6	3	1492	0	$\infty$	6	3	2004	0	$\infty$
6	4	1012	0	$\infty$	6	4	1590	0	$\infty$
6	5	720	0	$\infty$	6	5	720	0	$\infty$
8	2	330,624	0	$\infty$	8	2	486,240	0	$\infty$
8	3	142,160	0	$\infty$	8	3	242,400	0	$\infty$
8	4	95,696	0	$\infty$	8	4	149,128	0	$\infty$
8	5	72,616	0	$\infty$	8	5	127,472	0	$\infty$
8	6	53,760	0	$\infty$	8	6	89,280	0	$\infty$
8	7	40,320	0	$\infty$	8	7	40,320	0	$\infty$
10	2	48,867,840	0	$\infty$	10	2	70,413,120	0	$\infty$
10	3	20,498,976	0	$\infty$	10	3	41,840,304	0	$\infty$
10	4	12,365,520	0	$\infty$	10	4	24,038,688	0	$\infty$
10	5	9,097,392	0	$\infty$	10	5	15,670,512	0	$\infty$
10	6	7,406,232	0	$\infty$	10	6	7,733,520	0	$\infty$
10	7	6,048,480	0	$\infty$	10	7	11,146,704	0	$\infty$
10	8	4,339,440	0	$\infty$	10	8	7,896,240	0	$\infty$
10	9	3,628,800	0	$\infty$	10	9	3,628,800	0	$\infty$



**Fig. 5** Differences between prequential and batch AUC for different window sizes on the largest balanced dataset (50% examples of both classes). **a** Three-dimensional plot. **b** Two-dimensional top view

As Fig. 5 shows, most prequential estimates of AUC are very close to AUC calculated on the entire dataset. As it was observed for all three class imbalance ratios, one can notice single points above the bell curve directly above the zero difference value. This showcases that the most common difference between batch and prequential AUC is zero. This is not



**Fig. 6** Differences between block and batch AUC for different window sizes on the largest balanced dataset (50% examples of both classes). **a** Three-dimensional plot. **b** Two-dimensional top view

so obvious for block-based estimates, presented in Fig. 6. When compared with prequential AUC, block estimates have much “wider” bell curves without such strong peaks around zero.

Looking at the two-dimensional plots, it is worth noticing that for small window sizes (small compared to the class imbalance ratio) prequential AUC gives a more optimistic estimate compared to batch AUC. This issue is related to that of lower discriminancy for small windows or high class imbalance ratios (plots for the 14% minority class dataset have much fewer distinct points<sup>2</sup>). When there are no positive examples in the window, AUC for that window is equal to 1, which can lead to overestimating AUC over time. However, as the plots show, larger windows clearly mitigate this problem. The two-dimensional plots also show that, compared to prequential computation, block-based calculations are more prone to over- and under-estimation of AUC.

#### 4.2.4 Summary of results

The above analyses have shown that prequential AUC averaged over time is highly consistent and comparably discriminant to AUC calculated on the entire stream. We have also seen that the absolute difference between these two measures is small for most example orderings. Furthermore, we have noticed that the window size used for prequential calculation should be large enough to contain a sufficient number of positive examples at all time, to avoid overly optimistic AUC estimates and, as an effect, low discriminancy. Finally, prequential AUC proved to be more consistent, more discriminant, and closer in terms of absolute values to batch AUC than block-calculated AUC.

## 5 Processing speed, parameter sensitivity, and classifier evaluation

In this section, we experimentally evaluate prequential AUC on real and synthetic data streams. The proposed measure is first analyzed in terms of its processing time and compared with the Kappa statistic ( $\kappa$ ) [2]. Next, we check the sensitivity of the proposed measure with respect to the window size  $d$ . Finally, we conduct experiments that compare how prequential AUC, accuracy, recall, G-mean,  $\kappa$ , and  $\kappa_m$  [4] evaluate classifiers on imbalanced data streams with various types of drift.

## 5.1 Experimental setup

For experiments comparing prequential AUC with other evaluation measures, we used six commonly analyzed data stream classifiers [11, 14, 20]:

- Naive Bayes (NB),
- Hoeffding Tree (HT),
- Hoeffding Tree with ADWIN ( $HT_{ADWIN}$ ),
- Online Bagging (Bag),
- Accuracy Updated Ensemble (AUE),
- Selectively Recursive Approach (SERA).

Naive Bayes was chosen as a probabilistic classifier without any forgetting mechanism. The Hoeffding Tree was selected as a single stream classifier without (HT) and with a drift detector ( $HT_{ADWIN}$ ). The last three algorithms are ensemble classifiers representing: an online approach (Bag), a block-based approach with forgetting (AUE), and a dynamic block-based oversampling method designed for imbalanced streams (SERA).

All the algorithms and evaluation methods were implemented in Java as part of the MOA framework [3]. The experiments were conducted on a machine equipped with a dual-core Intel i7-2640M CPU, 2.8 GHz processor and 16 GB of RAM. For all the ensemble methods (Bag, AUE, SERA) we used 10 Hoeffding Trees as base learners, each with a grace period  $n_{\min} = 100$ , split confidence  $\delta = 0.01$ , and tie-threshold  $\psi = 0.05$  [20]. AUE and SERA were set to create new components every  $d = 1000$  examples.

## 5.2 Datasets

In experiments comparing prequential AUC with accuracy, recall, G-mean, and the Kappa statistic, we used 4 real and 13 synthetic datasets.<sup>3</sup> For the real-world datasets we chose four data streams which are commonly used as benchmarks [8, 25, 34, 43]. More precisely, we chose *Airlines* (Air) and *Electricity* (Elec) as examples of fairly balanced datasets, and *KDDCup* and *PAKDD* as examples of moderately imbalanced datasets. It is worth noting that we used the smaller version of the *KDDCup* dataset and transformed it into a binary classification problem, by combining every class other than “NORMAL” into one “ATTACK” class.

Synthetic datasets were created using custom stream generators prepared for this study. The *Ratio* datasets are designed to test classifier performance under different imbalance ratios without drift. Examples from the minority class create a five-dimensional sphere, whereas majority class examples are uniformly distributed outside that sphere. The *Dis* datasets are created in a similar manner, but the minority class is fragmented into spherical subclusters (playing the role of small disjuncts [32]). Datasets *Dis*<sub>2</sub>, *Dis*<sub>3</sub>, *Dis*<sub>5</sub> have 2, 3, and 5 clusters, respectively. In the *AppDis* datasets, every stream begins with a single well-defined cluster, and additional clusters are added as the stream progresses. New disjuncts appear suddenly in negative class space after 40 k (*AppDis*<sub>2,3,5</sub>), 50 k (*AppDis*<sub>3,5</sub>), 60 and 70 k (*AppDis*<sub>5</sub>) examples. In static data mining, the problem of small disjuncts is known to be more problematic than class imbalance per se [32], however, to the best of our knowledge this issue has not been analyzed in stream classification. Datasets *MinMaj*, *GradualRC*, *SuddenRC* contain class ratio changes over time. In *MinMaj* the negative class abruptly becomes the minority; such a virtual drift relates to a problem recently discussed in [46].

<sup>3</sup> Source code, datasets, stream generators, and test scripts are available at: <http://www.cs.put.poznan.pl/dbrzezinski/software.php>.

**Table 9** Characteristic of datasets

Dataset	No. of inst (k)	#Attrs	Class ratio	Noise (%)	No. of drifts	Drift type
Ratio <sub>5050</sub>	100	5	1:1	0	0	None
Ratio <sub>1090</sub>	100	5	1:9	0	0	None
Ratio <sub>0595</sub>	100	5	1:19	0	0	None
Ratio <sub>0199</sub>	100	5	1:99	0	0	None
Dis <sub>2</sub>	100	5	1:9	0	0	None
Dis <sub>3</sub>	100	5	1:9	0	0	None
Dis <sub>5</sub>	100	5	1:9	0	0	None
AppDis <sub>2</sub>	100	5	1:9	0	1	Sudden
AppDis <sub>3</sub>	100	5	1:9	0	2	Sudden
AppDis <sub>5</sub>	100	5	1:9	0	4	Sudden
MinMaj	100	5	1:19/19:1	0	1	Sud. virt.
Gradual <sub>RC</sub>	100	3	1:1 → 1:100	5	1	Grad. virt.
Sudden <sub>RC</sub>	100	3	1:1/1:100/1:10/1:1	10	3	Rec. virt.
Air	539	7	~24:30	–	–	Unknown
Elec	45	8	~19:26	–	–	Unknown
KDDCup	494	41	~1:4	–	–	Unknown
PAKDD	50	30	~1:4	–	–	Unknown
RBF <sub>20k</sub>	20	20	1:1	0	0	None
RBF <sub>20kSD</sub>	20	20	1:1	0	1	Sudden

Sudden<sub>RC</sub> was created using a modified version of the SEA generator [42] and contained three sudden class ratio changes (1:1/1:100/1:10/1:1) appearing every 25 k examples. Analogously, Gradual<sub>RC</sub> uses a modified Hyperplane generator [45] and simulates a continuous ratio change from 1:1 to 1:100 throughout the entire stream. We note that Ratio, Dis, AppDis, and MinMaj are unevenly imbalanced, i.e., for a class ratio of 1:99 a positive example can occur less (or more) frequently than every 100 examples.

Additionally, two small data streams created using the RBF generator [3] (RBF<sub>20k</sub>, RBF<sub>20kSD</sub>) were used to showcase the evaluation speed of prequential AUC. The characteristics of all the datasets are given in Table 9.

### 5.3 Prequential AUC evaluation time

Some researchers have argued that AUC is too computationally expensive to be used for evaluating data stream classifiers. In particular, the inability to calculate AUC efficiently made authors suggest, for example, that “the Kappa statistic is more appropriate for data streams than a measure such as the area under the ROC curve” [2]. Therefore, to verify whether AUC calculated prequentially overcomes these limitations, we compare its processing time with that of the Kappa statistic. We have already noted in Sect. 3 that the time required per example to calculate prequential AUC is constant; however, we want to also calculate evaluation time on a concrete data stream.

We examine the average evaluation time required per example for two small data streams: one without drift (RBF<sub>20k</sub>) and one with a single sudden drift (RBF<sub>20kSD</sub>). We use the MOA framework to compare the time required to evaluate a single Hoeffding Tree using both

**Table 10** Evaluation time per example (ms) using prequential AUC and  $\kappa$  on a window of  $d = 1000$  examples (averaged over 10 runs  $\pm$  standard deviation)

	Prequential AUC	Prequential $\kappa$
RBF <sub>20k</sub>	7.230 $\pm$ 0.158	7.189 $\pm$ 0.246
RBF <sub>20kSD</sub>	7.344 $\pm$ 0.519	7.287 $\pm$ 0.243

measures.<sup>4</sup> The calculation of the measures does not depend on the number of attributes, and therefore, we evaluate their running time only on two datasets averaged over ten runs. Table 10 presents evaluation time per example using prequential AUC and prequential Kappa ( $\kappa$ ) on a window of  $d = 1000$  examples (an identical window size was used in [2]).

As results in Table 10 show, evaluation using prequential AUC is only slightly slower than using the Kappa statistic on both datasets. The difference is very small and proportionally AUC is 0.57% slower for RBF<sub>20k</sub> and 0.79% for RBF<sub>20kSD</sub>. This difference could be larger for a larger window size; however, due to the properties of red-black trees, it would grow logarithmically. It is also worth noting that in these experiments, the difference between prequential AUC and the Kappa statistic may be obscured by the time required to train and test the classifier. However, this only shows that the evaluation time for both measures is comparably small even when related to the classification time of a single Hoeffding Tree.

As the described experiments show, prequential AUC offers similar evaluation time compared to the Kappa statistic for a window size of  $d = 1000$  examples. Furthermore, this difference can grow only logarithmically for larger window sizes. Thus, prequential AUC should not be deemed too computationally expensive for evaluating data stream classifiers. This stands in contrast to AUC calculated on entire streams and is one of the key features making prequential AUC applicable to real-world data streams.

### 5.4 Parameter sensitivity

In many data stream mining algorithms, the sliding window size is a parameter which strongly influences the performance of the algorithm [20,45]. Therefore, we decided to verify the impact of using different window sizes  $d$  for calculating prequential AUC. Table 11 presents the average prequential AUC of a Hoeffding Tree on different datasets while using  $d \in [100; 5000]$ .

Additionally, in Fig. 7 we present a box plot summarizing the differences in averaged AUC estimates for different window sizes. The plot was created by, first calculating average prequential AUC on each dataset over all window sizes, and later calculating and plotting the differences between the general mean and the value obtained for a certain  $d$ . For example, for the KDDCup dataset the mean value of average prequential AUC for all window sizes  $d \in [100; 5000]$  is 0.994. Therefore, the deviation for  $d = 100$ , where prequential AUC for KDDCup equals 0.997, is:  $\frac{0.997-0.994}{0.994} \times 100\% \approx 0.3\%$ .

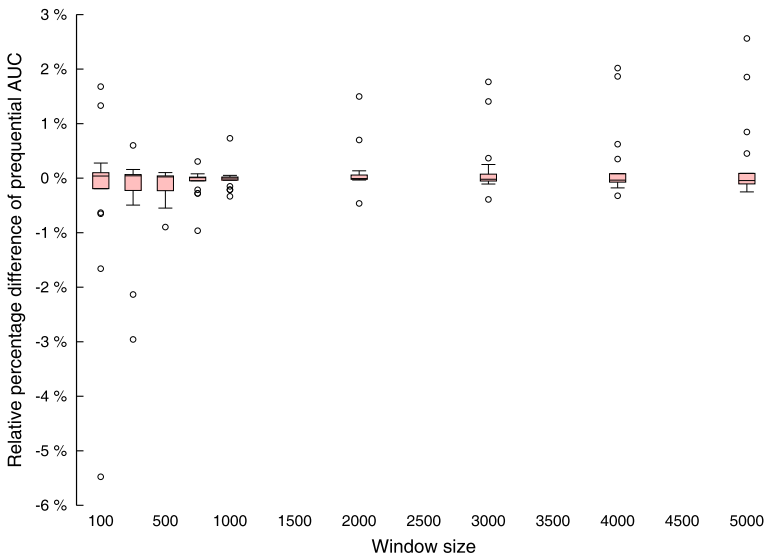
Analyzing values in Table 11, one can see that differences in each row are very small. The box plot in Fig. 7 confirms this observation and shows that all but one value are within 3% from the mean value on each dataset. The biggest deviations were obtained for window sizes  $d \in \{100, 200\}$  on Ratio<sub>0199</sub> and PAKDD datasets. PAKDD is the hardest of the analyzed datasets and requires larger windows for better AUC estimates. Ratio<sub>0199</sub>, on the other

<sup>4</sup> Originally, MOA calculates several evaluation metrics per example, and therefore, we created two separate evaluation functions which calculate only prequential AUC and prequential Kappa, respectively.

**Table 11** Average prequential AUC of a Hoeffding Tree using different window sizes  $d$

	100	250	500	750	1000	2000	3000	4000	5000	Mean
Ratio <sub>05050</sub>	0.982	0.983	0.983	0.984	0.984	0.984	0.984	0.984	0.984	0.984
Ratio <sub>1090</sub>	0.987	0.987	0.987	0.987	0.986	0.986	0.986	0.986	0.986	0.986
Ratio <sub>0595</sub>	0.988	0.987	0.987	0.987	0.987	0.986	0.986	0.986	0.986	0.987
Ratio <sub>0199</sub>	0.949	0.929	0.928	0.931	0.932	0.934	0.934	0.933	0.932	0.934
Dis <sub>2</sub>	0.965	0.965	0.965	0.965	0.965	0.964	0.964	0.964	0.963	0.964
Dis <sub>3</sub>	0.966	0.966	0.966	0.966	0.965	0.965	0.964	0.964	0.963	0.965
Dis <sub>5</sub>	0.965	0.964	0.964	0.964	0.964	0.964	0.964	0.963	0.963	0.964
AppDis <sub>2</sub>	0.985	0.985	0.984	0.984	0.984	0.984	0.984	0.984	0.984	0.984
AppDis <sub>3</sub>	0.982	0.982	0.982	0.982	0.982	0.981	0.981	0.981	0.981	0.982
AppDis <sub>5</sub>	0.980	0.981	0.981	0.981	0.981	0.981	0.981	0.981	0.981	0.981
MinMaj	0.986	0.985	0.985	0.984	0.985	0.984	0.984	0.984	0.984	0.985
GradualRC	0.537	0.530	0.541	0.541	0.545	0.550	0.554	0.557	0.560	0.546
SuddenRC	0.748	0.751	0.751	0.753	0.753	0.754	0.755	0.756	0.756	0.753
Air	0.651	0.652	0.652	0.653	0.653	0.655	0.657	0.659	0.660	0.655
Elec	0.871	0.864	0.860	0.857	0.856	0.855	0.856	0.856	0.857	0.859
KDDCup	0.997	0.996	0.994	0.994	0.994	0.994	0.993	0.992	0.992	0.994
PAKDD	0.544	0.563	0.573	0.577	0.579	0.584	0.585	0.586	0.586	0.575

The average was calculated based on samples gathered every 100 examples



**Fig. 7** Box plot of average prequential AUC for window sizes  $d \in [100; 5000]$ . The depicted values are the percentage deviation from the mean on each dataset

hand, is highly skewed and unevenly imbalanced, therefore, in parts of the stream minority examples appear less often than 100 or 200 examples (the largest positive class “gap” in this dataset is 700 examples). It is worth noting that this observation is in accordance with the



**Table 12** Average prequential performance values for Naive Bayes (NB) and Online Bagging (Bag)

	NB						Bag					
	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.
Ratio <sub>05050</sub>	0.95	0.98	0.95	0.91	0.91	0.99	0.98	0.99	0.98	0.96	0.96	1.00
Ratio <sub>1090</sub>	0.95	0.98	0.84	0.73	0.54	0.72	0.97	0.99	0.95	0.84	0.70	0.94
Ratio <sub>0595</sub>	0.96	0.98	0.68	0.54	0.28	0.46	0.98	0.99	0.81	0.71	0.51	0.68
Ratio <sub>0199</sub>	0.99	0.98	0.12	0.07	0.01	0.04	0.99	0.98	0.00	0.00	-0.02	0.00
Dis <sub>2</sub>	0.92	0.95	0.56	0.41	0.22	0.32	0.94	0.97	0.72	0.59	0.39	0.56
Dis <sub>3</sub>	0.92	0.96	0.53	0.38	0.21	0.28	0.94	0.98	0.75	0.63	0.45	0.59
Dis <sub>5</sub>	0.92	0.96	0.51	0.37	0.21	0.26	0.95	0.98	0.75	0.64	0.45	0.59
AppDis <sub>2</sub>	0.94	0.94	0.76	0.63	0.43	0.60	0.97	0.99	0.91	0.82	0.68	0.84
AppDis <sub>3</sub>	0.94	0.94	0.73	0.60	0.41	0.58	0.97	0.99	0.90	0.81	0.68	0.82
AppDis <sub>5</sub>	0.94	0.95	0.74	0.62	0.45	0.60	0.97	0.99	0.90	0.83	0.71	0.83
MinMaj	0.95	0.98	0.80	0.60	0.05	0.69	0.98	0.99	0.85	0.80	0.69	0.77
Gradual <sub>RC</sub>	0.93	0.67	0.25	0.16	0.47	0.12	0.93	0.66	0.33	0.20	0.48	0.14
Sudden <sub>RC</sub>	0.81	0.71	0.48	0.33	0.40	0.50	0.88	0.77	0.64	0.51	0.64	0.53
Air	0.65	0.66	0.54	0.20	0.02	0.37	0.65	0.66	0.56	0.22	0.03	0.39
Elec	0.73	0.78	0.61	0.40	0.37	0.91	0.82	0.91	0.80	0.63	0.58	0.86
KDDCup	0.98	0.98	0.74	0.28	0.40	0.94	1.00	1.00	0.87	0.51	0.95	0.99
PAKDD	0.54	0.64	0.51	0.11	-1.32	0.64	0.80	0.62	0.00	0.00	0.00	0.00

analysis performed in Sect. 4, where we noted that the window size should be large enough to incorporate at least one positive example at all times. In terms of other dependencies upon  $d$ , as discussed in previous sections the memory required to calculate AUC grows linearly to  $d$ , whereas the processing time for single example grows logarithmically with respect to  $d$ .

Due to the small dependency upon  $d$  in terms of AUC estimation, in the remaining experiments we decided to use  $d = 1000$  as a value with low variance, relatively low time and memory consumption, and one that was often proposed in other studies [2, 21, 45]. However, as the presented results show, using any window size from the analyzed range would yield very similar results.

### 5.5 Comparison of imbalanced stream evaluation measures

We experimentally compared classifier evaluations performed using prequentially calculated accuracy (Acc.), AUC, G-mean ( $G_\mu$ ), Cohen’s Kappa ( $\kappa$ ), Kappa M ( $\kappa_m$ ), and positive class recall (Rec.). For this purpose we tested six classifiers: NB, Bag, HT, HT<sub>ADWIN</sub>, AUE, and SERA. The results were obtained using a sliding window of  $d = 1000$  examples [2, 21] and by sampling performance values every 100 examples. Tables 12, 13 and 14 present average results for all six measures.

Comparing performance values on all datasets, one can notice that all classifiers showcase high accuracy. This is caused mainly by the fact that practically all datasets are highly imbalanced and even a simple majority stub would yield high accuracies. The remaining five measures are better at differentiating classifier performance and showcasing difficulties in the datasets.

**Table 13** Average prequential performance values for a Hoeffding Tree without (HT) and with a drift detector (HT<sub>ADWIN</sub>)

	HT						HT <sub>ADWIN</sub>					
	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.
Ratio <sub>5050</sub>	0.97	0.98	0.97	0.95	0.95	1.00	0.98	0.99	0.98	0.96	0.95	1.00
Ratio <sub>1090</sub>	0.97	0.99	0.94	0.83	0.68	0.92	0.97	0.99	0.95	0.83	0.68	0.93
Ratio <sub>0595</sub>	0.97	0.99	0.83	0.71	0.49	0.71	0.97	0.98	0.84	0.70	0.46	0.73
Ratio <sub>0199</sub>	0.99	0.93	0.04	0.02	-0.02	0.01	0.99	0.93	0.04	0.02	-0.01	0.01
Dis <sub>2</sub>	0.93	0.96	0.73	0.58	0.34	0.57	0.93	0.96	0.74	0.58	0.33	0.59
Dis <sub>3</sub>	0.94	0.97	0.74	0.60	0.39	0.57	0.94	0.96	0.77	0.63	0.41	0.63
Dis <sub>5</sub>	0.94	0.96	0.74	0.61	0.39	0.57	0.94	0.97	0.77	0.63	0.40	0.63
AppDis <sub>2</sub>	0.97	0.98	0.91	0.81	0.65	0.84	0.97	0.98	0.91	0.81	0.65	0.86
AppDis <sub>3</sub>	0.96	0.98	0.89	0.80	0.65	0.82	0.97	0.98	0.92	0.82	0.67	0.86
AppDis <sub>5</sub>	0.97	0.98	0.90	0.82	0.69	0.83	0.96	0.98	0.88	0.79	0.65	0.79
MinMaj	0.98	0.98	0.87	0.80	0.68	0.79	0.98	0.98	0.87	0.80	0.66	0.82
Gradual <sub>RC</sub>	0.93	0.63	0.36	0.22	0.48	0.16	0.93	0.63	0.37	0.22	0.48	0.17
Sudden <sub>RC</sub>	0.88	0.75	0.64	0.51	0.63	0.53	0.88	0.75	0.65	0.52	0.64	0.54
Air	0.65	0.65	0.56	0.22	0.04	0.39	0.64	0.61	0.54	0.18	0.00	0.44
Elec	0.79	0.86	0.77	0.56	0.51	0.83	0.83	0.87	0.81	0.65	0.61	0.89
KDDCup	1.00	0.99	0.91	0.50	0.92	0.99	1.00	0.99	0.80	0.46	0.89	0.99
PAKDD	0.80	0.58	0.05	0.00	-0.01	0.01	0.80	0.59	0.04	0.00	-0.01	0.01

**Table 14** Average prequential performance values for AUE and SERA

	AUE						SERA					
	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.	Acc.	AUC	$G_\mu$	$\kappa$	$\kappa_m$	Rec.
Ratio <sub>5050</sub>	0.97	0.99	0.97	0.95	0.94	0.98	0.98	0.99	0.97	0.95	0.95	0.98
Ratio <sub>1090</sub>	0.97	0.99	0.92	0.85	0.75	0.90	0.98	0.99	0.95	0.89	0.80	0.93
Ratio <sub>0595</sub>	0.98	0.98	0.78	0.69	0.52	0.66	0.98	0.99	0.91	0.82	0.68	0.85
Ratio <sub>0199</sub>	0.99	0.97	0.00	0.00	-0.02	0.00	0.99	0.98	0.41	0.36	0.23	0.28
Dis <sub>2</sub>	0.96	0.98	0.80	0.71	0.59	0.70	0.98	0.99	0.92	0.85	0.75	0.89
Dis <sub>3</sub>	0.96	0.98	0.80	0.72	0.59	0.70	0.97	0.98	0.91	0.84	0.75	0.87
Dis <sub>5</sub>	0.95	0.97	0.74	0.65	0.51	0.61	0.97	0.98	0.90	0.83	0.72	0.85
AppDis <sub>2</sub>	0.97	0.98	0.89	0.82	0.72	0.83	0.97	0.98	0.90	0.83	0.73	0.84
AppDis <sub>3</sub>	0.97	0.98	0.87	0.81	0.70	0.81	0.97	0.98	0.88	0.81	0.70	0.80
AppDis <sub>5</sub>	0.97	0.98	0.87	0.81	0.71	0.80	0.97	0.98	0.85	0.79	0.68	0.75
MinMaj	0.98	0.98	0.81	0.76	0.65	0.74	0.98	0.99	0.93	0.81	0.62	0.89
Gradual <sub>RC</sub>	0.92	0.64	0.17	0.09	0.43	0.07	0.93	0.65	0.46	0.33	0.52	0.24
Sudden <sub>RC</sub>	0.87	0.76	0.64	0.51	0.61	0.54	0.87	0.76	0.65	0.52	0.62	0.55
Air	0.67	0.67	0.55	0.23	0.08	0.44	0.61	0.61	0.55	0.16	-0.07	0.42
Elec	0.76	0.82	0.71	0.49	0.42	0.81	0.75	0.82	0.69	0.46	0.40	0.81
KDDCup	0.87	0.96	0.38	0.11	-15.83	0.43	0.99	0.99	0.81	0.57	-0.06	0.97
PAKDD	0.80	0.62	0.01	0.00	0.00	0.00	0.80	0.64	0.14	0.03	0.00	0.03

**Table 15** Average algorithm ranks in the Friedman test

	NB	Bag	HT	HT <sub>ADWIN</sub>	AUE	SERA
Accuracy	5.47	<b>2.47</b>	3.59	3.88	3.06	2.53
AUC	4.88	<b>1.59</b>	3.76	4.29	3.82	2.65
G-mean	5.35	3.21	3.12	2.53	4.38	<b>2.41</b>
Kappa	5.24	3.06	3.76	3.24	3.71	<b>2.00</b>
Kappa M	5.41	<b>2.41</b>	3.88	4.00	2.76	2.53
Recall	4.94	3.38	3.35	<b>2.41</b>	4.32	2.59

Best rank marked in bold

Another observation worth noting is that G-mean,  $\kappa$ ,  $\kappa_m$ , and recall show similar values for consecutive datasets. For example, with growing class imbalance between the `Ratio` datasets (from 1:1 to 1:99), values of all four aforementioned measures systematically drop. AUC, on the other hand, performs differently. This is the result of AUC being a measure that assesses the ranking capabilities of a classifier. On datasets where the classifiers rank most examples correctly (`Ratio`, `Dis`, `AppDis`, `MinMaj`) AUC reports values similar to accuracy. On the other hand, on datasets where the classifiers are unable to correctly rank examples (`GradualRC`, `SuddenRC`, `PAKDD`) AUC values are lower than accuracy. This phenomenon is known from static data mining, where it was reported that AUC and accuracy are consistent measures, with AUC being more discriminating on some datasets [30].

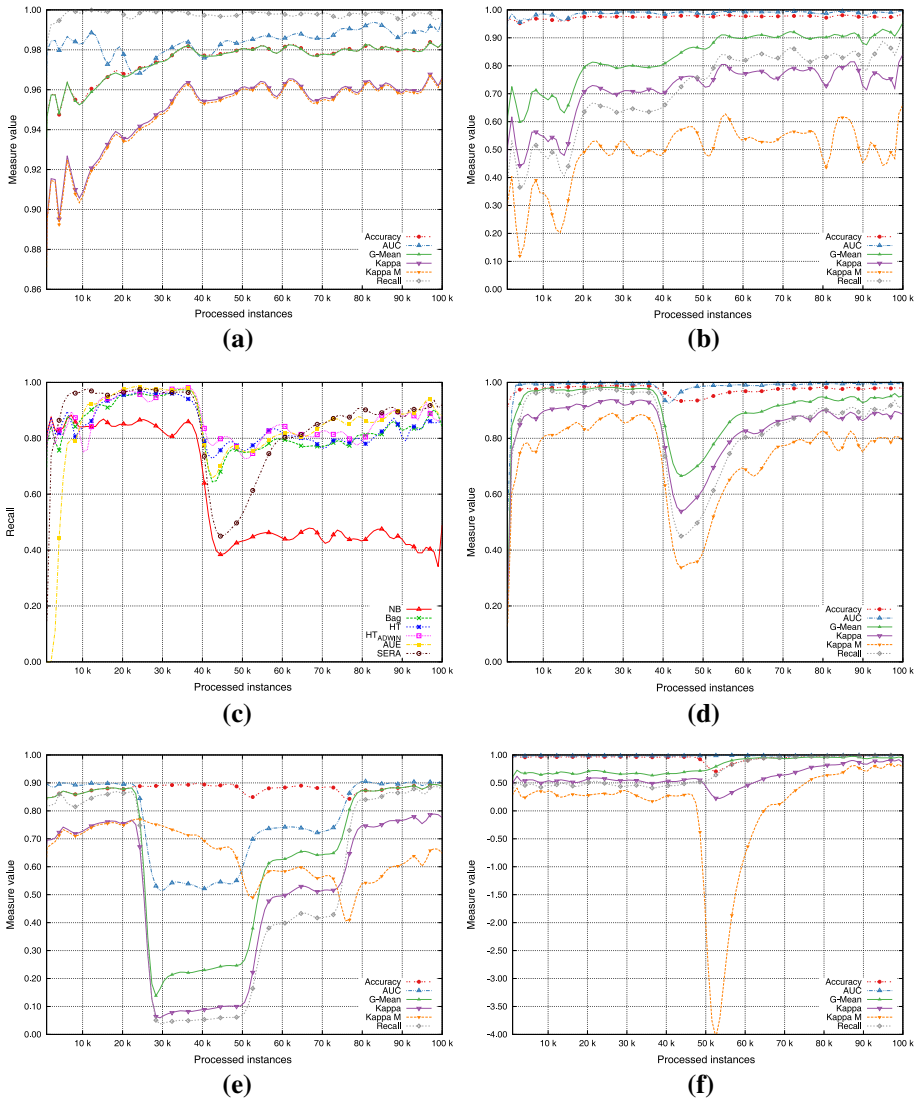
To obtain a broader view of how each of the analyzed measures assessed the classifiers, we performed a Friedman test with Nemenyi post-hoc analysis [12]. The resulting mean ranks are presented in Table 15, whereas critical distance plots are available in Appendix B in the supplementary materials.

The null hypothesis that all the classifiers perform similarly was rejected for each measure with  $p < 0.001$ . Results based on G-mean and  $\kappa$  showcase SERA as the best classifier. Similarly, SERA is second best according to accuracy, AUC,  $\kappa_m$ , and recall, outperformed only three times by Bag and HT<sub>ADWIN</sub>, respectively. Indeed, SERA is the only tested algorithm designed specifically for imbalanced streams, one which oversamples positive examples from previous data chunks. However, Figure S5 in the supplementary materials shows that groups of classifiers which are not significantly different (at  $\alpha = 0.05$ ) are distinct for each measure. We can see that AUE is ranked fairly high according to accuracy and  $\kappa_m$ , but no so according to the remaining measures. We also note that HT<sub>ADWIN</sub>, which is the only algorithm with active drift detection, achieved highest mean rank reported using recall, but was ranked much lower by other measures.

Differences between the analyzed measures were also visible on charts depicting performance over time. Figure 8 presents selected plots, which best characterize the differences between all the analyzed measures. We note that the presented line charts were smoothed with Bezier curves to make the plots less noisy.

Figure 8a, b compares evaluations of HT on a balanced and imbalanced stream. We can see that the biggest difference involves recall, which changes from almost perfect on the balanced dataset to the second lowest statistic on the imbalanced one. Moreover, because the concept generating the examples did not change, rankings produced by classifiers remained similar and AUC remained at a high level, whereas measures that count crisp misclassifications changed their absolute values.

Looking at Fig. 8c, one can notice that the Naive Bayes algorithm has difficulties in adapting to new minority class subclusters that appear after 40k examples. SERA has a similar problem because it oversamples positive examples by selecting the most similar ones from



**Fig. 8** Comparison of prequential performance over time (dataset: measure(s)). **a** Ratio<sub>0500</sub>: HT, all measures. **b** Ratio<sub>0595</sub>: HT, all measures. **c** AppDis<sub>2</sub>: Recall. **d** AppDis<sub>2</sub>: SERA, all measures. **e** Sudden<sub>RC</sub>: Bag, all measures. **f** MinMaj: NB, all measures

previous data chunks, which become outdated when the minority class is split. This shows that recursive oversampling algorithms proposed for data streams could be improved to take into account appearing subconcepts. Figure 8d additionally exemplifies that an algorithm can adapt its ranking model fairly quickly (AUC), but due to static decision thresholds requires more examples to offer better minority class predictions ( $\kappa$ ,  $\kappa_m$ , G-mean, recall). In other words, according to AUC, SERA recovered from the drift very quickly, but the threshold converting example scores into crisp classifications ruled out many correct minority class predictions. Therefore, it would be interesting to consider decision thresholds dynamically tuned in the event of concept drift.

Figure 8e shows that all measures except accuracy are capable of depicting sudden class ratio changes over time. Similar observations were made for gradual ratio changes ( $\text{Gradual}_{\text{RC}}$ ). Finally, Fig. 8f shows that accuracy,  $\kappa$ , and  $\kappa_m$  are sensitive to minority-majority class swaps. Conversely, recall and G-mean report the improving positive class predictions, whereas AUC shows that ranking performance, and thus probably the class boundaries, did not change. Figure 8f constitutes an example of the complementary properties of the analyzed measures.

## 6 Conclusions and outlook

In case of static data, AUC is one of the most popular measures for evaluating classifiers, both on balanced and imbalanced classes. However, due to its computational costs, until now AUC has not been successfully adapted to data stream mining. To overcome these limitations, in [9] we proposed an efficient algorithm which calculates AUC with forgetting on evolving data streams, by using a sorted tree structure with a sliding window. In this paper, we investigated the basic properties of the resulting evaluation measure, called prequential AUC, in the context of data stream classification.

Below, we summarize the main findings of this study:

- Prequential AUC was compared with earlier, limited attempts to calculate AUC for data streams. As a result, prequential AUC was found to give less pessimistic estimations and, therefore, preferable to batch, incremental, and block AUC estimations for streams with concept drifts.
- The proposed measure was found to be statistically consistent and comparably discriminant with AUC calculated on stationary data.
- Prequential AUC was also positively evaluated in terms of processing speed. This stands in contrast to batch-calculated AUC, and is one of the key features making prequential AUC applicable to real-world streams.
- The window size used for calculating AUC with forgetting was found to have negligible impact on the resulting AUC estimation.
- We performed a comparative study of prequential accuracy, AUC, Kappa, Kappa M, G-mean, and recall. On a set of imbalanced data streams with various difficulty factors, the analyzed measures reacted differently to drifts and showcased characteristic properties concerning: focus on one or both classes, sensitivity to class ratio changes or minority-majority class swaps, ranking and crisp classification assessment.

The conclusions drawn in this paper can have important implications in evaluating and designing data stream classifiers. Prequential AUC can be considered an additional tool for comparing online learning algorithms on evolving data, especially ranking classifiers. This can help open new studies, as the topics of ranking or probability calibration have still not been thoroughly tackled in the context of evolving streams. Furthermore, the adaptation of AUC for data streams opens questions about the possibility of introducing other ranking measures, such as the H-measure, or in-depth ROC analysis to data stream mining. Our future studies, already in progress, will also deal with the impact of additional class imbalance difficulty factors [38] on stream classifiers. By analyzing ratio changes, small disjuncts, majority class swaps, borderline examples, rare cases, outliers, and possible drifts between these classes, we aim to create a complex testbed for algorithms learning from imbalanced streams.

**Acknowledgements** The authors would like to thank Maciej Piernik for insightful comments on a draft of this paper, as well as Tomasz Pewinski and Jan Staniewicz for helping implement part of the experiments. We also gratefully acknowledge the insightful comments made by the three anonymous reviewers. This research was funded by the Polish National Science Center under Grant No. DEC-2013/11/B/ST6/00963. Dariusz Brzezinski acknowledges the support of an FNP START scholarship.

**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.

## References

1. Bayer R (1972) Symmetric binary B-trees: data structure and maintenance algorithms. *Acta Inf* 1:290–306
2. Bifet A, Frank E (2010) Sentiment knowledge discovery in twitter streaming data. In: Proceedings of 13th discovery science international conference. Lecture notes in computer science, vol 6332, pp 1–15
3. Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. *J Mach Learn Res* 11:1601–1604
4. Bifet A, Morales GDF, Read J, Holmes G, Pfahringer B (2015) Efficient online evaluation of big data stream classifiers. In: Proceedings of 21st ACM SIGKDD international conference on knowledge discovery data mining, pp 59–68
5. Bouckaert RR (2006) Efficient AUC learning curve calculation. In: Proceedings of Australian conference on artificial intelligence. Lecture notes in computer science, vol 4304, pp 181–191
6. Branco P, Torgo L, Ribeiro RP (2016) A survey of predictive modelling under imbalanced distributions. *ACM Comput Surv* 49(2):31:1–31:50
7. Brzezinski D, Stefanowski J (2016) Stream classification. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning. Springer, Berlin. doi:10.1007/978-1-4899-7502-7\_908-1
8. Brzezinski D, Stefanowski J (2014) Combining block-based and online methods in learning ensembles from concept drifting data streams. *Inform Sci* 265:50–67
9. Brzezinski D, Stefanowski J (2015) Prequential AUC for classifier evaluation and drift detection in evolving data streams. In: New frontiers in mining complex patterns. Lecture notes in computer science, vol 8983, pp 87–101
10. Chawla NV (2010) Data mining for imbalanced datasets: an overview. In: Maimon O, Rokach L (eds) Data mining and knowledge discovery handbook, 2nd edn. Springer, Berlin, pp 875–886
11. Chen S, He H (2009) SERA: selectively recursive approach towards nonstationary imbalanced stream data mining. In: International joint conference on neural networks. IEEE Computer Society, pp 522–529
12. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res* 7:1–30
13. Ditzler G, Polikar R (2013) Incremental learning of concept drift from streaming imbalanced data. *IEEE Trans Knowl Data Eng* 25(10):2283–2301
14. Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. *IEEE Comp Intell Mag* 10(4):12–25
15. Egan JP (1975) Signal detection theory and ROC analysis. Cognition and perception. Academic Press, London
16. Fawcett T (2001) Using rule sets to maximize ROC performance. In: Proceedings 2001 IEEE international conference on data mining, pp 131–138
17. Flach PA (2003) The geometry of ROC space: understanding machine learning metrics through ROC isometrics. In: Proceedings of 20th international conference on machine learning, pp 194–201
18. Flach PA (2010) ROC analysis. In: Sammut C, Webb GI (eds) Encyclopedia of machine learning. Springer, Berlin, pp 869–875
19. Flach PA, Hernández-Orallo J, Ramirez CF (2011) A coherent interpretation of AUC as a measure of aggregated classification performance. In: Proceedings of 28th international conference on machine learning. Omnipress, pp 657–664
20. Gama J (2010) Knowledge discovery from data streams. Chapman and Hall, London
21. Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. *Mach Learn* 90(3):317–346
22. Gama J, Zliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. *ACM Comput Surv* 46(4):44:1–44:37

23. Hand DJ (2009) Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Mach Learn* 77(1):103–123
24. Hanley JA, Mcneil BJ (1982) The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143:29–36
25. Harries M (1999) Splice-2 comparative evaluation: electricity pricing. Technical report, The University of South Wales
26. He H, Garcia EA (2009) Learning from imbalanced data. *IEEE Trans Knowl Data Eng* 21(9):1263–1284
27. He H, Ma Y (eds) (2013) *Imbalanced learning: foundations, algorithms, and applications*. Wiley-IEEE Press, London
28. Hoens TR, Chawla NV (2012) Learning in non-stationary environments with class imbalance. In: Proceedings of 18th ACM SIGKDD international conference on Knowledge discovery data mining, pp 168–176
29. Hu B, Dong W (2014) A study on cost behaviors of binary classification measures in class-imbalanced problems. *CoRR arXiv:1403.7100*
30. Huang J, Ling CX (2005) Using AUC and accuracy in evaluating learning algorithms. *IEEE Trans Knowl Data Eng* 17(3):299–310
31. Japkowicz N, Shah M (2011) *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, Cambridge
32. Jo T, Japkowicz N (2004) Class imbalances versus small disjuncts. *SIGKDD Explor* 6(1):40–49
33. Kirkby R (2007) Improving Hoeffding trees. Ph.D. thesis, Department of Computer Science, University of Waikato
34. Kosina P, Gama J (2015) Very fast decision rules for classification in data streams. *Data Min Knowl Discov* 29(1):168–202
35. Kubat M, Holte RC, Matwin S (1997) Learning when negative examples abound. In: Proceedings of 9th European conference on machine learning. Lecture notes in computer science, Springer, vol 1224, pp 146–153
36. Lichtenwalter R, Chawla NV (2009) Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. In: PAKDD Workshops, Lecture Notes in Computer Science, vol 5669, pp 53–75
37. Metz CE (1978) Basic principles of ROC analysis. *Semin Nuclear Med* 8(4):283–298
38. Napierala K, Stefanowski J (2016) Types of minority class examples and their influence on learning classifiers from imbalanced data. *J Intell Inf Syst* 46(3):563–597
39. Philip Dawid A, Vovk VG (1999) Prequential probability: principles and properties. *Bernoulli* 5(1):125–162
40. Provost FJ, Domingos P (2003) Tree induction for probability-based ranking. *Mach Learn* 52(3):199–215
41. Provost FJ, Fawcett T, Kohavi R (1998) The case against accuracy estimation for comparing induction algorithms. In: Proceedings of 15th international conference on machine learning, pp 445–453
42. Street WN, Kim Y (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of 7th ACM SIGKDD international conference on knowledge on discovery data mining, pp 377–382
43. Theeramunkong T, Kijssirikul B, Cercone N, Ho TB (2009) PAKDD data mining competition
44. Wang B, Pineau J (2013) Online ensemble learning for imbalanced data streams. *CoRR arXiv:1310.8004*
45. Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of 9th ACM SIGKDD international conference on knowledge discovery on data mining, pp 226–235
46. Wang S, Minku LL, Yao X (2015) Resampling-based ensemble methods for online class imbalance learning. *IEEE Trans Knowl Data Eng* 27(5):1356–1368
47. Wang S, Minku LL, Yao X (2016) Dealing with multiple classes in online class imbalance learning. In: Proceedings on 25th international joint conference on artificial intelligence. IJCAI/AAAI Press, pp 2118–2124
48. Wu S, Flach PA, Ramirez CF (2007) An improved model selection heuristic for AUC. In: Proceedings of 18th European conference on machine learning. Lecture notes in computer science. Springer, vol 4701, pp 478–489
49. Zliobaite 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
50. Zliobaite I, Pechenizkiy M, Gama J (2016) An overview of concept drift applications. In: Japkowicz N, Stefanowski J (eds) *Big data analysis: new algorithms for a new society, studies in big data*, vol 16. Springer, Berlin, pp 91–114



**Dariusz Brzezinski** received his M.Sc. and Ph.D. degrees in Computer Science from Poznan University of Technology, Poland, in 2010 and 2015 respectively. He is currently a research assistant in the Department of Data Processing Technologies at Poznan University of Technology. His research interests include data stream mining, concept drift, and machine learning applications in structural crystallography.



**Jerzy Stefanowski** is an Associate Professor in the Institute of Computing Science, Poznan University of Technology. He received the Ph.D. and Habilitation degrees in computer science from this university. His research interests include machine learning, data mining and intelligent decision support—in particular rule induction, multiple classifiers, class imbalance, data preprocessing, and handling uncertainty in data.