# Supplemental Material

## A Procedure to Continuously Evaluate Predictive Performance of Just-In-Time Software Defect Prediction Models During Software Development

Liyan Song, Leandro L. Minku, *Senior Member, IEEE*

## I. Summary of Notations

| Notation | Description |
|---|---|
| $X_u$ | Input features describing the software change at (commit or test) time step $u$ |
| $y_u$ | True label of $X_u$, 1 for defect-inducing software change and 0 for clean change |
| $y^*_{u,t}$ / $y^*_{u,T}$ | Observed label of the software change at commit time step $u$ and labeled at a later time step $t$ (timestamp $T$) |
| $\widehat{y}_t$ | Predicted label of a software change at time step $t$ |
| $\mathcal{M}_{t,W}(\cdot)$ | JIT-SDP model produced at time step $t$ with a waiting time $W$ |
| $\mathbb{T}^{s,*}$ | Training data stream |
| $\mathbb{C}$ | Commit data stream |
| $\mathbb{E}$ | Test (evaluation) data stream |
| $\mathbb{E}^s$ | Test data stream based on surrogate time steps |
| $\mathbb{E}^{s,*}$ | Test data stream based on surrogate time steps and observed labels |
| $E_{cn}$ | True continuous performance of a JIT-SDP model for the whole test data stream $\mathbb{E}$ |
| $E^s_{cn}$ | Estimated continuous performance of a JIT-SDP model based on surrogate time steps for the whole test data stream $\mathbb{E}^s$ |
| $E^{s,*}_{cn}$ | Estimated continuous performance of a JIT-SDP model based on surrogate time steps and observed labels for the whole test data stream $\mathbb{E}^{s,*}$ |
| $\eta_{cn}$ | Continuous label noise associated to a waiting time |
| $\Delta_{cn}(\eta)$ | Validity of the continuous performance evaluation given the amount of label noise $\eta$ |
| $\Delta_{cn}(W)$ | Validity of continuous performance evaluation, given the waiting time $W$ |
| $\Theta_{cn}(W)$ | Validity of continuous model ranking in JIT-SDP, given the waiting time $W$ |
| $T$, $t$ | Uppercase letters such as $T$, $U$ and others will be used to denote a Unix timestamp $T$, whereas lowercase letters such as $t$, $u$ and others denote their corresponding time step |

## II. Mathematical Formulation of the Training Process

As true labels of software changes are usually unknown in practice due to verification latency, observed labels are actually used for training JIT-SDP models. The training data stream is constructed according to the timestamp when each software change receives its observed label.

We use $y^*_{u,t}$ to denote the label observed at time step $t$ for the software change produced at commit time step $u$, where $U < T$ and "$*$" is used to indicate that this is an observed label. We can use $y^*_{u,T}$ to denote the same label. We will denote an observed label in the form of $y^*_{u,t}$ or $y^*_{u,T}$ when we need to emphasize the time step or the Unix timestamp of the labeling time. We use the value 0 to indicate clean and 1 to indicate defect-inducing.

In the training process, given a software change received at commit time step $u$, if it is not found to be defect-inducing until Unix timestamp $T = U + W$, where $W$ is the waiting time parameter, $y^*_{u,T}$ is labeled as clean, and its corresponding example $(X_u, 0)$ can be used for training at training time step $t$. If it is found to be defect-inducing at a Unix timestamp $T' < T$, then $y^*_{u,T'}$ is labeled defect-inducing. Its corresponding example $(X_u, 1)$ can then be used for training at time step $t'$. It may also happen that the software change is first labeled as clean at timestamp $T$, but later on is found to have induced a defect at timestamp $T'' > T$. This happens when the waiting time is not enough to correctly label a training example, resulting in this example being mistakenly labeled as clean and then found to be defect-inducing afterwards. In such case, a clean training example $(X_u, 0)$ is first produced and made available for training at time step $t$, and then a new defect-inducing training example $(X_u, 1)$ is produced and made available for training at time step $t''$.

Given a waiting time $W$, the timestamp $T_s = T - W$ denotes the *surrogate timestamp* of $T$, which is $W$ length of time earlier than the current timestamp $T$. An example is given in the illustrative Figure 1. The term 'surrogate' is used to indicate

L. Song is with Department of Computer Science and Engineering, Southern University of Science and Technology, China. E-mail: songly@sustech.edu.cn.
L.L. Minku is with the School of Computer Science, University of Birmingham, UK. Email: L.L.Minku@bham.ac.uk.

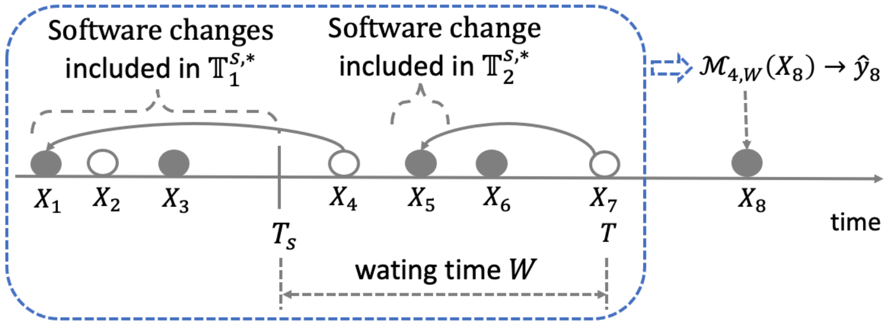

Software changes included in $\mathbb{T}_1^{s,*}$
Software change included in $\mathbb{T}_2^{s,*}$
$\mathcal{M}_{4,W}(X_8) \rightarrow \hat{y}_8$
$X_1$
$X_2$
$X_3$
$X_4$
$X_5$
$X_6$
$X_7$
$X_8$
$T_s$
$T$
time
wating time $W$

Fig. 1. Illustration of the train-prediction process in JIT-SDP. Software changes (input features) are denoted by '$X$' followed by a number indicating their order of commit time. Gray (white) circles denotes defect-inducing (clean) software changes. An arrow links the defect-inducing software change and the one that fixes the defect. At a given timestamp $T$, the first part of the training set is composed of software changes that are $W$ days or older. Their labels will be determined based on the labeling procedure explained in Section 3.3 of the paper, leading to labeled examples $(X_1, 1)$, $(X_2, 0)$, $(X_3, 0)$. The second part contains software changes that have been committed during the most recent $W$ days, but have already been found to be defect-inducing by timestamp $T$, leading to labeled example $(X_5, 1)$. The most up-to-date model $\mathcal{M}_{4,W}$ used to predict software change $X_8$ is the model that has been updated with this whole training set, which is composed of 4 training examples $(X_1, 1)$, $(X_2, 0)$, $(X_3, 0)$ and $(X_5, 1)$.

that most[1] knowledge about labeled examples we have access to in order to train a model at timestamp $T$ comes from examples labeled until timestamp $T_s$. So, this knowledge at timestamp $T_s$ has to be used in an attempt to train a model to represent the defect generating process at timestamp $T$. Whether or not such model will appropriately represent the process active at $T$ depends on whether or not concept drift has occurred.

Based on the defined surrogate time step, we can set up the training data stream at an arbitrary Unix timestamp $T$ when a JIT-SDP model can be trained by using (1) software changes that have at least $W$ waiting time for their observed labels and (2) those that are found defect-inducing after the surrogate timestamp $T_s = T - W$ but before $T$. The first part can be formulated as

$$\mathbb{T}_1^{s,*} = \{(X_u, y_{u,T}^*)\}_{u=1}^{t_s},$$

where $\mathbb{T}$ is used to represent this is a training data stream, $\square^{s,*}$ is used to indicate that this data stream is based on observed labels of software changes at surrogate time steps, $X_u$ denotes input features describing the software change produced at commit time step $u$, $y_{u,T}^*$ denotes the label for $X_u$ observed at $T$ and $t_s$ denotes the surrogate time step of $T$. The second part can be formulated as

$$\mathbb{T}_2^{s,*} = \{(X_u, 1)\}_{u=t_s+1}^{t},$$

where $t$ is the current training time step. Altogether, the training data stream at $T$ is formulated as

$$\mathbb{T}^{s,*} = \mathbb{T}_1^{s,*} \cup \mathbb{T}_2^{s,*}.$$

In this way, the whole training data stream is produced and updated with time. The two parts of training data stream at timestamp $T$ are illustrated in the blue box of Figure 1. In the illustrative example given in this figure, $\mathbb{T}_1^{s,*} = \{(X_1, 1), (X_2, 0), (X_3, 0)\}$. Note that $X_3$ is labeled as clean at this stage, because no defect has found to be induced by it yet. In addition, $\mathbb{T}_2^{s,*} = \{(X_5, 1)\}$ contains a single software change, because only this software change has already been found to induce defects within the most recent $W$ days.

We use $\mathcal{M}_{t,W}(\cdot)$ to denote a JIT-SDP model that is trained at time step $t$ based on the training data stream $\mathbb{T}^{s,*}$ with waiting time $W$, which can then be used to predict the label of new software changes committed at later time steps. The calligraphic $\mathcal{M}$ is used to indicate this is a JIT-SDP model. Different waiting times may produce different training data streams and consequently different JIT-SDP models. In the example shown in Figure 1, the JIT-SDP model produced at timestamp $T$ is $\mathcal{M}_{4,W}$. The index 4 is used here because only 4 training examples have been used to produce this model, i.e., the training time step corresponding to the timestamp $T$ in this case is $t = 4$, despite the commit time step being $t = 7$.

We should also note that how exactly the model $\mathcal{M}_{t,W}$ is trained on the data stream depends on the underlying machine learning algorithm. Several online learning algorithms will be able to simply update a given model $\mathcal{M}_{t,W}$ with new training examples that become available over time, instead of having to retrain on past training examples whenever the model needs to be updated [1].

[1]We say "most" and not "all" knowledge here, because examples could be labeled as defect-inducing after $T_s$ and used for training, if defects are found to be associated to them later on.

## III. Mathematical Formulation of the Commit-Time Prediction Process

Software changes are predicted over time according to a commit data stream

$$\mathbb{C} = \{X_u\}_{u=1}^{t},$$

where the mathematical bold $\mathbb{C}$ is used to indicate that this is a commit data stream, $X_u$ denotes the input features describing the software change at time step $u$ and $t$ is the most recent commit time step. Note that the true labels $\{y_u\}$ are unavailable at commit time in realistic scenarios.

When a new software change $X_v$ is committed at time step $v = t + 1$, it is predicted as defect-inducing or clean using the JIT-SDP model created at the most recent *training* time step $t'$, where $T' < V$. This can be formulated as

$$\widehat{y}_{t+1} = \mathcal{M}_{t',W}(X_{t+1}),$$

where $\mathcal{M}_{t',W}(\cdot)$ denotes the JIT-SDP model trained at time step $t'$ with the waiting time $W$ adopted in the training process and $\widehat{y}_{t+1}$ denotes the predicted label. For example, in Figure 1, model $\mathcal{M}_{4,W}$ produced at training time step $t = 4$ is the model used to predict the software change $X_8$.

This mechanism ensures that chronology is respected. No training example being actually unavailable at Unix timestamp $T$ would be used to train the JIT-SDP model, which will then be used to predict the software change committed at Unix timestamp $T$.

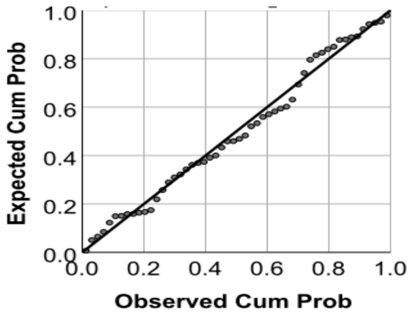
## IV. Checking the Assumptions of Linear Regression

To use the linear regression to investigate the impact of label noise on the validity of continuous performance evaluation, we need to check to what extent the assumptions of linear regression, including *normality*, *homoscedasticity* and absence of *collinearity* (in the bi-variate case), can be satisfied for answering RQ2 (and RQ3) [2], [3]. Similarly, some of the analysis performed for RQ1, namely the investigation of the positive impact of continuous verification latency on continuous label noise, also makes use of linear regression, and so such assumptions also need to be checked. Note that we do not need to check the assumption *independence of residuals* (i.e., non-autocorrelation of the residuals) as the linear regression is being conducted across the data streams (i.e., projects) rather than within a data stream (i.e., time series data) [4].

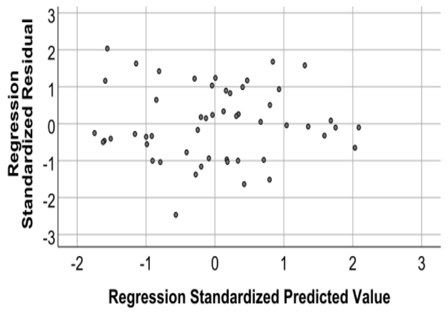
Linear regression is robust to violations of the normality assumption when the sample size is not small and to mild violations of the homoscedasticity assumption [4], [5]. There are a few different ways to check the assumptions and there generally is no uniformly optimal approach [5]. A scatter plot of residuals versus predicted values is a good way to check for homoscedasticity, and as long as there is no clear pattern in the distribution, such as a cone-shaped pattern, the data satisfies the homoscedasticity fairly well [2], [4]. Similarly, P-P plots are frequently used to judge normality. In a bi-variate case, Spearman correlation between the two independent variables is a typical indicator for the *collinearity* assumption, for which a non-strong correlation (<0.70) can indicate a non-serious problem especially given the sample size is not small [6].

### A. Analysis for RQ1

In Section 8.1, we used linear regression to investigate the impact of continuous verification latency and waiting time on continuous label noise. Figures 2(a) and (b) illustrate the P-P plot to check the *normality* assumption and the scatter plot to check the *homoscedasticity* assumption for this analysis, respectively. As we can see, the two assumptions are satisfied well.



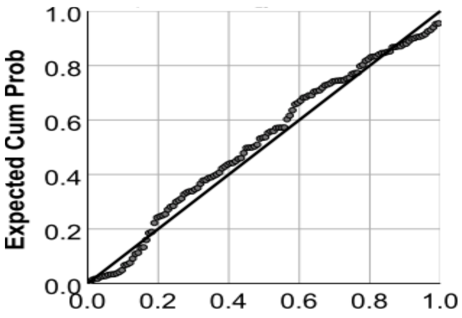
(a) P-P plot to check *normality* assumption.



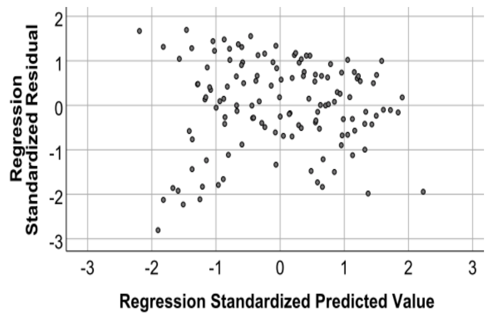
(b) Scatter plot to check *homoscedasticity* assumption.

Fig. 2. To answer RQ1, checking the assumptions of *normality* and *homoscedasticity* for using linear regression analysis to investigate the relation between continuous verification latency and continuous label noise.

Spearman correlation between the continuous verification latency and waiting time for checking the *collinearity* assumption is 0.055 (very weak). Therefore, we can use bi-variate linear regression to investigate the impact of the two independent variables on the continuous label noise.



(a) P-P plot to check *normality* assumption.



(b) Scatter plot to check *homoscedasticity* assumption.

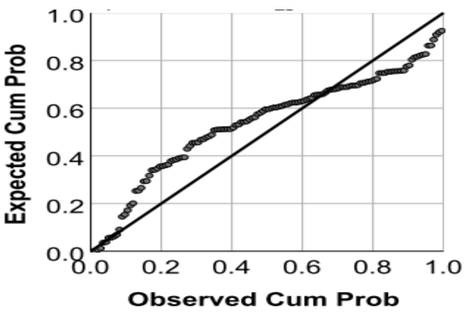
Fig. 3. To answer RQ2, checking the assumptions of *normality* and *homoscedasticity* for using linear regression analysis to investigate the impact of test (and training) label noise on the validity of continuous performance evaluation.

## B. Analysis for RQ2

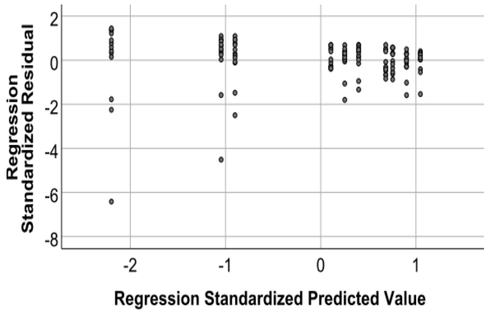
In Section 8.2.1, we have checked the assumptions of using linear regression analysis to investigate the impact of both continuous training and continuous test label noise on the validity of continuous performance evaluation. Figures 3 (a) and (b) illustrate the P-P plot to check the the *normality* assumption and the scatter plot to check and *homoscedasticity* assumption, respectively. As we can see, violations to these two assumptions are not large. Spearman correlation between continuous training and continuous test label noise for checking the *collinearity* assumption is 0.761 (strong), showing strong correlation between the two independent variables. Therefore, we have conducted ridge regression to investigate the impact of two independent variables on the validity of continuous performance evaluation, as it is a technique for analyzing multiple regression data that suffer from collinearity.

## C. Analysis for RQ3

Similar to RQ2, in order to use linear regression to investigate the impact of waiting time on the validity of continuous performance evaluation, we need to check to what extent the assumptions of linear regression are satisfied for answering RQ3.



(a) P-P plot to check *normality* assumption.

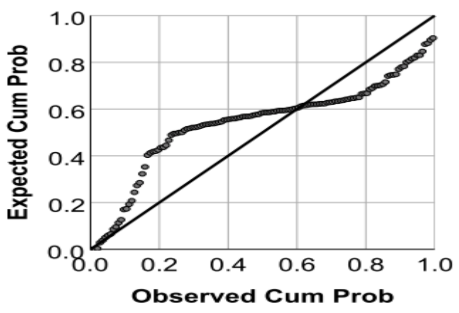


(b) Scatter plot to check *homoscedasticity* assumption.

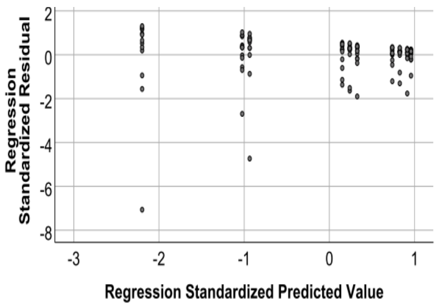
Fig. 4. To answer RQ3, checking the assumptions of *normality* and *homoscedasticity* for using linear regression analysis to investigate the impact of training and test waiting time on the validity of continuous performance evaluation.

In Section 8.3.1, we have checked the assumptions of using linear regression to investigate the impact of both training and test waiting time on the validity of continuous performance evaluation. Figures 4 (a) and (b) illustrate the P-P plot to check the *normality* assumption and the scatter plot to check the *homoscedasticity* assumption, respectively. As we can see the violation to normality is a bit larger than for RQ1-2, but can still be considered acceptable given linear regression's robustness to the normality assumption mentioned in the beginning of Section IV. Spearman correlation between the training and the test waiting time for checking the *collinearity* assumption is 0.477 (moderate). This moderate correlation can be partially explained by the requirement that the training waiting time should be no smaller than the test waiting time. As the Spearman correlation is smaller than 0.70 (i.e., not strong), we have conducted bi-variate linear regression to investigate the impact of the two independent variables on the validity of continuous performance evaluation.

In the rest of this section, we will check to what extent the assumptions of linear regression are satisfied when conducting further analysis on RQ3 in view of the impact of concept drift on the estimation of performance evaluation. In Section 8.3.3, we have checked the assumptions of using linear regression to investigate the impact of waiting time on the validity of continuous



(a) P-P plot to check *normality* assumption.



(b) Scatter plot to check *homoscedasticity* assumption.

Fig. 5. To further investigate RQ3, checking the assumptions of *normality* and *homoscedasticity* for using linear regression analysis to investigate the impact of training and test waiting time in view of an obsolete performance evaluation on the validity of continuous performance evaluation.

performance evaluation in view of concept drift. Figures 5 (a) and (b) illustrate the P-P plot to check the *normality* assumption and the scatter plot to check the *homoscedasticity* assumption, respectively. The violation to normality is a bit larger than for RQ1-2, but can still be considered acceptable given linear regression's robustness to the normality assumption as discussed previously. As we still use the training and the test waiting time as the independent variables, the result for the *collinearity* assumption remains the same as it is for answering RQ3.

## V. Chosen Parameter Settings for Each Dataset (Data Stream)

TABLE I
The parameter choices for each of the datasets investigated in this study given a training waiting time.

(a) Training waiting time 15 days.

| Dataset | Decay Factor | Ensemble Size |
|---|---|---|
| Brackets | 0.99 | 20 |
| Broadleaf | 0.99 | 20 |
| Camel | 0.99 | 10 |
| Fabric | 0.99 | 10 |
| jGroup | 0.99 | 10 |
| Nova | 0.9 | 5 |
| Django | 0.99 | 20 |
| Rails | 0.99 | 20 |
| Corefx | 0.9 | 20 |
| Rust | 0.99 | 20 |
| Tensorflow | 0.99 | 20 |
| VScode | 0.9 | 5 |
| wp-Calypso | 0.9 | 5 |

(b) Training waiting time 30 days.

| Dataset | Decay Factor | Ensemble Size |
|---|---|---|
| Brackets | 0.99 | 5 |
| Broadleaf | 0.9 | 5 |
| Camel | 0.9 | 5 |
| Fabric | 0.9 | 20 |
| jGroup | 0.99 | 20 |
| Nova | 0.9 | 5 |
| Django | 0.9 | 10 |
| Rails | 0.9 | 20 |
| Corefx | 0.9 | 20 |
| Rust | 0.9 | 5 |
| Tensorflow | 0.99 | 5 |
| VScode | 0.9 | 5 |
| wp-Calypso | 0.9 | 5 |

(c) Training waiting time 90 days.

| Dataset | Decay Factor | Ensemble Size |
|---|---|---|
| Brackets | 0.99 | 5 |
| Broadleaf | 0.99 | 5 |
| Camel | 0.99 | 5 |
| Fabric | 0.9 | 5 |
| jGroup | 0.9 | 5 |
| Nova | 0.9 | 5 |
| Django | 0.9 | 5 |
| Rails | 0.9 | 5 |
| Corefx | 0.9 | 5 |
| Rust | 0.9 | 5 |
| Tensorflow | 0.9 | 5 |
| VScode | 0.9 | 5 |
| wp-Calypso | 0.9 | 5 |

(d) Training waiting time 90 days.

| Dataset | Decay Factor | Ensemble Size |
|---|---|---|
| Brackets | 0.9 | 5 |
| Broadleaf | 0.99 | 5 |
| Camel | 0.99 | 5 |
| Fabric | 0.9 | 20 |
| jGroup | 0.99 | 20 |
| Nova | 0.9 | 5 |
| Django | 0.9 | 5 |
| Rails | 0.9 | 5 |
| Corefx | 0.9 | 5 |
| Rust | 0.9 | 5 |
| Tensorflow | 0.9 | 5 |
| VScode | 0.9 | 5 |
| wp-Calypso | 0.9 | 5 |

## References

[1] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in non-stationary environments: A survey," *IEEE CIM*, vol. 10, no. 4, pp. 12–25, 2015.
[2] S. Solution, "Assumptions of multiple linear regression," https://www.statisticssolutions.com/assumptions-of-multiple-linear-regression/.
[3] C. Dhakal, "Interpreting the basic outputs (SPSS) of multiple linear regression," *International Journal of Science and Research*, pp. 1448–1452, 01 2018.

[4] A. F. Ernst and C. J. Albers, “Regression assumptions in clinical psychology research practice: A systematic review of common misconceptions,” PeerJ 5:e3323; DOI 10.7717/peerj.3323, 2017.
[5] C. S. and H. AS, *Regression analysis by example. Third Edition*. New York: Routledge: Hoboken: John Wiley and Sons, 2003.
[6] T. Baguley, *Serious Stats: A guide to advanced statistics for the behavioral sciences*. Palgrave, 2012.