Cross-project online just-in-time software defect prediction

Tabassum, Sadia; Minku, Leandro; Feng, Danyi

DOI:
10.1109/TSE.2022.3150153

Document Version
Peer reviewed version

Citation for published version (Harvard):
Tabassum, S, Minku, L & Feng, D 2022, 'Cross-project online just-in-time software defect prediction', *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2022.3150153

Cross-Project Online Just-In-Time Software Defect Prediction

Sadia Tabassum, Leandro L. Minku, *Senior Member, IEEE,* Danyi Feng

Abstract—Cross-Project (CP) Just-In-Time Software Defect Prediction (JIT-SDP) makes use of CP data to overcome the lack of data necessary to train well performing JIT-SDP classifiers at the beginning of software projects. However, such approaches have never been investigated in realistic online learning scenarios, where Within-Project (WP) software changes naturally arrive over time and can be used to automatically update the classifiers. We provide the first investigation of when and to what extent CP data are useful for JIT-SDP in such realistic scenarios. For that, we propose three different online CP JIT-SDP approaches that can be updated with incoming CP and WP training examples over time. We also collect data on 9 proprietary software projects and use 10 open source software projects to analyse these approaches. We find that training classifiers with incoming CP+WP data can lead to absolute improvements in G-mean of up to 53.89% and up to 35.02% at the initial stage of the projects compared to classifiers using WP-only and CP-only data, respectively. Using CP+WP data was also shown to be beneficial after a large number of WP data were received. Using CP data to supplement WP data helped the classifiers to reduce or prevent large drops in predictive performance that may occur over time, leading to absolute G-Mean improvements of up to 37.35% and 48.16% compared to WP-only and CP-only data during such periods, respectively. During periods of stable predictive performance, absolute improvements were of up to 29.03% and up to 41.25% compared to WP-only and CP-only classifiers, respectively. Our results highlight the importance of using both CP and WP data together in realistic online JIT-SDP scenarios.

Index Terms—Software defect prediction, cross-project learning, transfer learning, online learning, verification latency, concept drift.

1 Introduction

The primary objective of software quality assurance activities is to reduce the number of defects in software products [2]. It is a challenging problem considering the limitation of budget and time allocation for such activities. Software Defect Prediction (SDP) helps to reduce the time and effort required for testing software products by predicting which parts of the software are more likely to contain defects. Different machine learning approaches have been proposed for SDP [3]. Many studies have focused on identifying defect prone components (e.g., modules or files) [4]. Recent studies have been increasingly focusing on identifying defect-inducing software changes. This is known as Just-In-Time Software Defect Prediction (JIT-SDP) [5]. Advantages of JIT-SDP over component level SDP include [5]: (1) prediction made at an early stage, facilitating code inspection, (2) finer granularity of the predictions, making it easier to find defects, and (3) straightforward allocation of developers to inspect the code.

Similar to SDP at the module/file level, JIT-SDP classifiers require sufficient amount of training data which is not available during the initial phase of a project. To overcome this problem, previous work has proposed Cross-Project (CP) JIT-SDP, where historical data from other projects are used to train the classifier [6]. Existing CP JIT-SDP work [6] assumes an offline learning scenario, where classifiers are built based on a pre-existing training set and never get updated after that. This means that the CP classifier is never trained on Within-Project (WP) data. However, in practice, JIT-SDP is an online learning problem [7], i.e., a problem where both additional CP and WP training examples arrive over time, forming a data stream that can be used to update JIT-SDP models whenever a new labelled software change arrives.

The role of CP data in such online learning scenario is unclear. CP JIT-SDP has never been investigated in online mode before. In particular, it is unknown whether CP data is only helpful at the very early stages of the project when there is little WP data, or if it brings a prolonged benefit to the predictive performance of the classifier. For instance, it may be that CP approaches using an augmented training data stream formed by both WP and CP examples lead to increased predictive performance even at later stages of the project, given that classifiers are built using more data than WP classifiers trained only with WP data. Or, it may be that CP data cause such approaches to obtain worse predictive performance than WP classifiers once enough WP data is used for training. Besides, prediction quality can fluctuate due to variations (concept drifts) in the underlying defect generating process [7], [8], rendering JIT-SDP classifiers unreliable. The use of CP training data could potentially help to handle concept drift. This is specially the case considering that JIT-SDP is a class imbalance problem, where the number of defect-inducing software changes is typically much smaller than that of clean changes. In such kind of problem, it would take a lot of time to collect new WP defect-inducing examples to recover well from concept drift. CP training examples could potentially help to recover from concept drift more quickly.

This paper thus aims at investigating when and to what

A preliminary version of this paper appeared in [1].

- *S. Tabassum and L.L. Minku are with the School of Computer Science, University of Birminigham, UK, emails: sxt901@cs.bham.ac.uk and L.L.Minku@bham.ac.uk.*
- *D. Feng is with Xiliu Tech, China, email: danyi@ouchteam.com.*
- *Corresponding author: L.L. Minku.*

- *This work was funded by EPSRC Grant No. EP/R006660/2.*

extent CP JIT-SDP data can be helpful in a realistic online learning scenario. It answers the following Research Questions (RQs):

RQ1 Can CP data help to improve predictive performance in the initial phase of a project, when there is little or no WP training data available? For how long and to what extent?

RQ2 Can CP data help to prevent sudden drops in predictive performance, which may be caused by concept drift? To what extent?

RQ3 What are the effects of CP data on the predictive performance during stable periods of the projects? Could the CP data be detrimental during stable periods, given that enough WP data may be available for training?

RQ4 Is it necessary to update CP approaches over time once a project starts, or is it enough to use models trained only with CP data produced before the project being predicted commences? If it is enough, online learning is not really necessary.

In online scenarios, both WP and CP training data arrive over time as their collection can be automated. If incoming CP data is used for updating a CP classifier over time, it is natural and reasonable to also use any WP data that may arrive over time. This is because such WP data should not hurt (and could potentially improve) predictive performance. Therefore, in online learning, we use the term CP when referring to approaches that make use of both incoming CP and WP data for training. This term is adopted with this meaning in our RQs and throughout the rest of this paper. This means that, for online learning scenarios, when we refer to the benefits of CP data, we do not mean the benefits of CP data in isolation, but the benefits of CP data used along with incoming WP data. Indeed the term CP means Cross-Project, i.e., data coming from multiple projects, and thus there is no reason for this term to exclude WP data in the context of online learning. When discussing realistic online learning scenarios in this paper, we will use the term Other-Project (OP) to refer to approaches that use data that come only from other projects than the project we are interested in. In this paper, these approaches are CP approaches trained only on CP data that are available before the project of interest commences, i.e., they are CP approaches that have no access to WP data. We use the term WP when referring to approaches that only use WP data.

To answer our RQs, we investigate four approaches: (1) a single online learning CP classifier trained on incoming WP and CP training examples, (2) an online learning CP ensemble where each classifier is trained on incoming training examples from a different project, (3) a single online learning CP classifier that filters out CP training examples that are likely to be very different from recent WP examples and (4) an OP classifier that is trained only with CP training data which arrive before the project of interest commences, and is never updated with incoming WP data. The first 3 approaches are online and are enhancements of the approaches used in the JIT-SDP literature [6], so that they can operate in online mode. The last approach represents CP approaches from the offline learning literature, which make use of only OP data. All of these approaches are compared against online WP [7], [9] approaches. Our experiments based on 9 proprietary and 10 open source software repositories show that the first and third approaches are helpful to improve predictive performance in JIT-SDP compared to WP classifiers, while the second and fourth are not.

Overall, our contributions are the following:

- We provide the first investigation of CP JIT-SDP in a realistic online learning scenario.
- We show how to adapt CP JIT-SDP approaches so that they can be used in an online learning scenario.
- We develop a new methodology to systematically analyse approaches during periods of drop in predictive performance (and conversely also periods of stability) in online learning. This enables us to know how much more robust CP JIT-SDP approaches are in terms of preventing such drops (and thus leading to more reliable predictions over time) compared to WP JIT-SDP approaches.
- By answering RQ1-4, we provide a detailed analysis revealing that the extent to which CP can be helpful in JIT-SDP is much larger than previously thought. It can not only provide significant improvements in predictive performance during the initial stages of a project (RQ1), but also can prevent drops in predictive performance over time (RQ2) and improve predictive performance during stable periods (RQ3).
- We show that online CP approaches outperform OP approaches (RQ4), highlighting the benefits of online learning in JIT-SDP.
- We show that it is better to use CP and WP data together to build a single classifier, rather than building different classifiers with disjoint subsets of the data.
- We provide an analysis of sensitivity to hyperparameters for the most promising CP approach (Filtering), providing an understanding of their effect on predictive performance.

This paper is further organized as follows. Section 2 presents related work. Section 3 introduces our online CP JIT-SDP approaches. Section 4 presents the investigated datasets. Section 5 explains the experimental setup for answering the RQs. Section 6 explains the results of the experiments. Section 7 presents the sensitivity analysis. Section 8 presents the analysis of computational cost. Section 9 explains the implications to practice. Section 10 explains the threats to validity. Section 11 presents the conclusions and future work.

2 Related Work

There are many SDP studies [3], [10], [11], including recent studies investigating class imbalance techniques [12], automated feature engineering [13], ensemble learning [14], among others. In this section, we discuss three main research areas of SDP that are closely related to this work: CP SDP at the component level (Section 2.1), CP JIT-SDP (Section 2.2) and Online JIT-SDP (Section 2.3).

2.1 CP SDP at the Component Level

There have been several studies on CP SDP at the component level. An initial study provided guidelines for choosing

training projects [15]. It proposed an approach to identify factors that influence CP prediction success, such as data and process factors. Another study [16] showed that carefully selected CP training data may provide better prediction results than training data from the same project. Peters et al. [17] focused on selecting suitable CP training data based on the similarities between the distribution of the test and potential training data. They used a similarity metric and feature subset selection to remove irrelevant training data. Canfora et al. [18] proposed a multi-objective approach for CP defect prediction. They attempted to achieve a compromise between amount of code to inspect and number of defect-prone artifacts. This approach performed better than WP models. Panichella et al. [19] analysed the equivalence of different defect predictors and proposed a combined approach CODEP (COmbined DEfect Predictor) that uses machine learning to combine different and complementary classifiers. This combination performed better than the stand-alone CP technique. Nam et al. [20] applied Transfer Component Analysis (TCA) to CP SDP. TCA is a transfer learning approach that maps the data to a common latent space where CP and WP data are similar to each other. They also proposed a new approach called TCA+, which selects suitable normalisation options for TCA. Other studies [21], [22], [23] consider class imbalance learning for CP SDP. For instance, Ryu et al. [21] proposed an approach that uses similarity weight drawn from distributional characteristics and the asymmetric misclassification cost to balance imbalanced distributions.

Overall, these studies demonstrate that data distributional characteristics are important for CP SDP. They proposed approaches to select CP data that are similar to WP data, or to map CP and WP data into a latent space where they are similar. However, none of these studies were in the context of JIT-SDP or online SDP.

2.2 CP JIT-SDP

The first CP JIT-SDP study [6] carried out an empirical evaluation of CP JIT-SDP performance by using data from 11 open source projects. They investigated five CP JIT-SDP approaches based on project similarity, three variations of data merging approaches, and ensemble approaches where each model was trained on data from a different project. All approaches were based on random forests as base learners. They found that simple merging of all CP data into a single training set and ensemble approaches obtained similar predictive performance to that of WP models. Different from SDP at the component level, other more complex approaches, including similarity-based approaches, did not offer any advantage compared to these.

Another study [24] investigated CP JIT-SDP in mobile platforms using 14 apps extracted from the Commit.Guru platform [25]. They compared the CP performance of four different well-known classifiers and four ensemble techniques. Naive Bayes performed best compared to other classifiers and some ensemble techniques. They did not check how CP compared against WP results.

Chen et al. [26] considered JIT-SDP as a multi-objective problem to maximise the number of identified defect-inducing changes while minimising the effort required to fix the defects. They proposed a multi-objective optimization-based supervised method called MULTI to build logistic regression JIT-SDP models. They used six open source projects. MULTI was evaluated on three different model performance evaluation scenarios (cross-validation, cross-project-validation, and timewise-cross-validation) against 43 state-of-the-art supervised and unsupervised methods. They found that it can perform significantly better than WP methods.

Zhu et al. [27] proposed a JIT-SDP approach named DAECNN-JDP based on denoising autoencoder and convolutional neural network. WP and CP defect prediction experiments were performed on six large open source projects and DAECNN-JDP was compared with 11 baseline models, including eight machine learning models, EALR, Deeper and CNN-JDP. The results show that DAECNN-JDP achieved better predictive performance than the baseline models for both CP and WP defect prediction. However, the predictive performance of CP and WP approaches was not compared against each other.

Despite showing that CP JIT-SDP can obtain promising results compared to WP JIT-SDP, none of the studies above considered a realistic online learning scenario, where CP and WP training data arrive over time.

2.3 Online JIT-SDP

Some studies have highlighted the importance of respecting chronology in component-level SDP [28], [29]. However, very few studies explored JIT-SDP in online mode. Tan et al. [30] investigated JIT-SDP in a scenario where new batches of training examples arrive over time and can be used for updating the classifiers, using one proprietary and six open source projects. They considered the fact that the labels of training data may arrive much later than the commit time. This is known as *verification latency* [31] and is important to be taken into account in realistic scenarios. They used resampling techniques to deal with the class imbalanced data issue and updatable classification to learn over time. However, their approach assumes that there is no concept drift, i.e., that the defect generating process does not suffer variations over time.

McIntosh et al. [8] performed a longitudinal case study of 37,524 changes from the rapidly evolving QT and OPENSTACK systems and found that fluctuations in the properties of fix-inducing changes can impact the performance of JIT models. They showed that JIT models typically lose predictive power after one year, possibly as a result of concept drift. Hence, the JIT model should be updated with more recent data.

Cabral et al. [7] proposed a method called Oversampling Rate Boosting (ORB) to tackle class imbalance evolution in an online JIT-SDP scenario taking verification latency into account. Class imbalance evolution is a type of concept drift where the proportion of defect-inducing and clean examples fluctuates over time. ORB has an automatically adjustable resampling rate to tackle class imbalance evolution, being able to improve predictive performance over JIT-SDP approaches that assume a fixed level of class imbalance.

The existing online JIT-SDP studies build JIT-SDP models that are trained only on the WP data. They do not use

any CP software changes for training. None of the online JIT-SDP studies above investigated CP JIT-SDP.

2.4 Online CP JIT-SDP

There is no previous work on online CP JIT-SDP except for a preliminary version of this paper, which was published in [1]. The key additions to the current paper compared to that one [1] are the following:

- The investigation of RQ2 was previously based on a visual analysis of the predictive performance plots, being subjective. We have now developed a new methodology to systematically and quantitatively analyse whether a given approach can help to prevent large drops in predictive performance that may occur over time. Therefore, the whole analysis of RQ2 has been re-done.
- The new methodology above also enables us to distinguish between periods of sudden drops and periods of stability of predictive performance. This is important because preventing drops is not so helpful if this results in poor performance during stable periods. We can now systematically investigate the predictive performance of the approaches also during stable periods, leading to the new RQ3.
- Existing CP JIT-SDP literature assumes that CP data used to train JIT-SDP classifiers are the data collected prior to the beginning of the software project of interest. If such OP data is enough to improve predictive performance over time in JIT-SDP, further training with CP data would be unnecessary, enabling companies to save resources. However, OP approaches were not investigated in the previous paper. We have now added RQ4 for that.
- We provide a sensitivity analysis of hyperparameters for the CP approach that achieved the most promising results. This sensitivity analysis was not available in the previous work and gives insights into how to choose hyperparameter values for this approach.
- We provide an analysis of the computational cost of the approaches, which was not available in the previous work.
- The previous study was based on 3 proprietary data sets and 10 open source projects. We have now collected more 6 proprietary data sets, making this the largest JIT-SDP study to date in terms of the number of proprietary software repositories. This improves the external validity of our study.

3 Online CP JIT-SDP Approaches

In this section, we modify and enhance three CP JIT-SDP approaches adopted in [6] to enable them to be applied to online JIT-SDP. All our algorithms fully respect chronology. In particular, they never use future CP/WP training examples, future knowledge about labels (i.e., defect-inducing or clean), or test examples, for training a model used for testing the present. Our CP approaches can be trained not only with CP and WP training examples that are made available over time after the first WP commit, but also with CP training examples produced before the first WP commit.

Training examples are generated using the online procedure recommended by Cabral et al. [7] to take verification latency into account for all approaches studied in this paper. In particular, a software change becomes a training example either when a defect is found to be associated to it, or once a pre-defined waiting period of w days has passed, whichever is earlier. This waiting period represents the amount of time that it takes for one to be confident that the software change in question is clean. In other words, if no defect is found to be associated to the software change during the waiting period, a training example of the clean class is created to represent this software change. Otherwise, a training example of the defect-inducing class is created immediately after the defect is found. If, after the waiting period, a defect is found to be associated to a change that was previously considered clean, a new defect-inducing training example is created for it. Training examples are used to update the classifier as soon as they are created.

3.1 All-in-One (AIO) Approach

Existing offline JIT-SDP work [6] assumes that CP classifiers are not trained with any WP data. Unlike offline approaches, the AIO online CP approach includes WP data for training. All incoming CP and WP training data are considered as part of a single data stream of training examples, which are used to train a single online classifier as soon as they are produced. The data stream is in chronological order, i.e., the training examples are sorted based on the Unix timestamp of their author creation. A new incoming change that does not belong to the target project is used for training (respecting the chronology and verification latency), but not for prediction. Algorithm 1 shows the pseudocode for the AIO approach.

The classifier is initialised as an empty model that always predicts "clean". When a new incoming change x_p^t is received at time step t (line 3), the algorithm first checks if this change belongs to the target project, i.e., to the project whose changes are being predicted (line 4). If it does, a prediction is provided for this change. After that, x_p^t is stored in a queue for a pre-defined waiting period (line 7). All software changes in this queue are checked to see whether they can be used for training (line 8 to 21). If a defect is found to be associated to a given software change in the queue during the waiting period (lines 9 to 12), a defect-inducing training example is created to represent this change and is used for training. If a defect is not found by the end of the waiting period of a given change (lines 13 to 20), a clean training example is created for this change and is used for training. After that the change is removed from the queue. The algorithm also checks whether there is any past change found to be defect-inducing, but that was previously considered as a clean training example (lines 22 to 26). If there is, the classifier is trained using that change as a defect-inducing training example.

The key difference between our approach and the data merging approaches used by Kamei et al. [6] is that their learning was offline (without taking into account incoming training examples and verification latency) and they did not include WP data for training. In our proposed approach, the learning is online, takes verification latency into account,

and the classifier is trained on both CP and WP data whose labels are produced before the current timestamp.

3.2 Ensemble Approach

The Ensemble approach uses an ensemble of classifiers rather than a single classifier. A separate classifier is built from each project's separate training data stream (e.g., for 10 projects there will be 10 different classifiers). This includes both CP and WP data streams. Each change belonging to the target project is then predicted by all the classifiers, and the mean of the predicted probabilities retrieved by the classifiers is calculated. This mean is used to predict whether the change is clean or defect-inducing. The pseudocode for the Ensemble approach is similar to that of the AIO approach, and can be found in the supplementary material [32]. As with the AIO approach, the chronological order of the training examples is always respected.

The predictions given by the ensemble are based on simple voting. The key differences between our approach and the approach used by Kamei et al. [6] are that our approach is online, and our ensemble contains a classifier built from WP training examples that have been labelled up to the current timestamp, rather than using only CP data classifiers.

3.3 Filtering Approach

We propose the following Filtering approach for online CP JIT-SDP. First, a fixed-size window of most recent incoming WP training examples is maintained. Whenever a CP training example arrives, it is compared with the training examples in the WP window. As with the AIO and Ensemble approaches, the chronological order of the training examples is always respected. Euclidean distances between the input features of the CP training example and each of the WP training examples in the window that have the *same label* as the CP training example are calculated to check how similar the CP training example is to recent WP examples. All input features used to describe the software changes in the data stream are used for calculating the Euclidean distance. It is important to use only training examples with the same label to compute the distance. Otherwise, the approach would consider that a CP clean training example described by similar features as a WP defect-inducing training example are similar training examples. However, they are different due to the different label.

The average of the smallest K distances is calculated. If this average distance is equal to or lower than a maximum threshold, the CP training example is allowed to train the classifier. Discarded CP training examples are kept in a fixed-sized queue. This queue is checked in every iteration to see whether any old discarded CP training example has now become suitable for training. This can be useful in case concept drift causes such discarded examples to become relevant at some point.

Algorithm 2 shows the pseudocode for the Filtering approach. The classifier is initialised as an empty model that always predicts "clean". When a new incoming change x_p^t from project p is received at time step t (line 3), the algorithm checks whether there are any old CP training examples that were previously not used for training due to their dissimilarity to WP examples, but that are now suitable for training due to their similarity to the current WP sliding window (lines 4–7). If there are, they are used for training. Then, a prediction is given if the change x_p^t belongs to target project (lines 8–10). The change x_p^t is then stored in a queue (line 11), waiting to be labelled. All changes in this queue are checked to see whether they can be labelled (lines 12 to 30). If they can, corresponding training examples are created and used for training only if they are similar enough to the WP sliding window (lines 21–23). If they are not similar enough and are CP examples, they are stored in the queue $CP\text{-}Q$ of discarded CP examples for possible future use (line 25). The sliding window is updated (slided) if the training example is WP (line 28). The algorithm also checks whether any change that was previously considered as clean has now been associated to a defect and uses it for training, but only if it is similar enough to the WP window (lines 31–39).

Algorithm 1 All-in-One (AIO) Approach

Input: S = stream of incoming changes from several projects, b = index identifying the target project, w = waiting period

1: Initialise predictive model m
2: **for** each incoming change $x_p^t \in S$ **do** // x_p^t is a change arriving from project p at timestamp t
3: **if** $p = b$ **then**
4: $\hat{y} \leftarrow$ predict(m, x_p^t)
5: **end if**
6: store x_p^t in a queue $WFL\text{-}Q$ // $WFL\text{-}Q$ is a queue of incoming examples waiting to be used for training
7: **for** each item q^i in $WFL\text{-}Q$ **do**
8: **if** a defect was linked to q^i at a timestamp $\leq t$ **then**
9: create a defect-inducing $training\_example$ for q^i
10: train(m, $training\_example$)
11: remove q^i from $WFL\text{-}Q$
12: **else**
13: **if** q^i is older than w **then**
14: create a clean $training\_example$ for q^i
15: train(m, $training\_example$)
16: remove q^i from $WFL\text{-}Q$
17: store $training\_example$ in $CL\text{-}H$ // $CL\text{-}H$ is a hash of clean training examples
18: **end if**
19: **end if**
20: **end for**
21: **if** a defect was linked to a $training\_example$ in $CL\text{-}H$ at a timestamp $\leq t$ **then**
22: Swap the label of $training\_example$ to defect-inducing
23: train(m, $training\_example$)
24: remove $training\_example$ from $CL\text{-}H$
25: **end if**
26: **end for**

Algorithm 2 Filtering Approach

Input: S = stream of incoming changes from several projects, b = index identifying the test project, w = waiting period, windowSize = size of the WP-Q sliding window, K = number of top short distances to be used, $maxDist$ = distance threshold for similarity, $cpqSize$ = maximum size of the queue CP-Q of dissimilar CP instances to be rechecked for similarity in the future

Initialise predictive model m
for each incoming change $x_p^t \in S$ **do** // x_p^t is a change arriving from project p at timestamp t
  if avgDist($training\_example$, WP-Q,K) $\leq$ $maxDist$ for any $training\_example$ in CP-Q **then**
    train(m, $training\_example$)
    remove $training\_example$ from CP-Q
  end if

  if $p = b$ **then**
    $\hat{y}$ = predict(m, x_p^t)
  end if
  store x_p^t in a queue WFL-Q // WFL-Q is a queue of incoming examples waiting to be used for training

  for each item q^i in WFL-Q **do**
    if a defect was linked to q^i at a timestamp $\leq t$ **then**
      Create a defect-inducing $training\_example$ for q^i
    else
      if q^i is older than w **then**
        Create a clean $training\_example$ for q^i
        store q^i in CL-H // CL-H is a hash of clean training examples
      end if
    end if
    if a $training\_example$ was created for q^i **then**
      if avgDist($training\_example$, WP-Q,K) $\leq$ $maxDist$ or this is a WP change **then**
        train(m, $training\_example$)
      else
        Add $training\_example$ to CP-Q
      end if
      Remove q^i from WFL-Q
      Slide WP-Q if $training\_example$ is WP
    end if
  end for

  if a defect was linked to a $training\_example$ in CL-H before time t **then**
    swap label of $training\_example$ to defect-inducing
    remove $training\_example$ from CL-H
    if avgDist($training\_example$,WP-Q,K) $\leq$ $maxDist$ **then**
      train(m, $training\_example$)
    else
      add $training\_example$ to CP-Q
    end if
  end if
end for

4 Datasets

We have extracted data from nine proprietary software development project repositories from a Chinese software development company for the purpose of this study. The proprietary data collection used the same version of Commit Guru prepared for Chinese language [33] as in [1]. A description of the procedure used to prepare Commit Guru to collect data from Chinese software development companies can be found in [1]. We have also used ten existing datasets extracted from open source GitHub projects based on Commit Guru [25], which were made available by Cabral et al. [7].

Some information on the datasets is shown in Table 1. The input features include 14 change metrics: NS (number of modified subsystems), ND (number of modified directories), NF (number of modified files), Entropy (distribution of modified code across each file), LA (lines of code added), LD (lines of code deleted), LT (lines of code in a file before the change), FIX (whether or not the change is a defect fix), NDEV (number of developers that changed the modified files), AGE (average time interval between the last and the current change), NUC (number of unique changes to the modified files), EXP (developer experience), REXP (recent developer experience) and SEXP (developer experience on a subsystem). These change metrics have been shown to be adequate for JIT-SDP in previous work [5] and have been adopted in previous online JIT-SDP work [7], [8].

All software changes were chronologically ordered based on the author timestamp, which is provided by Git as the timestamp when the commit was first created by the author. Author timestamp is recommended over commiter timestamp for studies involving time-based Git data [34]. This timestamp is used in our paper as the moment in time when a JIT-SDP model is required to provide a prediction for this software change. The timestamp when a software change is found to be defect-inducing is the author timestamp of the commit that fixes this software change, as retrieved by Commit Guru [25]. Such timestamps and the waiting period w are used to determine when the training examples become available for training, following the procedure [7] explained in the beginning of Section 3. In this way, both the moment when software changes are predicted and the moment when software changes become available for training fully respect chronology. Commit guru uses git log to get the commits. Git log lists all commits in the current branch. If another branch is merged into it, the commits from that other branch will also show, and are chronologically ordered with the other commits based on author timestamp.

5 Experimental Setup

This section explains the experimental setup for answering the RQs introduced in Section 1. A replication package is available at https://zenodo.org/badge/latestdoi/455513474.

5.1 Compared Approaches

RQ1-3 will be answered by comparing the predictive performance of the online CP approaches presented in Section 3

TABLE 1: An overview of the projects

Project	Total Changes	# Defect-inducing Changes	% Defect-inducing Changes	Median Defect Discovery Delay (days)	Time Period	Main Language	Project Type
Tomcat	18907	5207	27.54	200.5798	27-03-2006 - 06-12-2017	Java	Open source [7]
JGroups	18325	3153	17.21	116.1565	09-09-2003 - 05-12-2017	Java	Open source [7]
Spring-integration	8750	2333	26.66	415.1201	14-11-2007 - 16-01-2018	Java	Open source [7]
Camel	30575	6255	20.46	28.1947	19-03-2007 - 07-12-2017	Java	Open source [7]
Brackets	17364	4047	23.31	14.454	07-12-2011 - 07-12-2017	JavaScript	Open source [7]
Nova	48989	12430	25.37	88.5615	28-05-2010 - 28-01-2018	Python	Open source [7]
Fabric8	13106	2589	19.75	39.1833	13-04-2011 - 06-12-2017	Java	Open source [7]
Neutron	19522	4607	23.6	82.5097	01-01-2011 - 27-12-2017	Python	Open source [7]
Npm	7920	1407	17.77	111.514	29-09-2009 - 28-11-2017	JavaScript	Open source [7]
BroadleafCommerce	15010	2531	16.86	42.5818	19-12-2008 - 21-12-2017	Java	Open source [7]
C1	1030	confidential	confidential	18.17	06-09-2018 - 17-07-2019	Javascript	proprietary
C2	601	confidential	confidential	1.93	16-10-2018 - 19-07-2019	Javascript and 3D studio	proprietary
C3	417	confidential	confidential	2.98	05-09-2018 - 19-07-2019	Python	proprietary
C4	341	confidential	confidential	4.13	17-04-2019 - 17-10-2019	Javascript and 3D studio	proprietary
C5	323	confidential	confidential	4.24	22-05-2019 - 13-11-2019	Javascript and 3D studio	proprietary
C6	555	confidential	confidential	2.92	13-05-2019 - 13-12-2019	Javascript and 3D studio	proprietary
C7	546	confidential	confidential	3.33	16-04-2019 - 18-11-2019	Javascript and 3D studio	proprietary
C8	798	confidential	confidential	0.81	22-05-2019 - 16-09-2019	Python	proprietary
C9	694	confidential	confidential	0.77	30-04-2019 - 21-08-2019	Python	proprietary

against that of online WP approaches, providing a detailed understanding of when and to what extent CP data can be helpful to improve predictive performance in JIT-SDP. Comparisons of online CP against OP approaches will be used to answer RQ4, revealing whether online learning is really helpful in JIT-SDP. As with the online CP approaches, the WP and OP approaches take verification latency into account following Cabral et al. [7]'s waiting time procedure and always respect chronology. A dummy classifier that predicts clean or defect inducing uniformly at random is also used for a preliminary investigation.

The online WP approaches are approaches able to learn WP training examples over time, but learn no CP data. The OP approaches represent the core CP strategies adopted in the offline CP JIT-SDP literature [6]. They do not use WP data for training, except for when filtering is adopted, in which case WP data are only used to decide which CP data to filter out from the training process. As existing CP JIT-SDP work does not adopt online learning, its corresponding approaches would only train JIT-SDP models with CP data produced prior to the development of the project of interest. Therefore, this is also the case for the OP approaches investigated in this study. Both an AIO and a Filtering OP approach are adopted. The OP AIO uses all the OP data to train the same predictive model, as in [6]. To ensure fair comparisons for the purpose of answering RQ4, the OP Filtering approach makes use of the same filtering mechanism explained in Section 3.3.

For simplicity, when referring to the CP approaches AIO and Filtering presented in Section 3, we will omit the term "CP". When referring to the OP AIO and Filtering approaches, we will make use of the term "OP" explicitly.

Given an open source repository corresponding to a project of interest, for CP approaches, all 10 open repositories are considered as the CP data. For the OP approaches, the other 9 open source repositories are considered as the OP data. Given a proprietary repository, the AIO approaches were investigated in two different ways: (a) combining both 10 open source and the 9 proprietary repositories as the CP data and (b) only with the 9 proprietary repositories as the CP data. For OP AIO, cases similar to (a) and (b) were used, but the OP data excluded the data from the project of interest. CP and OP Filtering approaches were investigated only in the scenario (a), again with the OP data excluding the data from the project of interest. They were not investigated using only proprietary data because filtering reduces the amount of training data by filtering dissimilar instances, and the amount of proprietary data is already small compared to the amount of open source data. Hence, using only proprietary data would likely be insufficient to train models for Filtering approaches. In addition, due to the poor results obtained by the Ensemble approach on the open source data, this approach was not run for the proprietary data.

JIT-SDP is a class imbalance problem [5], [6], [7], [30], and the open source data used in this study are known to be affected by this issue [7]. Therefore, learning approaches need to use online learning classifiers that can deal with that. Two state-of-the-art approaches for online class imbalanced learning are Improved Oversampling Online Bagging (OOB) and Improved Undersampling Online Bagging (UOB) [9]. Also, Cabral et al. [7] proposed a new approach called Oversampling Rate Boosting (ORB), which improves the predictive performance further in the context of WP JIT-SDP. The three approaches are ensembles of Hoeffding Trees [35]. These are online learning base classifiers, which are updated incrementally with each new training example, preserving old knowledge without requiring storage of old examples. Previously, OOB and UOB achieved similar performance in WP JIT-SDP [7], and ORB performed the best. Thus, only OOB and ORB are selected as the base classifiers for the approaches investigated in our study.

5.2 Predictive Performance

We define a time step as a sequential number indicating the order of WP commits according to their author timestamp. Each WP commit requires a WP software change to be predicted as defect-inducing or clean. Therefore, to evaluate predictive performance, we investigate how well a classifier can predict the WP commit received at each time step. At this time step, the online CP and WP classifiers used to give a prediction will have been updated with all CP+WP and WP labelled training examples that became available *prior* to the Unix timestamp corresponding to this time step, respectively. Therefore, they correspond to classifiers that could really have been used in practice to make such prediction. The OP approach always uses the OP classifier that was created based on all the labelled CP training examples

that were available prior to the beginning of the project being predicted. The analyses done for RQ1, RQ2 and RQ3 will concentrate on the predictive performance (1) in the beginning of the projects, (2) during periods of time where we can observe sudden drops in predictive performance of the WP approach and (3) during periods of stable predictive performance of the WP approach, respectively. The analysis for RQ4 will discuss all of (1-3). The procedure used to distinguish between (1), (2) and (3) is described in Section 5.3.

We adopt the Geometric Mean (G-Mean) of Recall0 and Recall1 as measures of predictive performance, where Recall0 is the recall on the clean class and Recall1 is the recall on the defect-inducing class. These metrics were computed prequentially and using a fading factor to enable tracking changes in predictive performance over time, as recommended for problems that may suffer concept drift [36]. If the current example belongs to class i, $Recall_i^{(t)} = \theta Recall_i^{(t-1)} + (1-\theta)\mathbb{1}_{\hat{y}=i}$, where i is zero or one, t is the current time step, θ is a fading factor set to 0.99 as in [7], $\hat{y}$ is the predicted class, and $\mathbb{1}_{\hat{y}=i}$ is the indicator function, which evaluates to one if $\hat{y}=i$ and to zero otherwise. If the current example does not belong to class i, $Recall_i^{(t)} = Recall_i^{(t-1)}$. Also, $G\text{-}Mean^{(t)} = \sqrt{Recall_0^{(t)} \times Recall_1^{(t)}}$. It is worth noting that $Recall_0 = 1 - FalseAlarmRate$, i.e., false alarms are taken into account through Recall0 and G-Mean.

These metrics were chosen because they are the most recently recommended for online class imbalance learning [9]. We opted for G-Mean instead of Area Under the ROC Curve (AUC). AUC has been discouraged in the context of software defect prediction [37] because it incorporates several threshold values that are not meaningful in practice, making the comparison among approaches difficult. Other metrics such as Matthews Correlation Coefficient (MCC) and F1-Score were not used because they are biased when there is class imbalance [9], [38]. Such bias is particularly problematic when the imbalance ratio is changing overtime as in the case of JIT-SDP [7]. This is because the performance metric would vary over time based on the current imbalance ratio, even if the quality of the predictions given by the classifier remains the same. Therefore, G-Mean is more adequate for our study.

5.3 Identifying Initial Periods and Periods of Sudden Drops in Predictive Performance

We define the initial phase of the open source projects as the period of time ranging from the first time step until the time step where the G-Mean of the WP approach reaches the value of its average G-Mean across time steps. It represents the time it takes for the G-Mean of this approach to reach its typical values for a given project. Knowledge of how long such period can be and how much it may vary from project to project can be useful to enable: (a) more informed decisions in terms of whether and when to trust the predictions given by a JIT-SDP model, and (b) to check which approaches can learn more quickly so as to present a better G-Mean during this initial stage.

For the proprietary data, the total number of time steps is too small to use the average G-Mean across time steps for this purpose. In particular, had longer periods of time been observed, the G-Mean values would be likely to improve further, given the trends in G-Mean at the end of the period analyzed for these projects (see Fig. 2, which will be discussed later in this section). Therefore, instead of using the average G-Mean across all time steps, we have used the average G-Mean across the last 40 time steps to determine the initial phase. We chose to fix the value of 40 so that enough time steps are being used for the calculation while ensuring that only the very last time steps are used, as they would be the time steps most likely to have increased predictive performance.

A systematic approach is applied to identify the periods of sudden drop in predictive performance. Its pseudocode is shown in the supplementary material [32]. First, the peaks in the $G\text{-}Mean^{(t)}$ (Section 5.2) obtained by the WP approach across time steps are identified for each of the datasets using a python module called 'Detecta' [39]. This module provides a function to detect or identify the change or occurrence of a particular event in the data, based on a threshold. Then, we start analysing the $G\text{-}Mean^{(t)}$ for each time step t. In particular, we compare $G\text{-}Mean^{(t)}$ against the simple average G-Mean obtained based on the WP approach's predictions over all software changes between the immediate previous peak identified by Detecta and the current time step t. If $G\text{-}Mean^{(t)}$ drops below 20% of the simple average G-Mean, then the time period from the immediate previous peak to the next peak following this time step is considered as a period of sudden drop in G-Mean. The threshold 20% was chosen after investigating several different threshold values. Smaller thresholds than 20% caused too many small drops in performance to be considered as sudden drops, whereas larger thresholds resulted in most sudden drops in performance to be missed. Examples of sudden drop periods identified by this systematic approach can be seen in the grey shaded areas of Figures 1 and 2. As we can see, the approach is able to identify periods of time where the G-Mean of the WP approach presents severe drops.

Considering only the sudden drop periods without knowing what happens in the periods of stable performance would give us incomplete insights. In particular, an approach that is able to prevent sudden drops in performance, but performs much worse during stable periods would not be so useful. Therefore, we also investigate periods of stable performance. Any period after the initial period excluding the sudden drop periods is considered as a stable period.

5.4 Statistical Tests and Effect Size

The predictive performances obtained during each period of the projects (initial phase, periods of sudden drop and stable periods) will be compared across data sets using the Scott-Knott procedure [40], which ranks the models and separates them into clusters. This test is used to select the best subgroup among different models. Non-parametric bootstrap sampling is used to make the test non-parametric, as recommended by Menzies et al. [41]. As explained by Demsar [42], non-parametric tests are adequate for comparison across data sets.

In addition, the Scott-Knott test adopted in this paper uses A12 effect size [43] to rule out any small differences in

performance. Specifically, Scott-Knott only performed statistical tests to check whether groups should be separated if the A12 effect size was not insignificant [41]. If the A12 effect size was insignificant, groups were *not* separated. We will refer to Scott-Knott based on Bootstrap sampling and A12 as Scott-Knott.BA12. Smaller Scott-Knott.BA12 rankings are better rankings.

We also report the A12 effect sizes against the WP approach for each dataset individually to support the analysis. Symbols [*], [s], [m] and [b] represent insignificant, small, medium and large A12 effect size. Presence/absence of the sign "-" in the effect size means that the corresponding approach was worse/better than the corresponding WP approach.

5.5 Parameter Tuning

The parameters for the Filtering approach were chosen by performing grid search on the initial portion (1000 commits and one-third commits respectively for open source and proprietary data) of the datasets using the following set of values: $windowSize$ ={500,600,700,1000}, number of top short distances tobe used K={50,100,200}, distance threshold for similarity $maxDist$={0.6,0.7,0.8} and maximum size of the queue of dissimilar CP instances $cpqSize$ = {500, 1000}. Two ORB parameters are also tuned by grid search: moving average window size={50, 100, 200} and n={3, 5, 7} for open source and proprietary data. The rest of the ORB parameters were kept to the same values as in [1] for open source datasets, as they have already been tuned for these datasets. However, for the proprietary datasets, OOB parameters were further tuned using the following set of values: ensemble={5,10,20,30,40} and theta={0.9,0.99,0.999}. We will also provide an analysis of sensitivity of the online CP approach that obtained the most promising results (Filtering) to its parameters in Section 7.

The waiting period was 90 days for the open source datasets as in [1] and 30 days for the proprietary datasets, due to their lower defect discovery delay (see Table 1). When open source data were used in combination with proprietary data, waiting period of 30 days was used for both types of datasets. Cabral et al. [7]'s study on WP JIT-SDP recommended to use waiting periods that are close to the expected median time to discover defects in the project. However, due to concept drift, it is advisable to use shorter waiting periods when some of the CP data have shorter defect discovery delay. This is because shorter waiting periods enable speeding up recovery from concept drift. The choice of waiting period may cause some defect-inducing commits to be initially labelled as clean and then relabelled to defect-inducing once a defect is found to be associated to them. However, once an example is relabelled, it is immediately used for training again. When that happens, the resampling rates adopted by OOB and ORB are likely to help emphasising the new defect-inducing label, given that the defect-inducing class is usually a minority class.

Thirty executions of each approach with each of the base classifiers have been performed on each dataset.

6 Experimental Results

Tables 2 to 7 include the results obtained by the CP, WP and OP approaches for the different periods of time investigated in this study. Results obtained by the ORB-based approaches are shown in the supplementary material, for space considerations. Section 6.1 presents a preliminary comparison of the approaches against the dummy classifier. Sections 6.2 to 6.4 will focus on the comparisons between the CP and WP approaches, whereas Section 6.5 will also consider the OP approaches.

6.1 Preliminary Analysis

We compared all approaches against a dummy classifier as a preliminary analysis. A JIT-SDP classifier that outperforms a dummy classifier in terms of overall predictive performance is a classifier that was able to learn relevant JIT-SDP knowledge. Filtering and AIO were always able to outperform a dummy classifier in terms of overall G-Mean across times steps, with large effect size. These results are in Table 7 of the supplementary material, for space considerations. The Scott-Knott.BA12 ranking of these approaches across datasets was much better than that of the dummy classifier. The WP and Ensemble approaches also achieved better overall G-Mean ranking than the dummy classifier, even though they were unable to consistently outperform the dummy classifier in all individual datasets. The OP approaches obtained worse overall G-Mean ranking than the dummy classifier. This means that it was unable to learn enough JIT-SDP knowledge to be of practical use for these projects.

Table 8 of the supplementary material shows the overall G-Mean across time steps obtained by all approaches for the proprietary projects. The dummy classifier outperformed all approaches, achieving the top ranking in terms of overall G-Mean across datasets. It is known that a small number of training examples from the underlying problem distribution negatively affects machine learning models [44]. The small number of WP training examples (Table 1) is likely the reason for such poor results on the proprietary projects. Nevertheless, it is still useful to further compare the existing approaches on the proprietary datasets in the subsequent sections of this paper. Approaches that perform better than other approaches on these datasets are likely to learn faster (with less training examples) than approaches that perform worse.

An ideal JIT-SDP approach would also outperform a dummy classifier at different individual stages of the learning process. However, this is unlikely to be achievable in practice. In particular, during the first time steps of the learning, there would always be a very small number of WP examples to learn from. Therefore, so long as WP data is relevant to achieve good predictive performance, there would always be at least some very initial time steps where a given approach performs worse than a dummy classifier. During latter periods of time, concept drifts may sometimes also have a similar effect to that of a small number of training examples. This is because concept drifts may uncover areas of the input space that have not yet been learned. To gain a better understanding of how well the approaches perform against the dummy classifier, we have compared each approach against the dummy classifier separately during the initial, performance drop and stable periods.

Table 2 (and Table 1 of the supplementary material) shows the results obtained during initial period for the

opensource projects. From the Scott-Knott.BA12 test results, we can see that AIO-OOB and Filter-OOB were better ranked than the dummy classifier in terms of G-Mean across open source projects for both OOB and ORB. When analysing individual open source projects, AIO-OOB and Filter-OOB outperformed the dummy classifier with large effect size in 6 and 7 out of 10 projects, respectively. Similar results were achieved when adopting ORB. Absolute improvements in G-Mean were up to 25.09% and 24.98%, for Neutron using AIO-ORB and Filter-ORB, respectively. All other approaches were unable to rank better than the dummy classifier during the initial period.

From Table 3 (and Table 2 in the supplementary material) we can see that all approaches ranked worse than the dummy classifier on the initial period of proprietary projects. This is because the number of commits in the initial period for proprietary data is very small. In reality, the actual initial period might be much longer if we had more commits from these projects. It is possible that the whole period of time analysed for these projects is actually part of the initial period. In most cases, the whole period of the proprietary projects is smaller than the initial period of the open source projects. In particular, the median total number of software changes in the proprietary projects is 555, whereas the median initial period of the open source projects is 17844.5. This explains the poor performance of all approaches on the proprietary projects.

Table 4 (and Table 3 of the supplementary material) shows the results obtained during performance drop periods for open source projects. AIO and Filtering obtained better Scott-Knott.BA12 ranking than the dummy classifier, whereas the other approaches did not. When looking at individual datasets, AIO and Filtering were able to outperform the dummy classifier with large effect size in the performance drop periods of all projects except for JGroups when using OOB and Npm when using ORB. This is an encouraging result for AIO and Filtering, as the performance drop periods are periods where we expect the approaches to struggle to perform well. The absolute improvements were up to 17.39% for BroadleafCommerce and up to 17.48% for Nova when using AIO-OOB and AIO-ORB, respectively in terms of G-Mean.

For the proprietary projects (Table 5 of the paper and Table 4 of the supplementary material), all approaches obtained worse Scott-Knott.BA12 ranking than the dummy classifier during the performance drop period.

Table 6 (and Table 5 of the supplementary material) shows the results obtained during stable periods for open source projects. According to the Scott-Knott.BA12 tests, the AIO, Filtering, Ensemble and WP approaches performed better than the dummy classifier during the stable periods. There were improvements in G-Mean with large effect sizes for all datasets, except for the Ensemble approach on JGroups. In particular, Filtering-OOB led to absolute improvements in G-Mean of up to 32.22% (for Neutron) and AIO-ORB approach led to absolute improvements in G-Mean of up to 31.87% (for Neutron) compared to the dummy approach (Tables 6 and Table 5 of the supplementary material). The OP approaches were unable to achieve better ranking than the dummy classifier across datasets.

For proprietary projects (Table 7 of the paper and Table 6 of the supplementary material), according to Scott-Knott.BA12, AIO-OOB-combined and Filter-ORB-combined were able to outperform the dummy classifier across datasets. Filter-OOB-combined achieved the largest absolute improvement in G-Mean of 9.42% for C2 among CP approaches. AIO-ORB-proprietary achieved absolute G-Mean improvements of up to 12.7% (for C3). These results are encouraging. They show that, at least during periods of stability, some AIO approach is able to outperform the dummy classifier, despite the overall G-Mean of all approaches having been worse than that of the dummy classifier. This strengthens our believe that, had they been given more WP training data for training, the proposed approaches would also perform better than the dummy classifier in terms of overall G-Mean.

The preliminary analysis shows that the proposed CP approaches are able to learn relevant JIT-SDP knowledge. They outperformed the dummy classifier in terms of overall G-Mean for the open source projects, despite not always outperforming the dummy classifier on every individual initial and performance drop periods. Other approaches struggled more to outperform the dummy classifier. All approaches struggled to perform well on the proprietary projects, due to the limited number of WP training examples.

6.2 RQ1: Initial Phase of the Project

Tables 2 and 3 show the number of time steps of the initial phase, the average G-Mean, the effect size A12 against the dummy approach, and the Scott-Knott.BA12 results during this period for the OOB approaches on the open source and proprietary data, respectively. Results for ORB can be found in the supplementary material. Note that the time steps corresponding to the initial period (and also the stable and sudden drop periods) are different for OOB and ORB, such that we cannot compare these approaches against each other in these separate periods. A comparison between these two approaches across all time steps is shown in the supplementary material.

The initial phase of the open source projects lasted from 461 to 6271 time steps with a median of 1418 when using OOB (Table 2). This length of time was also considerable when using ORB, lasting from 900 to 6271 time steps with a median of 1553. The initial phase of the proprietary projects was typically much smaller than that of the open source projects, lasting from 8 to 581 time steps for OOB (Table 3) and from 62 to 671 time steps for ORB. This is probably related to the fact that the duration of these projects as a whole was in general much shorter than that of the open source projects.

For open source projects, AIO and Filtering ranked the best for both OOB and ORB according to Scott-Knott.BA12. A12 effect sizes against WP learning for individual datasets were typically large. The average G-Means of the AIO and Filtering approaches were typically higher during the initial phase than those of the WP approach (absolute improvements in G-Mean of up to 44.81% and 53.89%, for Brackets using Filtering-OOB and AIO-ORB respectively). For the Filtering approach, the G-Means were better than

TABLE 2: Number of initial time steps of the initial phase, average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 ranking to compare OOB-based approaches on this initial phase for open source data

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB	Filter-OOB	Ensemble-OOB	OP-AIO-OOB	OP-Filter-OOB
Tomcat	2006	50	43.62(4.84)[-b]	51.95(1.25)[b]	52.5(0.71)[b]	33.67(0.63)[-b]	53.12(3.28)[b]	52.86(1.15)[b]
JGroups	1268	50	38.6(0.83)[-b]	38.29(0.84)[-b]	39.01(0.75)[-b]	16.64(0.36)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
Spring-Integration	461	50	27.3(0.12)[-b]	24.28(0.25)[-b]	38.69(0.49)[-b]	19.48(0.22)[-b]	11.15(6.13)[-b]	29.16(6.34)[-b]
Camel	3112	50	46.82(1.71)[-b]	57.75(0.74)[b]	57.07(0.82)[b]	39.42(0.52)[-b]	47.0(1.06)[-b]	46.94(0.96)[-b]
Brackets	1569	50	21.37(0.03)[-b]	64.89(1.32)[b]	66.17(0.74)[b]	46.83(0.91)[-b]	50.46(1.05)[s]	59.45(1.33)[b]
Nova	6271	50	55.42(0.41)[b]	63.15(0.68)[b]	64.39(0.34)[b]	51.66(0.29)[b]	62.55(1.16)[b]	62.88(4.43)[b]
Fabric8	795	50	27.01(0.3)[-b]	51.51(1.42)[b]	58.63(1.57)[b]	40.3(0.69)[-b]	56.57(3.12)[b]	45.52(1.08)[-b]
Neutron	917	50	44.84(5.75)[-b]	73.55(0.99)[b]	67.29(2.45)[b]	55.07(1.1)[b]	67.03(0.41)[b]	73.15(0.7)[b]
Npm	2536	50	26.92(1.1)[-b]	48.37(1.34)[-b]	45.75(1.94)[-b]	42.02(0.34)[-b]	42.89(6.01)[-b]	15.7(1.15)[-b]
BroadleafCommerce	677	50	26.37(0.26)[-b]	50.32(1.55)[*]	53.16(1.99)[b]	37.26(1.58)[-b]	40.29(2.8)[-b]	30.63(3.6)[-b]
Ranking		2	4	**1**	**1**	4	3	3

TABLE 3: Number of initial time steps of the initial phase, average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 ranking to compare OOB-based approaches on this initial phase for proprietary data

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB combined	AIO-OOB proprietary	Filter-OOB combined	OP-AIO-OOB combined	OP-AIO-OOB proprietary	OP-Filter-OOB combined
C1	284	50	15.76(4.12)[-b]	35.07(3.74)[-b]	15.97(3.9)[-b]	42.5(2.11)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C2	581	50	24.62(0.37)[-b]	31.03(0.43)[-b]	32.71(5.38)[-b]	46.82(0.76)[-b]	43.84(1.9)[-b]	5.62(6.26)[-b]	45.33(2.97)[-b]
C3	82	50	8.48(0.0)[-b]	23.36(0.0)[-b]	35.9(0.0)[-b]	8.62(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C4	24	50	7.59(0.13)[-b]	18.93(2.73)[-b]	18.63(3.19)[-b]	0.0(0.0)[-b]	20.1(0.14)[-b]	14.47(0.39)[-b]	0.0(0.0)[-b]
C5	8	50	11.62(0.0)[-b]	1.69(2.63)[-b]	31.66(4.57)[-b]	10.14(3.89)[-b]	5.16(6.01)[-b]	1.99(4.52)[-b]	39.83(0.0)[-b]
C6	20	50	3.6(0.0)[-b]	1.8(0.0)[-b]	1.8(0.0)[-b]	1.38(0.77)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C7	29	50	1.17(0.0)[-b]	42.76(2.42)[-b]	50.92(3.65)[-s]	0.0(0.0)[-b]	40.61(8.02)[-b]	23.0(10.38)[-b]	0.0(0.0)[-b]
C8	39	50	0.76(0.0)[-b]	5.87(4.91)[-b]	40.16(8.41)[-b]	0.66(0.26)[-b]	16.56(8.28)[-b]	37.57(6.39)[-b]	0.0(0.0)[-b]
C9	62	50	10.44(1.43)[-b]	15.4(6.94)[-b]	25.12(2.67)[-b]	0.55(0.0)[-b]	21.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
Ranking		**1**	4	3	2	4	4	4	4

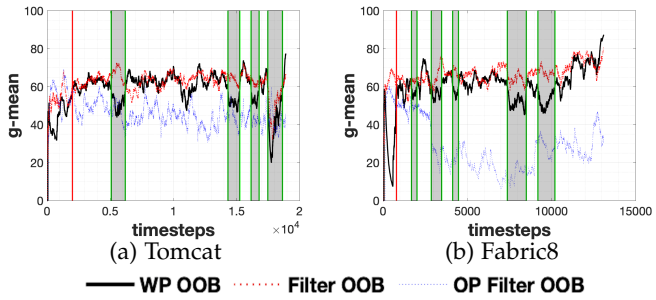


Fig. 1: G-Mean for open source datasets through time using OOB. The vertical red bar indicates the last time step of the initial phase of the project. The periods highlighted in grey background are the sudden drop periods.

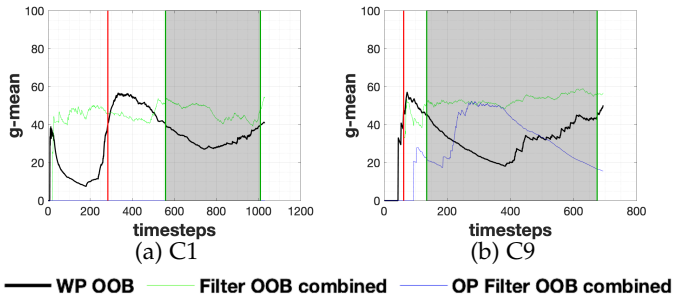


Fig. 2: G-Mean for proprietary datasets through time using OOB. The vertical red bar indicates the last time step of the initial phase of the project. The periods highlighted in grey background are the sudden drop periods.

those of the WP classifiers for the initial period of all open source datasets. Fig. 1 illustrates some sample results (see G-Means up to the red vertical line). Other figures are in the supplementary material. AIO also performed better than WP except for AIO-OOB on JGroups and Spring-integration. The Ensemble approach ranked worse than AIO and Filtering, even though it was able to improve over WP when using ORB according to Scott-Knott.BA12.

The fact that AIO and Filtering ranked best across datasets means that these approaches typically outperform the others. However, this does not mean that they will always outperform the others. For instance, WP-OOB performed better than AIO-OOB on JGroups and Spring-integration at the initial phase. In the case of JGroups, there were no CP training examples arriving during the initial period because of the chronological order. So, AIO was trained only with WP data during this period. Hence, for JGroups, the predictive performance obtained by WP and AIO is very similar.

For Spring-Integration, the initial period is small (461 commits for OOB) and has CP data from only 2 projects (Tomcat and JGroups). It is possible that these CP projects are dissimilar to Spring-Integration. A proportion of 82% of the CP commits in the initial phase of Spring-Integration are from project JGroups. There are large differences in the input features of JGroups and Spring-integration. For instance, the averages of the developers experience (EXP) is 4790 and 1039, of the number of defective commits is 1918 and 123, and of the time interval between last and current change (AGE) is 25 and 4.57 in JGroups and Spring-integration, respectively. This indicates that during this initial period, the JIT-SDP model is trained with very dissimilar commits, possibly causing AIO-OOB to perform worse than WP. Filter-OOB was able to overcome this issue by filtering out commits that are very dissimilar.

For proprietary projects, AIO-proprietary was better

ranked than the WP approach for both OOB and ORB, being followed by AIO-combined according to Scott-Knott.BA12. A possible reason why AIO-proprietary performed better than AIO-combined is the high dissimilarity between the open source and proprietary projects. Together with the very small number of WP training examples during the initial phase, such high dissimilarity can prevent the WP training examples from playing a significant role when training the AIO-combined classifiers. This is because they may be perceived almost as noise by the learning process. When using AIO-proprietary, there is not only smaller amount of CP data (enabling the WP data to play a more significant role), but also those CP data are more similar to WP data as they are all projects of similar nature. This may enable the AIO-proprietary approach to learn the underlying defect generating process better.

AIO-proprietary's G-Means were better than those of WP classifiers in the initial phase (absolute improvements in G-Mean of up to 49.75%, for C7) for most of the datasets except for datasets C6 (OOB) and C1 (ORB). Filtering with OOB's G-Means were lower than WP's for several datasets especially where the initial periods were very small (ranging from 8 to 62). Sample results when it performed better and worse are shown in Fig. 2. For Filtering with ORB, the G-Means were better than those of WP classifiers in the initial phase for all of the proprietary datasets.

It is also interesting to note that AIO performed better than Filtering for the proprietary projects. Specifically, when comparing AIO-OOB combined against Filter-OOB combined, we can see that filtering itself was detrimental. It is likely that filtering resulted in the removal of relevant training examples, which was particularly a problem given the small length of the initial phase of these datasets.

The G-Means obtained for proprietary datasets were in general low. As AIO-proprietary performed significantly better than WP and AIO-combined (see Scott-Knott.BA12 results in Table 3), the CP proprietary data were usually important for improving predictive performance in the initial period. Therefore, the small number of CP proprietary software changes compared to the open source projects is likely to be part of the reason for the poor G-Means obtained for the proprietary projects, even though other factors such as the specific distribution of the WP data may also affect the G-Mean on individual projects.

The G-Means of all approaches during the initial periods were relatively poor compared to the stable periods. This is expected, because the initial period is by definition a period of time when JIT-SDP classifiers have not yet reached their typical average G-Mean, i.e., it corresponds to the period of time when the approaches perform below average. Reaching the average G-Mean value requires training on more WP data. Despite that, our experiments have shown that AIO-OOB outperformed other existing approaches during such periods, having obtained better initial period G-Mean ranking.

RQ1: The initial phase of the projects had considerably large length, lasting from 461 to 6271 and from 8 to 582 time steps for the open source and proprietary projects, respectively. CP data was helpful when there was no or little WP training data available, in particular when using AIO approaches for both open source projects and proprietary projects. Filtering was also top performing for the open source projects. Absolute improvements in average G-Mean were up to 53.89%, avoiding extremely low G-Means.

6.3 RQ2: Periods with Sudden Drops in WP Classifier's Predictive Performance

In several datasets, after the initial phase, the WP approach suffered periods of large drops in G-Means. Figs. 1 and 2 show some examples of that highlighted in grey background. Tables 4 and 5 show further information about the sudden drop periods suffered by OOB for open source and proprietary projects, respectively. Other tables and results for ORB are in the supplementary material.

The periods of sudden drop have considerable length for the open source projects, covering in total from 1202 to 6009 time steps when using OOB (Table 4) and from 936 to 4408 when using ORB. Due to the shorter duration of the proprietary projects, the total length of the sudden drop periods in those projects was smaller, ranging from 210 to 541 time steps for OOB (Table. 5) and 23 to 238 time steps for ORB.

For open source projects, when using OOB, Filtering was the best ranked approach, followed by AIO, according to Scott-Knott.BA12. When using ORB, AIO was top ranked together with Filtering. Drop periods are likely to occur as result of concept drifts, meaning that most recent WP data are dissimilar to past WP data. Filtering out dissimilar examples may help to focus more on the current concept being learned, enabling the model to adapt quicker than with AIO. However, ORB is more robust to concept drift than OOB, due to its automatically boosted resampling rate [7]. So it may not always require the extra focus on the current concept. The Ensemble approach was unable to consistently perform better than the WP approach across open source datasets, obtaining the same Scott-Knott.BA12 ranking as the WP approach.

AIO and Filtering outperformed the WP approach in all open source datasets except Nova during such periods (absolute improvements of up to 27.17% and up to 29.01% when using Filtering-OOB and Filtering-ORB, respectively, for Spring-integration). Fig. 1 shows some examples of that, highlighted in grey background. As we can see, Filtering managed to reduce or even eliminate the drops in G-Mean suffered by the WP approach.

For proprietary projects, Filtering was also better than the WP, AIO and OP approaches, based on Scott-Knott.BA12. Therefore, even when using ORB, Filtering was important during the performance drop periods of the proprietary projects. The effect size of the Filtering and AIO approaches against WP were frequently large. The absolute improvements in average G-Mean over the WP approach were up to 37.35% for OOB when using AIO-combined and up to 30.34% for ORB when using Filtering-ORB (for C8). Fig. 2 shows some examples of results achieved by Filtering, where it is clear that this approach outperformed the WP approach. The AIO approach also outperformed WP approach in most of the datasets. AIO-combined achieved better results than AIO-proprietary when using OOB, but

TABLE 4: Total number of time steps of the performance drop periods, average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 to compare approaches using OOB for open source data during the drop periods

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB	Filter-OOB	Ensemble-OOB	OP-AIO-OOB	OP-Filter-OOB
Tomcat	3855	50	48.52(1.17)[-b]	58.86(0.72)[b]	60.22(1.0)[b]	52.26(0.32)[b]	36.42(6.24)[-b]	43.81(1.44)[-b]
JGroups	4353	50	43.95(1.44)[-b]	48.01(1.05)[-b]	48.08(0.89)[-b]	31.41(0.5)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
Spring-Integration	3973	50	32.98(0.7)[-b]	59.01(0.53)[b]	60.15(0.97)[b]	45.68(0.43)[-b]	52.05(5.34)[s]	34.89(4.74)[-b]
Camel	6009	50	53.15(1.68)[b]	56.17(0.98)[b]	57.0(0.92)[b]	52.4(0.41)[b]	50.09(1.54)[*]	50.68(0.69)[b]
Nova	4427	50	66.99(0.06)[b]	66.99(0.18)[b]	66.85(0.26)[b]	57.19(0.66)[b]	65.4(3.73)[b]	67.1(6.49)[b]
Fabric8	3443	50	53.79(2.18)[b]	56.34(0.76)[b]	65.57(1.04)[b]	53.98(0.4)[b]	23.23(6.54)[-b]	24.48(6.65)[-b]
Npm	1202	50	38.64(1.95)[-b]	62.97(1.47)[b]	65.37(1.25)[b]	51.13(0.62)[b]	44.42(7.52)[-s]	17.36(3.61)[-b]
BroadleafCommerce	3579	50	50.18(0.55)[s]	67.39(1.05)[b]	66.05(0.66)[b]	54.38(0.25)[b]	52.72(1.17)[b]	56.08(2.43)[b]
Ranking		3	3	2	**1**	3	4	4

TABLE 5: Total number of time steps of the performance drop periods, average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 to compare approaches using OOB for proprietary data during the drop periods

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB combined	AIO-OOB proprietary	Filter-OOB combined	OP-AIO-OOB combined	OP-AIO-OOB proprietary	OP-Filter-OOB combined
C1	452	50	32.27(4.27)[-b]	41.37(1.45)[-b]	43.4(4.45)[-b]	46.66(1.08)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C3	210	50	42.79(1.72)[-b]	47.75(1.31)[-b]	39.1(1.42)[-b]	52.17(1.3)[b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C4	249	50	33.0(0.95)[-b]	51.47(1.74)[b]	60.05(2.89)[b]	56.78(1.76)[b]	57.74(1.11)[b]	47.1(1.74)[-b]	51.35(0.0)[b]
C6	466	50	21.34(1.35)[-b]	43.64(1.49)[-b]	45.89(5.02)[-b]	44.32(1.98)[-b]	43.53(0.41)[-b]	37.47(0.18)[-b]	45.1(0.47)[-b]
C7	255	50	25.74(1.19)[-b]	41.5(5.94)[-b]	32.34(1.51)[-b]	40.34(1.02)[-b]	50.43(2.6)[-*]	36.26(0.52)[-b]	30.73(0.4)[-b]
C8	287	50	7.3(0.89)[-b]	44.65(0.96)[-b]	34.6(5.43)[-b]	40.14(1.82)[-b]	44.3(1.81)[-b]	45.49(9.63)[*]	25.57(8.7)[-b]
C9	541	50	31.75(1.77)[-b]	51.34(1.76)[b]	50.08(2.77)[s]	53.25(1.58)[b]	40.3(5.22)[-b]	34.92(3.87)[-b]	34.36(0.55)[-b]
Ranking		**1**	5	3	4	2	4	5	5

similar when using ORB. This shows that open source data can sometimes help to improve predictive performance on proprietary projects during sudden drop periods.

Given that existing literature indicates that JIT-SDP suffers from concept drift [8], it is likely that WP classifiers suffer drops in performance due to changes in the characteristics of WP training data over time. However, in CP learning, training data comes from different projects. Some of the CP training data may have similar distribution as the target project currently has, helping to reduce the negative effect of differences in the distribution over time. This is a potential reason for CP data (including mixed proprietary and open source data) to be helpful in case of sudden performance drops.

Interestingly, the Filtering approach typically achieved better results than the AIO approach during the sudden drop periods. Improvements in G-Means achieved by Filtering-OOB compared to AIO-OOB were up to 9.21% (for Fabric8) for open source and up to 13.16% (for C3) for the proprietary data. This suggests that even though CP data may prevent performance drops resulting from changes in characteristics of the data, it might introduce other performance drops due to the use of too dissimilar CP data. This indicates that filtering dissimilar instances can be particularly helpful to reduce or prevent sudden drops of performance, which has not been captured by the offline JIT-SDP literature when they concluded that AIO and Filtering approaches perform similarly [6]. Unlike in the initial period, CP approaches benefited from a combination of both open source and proprietary data during drop periods, specially for OOB, where combined data outperformed proprietary-only approaches both through Filtering and AIO. This suggests that the diversity of data from multiple sources may help with dealing with concept drift in JIT-SDP.

Brackets, Neutron, C2 and C5 did not present drops in predictive performance. Together with the fact that no large sudden changes in the values of the input features were found for these projects over time, we believe that these projects are more stable than the others, being less negatively affected by concept drift.

RQ2: The periods of sudden drop in performance were of considerable length, covering in total from 936 to 6009 time steps for open source, and from 23 to 541 time steps for proprietary data. CP approaches frequently helped to reduce or even prevent sudden drops in G-Mean compared to WP approaches, with absolute improvements of up to 37.35% during such periods. Filtering was the top ranked approach across datasets.

6.4 RQ3: Effects of CP Data on the Predictive Performance During Stable Periods of The Projects

Tables 6 and 7 show information about the stable periods for the open source and proprietary projects, respectively, when using OOB. Tables for ORB are in the supplementary material.

The smallest and largest total length of stable periods for open source projects are 4182 and 38291 for OOB (Table 6), and 5490 and 38310 for ORB. For proprietary projects, they are 20 and 472 for OOB (Table 7) and 19 and 486 for ORB. Due to the long duration of the open source projects, it is expected that their stable periods are longer than the periods of sudden drop in performance, even though the periods of sudden drop are still considerable periods worth addressing. For the proprietary data, the stable periods were short compared to the sudden drop periods. Possibly, due to the relatively small number of software changes in the proprietary projects, the whole period of time covered by the proprietary projects is more unstable.

Given the length of time covered by stable periods, it is important that the better results obtained by CP approaches during the initial and sudden drop periods are not counteracted by poor performance during stable periods.

TABLE 6: Number of time steps of the stable periods, and average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 to compare OOB-based approaches on stable periods for open source data

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB	Filter-OOB	Ensemble-OOB	OP-AIO-OOB	OP-Filter-OOB
Tomcat	13046	50	62.69(0.81)[b]	61.36(0.5)[b]	63.36(0.64)[b]	50.57(0.22)[b]	39.68(6.07)[-b]	45.23(1.56)[-b]
JGroups	12704	50	60.08(0.66)[b]	58.74(0.72)[b]	58.51(0.54)[b]	38.8(0.3)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
Spring-Integration	4316	50	64.86(1.01)[b]	65.99(0.35)[b]	66.96(0.77)[b]	56.02(0.33)[b]	55.88(8.08)[b]	45.51(8.94)[-*]
Camel	21454	50	67.33(0.56)[b]	64.48(0.67)[b]	65.05(0.62)[b]	54.61(0.35)[b]	41.45(1.55)[-b]	41.52(1.26)[-b]
Brackets	15795	50	68.27(0.12)[b]	71.44(0.48)[b]	71.38(0.42)[b]	64.25(0.54)[b]	38.19(13.68)[-b]	66.67(0.36)[b]
Nova	38291	50	79.84(0.26)[b]	82.03(0.21)[b]	81.54(0.2)[b]	68.72(0.79)[b]	73.99(16.63)[b]	75.19(0.5)[b]
Fabric8	8868	50	65.0(0.77)[b]	63.53(0.62)[b]	67.64(0.64)[b]	54.15(0.4)[b]	26.11(4.94)[-b]	28.23(4.47)[-b]
Neutron	18605	50	81.13(0.27)[b]	81.87(0.23)[b]	82.22(0.25)[b]	68.93(1.5)[b]	74.53(1.79)[b]	71.53(3.95)[b]
Npm	4182	50	59.52(1.71)[b]	64.96(0.63)[b]	66.49(0.99)[b]	55.3(0.46)[b]	29.24(2.24)[-b]	40.11(1.15)[-b]
BroadleafCommerce	10754	50	65.96(1.12)[b]	70.73(0.62)[b]	70.13(0.55)[b]	59.55(0.2)[b]	54.26(0.8)[b]	60.42(1.07)[b]
Ranking		4	2	**1**	**1**	3	5	4

TABLE 7: Number of time steps of the stable periods, and average G-Means, A12 effect sizes against the Dummy approach and Scott-Knott.BA12 to compare OOB-based approaches on stable periods for proprietary data

Dataset	#Time Steps	Dummy	WP-OOB	AIO-OOB combined	AIO-OOB proprietary	Filter-OOB combined	OP-AIO-OOB combined	OP-AIO-OOB proprietary	OP-Filter-OOB combined
C1	294	50	49.55(1.47)[-*]	43.65(2.33)[-b]	43.4(6.94)[-b]	45.75(2.02)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C2	20	50	64.59(0.23)[b]	52.86(1.75)[b]	55.58(2.72)[b]	59.42(0.62)[b]	36.49(0.93)[-b]	2.8(3.22)[-b]	37.36(1.46)[-b]
C3	125	50	54.54(1.05)[b]	54.01(0.43)[b]	33.69(2.6)[-b]	57.0(0.61)[b]	0.0(0.0)[-b]	0.0(0.0)[-b]	0.0(0.0)[-b]
C4	68	50	45.72(1.21)[-b]	53.4(1.23)[b]	56.98(4.85)[b]	27.16(8.92)[-b]	46.76(0.61)[-b]	44.62(2.6)[-b]	21.15(0.0)[-b]
C5	315	50	23.2(0.43)[-b]	46.55(1.35)[-b]	42.6(0.99)[-b]	49.93(1.71)[-s]	44.14(5.2)[-b]	49.53(2.92)[-s]	41.15(4.14)[-b]
C6	69	50	41.71(0.64)[-b]	43.05(0.65)[-b]	49.84(2.73)[-*]	41.83(3.36)[-b]	24.67(0.34)[-b]	14.76(0.07)[-b]	21.1(0.26)[-b]
C7	262	50	45.55(1.0)[-b]	52.02(5.6)[s]	52.62(2.11)[b]	52.35(2.27)[b]	57.67(1.5)[b]	52.6(1.18)[b]	40.38(0.7)[-b]
C8	472	50	26.13(0.22)[-b]	55.16(0.91)[b]	47.01(3.71)[-b]	40.38(1.65)[-b]	45.29(1.94)[-b]	48.99(5.2)[s]	41.14(2.6)[-b]
C9	91	50	49.63(1.59)[-b]	56.46(2.26)[b]	59.12(6.12)[b]	46.6(2.06)[-b]	29.03(2.98)[-b]	8.62(1.08)[-b]	14.33(2.11)[-b]
Ranking		2	3	**1**	2	3	4	5	5

Except for the Ensemble approach, Scott-Knott.BA12 shows that all CP approaches performed at least as well as the WP approach across datasets during the stable periods. This was the case both when adopting OOB and ORB, for both open source and proprietary data. This is very encouraging, as it shows that the G-Mean improvements achieved by Filtering and AIO during the initial and sudden drop periods are not cancelled out by worse G-Mean during stable periods.

For the open source data, according to Scott-Knott.BA12, both AIO and Filtering approaches were ranked the best for OOB and ORB during stable periods in terms of G-Mean. Moreover, except for Ensemble, the CP approaches were frequently able to outperform the WP approach in terms of G-Mean during the stable period, though by a smaller amount than they did for the initial and sudden drop periods. The Filtering approach performed up to 6.97% (for Npm) and 8.39% (for Spring-integration) better compared to the WP approach for OOB (Table 6) and ORB, respectively.

For proprietary data, according to Scott-Knott.BA12, AIO-OOB-combined was ranked the best for OOB (Table 7). The largest improvement in G-Mean compared to WP approaches was achieved by AIO-OOB-combined, which had an improvement of 29.03% for C8. When using ORB, Filtering was the top ranked approach. Filtering-ORB achieved G-Mean improvements of up to 15.92% (for C8).

Filtering was helpful for open source projects, but not so much for proprietary projects, where Filtering-OOB performed similar to WP-OOB. Therefore, the more competitive performance achieved by Filtering compared to AIO during the sudden drop periods can possibly be counteracted by less competitive results during stable periods compared to AIO. Nevertheless, it is worth reiterating that both Filtering and AIO performed at least as well as the WP approach during the stable periods across datasets, and that both of them outperformed WP approaches during the periods of sudden drop, based on Scott-Knott-BA12.

Although the Ensemble approach performed similar to the WP approach for initial and drop periods, it was worse for stable periods. A further analysis of the results obtained by each classifier within the ensemble reveals that their individual G-Means were not high, which resulted in the poor G-Mean of the ensemble as a whole. This could be due to lack of enough training data for the individual classifiers, given that each classifier in the ensemble is trained with data from one project. This suggests that the larger amount of varied data to train a model was crucial to improve predictive performance in online JIT-SDP. It also explains why the Ensemble approach worked in previous offline mode but not in online mode, as in offline mode chronology would have been ignored, enabling each base model to be trained with larger amounts of data that would not have been available in practice. In addition, these results suggest that using a weighted ensemble instead of simple votes as in this study would also unlikely work well. This kind of strategy would emphasise the individual models that perform best for the project being predicted, but if all models underperform, emphasising the best models would not help in this case.

RQ3: Except for the Ensemble approach, CP approaches performed at least as well as WP approaches in terms of G-Mean during stable periods. In particular, Filtering with ORB obtained up to around 8.39% improvement in G-Mean for open source data and AIO-OOB-combined achieved up to 29.03% absolute improvement in G-Mean for proprietary data than the WP approach during such periods.

6.5 RQ4: The Need for Updating CP Approaches Over Time

In traditional offline JIT-SDP, models are built with only CP training data and are never updated with new incoming training data over time once they start predicting. Such OP approaches have been successful in achieving similar predictive performance to WP approaches in the offline learning literature [6]. Even though there has been work on online JIT-SDP [7], [30], it is unclear whether OP approaches would perform well enough such that online learning is after all unnecessary. In particular, if OP approaches trained on data received before a given project perform well enough when making predictions in realistic online scenarios, then it would be unnecessary to update CP or WP models over time with incoming WP examples. It would be sufficient for the model to be trained just once with all the available CP data in offline mode.

We investigated this using a JIT-SDP OP model only with CP data that are available until the first WP instance arrives. Once the first WP instance arrives for prediction, training of the OP model is ceased. Comparing the results between OP and CP approaches during initial period, drift period and stable period will provide a better understanding of the necessity of updating CP approaches over time. There are several cases when OP approaches have G-Mean of zero. This happened because the OP model did not have enough training data. When calculating differences in G-Mean between OP and CP, these cases are excluded because they may not reflect the actual magnitude of improvement.

During initial phase, for open source data, the OP AIO and Filtering approaches usually performed worse than the AIO and Filtering approaches (Table 2). CP approaches obtained absolute G-Mean improvements of up to 32.15% (AIO-ORB for Nova) for open source data and up to 35.02% (AIO-ORB-proprietary for C9) for proprietary data during this period, when compared to OP approaches. However, OP's G-Means were frequently better than those of WP classifiers in the initial phase. This is supported by the Scott-Knott.BA12 which ranked the OP-AIO and OP-Filtering better (third) than WP (fourth) for OOB (Table 2). For proprietary data, when using OOB, whereas Filtering outperformed the WP approach according to ScottKnott.BA12, all OP approaches performed similarly to WP approaches. In general, we can see that performing online CP learning over time is important to improve G-Means in the initial period, even though data from other projects was also useful to improve G-Means over WP approaches through OP.

During the sudden drop periods, CP approaches obtained absolute improvements in G-Mean of up to 48.16% (Filter-OOB for Npm) compared to OP for open source data, and up to 23.9% (AIO-ORB-proprietary for C9) for proprietary data. While AIO and Filtering were very successful in preventing sudden drops in predictive performance suffered by the WP approach for the open source data, the OP approaches did not perform so well during these periods, being ranked worse than the WP approach based on Scott-Knott.BA12 (Table 4). The OP results were a bit more competitive for the proprietary data (Table 5), but still not in the top ranked group. These results show that OP data are not enough. Possibly, even though OP data may be prepared to reduce the sudden drop in predictive performance caused by concept drifts, in general it cannot learn well the underlying defect generating process of the project of interest, due to the lack of data coming from it. This suggests that the combined use of WP and data from other projects is important to prevent sudden drops in predictive performance.

For stable periods, OP approaches ranked worse than all CP and WP approaches for both open source and proprietary data (Tables 6 and 7). There were also some datasets (Fabric8 and Npm for OOB) where the differences in G-Means between OP and WP approaches were quite large. CP approaches obtained absolute G-Mean improvements of up to 41.25% (AIO-ORB for Nova) compared to OP for open source data, and 37.94% (AIO-OOB-proprietary for C9) for proprietary data during the stable period. These results further confirm that using OP data is not enough to achieve good predictive performance in JIT-SDP when considering realistic online learning scenarios. For JGroups dataset, OP approaches obtained G-Mean of zero. This is because the first change of this project was created before any training data from other projects was available. Hence, the models were not trained, resulting in them always predicting the non-defect inducing class, which led to a G-Mean of zero.

Overall, these results suggest that a combination of CP and WP data is important to improve predictive performance in JIT-SDP. Using only WP data can lead to very poor performance during the initial phase of the projects, and severe drops in predictive performance over time. Using only OP data can help during the initial period, but overall is not enough to achieve competitive results. Tables 7 and 8 of the supplementary material contain a further comparison of all approaches across the whole data streams formed by the projects. It confirms that OP approaches are not well ranked in terms of G-Mean when considering the data streams as a whole. OP approaches performed up to 39.36% (for Nova with ORB) and 36.64% (for C9 with ORB) worse compared to corresponding CP approaches across all time steps.

Online approaches can make use of both CP and WP incoming data whereas OP approaches only train with data from other projects. As a result, online approaches use more training data than OP approaches, which contributes towards better predictive performance. The presence of WP data is also important along with larger amount of data. This can be realised when we observe the experiments with proprietary projects as proprietary projects have very small amount of WP data and their predictive performance was much worse. This indicates that data from other projects help, but we still need WP data. Therefore, online approaches achieved better predictive performance compared to OP approaches most of the time.

RQ4: CP approaches achieved absolute G-Mean improvements of up to 32.15%, 48.16% and 41.25% for open source data and up to 35.02%, 23.9% and 37.94% for proprietary data compared to OP approaches during initial, sudden drop and stable periods respectively. This highlights the importance of updating the JIT-SDP model with CP (including WP) data received over time.

7 SENSITIVITY ANALYSIS

Filtering and AIO were the most competitive approaches according to our experiments. Filtering should be preferred over AIO especially if one wants to strongly focus on prevention of sudden drops in G-Mean. However, Filtering has more hyperparameters to be tuned than AIO, and it is unclear how these hyperparameters affect its performance. To avoid loosing such advantage, it is important to understand the effect of these hyperparameters better. We will focus on Filtering-OOB, as overall it achieved similar or better rankings than Filtering-ORB (see Tables 7 and 8 of the supplementary material). Results for Filtering-ORB can be found in the supplementary material.

To investigate this, Analysis of Variance (ANOVA) was performed to analyse the influence of each hyperparameter as well as their interactions with each other on the average G-Mean. The following within-subject factors are investigated: $windowSize \in \{500, 700, 1000\}$, number of top short distances to be used $K \in \{50, 100, 200\}$, distance threshold for similarity $maxDist \in \{0.6, 0.7, 0.8\}$ and maximum size of the queue of dissimilar CP instances $cpqSize \in \{500, 1000\}$. The $dataset$ is a between-subjects factor. Therefore, a Split-Plot (mixed) ANOVA design was used. Mauchly's sphericity test were adopted with a level of significance of 0.05 to evaluate whether or not the sphericity assumption is violated. If violated, Greenhouse-Geisser corrections were adopted to take that into account. Partial eta-squared η_p^2 was used as a measure of effect size.

The analysis was based on Filtering-OOB with six datasets: JGroups, Fabric8 and Neutron (for open source) and C5, C6 and C8 (for proprietary data). These datasets were chosen because they are the open source and proprietary datatsets where this approach achieved the best, worst and average G-Mean values, respectively. Thirty runs for each combination of hyperparameter values were carried out on each dataset.

Table 8 shows the sensitivity analysis for the open source projects with OOB. Several factors and interactions have large ($\eta_p^2 \geq .244$) effect size. Many of them involve the factors K, *maxDist* and their interaction with dataset. Therefore, we examine them further through Fig. 3. From this figure we can see that despite the choice of hyperparameters leading to significantly different G-Means, such differences did not have large magnitude except for Fabric8. Therefore, in general, small K with small *maxDist* led to top or close to top G-Means for the open source projects, even when this combination of values did not lead to the best G-Mean values. Smaller K means that Filtering can use examples for training so long as they are similar to a small number of recent WP software changes. Small *maxDist* means that the similarity measure is more strict, not allowing very dissimilar training examples. Small K with small *maxDist* also led to top or close to top G-Means when using ORB.

The hyperparameter values selected based on the grid search used for hyperparameter tuning in Section 5.5 generally matched the best hyperparameter values obtained in the sensitivity analysis. This indicates that the hyperparameter tuning procedure worked well for the open source projects.

Table 9 shows the sensitivity for proprietary projects with OOB. Hyperparameters $maxDist$, K, interaction between K*$maxDist$, $maxDist$*dataset* and K*$maxDist$*dataset* had very large effect size ($\eta_p^2 \geq .516$) compared to the other hyperparameters and interactions, whose effect sizes were always $\eta_p^2 \leq .134$. As the largest effect sizes involve hyperparameters $maxDist$, K and their interaction with *dataset*, we also examine them further through Fig. 4.

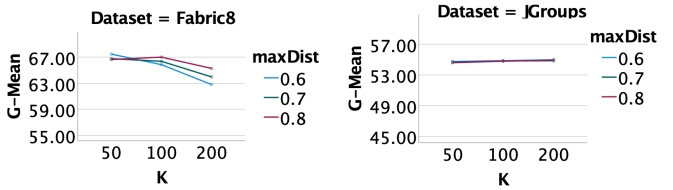
From this figure, we can see that the magnitudes of the differences in G-Mean were larger than for the open source projects. This suggests that Filtering becomes more sensitive to hyperparameter choice when dealing with smaller datasets. Larger values of K usually led to top or close to top G-Means. A larger K requires CP training examples to be similar to a larger number of WP examples. This may have helped the learning process to focus more on learning the core aspects of the current defect generating process, which are shared by a larger proportion of WP training examples. As the number of WP training examples is small in the proprietary datasets, such additional focus may be important. The results for different values of *maxDist*

TABLE 8: ANOVA and Effect Size Results for Open Source Data

Within Subject Test for Filtering-OOB (Open Source Data)					
Factor/Int.	SS	DF	MS	F	η_p^2
K*Dataset	1791.279	4	447.82	2038.001	0.979
K	905.532	1.815	498.873	2060.511	0.959
maxDist*Dataset	304.904	4	76.226	363.86	0.893
K*maxDist*Dataset	326.35	8	40.794	248.202	0.851
windowSize*K*Dataset	254.511	8	31.814	168.637	0.795
K*maxDist	213.227	4	53.307	324.334	0.788
windowSize*Dataset	124.758	4	31.189	160.438	0.787
windowSize*K	242.986	3.467	70.079	322.001	0.787
windowSize	101.327	1.806	56.103	260.612	0.75
windowSize * maxDist * Dataset	60.926	8	7.616	39.002	0.473
windowSize*K*maxDist*Dataset	116.255	16	7.266	38.747	0.471
windowSize*K*maxDist	112.351	8	14.044	74.891	0.463
windowSize*maxDist	36.793	4	9.198	47.106	0.351
maxDist	11.785	2	5.893	28.128	0.244
K*maxDist*cpqSize	7.657	4	1.914	11.486	0.117
K*maxDist*cpqSize*Dataset	7.695	8	0.962	5.772	0.117
windowSize*K*cpqSize*Dataset	8.666	8	1.083	5.527	0.113
K*cpqSize*Dataset	3.712	4	0.928	4.284	0.09
maxDist*cpqSize*Dataset	3.019	4	0.755	3.226	0.069
windowSize*cpqSize*Dataset	1.642	4	0.411	2.825	0.061
maxDist*cpqSize	1.498	1.725	0.869	3.202	0.036
windowSize*maxDist*cpqSize	0.757	3.596	0.21	0.988	0.011
windowSize*cpqSize	0.171	2	0.085	0.588	0.007
K*cpqSize	0.271	2	0.136	0.626	0.007
Between Subject Test for Filtering-OOB					
Dataset	56.040 $\times 10^4$	2	28.020 $\times 10^4$	122.836 $\times 10^4$	1

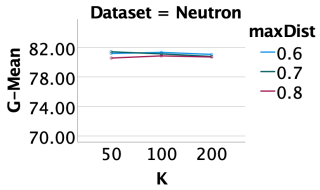
TABLE 9: ANOVA and Effect Size Results for Proprietary Data

Within Subject Test for Filtering-OOB (Proprietary Data)					
Factor/Int.	SS	DF	MS	F	η_p^2
maxDist*dataset	35880.183	4	8970.046	3254.373	0.986
K*dataset	7718.661	4	1929.665	729.317	0.942
K*maxDist*dataset	4795.522	8	599.44	232.216	0.838
maxDist	1289.891	2	644.946	233.989	0.722
K	1076.273	1.858	579.372	203.389	0.693
K*maxDist	989.154	3.577	276.565	95.796	0.516
cpqSize*dataset	44.216	2	22.108	6.963	0.134
cpqSize	33.326	1	33.326	10.497	0.104
maxDist*cpqSize	56.189	2	28.095	8.667	0.088
maxDist*cpqSize*dataset	55.978	4	13.995	4.317	0.088
K*cpqSize*dataset	34.787	4	8.697	2.972	0.062
K*maxDist*cpqSize	63.415	3.472	18.265	5.56	0.058
Between Subject Test for Filtering-OOB					
dataset	74431.101	2	37215.55	14019.740	0.997



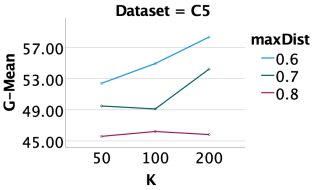
(a) Effect of $K * maxDist$ with Fabric8

(b) Effect of $K * maxDist$ with JGroups

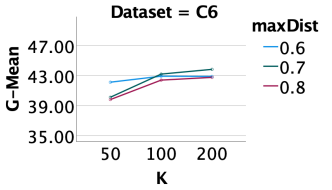


(c) Effect of $K * maxDist$ with Neutron

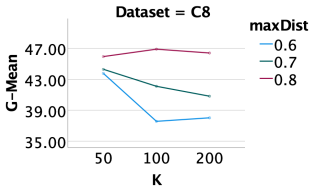
Fig. 3: Plots of Marginal Means for Filtering-OOB's Factors K, $maxDist$ and $dataset$ (Opensource data).



(a) Effect of $K * maxDist$ with C5



(b) Effect of $K * maxDist$ with C6



(c) Effect of $K * maxDist$ with C8

Fig. 4: Plots of Marginal Means for Filtering-OOB's Factors K, $maxDist$ and $dataset$ (Proprietary data).

were varied, indicating that this hyperparameter was more dependent on the characteristics of specific datasets. The findings for ORB were similar for K, but large *maxDist* led to good results in all datasets when using ORB.

The best values of hyperparameter *maxDist* selected by the grid search were either small or large for datasets C5, C6 and C8, which aligns with the above finding. The value of hyperparameter K selected by the grid search was not large for datasets C5 and C6. This means that Filtering was able to achieve competitive results using sub-optimal K value, and it is possible for it to achieve better predictive performance for these datasets if this parameter is better tuned for the dataset of interest.

It is worth noting that the hyperparameter tuning done in this paper was based on G-Mean, so as to achieve a good trade-off between Recall0 and Recall1. However, one could focus on another performance metric should they wish to focus more on a specific class. For instance, they could focus on Recall0 (or false positive rate) to target a given pre-defined false positive rate such as the value of 30% suggested in [45]. We have performed an additional investigation to check whether different hyperparameter configurations could enable to tune the false positive rate of the approaches. In 8 out of 10 open source datasets, at least one of these configurations led to an average overall false positive rate that is close to 30%: Spring-Integration: 21.63%, Neutron: 25.73%, Tomcat: 27.45%, BroadLeaf: 29.54%, Nova: 29.67%, Brackets: 29.79%, Fabric8: 33.79%, JGroups: 34.08%, Camel: 39.61%, Npm: 40.34%.

The hyperparameter sensitivity analysis suggests that the G-Mean obtained by Filtering is more sensitive to hyperparameters when the datasets are smaller. Smaller/larger values for K may be preferable for larger/smaller datasets for the purpose of improving G-Mean. While smaller *maxDist* led to acceptable G-Means on the larger datasets, the choice of *maxDist* was very dependent on the dataset and/or underlying machine learning algorithm when the dataset was small.

8 Analysis of Computational Cost

Table 10 shows total average runtime of AIO, Filter, OP-AIO and OP-Filter with OOB in columns 2–5, computed across 5 runs for 3 open source projects on an Intel(R) Xeon(R) CPU E5-2690 v3 at 2.60GHz and 16Gb of RAM. Tomcat, Npm and Brackets were chosen as they have small, medium and large number of OP projects, respectively.

The total runtime of the OP approaches is mostly spend for training prior to the beginning of the project being predicted. The total runtime of the AIO and Filter approaches includes both such prior training and the training performed over time, with training examples generated after the target project commenced. As we can see, updating the JIT-SDP classifier with additional training examples received over time leads to large increases in the total runtime in the AIO and Filter approaches compared to the OP approaches. The Filter approaches are slower than their corresponding AIO approaches, because they require the training process to compare the similarity between software changes to decide which training examples to filter out.

Nevertheless, the runtime of *all* approaches is actually very small compared to the number of software changes being learned. Columns 7–10 of Table 10 list the average runtime per year and per day for the AIO and Filter approaches. This value is computed by dividing the runtime spent after the target project commenced by the duration of the target project in years and days, respectively. This is done because the learning process of these approaches occurs continuously over time. OP was not included as all its training process is spent before the target project commences. We can see that the highest runtimes are 17.067 and 506.6 seconds per year for AIO and Filter, respectively. They correspond to only 0.05 and 1.39 seconds per day, respectively. As these runtimes are very small, these approaches can be used in practice without causing much computational overhead.

Overall, Filter and AIO had much higher computational cost than the OP approaches. However, such higher computational cost is still very small (less than 1.5 seconds per day).

TABLE 10: Runtime Analysis of CP Approaches

Dataset	AIO-OOB (s)	Filter-OOB (s)	OP-AIO-OOB (s)	OP-Filter-OOB (s)	Duration (year)	Runtime Per Year (AIO)	Runtime Per Year (Filter)	Runtime Per Day (AIO)	Runtime Per Day (Filter)	# Training Examples After Target Project Starts (For AIO and Filter)
Tomcat	98.8	2414.6	3	4.6	11.7	8.188	205.983	0.02	0.56	227286
Brackets	111	3185.6	8.6	146	6	17.067	506.6	0.05	1.39	166978
Npm	102.6	3358.8	3.4	51.8	8.16	12.157	405.27	0.03	1.11	204398

9 Implications to Practice

Our work has practical implications as described below:

- Our study demonstrates the importance of the amount of training data, with both WP and CP data being important to improve JIT-SDP. Practitioners should consider collecting large amounts of both CP and WP training data when adopting JIT-SDP.
- The G-Means obtained by all JIT-SDP approaches were low on the proprietary projects, due to the small number of WP software changes that can be used as training examples in these projects. Therefore, if a software development company expects their projects to be very small like the proprietary software projects used in this study, current JIT-SDP approaches may not be recommended.
- In online JIT-SDP, if practitioners use CP data (along with WP data), this can help improving the performance of the model at the initial phase of a project. This can help practitioners to use JIT-SDP earlier during the development of a project (RQ1,RQ4).
- WP models can suffer performance drops which cause them to be unsuitable during certain periods of time. These drops mean that, at any given point in time, models may be performing very well or very poorly, being unreliable for practitioners. Using CP data (along with WP data) can help to overcome this issue by preventing or reducing such drops, enabling practitioners to more continuously use JIT-SDP throughout the lifetime of the project (RQ2,RQ4).
- The use of CP (along with WP data) also improves predictive performance of JIT-SDP during stable periods (RQ3,RQ4). Therefore, the benefits provided by CP data during the initial and sudden drop periods are not cancelled out by poor predictive performance during stable periods. This further confirms that CP approaches could be adopted throughout the software development process.
- In practice, it is not possible to know beforehand how long the initial period will last, or whether we are currently in a performance drop period. In particular, tracking predictive performance over time will not enable practitioners to know that. Due to verification latency, any tracking of the predictive performance over time in practice would have a delay of "waiting time" days. This means that the performance being calculated "now" by practitioners would correspond to the performance obtained "waiting time" days ago. Therefore, it is important to adopt an approach that is competitive during the stable periods, while not being so negatively affected during the initial or drop periods. As shown in Sections 6.2 to 6.5, the proposed approaches AIO and Filtering achieved the best results in terms of that.
- Practitioners may still wish to track such predictive performance over time despite the delay caused by verification latency. Once the drop in performance is noticed, practitioners may opt for stopping to use JIT-SDP until an improvement in performance is noticed. The caveat is that this also means that a JIT-SDP model that is good will only start being used "waiting period" days after its predictive performance becomes suitable. A similar caveat occurs when monitoring the performance during the initial phase to decide when to start using the JIT-SDP classifier. An alternative strategy for the initial phase is to start using the JIT-SDP classifiers after 1418 software changes have been produced, as this was the median length of the initial phase according to Table 2. After this length, JIT-SDP classifiers are likely to perform better. Even though this number will not work for all projects, it is a suitable strategy if used together with an approach such as AIO and Filtering, which reduce the risk of underperforming a dummy classifier during this period.

10 Threats to Validity

Internal validity: Each approach for each dataset with each classifier has been executed 30 times to mitigate threats to internal validity. Also, results can be influenced by hyperparameter choices. To ensure a fair comparison, a grid search was performed on a set of possible values for the hyperparameters of each approach based on an initial portion of the data stream (see Section 5). All approaches take verification latency into account and fully respect chronology. There is some chance that the author timestamps used to chronologically order the data contain errors resulting from, e.g., misconfigured clocks or problems when converting old repositories into GitHub. However, such issues tend to affect a very small proportion of commits [34], being unlikely to negatively affect our analyses. An additional threat is the use of Commit Guru to collect labelled software changes. This tool is based on SZZ, which can lead to label noise [46]. This threat is partly mitigated by the finding that the performance reductions caused by the mislabeling incurred by the original SZZ algorithm are not significant compared to the most recent SZZ version [46]. However, even the most recent SZZ version is likely to incur some label noise.

Construct validity: This work is mainly based on G-Mean as an evaluation metric. Recall0 and Recall1 are also reported in the supplementary material. Predictive performance is calculated in a prequential way with fading factor to discount older information across time, so that plots of predictive performance reflect the variations in predictive

performance observed over time. G-Mean with fading factor is a widely used metric appropriate for class imbalance learning [9]. Even though G-Mean has been criticised in [37], such criticism does not affect the comparisons between classifiers on the same datasets.

Statistical conclusion validity: Scott-Knott test was run with non-parametric bootstrap sampling considering A12 effect size to avoid concluding that there is a difference in predictive performance when this difference is likely to be irrelevant due to insignificant effect size.

External validity: This study used 10 open source projects and 9 proprietary projects of various characteristics such as programming language, starting date, number of commits per day, etc. As with any study involving machine learning, the results may not generalise to other types of projects. This study is based on the threshold-dependent performance metric G-Mean. Other threshold values could be used to increase recall on one class at the cost worsening the recall on the other class. As we have adopted OOB and ORB to increase recall on the minority class while attempting not to worsen recall on the majority class too much, we have kept the default threshold of 0.5. Conclusions may not generalise to other threshold values.

11 Conclusions

This study provided the first investigation of CP learning for JIT-SDP in a realistic online learning scenario, using both open source and proprietary data. We showed that CP approaches trained with incoming CP and WP data can help to improve predictive performance over WP approaches trained only with WP data and over OP approaches trained only with data from other projects. The AIO and Filtering CP approaches were particularly helpful during the initial phase of the project when there is not enough WP data available (RQ1), leading to up to 53.89% and 35.02% absolute improvements in G-Mean over WP and OP models, respectively. These approaches also helped to reduce or even eliminate sudden drops in predictive performance suffered by the WP model (RQ2) after the initial phase of the project, achieving up to around 37.35% and 48.02% absolute improvements in G-Mean during such periods of time compared with WP and OP models, respectively. Also, during stable periods (RQ3), these approaches achieved up to 29.03% and 41.25% absolute improvements in G-Mean compared to WP and OP models, respectively. These results highlight the importance of using CP data (including data from other projects and data from the project being predicted) over time for training JIT-SDP models.

When comparing CP, WP and OP, the best ranked approaches for initial, drift, non-drift and overall periods were always online CP approaches. For most cases, OP approaches were worse than and for few cases they were similar to online CP approaches. In general, Filtering is recommended over AIO if we wish to strongly focus on preventing sudden drops in predictive performance. If the focus is on the initial or stable periods, we recommend AIO, as it performed competitively and requires less hyperparameters. Even though the Ensemble approach was shown to perform well in offline learning [6], it was the worst approach when considering a realistic online learning scenario, obtaining average G-Means that were even lower than those of the WP approach. This indicates that splitting data from different projects may not be appropriate in online scenarios. Training a single model combining CP and WP together (AIO) significantly improved performance, hence is more suitable in online JIT-SDP. Our results indicate that both the number of CP and WP training examples is important for achieving good predictive performance in JIT-SDP. This is confirmed by the fact that G-Means of CP approaches obtained during initial phase, when there is less WP training examples, are lower compared to the drop and stable periods.

Our study found that the predictive performance on the proprietary projects was not high. Future work includes incorporating additional longer running industrial projects to investigate whether the predictive performance would improve. Moreover, such results indicate that WP data is still necessary to improve predictive performance in JIT-SDP. New transfer learning approaches to reduce the need for large amounts of WP data would be desirable. These approaches could focus especially on improving predictive performance during the initial phase of the projects, which was the most challenging for the proposed approaches. The predictive performance of the proposed approaches was also found to be affected by hyperparameter choice. However, hyperparameter tuning is inherently difficult in online learning, as the best hyperparameter values may change over time. Approaches able to automatically adjust their hyperparameters over time are desirable. Finally, even though the proposed approaches were able to improve predictive performance during the initial and performance drop periods, the predictive performance was still much lower during these periods than during the stable periods. Mechanisms able to alert practitioners of potential reductions in predictive performance are desirable.

References

[1] 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, pp. 554–565.

[2] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 2010, pp. 107–116.

[3] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE TSE*, vol. 38, no. 6, pp. 1276–1304, 2012.

[4] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE TSE*, vol. 31, no. 10, pp. 897–910, Oct. 2005.

[5] 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*, 2013.

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

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

[8] 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.

[9] 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.

[10] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[11] R. S. Wahono, "A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks," *Journal of Software Engineering*, vol. 1, no. 1, pp. 1–16, 2015.

[12] A. Agrawal and T. Menzies, "Is "better data" better than "better data miners"?: on the benefits of tuning smote for defect prediction," in *ICSE*, 2018, pp. 1050–1061.

[13] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *QRS*. IEEE, 2017, pp. 318–328.

[14] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *ASE*, vol. 23, no. 4, pp. 569–590, 2016.

[15] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *FSE*, 2009, pp. 91–100.

[16] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *ASE*, vol. 19, no. 2, pp. 167–199, 2012.

[17] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *ESEM*, 2013, pp. 45–54.

[18] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 252–261.

[19] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *CSMR-WCRE*. IEEE, 2014, pp. 164–173.

[20] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *ICSE*, 2013, pp. 382–391.

[21] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *EMSE*, vol. 21, no. 1, pp. 43–71, 2016.

[22] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE TSE*, vol. 43, no. 4, pp. 321–339, 2016.

[23] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *SQJ*, vol. 25, no. 1, pp. 235–272, 2017.

[24] G. Catolino, D. Di Nucci, and F. Ferrucci, "Cross-project just-in-time bug prediction for mobile apps: An empirical assessment," in *6th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, 2019.

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

[26] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "Multi: Multi-objective effort-aware just-in-time software defect prediction," *IST*, vol. 93, pp. 1–13, 2018.

[27] K. Zhu, N. Zhang, S. Ying, and D. Zhu, "Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network," *IET Software*, 2020.

[28] D. Falessi, J. Huang, L. Narayana, J. F. Thai, and B. Turhan, "On the need of preserving order of data when validating within-project defect classifiers," *EMSE*, vol. 25, no. 6, pp. 4805–4830, 2020.

[29] M. Harman, S. Islam, Y. Jia, L. Minku, F. Sarro, and K. Srivisut, "Less is more: Temporal fault predictive performance over multiple hadoop releases," in *SSBSE*, 2014, pp. 240–246.

[30] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *ICSE*, vol. 2, 2015, pp. 99–108.

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

[32] S. Tabassum, L. L. Minku, and D. Feng, "Cross-project online just-in-time software defect prediction – supplementary material," 2022.

[33] L. L. Minku, "Commitguru-chinese: First release," 2020. [Online]. Available: https://zenodo.org/record/3684635

[34] S. Flint, J. Chauhan, and R. Dyer, "Escaping the time pit: Pitfalls and guidelines for using time-based git data," in *MSR*, 2021, pp. 85–96.

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

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

[37] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2018.

[38] Q. Zhu, "On the performance of matthews correlation coefficient (MCC) for imbalanced dataset," *Pattern Recognition Letters*, vol. 136, pp. 71–80, 2020.

[39] M. Duarte, "detecta: A python module to detect events in data," https://github.com/demotu/detecta, 2020.

[40] N. Mittas and L. Angelis, "Ranking and clustering software cost estimation models through a multiple comparisons algorithm," *IEEE TSE*, vol. 39, no. 4, pp. 537–551, 2012.

[41] T. Menzies, Y. Yang, G. Mathew, B. Boehm, and J. Hihn, "Negative results for software effort estimation," *EMSE*, vol. 22, no. 5, pp. 2658–2683, 2017.

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

[43] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[44] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 2000.

[45] R. Moser, W. Pedrycz, and S. G., "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *ICSE*, 2008.

[46] Y. Fan, X. Xia, D. Costa, D. Lo, A. Hassan, and S. Li, "The impact of mislabelled changes by SZZ on just-in-time defect prediction," *IEEE TSE*, vol. 47, 2021.

Sadia Tabassum is a final year PhD candidate at the University of Birmingham (UK). She received the MSc degree in Computer Science from the University of Leicester (UK), in 2016. Her research interests include software engineering, software defect prediction, machine learning, online learning, transfer learning, class imbalanced learning and evolutionary optimisation.

Leandro L. Minku is an Associate Professor at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer in Computer Science at the University of Leicester (UK), and a Research Fellow at the University of Birmingham (UK). He received the PhD degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests include machine learning for software engineering, machine learning for non-stationary environments / data stream mining, class imbalance learning, ensembles of learning machines and search-based software engineering. Among other roles, Dr. Minku is Associate Editor-in-Chief for Neurocomputing, and Associate Editor for IEEE Transactions on Neural Networks and Learning Systems, Empirical Software Engineering and Journal of Systems and Software.

Danyi Feng is the founder of XiLiuTech. She received the MSc degree in Artificial Intelligence from the University of Edinburgh (UK) in 2008, and the BEng degree in Human Interactive Systems from the University of Birmingham (UK) and in Software Engineering from the Huazhong University of Science and Technology (China) in 2007. Ms. Feng's main interests are the application of artificial intelligence technology, software quality assurance and life science.