A Novel Data Stream Learning Approach to Tackle One-Sided Label Noise From Verification Latency

Liyan Song^{1,2}, Shuxian Li^{1,2}, Leandro L. Minku^{*3}, Xin Yao^{*1,2}
songly@sustech.edu.cn, lisx@mail.sustech.edu.cn, L.L.Minku@bham.ac.uk, xiny@sustech.edu.cn
^{1} Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, China
^{2} Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation,
Department of Computer Science and Engineering, Southern University of Science and Technology, China
^{3} School of Computer Science, University of Birmingham, UK

***Abstract*—Many real-world data stream applications suffer from verification latency, where the labels of the training examples arrive with a delay. In binary classification problems, the labeling process frequently involves waiting for a pre-determined period of time to observe an event that assigns the example to a given class. Once this time passes, if such labeling event does not occur, the example is labeled as belonging to the other class. For example, in software defect prediction, one may wait to see if a defect is associated to a software change implemented by a developer, producing a defect-inducing training example. If no defect is found during the waiting time, the training example is labeled as clean. Such verification latency inherently causes label noise associated to insufficient waiting time. For example, a defect may be observed only after the pre-defined waiting time has passed, resulting in a noisy example of the clean class. Due to the nature of the waiting time, such noise is frequently one-sided, meaning that it only occurs to examples of one of the classes. However, no existing work tackles label noise associated to verification latency. This paper proposes a novel data stream learning approach that estimates the confidence in the labels assigned to the training examples and uses this to improve predictive performance in problems with one-sided label noise. Our experiments with 14 real-world datasets from the domain of software defect prediction demonstrate the effectiveness of the proposed approach compared to existing ones.**

***Index Terms*—Data stream learning, one-sided label noise, verification latency, confidence level, concept drift, clustering, just-in-time software defect prediction.**

I. Introduction

Many real-world data stream applications, such as credit card fraud detection, credit card approval, network intrusion detection and Just-In-Time Software Defect Prediction (JIT-SDP), suffer from *verification latency*, where the labels of training examples arrive with a delay [1], [2]. In binary data stream classification problems, the labeling process of training examples with verification latency frequently involves waiting for a pre-determined period of time to observe an event that assigns the example to a given class. Once this time passes, if such labeling event does not occur, the example is labeled as belonging to the other class. Take JIT-SDP [3] as an example. A *defect-inducing* example created based on a software change implemented by a developer is created either when a defect is found to be associated to this software change or when a pre-defined period of time has passed for one to be confident that this change should be *clean*, whichever is earlier. The pre-determined period of time one needs to wait for labeling the clean class is called *waiting time*.

Verification latency inherently causes label noise associated to insufficient waiting time. Take JIT-SDP again as an example. If a defect is found to be associated to a software change only *after* the waiting time, a noisy training example labeled as clean will be created *at* the waiting time. If a larger waiting time had been used, the example may have not been labeled as clean. However, a too large waiting time would mean that the examples being labeled are obsolete, hindering predictive performance due to potential concept drifts. Therefore, as the time it takes to find a defect varies for different software changes, it is impossible to set a perfect waiting time for all changes, which inevitably leads to label noise. Due to the nature of the waiting time, such noise is frequently *one-sided*, i.e., it only occurs to examples of one of the classes. In JIT-SDP, only training examples of the defect-inducing class would be mislabeled as clean due to insufficient waiting time.

No existing data stream learning approach can handle such one-sided label noise associated with verification latency. This paper proposes a novel data stream learning approach to deal with such label noise by estimating the confidence in the labels assigned to training examples and uses this to improve predictive performance. The proposed method is named as **O**versampling based **Da**ta **S**tream bagging with **C**onfidence (ODaSC), which is then evaluated in the context of JIT-SDP. We answer the following research questions:

RQ1 How to estimate the label confidence of training examples? How accurate are such estimates in the context of JIT-SDP?

RQ2 How to use label confidence to improve the robustness of classification models against the one-sided label noise and improve predictive performance? How well does the proposed approach perform in the context of JIT-SDP?

Our main contributions are:

- we formulate the one-sided label noise associated to verification latency for the first time;
- we propose a method to estimate the confidence in the

[*] the corresponding authors.

labels assigned to the training examples to tackle one-sided label noise associated with verification latency and experimentally evaluate such estimates;

- we propose a novel data stream learning approach named ODaSC based on the estimates of label confidence and experimentally evaluate its predictive performance in the context of JIT-SDP. Our experiments based on 14 real world datasets demonstrate ODaSC's effectiveness.

II. Problem Statement

At each time step i, we consider that a new test data point X_i arrives, where $X_i \in \mathbb{R}^d$ are the d-dimensional features. The true label $y_i \in \{0, 1\}$ of X_i is unknown. We aim to use the most up-to-date model $\mathcal{M}(\cdot)$ to give a prediction to X_i, i.e., $\widehat{y}_i = \mathcal{M}(X_i)$. This process continues with time as more test examples arrive. To be kept up-to-date, before giving a new prediction, $\mathcal{M}(\cdot)$ is updated with any training example produced between time steps $i-1$ and i. Training examples are produced as explained over the next paragraphs.

Without loss of generality, assume that a training example $(X_j, y_j^* = 1)$ of class 1 is produced immediately when a class 1 labeling event associated to X_j occurs, where y_j^* is the observed label. For instance, in JIT-SDP, a training example $(X_j, y_j^* = 1)$ of the defect-inducing class is produced as soon as a defect is found to be associated to X_j. If no class 1 labeling event occurs during a pre-defined *waiting time* of W days, the example X_j is assumed to belong to class 0, producing a training example $(X_j, y_j^* = 0)$.

The time it takes for a class 1 labeling event to occur is unknown and may vary from example to example. Therefore, the class 1 labeling event may occur at any moment within or after W days. When it occurs after W days, a noisy example $(X_j, y_j^* = 0)$ is first produced upon W days. As such kind of noise resulting from verification latency can only affect examples with observed label $y_j^* = 0$, we refer to this type of noise as *one-sided label noise*. After W days, an example $(X_j, y_j^* = y_j = 1)$ is then produced upon the class 1 labeling event, where y_j is a label free of one-sided label noise. When the class 1 labeling event occurs within W days, only a single example $(X_j, y_j^* = y_j = 1)$ is produced. It is worth noting that not all noisy training examples $(X_j, y_j^* = 0)$ will be followed by a corresponding example $(X_j, y_j^* = y_j = 1)$, because the class 1 labeling event associated to X_j may never occur. For instance, in JIT-SDP, a defect associated to a given software change may never be found within the duration of the project.

We aim to propose a novel data stream learning approach to tackle one-sided label noise resulting from verification latency by estimating how confident we are that $y_j^* = y_j$ when $y_j^* = 0$.

III. Related Work

A. Data Stream Learning to Tackle Verification Latency

There have been a limited number of papers investigating data stream learning with verification latency [4]–[7]. Kuncheva et al. [4] investigated an online scenario where all training examples are labeled exactly at a pre-defined number of τ time steps after their features are generated. Zhang et al. [6] investigated a scenario where training data arrive in chunks: part of them can get their true labels and the rest can only get features without label. In both cases, the duration of the labeling delay is assumed to be fixed to the same value for all examples and known a priori. Pozzolo et al. [7] investigated an online scenario where it is known beforehand which training examples have their true labels arriving early and which are likely to have delayed labels. There are also papers investigating an extreme case of the online scenario with verification latency, where labeled training examples are received only in the initial learning stage and after that no additional labeled training examples become available [5], [8]. Nevertheless, no existing work investigated the online learning scenario of this paper, where the time to label training examples varies and there may be (one-sided) label noise.

B. Data Stream Learning to Tackle Label Noise

Only a few data stream learning approaches have been proposed to deal with label noise. Cesa-Bianchi et al. [9] proposed a method to improve linear and kernel-based classifiers to tackle label noise in online learning without verification latency, based on the assumption that a global bound on the noise variance is known. Frenay et al. [10] extended a prototype-based online classifier in a probabilistic manner to deal with label noise. Oliveira et al. [11] adopted a filter based on data hardness techniques, but requires a window of data to be stored for this purpose. There are also studies investigating "label" noise in regression problems under online learning scenarios without verification latency [12], [13]. Nevertheless, none of them aimed to handle the one-sided label noise resulting from verification latency as this paper investigates.

C. Just In Time Software Defect Prediction (JIT-SDP)

JIT-SDP is a typical problem suffering from one-sided label noise resulting from verification latency. However, no existing work has handled such one-sided label noise problem yet. Kamei et al. conducted a large-scale empirical study [14] investigating 14 features extracted from commits and bug reports for JIT-SDP. They showed them to be good indicators for yielding good predictive performance on both open source and commercial projects. However, they assumed JIT-SDP to be an offline learning problem. Since then, several studies have showed that *concept drift* often occurs during the life-cycle of the project development process, which would negatively impact performance of classifiers trained on old data [3], [15]. Thus, an online learning approach should be taken in JIT-SDP.

JIT-SDP also suffers from *class imbalance*, where the defect-inducing class is typically the minority compared to the clean class [3], [16], [17]. Shuo et al. [18] proposed oversampling and undersampling techniques to deal with the online class imbalance, creating two online ensemble learning approaches, namely Oversampling-based Online Bagging (OOB) and Undersampling-based Online Bagging (UOB). OOB was more robust against dynamic changes in class imbalance status [18] and performed competitively for JIT-SDP [19]. Therefore, we adopt OOB as the foundation framework for our study.

The challenges imposed by concept drift and class imbalance in JIT-SDP could be further exacerbated by verification latency. Model adaptation to concept drift may be delayed as a result of the waiting time, and class imbalance may become more extreme as a result of one-sided label noise [3]. In particular, when the waiting time is insufficient to obtain the true label of a defect-inducing software change due to verification latency, a noisy training example labeled as clean would be produced, further reducing the number of examples of the defect-inducing class, which is typically a minority.

IV. Label Confidence Estimation

This section aims for answering RQ1: *How to estimate the label confidence of training examples?* We propose to estimate the label confidence by using micro-clusters, which form a high-level description of the data stream without actually storing them [20], [21]. We adopt DenStream [21] to produce micro-clusters, as it can maintain a varying number of micro-clusters and be robust to the data stream containing noise. In DenStream, micro-clusters of the evolving data stream are created via damped windows, in which the weight of each example decreases exponentially with time t following a fading function $f(t) = 2^{-\lambda \cdot t}$, where $\lambda > 0$ balances the contribution of the past examples to micro-cluster construction. Our preliminary experiments demonstrated good and stable micro-clustering performance for $\lambda = 0.1$, so we use this value throughout this study. This section explains how to estimate label confidence based on the created micro-clusters.

A. Micro-Clusters with Label Information

DenStream does not maintain information about the class label distribution over time. We propose to incorporate such label information for micro-clusters by tracking the numbers of examples of each class over time.

Suppose that a micro-cluster C has been updated with n training examples at the current Unix timestamp t. The micro-cluster with label information is represented as

$$\{\overline{CF^1}, \overline{CF^2}, CT, C0, C1\}, \tag{1}$$

where $\overline{CF^1}$, $\overline{CF^2}$ and CW are internal micro-cluster statistics kept by DenStream [21], with $\overline{CF^1}$ being the weighted linear sum of the training examples; $\overline{CF^2}$ being the weighted squared sum of the training examples; $CT = \sum_{k=1}^{n} f(t - T_k)$ measuring the obsolescence of this micro-cluster and T_k being the Unix timestamp the k^{th} training example used to update the micro-cluster. $C0$ and $C1$ are counters of the number of training examples with $y^* = 0$ and $y^* = 1$, respectively. However, as these numbers may be affected by one-sided label noise, these counters are adjusted based on the estimates of label confidence, as will be explained in Section V-B.

Based on this, $P^{(0)}(C) := \frac{C0}{C0+C1}$ and $P^{(1)}(C) := \frac{C1}{C0+C1}$ are the estimated probabilities of examples of class 0 and class 1 in this micro-cluster, respectively. During training, several micro-clusters $\{C_j\}$ are usually produced to describe the data stream, based on which $P^{(0)}(D) := \frac{\sum_j C0_j}{\sum_j C0_j + C1_j}$ and $P^{(1)}(D) := \frac{\sum_j C1_j}{\sum_j C0_j + C1_j}$ are used to denote the probability of examples of class 0 and class 1 of the data stream, respectively.

B. Micro-Cluster Representativeness

This procedure aims to measure the degree with which a micro-cluster can individually represent classes 0 and 1. We use *representativeness* for a micro-cluster to measure the degree this micro-cluster as a whole to represent a specific data class. Figure 1(a) illustrates a case where $P^{(1)}(C_2) < P^{(0)}(C_2)$, so micro-cluster C_2 is less representative for class 1 than for class 0. In addition, $P^{(1)}(C_2) > P^{(1)}(D)$, so the representativeness of C_2 for class 1 should be tuned upwards from $P^{(1)}(C_2)$. Based on these ideas, the ***representativeness*** of micro-cluster C for class $l \in \{0, 1\}$ is defined as

$$rep^{(l)}(C) := P^{(l)}(C) \cdot [1 + d(P^{(l)}(C)||P^{(l)}(D))] \tag{2}$$

where $P^{(l)}(C)$ and $P^{(l)}(D)$ denote the probability of examples of class l in micro-cluster C and in the whole data stream, respectively. The value $d(P^{(l)}(C_j)||P^{(l)}(D)$ is the *directed probability distance*, defined as $d(P^{(l)}(C)||P^{(l)}(D)) := \frac{1}{2} \cdot \frac{P^{(l)}(C) - P^{(l)}(D)}{P^{(l)}(C) + P^{(l)}(D)}$. Specifically, when $P^{(l)}(C) \neq P^{(l)}(D)$, the representativeness of micro-cluster C for class l is tuned upwards when $P^{(l)}(C) > P^{(l)}(D)$ or downwards when $P^{(l)}(C) < P^{(l)}(D)$. Take micro-cluster C_2 of Figure 1(a) again as an example. Its representativeness for class 1 is $rep^{(1)}(C_2) = 0.4 \cdot [1 + (0.4 - 0.25)/(2 \cdot (0.4 + 0.25)] = 0.4 \cdot (1 + 3/26)$, being a slight increase over $P^{(1)}(C_2)$.

C. Micro-Cluster Weight

Given a training example, this procedure aims to compute the weight of the micro-cluster for the purpose of estimating the label confidence of this training example later. The proposed micro-cluster weight is computed by considering two factors: (a) the correlation between the training example and the micro-cluster and (b) the obsolescence of the micro-cluster.

We define the ***micro-cluster correlation*** between training example (X, y^*) and micro-cluster C as

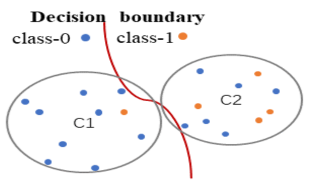
$$cor(X, C) := exp(-T \cdot d(X, C)) \tag{3}$$

where $T = 10$ is a *temperature* to enlarge the correlation for different examples, the distance $d(X, C) := max(0, ||X - o(C)||_2 - r(C))$, and $o(C)$ and $r(C)$ denote the centroid and the radius of micro-cluster C, respectively. When the training example locates within the micro-cluster, the correlation can reach the maximal 1; whereas, when it is far from the micro-cluster, the correlation is very small.

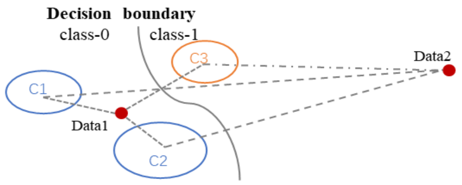
If a micro-cluster has not been updated with new examples for a relatively long time, this micro-cluster is getting obsolete and cannot depict the current status of the data stream. Thus, we should decrease the weight of this micro-cluster when we estimate the label confidence of this training example later. We define the ***micro-cluster obsolescence*** of C as

$$obs(C) := tanh(t(C)/T), \tag{4}$$

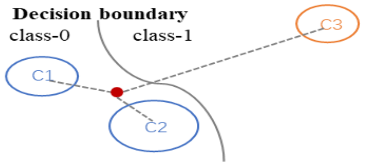
where $t(C) = CT > 0$ (Eq. (1)) of DenStream, $tanh(x) \in [0, 1]$ for $\forall x \in \mathbf{R}^+$ is chosen to convert the micro-cluster time



(a) Micro-cluster representativeness.



(b) Two data scenarios to compute class-1 label confidence.



(c) Micro-cluster pruning.

Fig. 1. Estimate of label confidence. Micro-clusters C_1 and C_2 of Figure 1(a) are produced by 10 training examples. The label information is $P^{(0)}(C_1) = 0.9$ and $P^{(1)}(C_1) = 0.1$ for C_1, and $P^{(0)}(C_2) = 0.6$ and $P^{(1)}(C_2) = 0.4$ for C_2. For the entire data stream D, $P^{(0)}(D) = 0.75$ and $P^{(1)}(D) = 0.25$. As micro-cluster C_3 of Figure 1(c) is far away from the training example (the red dot), so it is better excluded from the estimation of label confidence.

information to be within 0 and 1 in a monotonic way, and $T = 10$ is the *temperature*. Smaller values of obsolescence mean more obsolete micro-clusters and thus should play a smaller role in the estimate of label confidence later.

We define the ***micro-cluster weight*** of C on training example (X, y^*) as

$$w(X, C) := cor(X, C) \cdot obs(C) \tag{5}$$

where $cor(\cdot)$ and $obs(\cdot)$ are defined in Eqs. (3) and (4), respectively.

D. Class 1 Label Confidence of Training Examples

This procedure aims to compute the confidence a training example would belong to the class 1 free from one-sided label noise, by making use of one-sided label noise associated to verification latency. We name it the *class-1 label confidence* of training example, denoted as $\alpha^{(1)}(\cdot) \in [0, 1]$. Class 0 label confidence of the same training example is computed automatically as $1 - \alpha^{(1)}(\cdot)$. We propose two steps to achieve this aim, as explained next.

Step 1. Computing the class 1 label confidence of training examples.

There are two possible cases for the calculation of the class 1 label confidence of a training example: (1) when this example has micro-clusters in its neighborhood and (2) when this example does not have micro-clusters in its neighborhood, such that its class 1 label confidence cannot be calculated based on existing micro-clusters. Figure 1(b) shows an example of that, where "Data1" and "Data2" correspond to training examples under cases (1) and (2), respectively.

Given training example (X, y^*) and micro-clusters $\{C_j\}$, this training example belongs to case (1) if the criterion is satisfied that $\sum_j w(X, C_j) \geq \rho_w$, where $w(X, C_j)$ is defined in Eq. (5) and $\rho_w = 10^{-5}$ is a predefined threshold. Otherwise, the training example is considered to belong to case (2).

When the training example belongs to case (1), the class 1 label confidence of this example X is formulated as

$$\alpha^{(1)}(X, y^*) = \frac{\sum_{j \in F} rep^{(1)}(C_j) \cdot w(X, C_j)}{\sum_{l \in \{0,1\}} \sum_{j \in F} rep^{(l)}(C_j) \cdot w(X, C_j)}, \tag{6}$$

where $l \in \{0, 1\}$ is the class label, $rep^{(l)}(\cdot)$ and $w(\cdot)$ are defined in Eqs. (2) and (5), respectively, F is the "filtered" set of micro-clusters $\{C_j | w(X, C_j) \geq \rho(X)\}$, and $\rho(X)$ is a threshold used to filter out micro-clusters that have too low weight with respect to the example X. For example, in Figure 1(c), micro-cluster C3 is too far from the training example shown in red. Therefore, its weight would be very small (below the threshold) and it should not be used to compute the label confidence of this example.

Given a training example (X, y^*) and the set of micro-clusters $\{C_j\}$ available at current time, we define the *filtering threshold* $\rho(X)$ by simulating the "68-95-99" rule of the Gaussian distribution as $\rho(X) := \mu\{w(X, C_j)\} - \sigma\{w(X, C_j)\}$, where $\mu\{\cdot\}$ and $\sigma\{\cdot\}$ denote the mean and standard deviation of micro-cluster weights for this example (Eq. (5)).

When the training example belongs to case (2), the class 1 label confidence cannot be calculated based on the existing micro-clusters. Therefore, it is computed based on the entire data stream, as

$$\alpha^{(1)}(X, y^*) := \begin{cases} 1 & \text{for} \quad y^* = 1 \\ P^{(1)}(D) & \text{for} \quad y^* = 0 \end{cases} \tag{7}$$

where $P^{(1)}(D)$ denotes the probability of examples with $y^* = 1$ in the data stream.

Step 2. Calibrating the class 1 label confidence of training examples based on one-sided label noise.

As only examples whose observed label is $y^* = 0$ may suffer from one-sided label noise resulting from verification latency, we know that training examples with observed label $y^* = 1$ should have the maximum possible class 1 label confidence value of 1. Therefore, examples of class 1 can be used as "training data" to further calibrate the estimation of the class 1 label confidence derived previously. It is worth to noting that this step is only performed when a training example with observed label $y^* = 1$ arrives.

Without loss of generality, suppose that n new training examples become available between test time steps i and $i+1$, among which d examples are free from one-sided label noise. We derive the *benchmark class-1 label confidence* as

$$\begin{cases} \alpha^{(1)-} := \alpha^{(1)}(X, 1) & \text{if } d = 1 \\ \alpha^{(1)-} := max(\mu - k^{min} \cdot \sigma, 0) & \text{otherwise} \end{cases} \tag{8}$$

where $(X, 1)$ represents the only training example of class 1 when $d = 1$; μ and σ denote the mean and the standard deviation of the class 1 label confidence values of all d training examples of class 1, respectively, and k^{min} is the minimal $k \in \{1, 2, 3\}$, such that the below inequality that simulates the "68-95-99" rule of the Gaussian distribution is satisfied as $\mu - k \cdot \sigma \leq \alpha^{(1),min}$, where $\alpha^{(1),\,min} :=$

$min\{\alpha^{(1)}(X_i, y_i^*)|y_i^* = 1\}$ for $i = \{1, \cdots, n\}$ and we have $\alpha^{(1)-} \leq \alpha^{(1),min}$. Literally, the benchmark value approximates the lower bound of the class 1 label confidence a training example of class 1 can possibly have. Then, we find the training example(s) that have larger class 1 label confidence than the benchmark value (i.e., $\alpha^{(1)}(X_i, y_i) \geq \alpha^{(1)-}$), and calibrate their class 1 label confidence with the formula:

$$\alpha^{(1)}(X_i, y_i^*) \equiv min(1, \alpha^{(1)}(X_i, y_i^*)/\alpha^{(1),min}). \tag{9}$$

For simplicity, we keep using $\alpha^{(1)}(\cdot)$ to denote the final class-1 label confidence that may be calibrated.

Finally, we estimate the label confidence of training example (X, y^*) based on the calibrated class 1 label confidence, as

$$\alpha(X, y^*) := \begin{cases} \alpha^{(1)}(X) & \text{if} \quad y^* = 1 \\ 1 - \alpha^{(1)}(X) & \text{if} \quad y^* = 0 \end{cases} \tag{10}$$

Training examples with a larger label confidence would be used more times to update our online learning approach.

V. The Proposed Online Learning Approach

This section aims at answering RQ2: *How to use label confidence to improve predictive performance?* The main idea is to encode the derived label confidence of training examples into an online learning framework. As explained in Section III-C, we use Oversampling Online Bagging (OOB) [18] as our underlying online learning framework due to its competitive results [19], [22] in data stream problems that may suffer from class imbalance. As in previous work with this approach [19], [22], Hoeffding trees [23] are used as the base learners as they can be updated incrementally in a strict online manner.

A. Over-Sampling Based Online Bagging With Confidence

OOB [18] is an oversampling approach that tracks the proportion of examples of each class over time in the data stream based on exponential smoothing. Whenever a new training example of the minority class arrives, it is presented $k \sim Poisson(\lambda)$ times to the base learner (Hoeffding tree), where $\lambda = P_{maj}/P_{min}$ is a class imbalance factor to control the oversampling rate, and P_{maj} and P_{min} are the tracked proportions of examples of the current majority and minority classes, respectively. Poisson distribution is adopted for the same reason as in [18], [24]. An ensemble of M base learners is kept and k is drawn independently for each of them. Predictions are based on the majority vote among the predictions of each base learner.

As there may be one-sided label noise, we propose to change λ into a new factor Λ that incorporates label confidence, giving rise to our proposed **O**versampling based **Da**ta **S**tream bagging with **C**onfidence (ODaSC) approach.

Definition 1 (Confidence-based sampling rate)**.** The expected number of times a training example (X, y^*) of the minority class is used to train the online base learner is formulated as

$$\Lambda(X, y^*) := \lambda * e^{\alpha(X,y^*)-\delta} \tag{11}$$

where λ denotes the original class imbalance factor [18], $\alpha(X, y^*)$ is defined in Eq. (10) and $\delta \in [0, 1]$ denotes a *threshold* hyper-parameter that controls the increase / decrease in the oversampling rate. Particularly, the number of times an example is used for training is enlarged if the label confidence is larger than the threshold δ and is reduced if the label confidence is smaller than δ.

Each base learner of the online ensemble in our ODaSC approach is then trained k times with the training example (X, y^*), where $k \sim Poisson(\Lambda(X, y^*))$ is obtained independently for each base learner. Training is performed as soon as new training examples with label confidence become available.

B. Update Micro-Clusters

Given a new training example (X, y^*), after its label confidence $\alpha(X, y^*)$ is computed based on Section IV-D, the micro-clusters are updated with this new example.

Clustering elements $\overline{CF^1}$, $\overline{CF^2}$ and CW of C_j are updated by the DenStream learning algorithm [21]. $C0$ and $C1$ are updated by taking into account not only the numbers of examples received so far with $y^* = 0$ and $y^* = 1$, but also the label confidence as follows. For every micro-cluster C_j where $X \in C_j$, apply the following rules to update C_j's $C0$ and $C1$:

$$\begin{cases} C1^{new} = C1^{old} + 1, & \text{if } y^* = 1 \\ C0^{new} = C0^{old} + 1, & \text{if } y^* = 0 \\ C1^{new} = C1^{old} + 1, & \text{if } y^* = 0 \;\&\; \alpha(X, y^*) < 1 - \delta \end{cases} \tag{12}$$

where these rules are not mutually exclusive, $1 - \delta$ is a threshold representing the minimum amount of confidence in the label of an example with $y^* = 0$, and δ is the parameter in Eq. (11). When the confidence does not reach the threshold, the example will be counted both as an example of class 0 (through the second rule above) and as an example of class 1 (through the third rule above). This represents the fact that there is not enough confidence in its label and that the example may be suffering from one-sided label noise.

VI. Experimental Setup

A. Datasets

Our study uses 14 real-world datasets from the domain of JIT-SDP to demonstrate the effectiveness of our approach, as shown in Table I. They were chosen among projects with more than 4 years of duration, rich history (>10k commits) and a wide range of defect-inducing change ratio (2% ∼ 45%). The first six projects were made available by Cabral et al. [3] and the others were selected randomly among GitHub projects that match the aforementioned criteria. The `Commit Guru` [25] tool was used to collect the data with 14 features [14] (Section III-C). The labels are defect-inducing (class 1) or clean (class 0). For a uniform investigation, we use the first 10k software changes of each project in the experiments.

B. Competing Methods

To evaluate our proposed approach, we need to compare it against two special label confidence setups – the *optimal*

TABLE I
AN OVERVIEW OF THE DATASETS. THE TOP 10,000 SOFTWARE CHANGES (COMMITS) ARE USED IN OUR EXPERIMENTS.

Project	Total Changes	%Defect-inducing	%Defect-inducing	Time Period
Brackets	11,477	33.876	36.1	12/2011 - 12/2017
Broadleaf	12,034	20.542	22.79	11/2008 - 12/2017
Camel	29,860	20.8	34.5	03/2007 - 12/2017
Fabric	12,282	20.738	21.27	12/2011 - 12/2017
jGroups	17,947	17.524	20.34	09/2003 - 12/2017
Nova	22,872	43.516	52.56	08/2010 - 01/2018
Corefx	26,191	6.911	6.8	11/2014 - 10/2019
Django	25,662	42.156	47.89	07/2005 - 09/2019
Rails	56,049	25.374	36.74	11/2004 - 09/2019
Rust	68,344	2.032	6.2	06/2010 - 10/2019
Tensorflow	64,135	24.877	30.36	11/2015 - 11/2019
Tomcat	18,559	27.933	33.25	03/2006 - 06/2017
VScode	51,459	2.268	3.79	11/2015 - 10/2019
wp-Calypso	30,015	22.529	24.61	11/2015 - 10/2019

label confidence and the *random* label confidence. *Optimal label confidence* for training example (X, y^*) is defined as

$$\overline{\alpha}(X, y^*) := \begin{cases} 1 & \text{if} \quad y^* = y \\ 0 & \text{if} \quad y^* \neq y \end{cases}$$

where y^* is the observed label and y is the label free from one-sided label noise. The *random label confidence* for training example (X, y^*) is defined as $\underline{\alpha}(X, y^*) \sim U[0, 1]$, where $U[0, 1]$ denotes uniform distribution between 0∼1. Optimal label confidence is the best label confidence that could possibly be obtained based on the available information throughout the whole data stream, whereas random label confidence reflects an approach that estimates label confidence randomly.

By replacing our label confidence in our proposed ODaSC with the optimal and the random label confidences, two benchmark online approaches ODaSC-opt (short for ODaSC-optimal) and ODaSC-rnd (ODaSC-random) are produced. It is conjectured that ODaSC-opt would usually deliver superior predictive performance due to the optimal estimates of label confidence for training examples; on the contrary, ODaSC-rnd would provide a bottom line of predictive performance as a result of the random estimates on label confidence. A third benchmark classifier is produced based on training examples free from one-sided label noise, by filtering out examples with $y^* \neq y$, named as *ODaSC-flt* (short for ODaSC-filter). We should note that *ODaSC-opt* and *ODaSC-flt* are not applicable in practice as true labels of training examples are not accessible. They are used for the sole purpose of evaluating our proposed approach. Based on our conjectures, it would be desirable for our ODaSC to perform better than ODaSC-rnd and as close as possible to ODaSC-opt.

We also compare the proposed ODaSC against Online Bagging (OB) [24] and Oversampling-based Online Bagging (OOB) [18]. As explained in Section III, OOB ensemble of Hoeffding trees [23] has shown to gain state-of-the-art predictive performance in the context of JIT-SDP [19], but was not designed for dealing with the one-sided label noise.

C. Performance Evaluation

Root-Mean-Square-Error (RMSE) is chosen to evaluate the proposed method for estimating label confidence for RQ1. G-

TABLE II
COMPARISON OF THE MEAN PERFORMANCE ACROSS 30 RUNS. THE LAST THREE ROWS REPORT THE STATISTICAL TESTS ACROSS DATASETS. ODASC IS THE CONTROL METHOD ATTACHED WITH A STAR. SIGNIFICANT DIFFERENCE AGAINST ODASC IS HIGHLIGHTED IN YELLOW (LIGHT GRAY). SMALLER RANKINGS REPRESENT BETTER PERFORMANCE WHEN THERE IS STATISTICALLY SIGNIFICANT DIFFERENCE.

Project	RMSE		G-Mean					
	Ours	Random	ODaSC	#-opt	#-filter	#-rnd	OOB	OB
Brackets	0.4360	0.5770	0.6575	0.6525	0.6564	0.6201	0.6367	0.5186
Broadleaf	0.4310	0.5780	0.6062	0.6403	0.6406	0.4932	0.5732	0.1433
Camel	0.4150	0.5770	0.6498	0.6816	0.6753	0.6247	0.6495	0.3416
Corefx	0.3720	0.5780	0.6058	0.5978	0.5947	0.5958	0.5799	0.0914
Django	0.4170	0.5780	0.6838	0.6930	0.6926	0.6505	0.6534	0.4126
Fabric	0.3980	0.5780	0.6511	0.6596	0.6554	0.6402	0.6303	0.1889
jGroups	0.4650	0.5780	0.5738	0.5684	0.5800	0.5514	0.5453	0.3054
Nova	0.4130	0.5780	0.6475	0.6497	0.6409	0.5762	0.5883	0.4121
Rails	0.4460	0.5780	0.5650	0.5878	0.6228	0.4877	0.5004	0.2581
Rust	0.4530	0.5780	0.5689	0.5200	0.5215	0.5007	0.4948	0.1805
Tensorflow	0.4190	0.5770	0.6854	0.6923	0.7034	0.5927	0.6560	0.2912
Tomcat	0.4640	0.5770	0.6135	0.6234	0.6167	0.4998	0.4944	0.1318
VScode	0.2610	0.5780	0.5811	0.5731	0.4603	0.4225	0.5768	0.1568
wp-Calypso	0.4280	0.5780	0.5310	0.5578	0.5696	0.5319	0.5724	0.1523
aveRank	1.00	2.00	2.43	2.00	2.07	4.50	4.00	6.00
reject H0	1		*	0	0	1	1	1

mean is chosen to evaluate predictive performance of online learning approaches for RQ2, for being more robust to class imbalance [3], [22]. It is the geometric mean of the recall on class 0 (Recall-0) and on class 1 (Recall-1). In the online learning scenario, the performance metrics are computed prequentially by using a *fading factor* to track the changes of predictive performance over time [26]. The fading factor 0.99 is used following previous JIT-SDP studies [3], [19].

We conduct grid search to choose the best parameter setting for each approach. Three parameters are shared among ODaSC, ODaSC-opt, ODaSC-flt, and ODaSC-rnd: the ensemble size (# Hoeffding trees) $\in \{5, 10, 20, 30, 40\}$, the decay factor of class imbalance $\in \{0.9, 0.95, 0.99, 0.999\}$, and the resampling threshold (Eq. (11)) $\in \{0.8, 0.9\}$. OOB has two parameters: the ensemble size and the decay factor of class imbalance, with the same setup as ODaSC. OB has one tuning parameter: the ensemble size. We choose the parameter setting that achieves the best average G-mean across 30 runs based on the first 2,000 (out of the total 10,000) software changes in the data stream of each project. Hoeffding trees use the default parameters provided in the python package *scikit-multiflow* [27], following previous studies in JIT-SDP [3], [19].

The predictive performance of online learning approaches with the best parameter settings is evaluated based on the rest 2,000∼10,000 software changes of the project. Comparisons are conducted based on the mean performance across 30 runs. We report the results corresponding to the waiting time of 15 days for usually delivering good predictive performance with respect to the median G-means across projects.

VII. EXPERIMENTAL RESULTS

A. How Accurate Are the Estimates of Our Label Confidence?

This section aims to complete the answer to RQ1. The RMSE of our proposed method for estimating label confidence and of the random estimator are shown in the second and third columns of Table II. The RMSE of the random label confidence is always similar across different datasets, as the estimation is entirely random. The RMSE of the optimal label confidence is always zero.

The estimations given by our method did not reach RMSE very close to zero, meaning that there is room for improvement. However, its RMSEs were always better than those of the random label confidence method, showing that our method is effective. Wilcoxon signed rank tests with Holm-Bonferroni correction [28] at the significance level 0.05 across all datasets confirm that the superiority of our label confidence against random label confidence is significant. In particular, the null hypothesis (H0) that the two methods are equivalent is rejected with p-value 9.14E-5.

B. *How Well Does ODaSC Perform in JIT-SDP?*

This section aims to complete the answer to RQ2, based on the results in the "G-mean" part of Table II. Recall-1 and Recall-0 are also reported in Table III to support the analyses.

We can see that the proposed ODaSC outperformed OOB in terms of G-Mean in all datasets except for wp-Calypso. Some improvement ratios were as high as around 10.07% in Nova, 12.92% in Rails, 14.98% in Rust and 24.08% in Tomcat, being of large magnitude. Also, ODaSC performed no worse than ODaSC-opt and ODaSC-flt in all datasets except for Broadleaf, Camel, Rails, and wp-Calypso.

We conduct Friedman test [28] for statistical comparisons of all methods across all datasets. The null hypothesis (H0) states that all methods are equivalent in terms of G-mean. Friedman test with the significance level 0.05 rejects H0 with the p-value 0.000, showing that the six approaches obtained statistically different predictive performance. ODaSC was chosen as the control method to conduct post-hoc tests with Holm-Bonferroni corrections. ODaSC outperformed the state-of-the-art OOB significantly with p-value 0.0131. No significant difference was found between ODaSC vs ODaSC-opt (p-value is 0.7278) and vs ODaSC-flt (p-value is 0.6932). This is a very desirable behavior, given that ODaSC-opt is using the optimal label confidences and ODaSC-flt is optimally filtering out noisy examples. It means that even though our label estimations are not perfect (Section VII-A), they are good enough to achieve very competitive ODaSC results.

Friedman test also provides average ranks of the methods across datasets, which can provide a reasonable idea of how the methods compare to each other given rejection of H0 [28]. We can see from Table II that ODaSC, ODaSC-opt and ODaSC-flt usually outperformed OOB by having better average ranks, and achieved similar performance among themselves. Post hoc tests with Holm-Bonferroni corrections find significant superiority of ODaSC against ODaSC-rnd with p-value 0.0017. Such superiority of ODaSC over ODaSC-rnd also indicates the effectiveness of our label confidence in delivering better predictive performance, further supporting our answer to RQ1. OB performs the worst on all datasets as anticipated, having the worst average rank. This is probably due to its inability to deal with class imbalance, which occurs in JIT-SDP. Post hoc tests with Holm-Bonferroni corrections show the inferiority of OB compared to ODaSC to be significant with p-value 2.20E-07.

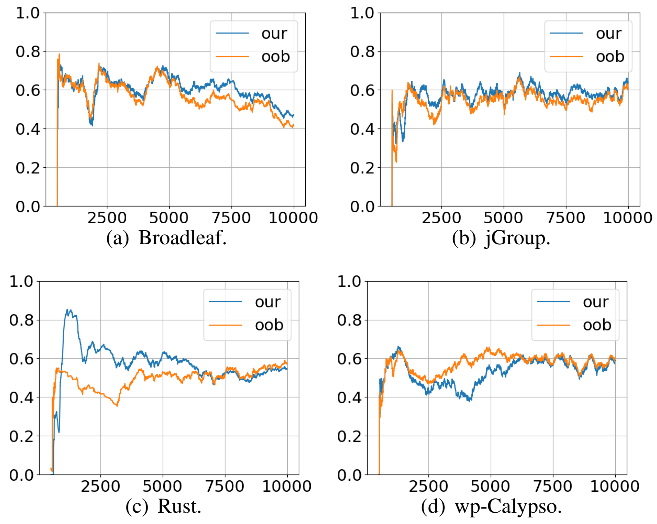


Fig. 2. Continuous performance in terms of G-mean throughout test steps. We choose some representative datasets for the space reason.

Figure 2 shows continuous performance throughout test time steps in terms of G-mean between OOB and ODaSC for some representative datasets. Other datasets were omitted due to space restrictions. ODaSC can constantly outperform OOB at most test time steps in datasets such as Brackets, Broadleaf, Corefx, Django, jGroup, Fabric, and Tensorflow. Two examples are shown in Figures 2(a) and 2(b). For some datasets such as Rust, Tomcat, Rails and Nova, the superiority of ODaSC against OOB can have large magnitude during some periods of time. An example of that is given in Figure 2(c). Only for wp-Calypso, ODaSC performed worse than OOB between test step 1,800$\sim$6,000, as illustrated in Figure 2(d), due to worse Recall-1 of ODaSC in that time span.

C. *ODaSC Gets Better Recall-1 by Sacrificing Recall-0*

This section aims to explore the reason why ODaSC usually got better G-mean than OOB. Table III reports the overall predictive performance in terms of Recall-1 and Recall-0. In terms of Recall-1, Wilcoxon signed rank tests with Holm-Bonferroni correction at the significance level 0.05 across datasets rejects the null hypothesis (H0) that ODaSC and OOB perform similarly with p-value 6.70E-04. This means that ODaSC obtained better Recall-1 than OOB. In terms of Recall-0, the null hypothesis was rejected with p-value 3.76E-03. This means that OOB obtained better Recall-0 than ODaSC.

Nevertheless, we can also find that the superior Recall-1 of ODaSC usually had differences above around 20% and sometimes can be more than 40% with respect to OOB, e.g., difference of 43.33% in Rails, 51.58% in Rust and 71.42% in Tomcat. Meantime, the inferior Recall-0 of ODaSC had differences below 15%, such as 11.83% in Rails, 7.73% in Rust and 15.95% in Tomcat with respect to OOB. This means that ODaSC usually achieved much better Recall-1 by

TABLE III
ODaSC vs OOB in terms of Recall-1 and Recall-0 across 30 runs. The last 3 rows report statistical tests across datasets. Significant difference against ODaSC is highlighted in yellow (light gray).

Project	Recall-1		Recall-0	
	ODaSC	OOB	ODaSC	OOB
Brackets	0.6416	0.5505	0.6879	0.7616
Broadleaf	0.4837	0.4097	0.7959	0.8445
Camel	0.6621	0.5409	0.6665	0.7900
Corefx	0.4371	0.3930	0.8473	0.8638
Django	0.6435	0.5119	0.7390	0.8501
Fabric	0.6390	0.5524	0.6768	0.7377
jGroups	0.5072	0.4056	0.6816	0.7644
Nova	0.6878	0.6217	0.6585	0.6431
Rails	0.4616	0.3221	0.7420	0.8415
Rust	0.4499	0.2968	0.7902	0.8564
Tensorflow	0.6634	0.5419	0.7187	0.8041
Tomcat	0.5517	0.3218	0.7114	0.8464
VScode	0.3919	0.3770	0.8799	0.9003
wp-Calypso	0.3368	0.4164	0.8657	0.8055
aveRank	1.07	1.93	1.86	1.14
p-value	6.70E-04		0.0038	

slightly sacrificing Recall-0, leading to better G-mean. This is understandable, as some examples with $y^* = 0$ will have their confidence reduced by ODaSC, making it less likely to predict examples belong to class 0. However, as ODaSC manages to estimate label confidence effectively, the reduction in Recall-0 is much smaller than the improvement in Recall-1.

VIII. Conclusions

We proposed a novel data stream learning approach to tackle one-sided label noise resulting from verification latency and evaluated it in the context of JIT-SDP by answering the two research questions as follows. **Answer to RQ1:** We proposed a method to estimate label confidence of training examples based on micro-clusters with label information and one-sided label noise. Experimental results demonstrate the effectiveness of our label confidence estimation method. **Answer to RQ2:** We proposed an online learning approach based on our label confidence estimation to tackle one-sided label noise in data streams. Experimental results show that the proposed ODaSC significantly outperformed the state-of-the-art OOB on real world JIT-SDP data streams.

Future work includes: (1) further evaluation of our ODaSC with data streams from other application domains and further investigation on the impact of the components of our approach; (2) extension of ODaSC to multi-class classification task; and (3) proposing mechanisms to cater for concept drift of the data stream whilst dealing with the one-sided label noise.

Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant No. 62002148), EPSRC (Grant No. EP/R006660/2), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), Research Institute of Trustworthy Autonomous Systems (RITAS), and a joint project between Huawei and Southern University of Science and Technology.

References

[1] G. Krempl and V. Hofer, "Classification in presence of drift and latency," in *ICDMW*, 2011, pp. 596–603.

[2] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[3] G. Cabral, L. Minku, E. Shihab, and S. Mujahid, "Class imbalance evolution and verification latency in just-in-time software defect prediction," in *ICSE*, Monteal, Canada, 2019, pp. 666–676.

[4] L. I. Kuncheva and J. S. Sanchez, "Nearest neighbour classifiers for streaming data with delayed labelling," in *ICDM*, 2008, pp. 869–874.

[5] K. Dyer, R. Capo, and R. Polikar, "Compose: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE TNNLS*, vol. 25, pp. 12–26, 2014.

[6] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *ICDM*, 2010, pp. 1175–1180.

[7] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi, "Credit card fraud detection: A realistic modeling and a novel learning strategy," *IEEE TNNLS*, vol. 29, no. 8, pp. 3784–3797, 2018.

[8] V. M. A. Souza, D. F., G. Batista, and J. Gama, "Classification of evolving data streams with infinitely delayed labels," in *ICMLA*, 12 2015, pp. 214–219.

[9] N. Cesa-Bianchi, S. Shalev-Shwartz, and O. Shamir, "Online learning of noisy data," *IEEE Trans. Inf. Theory*, vol. 57, no. 12, pp. 7907–7931, 2011.

[10] B. Frenay and B. Hammer, "Label-noise-tolerant classification for streaming data," in *IJCNN*, 2017, pp. 1748–1755.

[11] G. Oliveira, L. Minku, and A. Oliveira, "Tackling virtual and real concept drifts: An adaptive gaussian mixture model approach," *IEEE TKDE*, 2021.

[12] E. Moroshko and K. Crammer, "Online regression with controlled label noise rate," in *ECML-PKDD*, 2017, pp. 355–369.

[13] S. Lei, X. Zhang, L. Zhao, A. P. Boedihardjo, and C.-T. Lu, "Online and distributed robust regressions with extremely noisy labels," *ACM TKDD*, vol. 16, no. 3, 2021.

[14] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE TSE*, vol. 39, no. 6, pp. 757–773, 2013.

[15] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *IEEE TSE*, vol. 44, no. 5, pp. 412–428, 2018.

[16] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalance data," in *ICSE*, 2015, pp. 99–108.

[17] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *ESE*, vol. 21, no. 5, pp. 2072–2106, 2016.

[18] S. Wang, L. L. Minku, and X. Yao, "Resampling-based ensemble methods for online class imbalance learning," *TKDE*, vol. 27, no. 5, pp. 1356–1368, 2015.

[19] S. Tabassum, L. L. Minku, D. Feng, G. G. Cabral, and L. Song, "An investigation of cross-project learning in online just-in-time software defect prediction," in *ICSE*, 2020, p. 554–565.

[20] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *VLDB*, 2003, pp. 81–92.

[21] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *ICDM*, 2006.

[22] S. Wang, L. L. Minku, and X. Yao, "A systematic study of online class imbalance learning with concept drift," *IEEE TNNLS*, vol. 29, no. 10, pp. 4802–4821, 2018.

[23] P. Domingos and G. Hulten, "Mining high-speed data streams," in *ACM SIGKDD*, 2000, p. 71–80.

[24] N. Oza, "Online bagging and boosting," in *IEEE SMC*, vol. 3, 2005, pp. 2340–2345.

[25] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: analytics and risk prediction of software commits," in *FSE*, 2015, pp. 966–969.

[26] J. Gama, R. Sebastiao, and P. P. Rodrigues, "On evaluating stream learning algorithms," *JML*, vol. 90, no. 3, pp. 317–346, 2013.

[27] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *JML Research*, vol. 19, no. 72, pp. 1–5, 2018.

[28] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *JMLR*, vol. 7, pp. 1–30, 2006.