Noname manuscript No.
(will be inserted by the editor)

Supplemental Material 2

On the Validity of Retrospective Predictive Performance Evaluation Procedures in Just-In-Time Software Defect Prediction

Liyan Song · Leandro L. Minku* · Xin Yao*

the date of receipt and acceptance should be inserted later

Abstract This document presents additional information of the submitted paper "On the Validity of Retrospective Predictive Performance Evaluation Procedures in Just-In-Time Software Defect Prediction".

We report a summary table of related works discussed in Section 3.2 and 3.3 of this paper to assist readers to better track the related works discusses in this paper. In the table, the "ML approach" column lists the learning algorithms that were proposed or adopted in the literature. The "respect chronology" and "respect VL (Verification Latency)" columns report whether or not the study had taken the chronology and verification latency among software changes into consideration for the evaluation purpose, respectively. For the literature that respected the chronology, we also report the evaluation methodology taken in the study. The "feature and performance results" column summarizes related conclusions with respect to the effects of different features on predictive performance and the achieved predictive performance of the investigated learning machines. When the answer to each question is not applicable in a cell, "N/A" is reported.

Liyan Song (songly@sustech.edu.cn) · Xin Yao (xiny@sustech.edu.cn)
Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, China and Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, China.

Leandro L. Minku (L.L.Minku@bham.ac.uk)
School of Computer Science, the University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK

* the corresponding authors.

Table 1: Summary of related work in Section 3.2 and 3.3 of the paper. The "ML approach" column lists the learning algorithms that were proposed or adopted in the literature. The "respect chronology" and "respect VL (Verification Latency)" columns report whether or not the study had taken the chronology and verification latency among software changes into consideration for the evaluation purpose, respectively. For the literature that respected the chronology, we also report the evaluation methodology taken in the study. The "feature and performance results" column summarizes related conclusions with respect to the effects of different features on predictive performance and the achieved predictive performance of the investigated learning machines. When the answer to each question is not applicable in a cell, "N/A" is reported.

Paper	ML Approach	Online or Offline	Respect Chronology?	Respect VL?	# Projects	Feature and Performance Results
Mockus and Weiss (2000)	Logistic regression	Offline	Yes, but unclear.	No	1 proprietary project for telephone switching system called 5ESS (Five Element Switching System)	Feature result: Change diffusion and developer experience are essential to predicting failures. Performance result: N/A
Śliwerski et al. (2005)	N/A	N/A	N/A	N/A	2 open-source projects: MOZILLA and ECLIPSE	Feature result: The larger a change, the more likely it is to induce a fix. Performance result: N/A
Kim et al. (2008)	Support Vector Machine (SVM)	Offline	No	No	12 open-source projects: Apache, HTTP 1.3, Bugzilla, Columba, Gaim, GForge, JEdit, Mozilla, Eclipse, Plone, PostgreSQL, Scarab, and Subversion	Feature result: This work is the first to classify software changes based on extracted features from the software configuration management repository, showing potential to achieve promising prediction performance. Performance result: The trained classifier can classify changes as buggy or clean, with a 78% accuracy and a 60% buggy change recall on average.
Eyolfson et al. (2011)	N/A	Offline	No	No	2 open-source projects: Linux kernel and PostgreSQL	Feature result: Late-night commits are buggier than average; morning commits are less buggy. The bugginess of commits per day-of-week varies for different software projects. Developers who commit to a project on a daily basis write fewer buggy commits for that project, while day-job developers are more likely to produce bugs. Performance result: N/A

Table 1 continued from previous page

Paper	ML Approach	Online or Offline	Respect Chronology?	Respect VL?	# Projects	Feature and Performance Results
Kamei et al. (2013)	Logistic regression	Offline	No	No	6 open-source projects (Bugzilla, Columba, Eclipse JDT, Eclipse Platform, Mozilla, PostgreSQL) and 5 larger commercial properity projects (C-1, C-2, C-3, C-4, C-5)	Feature result:Different factors are effective for open source and for commercial projects. The number of files and whe ther or not the change fixes a defect are risk-increasing factors and the average time interval between the last and the current change is a risk-decreasing factor for open source projects. The size factors are consistently important for commercial projects. Performance result: The change-level prediction model can predict changes with 68% accuracy, 34% precision, and 64% recall. When factoring in the effort required to review the changes into the predictions, using only 20% of all effort suffices to identify 35% of all predicteddefect-inducing changes.
Chen et al. (2014)	N/A	N/A	N/A	N/A	20 Apache open-source projects: Abdera, Accumulo, Axis, Bookkeeper, Camel, Cassandra, Cocoon, CouchDB, CXF, Derby, Felix, Hive, OpenEJB, OpenJPA, Pig, Qpid, Shiro, Thrift, Wicket, Wink	Feature result: N/A Performance result: Typically 33% of the defects introduced in a version were reported in future versions as dormant bugs and performance evaluation that ignored dormant bugs could be misleading. Dormant bugs fixes are more complex than non-dormant bugs.
Tan et al. (2015)	Resampling; Bayes, instance-based learning, boosting, 1NN, and SVM	Online	Yes, holdout	Yes	6 open-source projects (Linux, PostgreSQL, Xorg, Eclipse, Lucene, Jackrabbit) and 1 proprietary project from Cisco	Feature result: N/A Performance result: Overlooking the chronology would lead to defect prediction models with deceptively better predictive performance than they could achieve in practice when chronology must be respected. Both resampling techniques and updatable classification improve the precision by 12.2% to 89.5% or 6.4% to 34.8%.

Table 1 continued from previous page

Paper	ML Approach	Online or Offline	Respect Chronology?	Respect VL?	# Projects	Feature and Performance Results
Misirli et al. (2016)	Regression and Random forest	Offline	No	No	6 open-source projects: GIMP1, Maven 22, Perl3, PostgreSQL4, Ruby on Rails5, and Rhino	Feature result:The lines of code added, the number of developers who worked on a change, and the number of prior modifications on the files modified during a change are the best indicators of high impact fix-inducing changes. Performance result: Based on the change factors designed in this paper, the adopted model can predict 56% to 77% of HIFCs with an average false alarm (misclassification) rate of 16%. The specilized model can provide inspection effort savings of 34% over the state-of-the-art models (at that moment).
Kamei et al. (2016)	Random forest; cross-project approaches	Offline	No	No	11 open-source projects, of which 6 (Bugzilla, Columba, Eclipse JDT, Mozilla, Eclipse Platform, PostgreSQL) are provided by Kamei et al. (2013) and 5 well-known ones(Gimp, Maven-2, Perl,Ruby on Rails, Rhino)	Feature result: The within-project performance of a JIT model is not a strong indicator of its performance in a cross-project context. Performance result: Predictive performance of JIT-SDP tends to imporove when using approaches that (1) select models trained using other projects that are similar to the testing project, (2) combine the data of several other projects to produce a larger pool of training data, and (3) combine the models of several other projects to produce an ensemble model.
McIntosh and Kamei (2018)	Nonlinear logistic regression model	Online	Yes, holdout	No	2 open-source projects: QT and OPENSTACK	Feature result:N/A Performance result: JIT models lose a large proportion of predictive performance one year after being trained. The magnitude of the importance scores of the six studied families of code change properties fluctuate as systems evolve. These fluctuations can lead to consistent overestimates (and underestimates) of the future impact of the studied families of code change properties.
Cabral et al. (2019)	Oversampling Rate Boosting (ORB)	Online	Yes, prequential	Yes	10 open-source projects (Fabric8, JGroups, Camel, Tomcat, Brackets, Neutron, Spring-integration, Broadlead, Nova, NPM)	Feature result: N/A Performance result: ORB was able to deal with class imbalance evolution, being top ranked both in terms of G-Mean and differences in recalls. In particular, its difference in recals was up to 45.38% lower than OOB's. Analyses of the typical defect discovery delay found that a waiting time of 90 days is adequate for the projects investigated in the paper.

Table 1 continued from previous page

Paper	ML Approach	Online or Offline	Respect Chronology?	Respect VL?	# Projects	Feature and Performance Results
Hoang et al. (2019)	DeepJIT with Convolutional Neural Network	Offline	No	No	2 open-source projects: QT and OPENSTACK	Feature result: Features extracted by deep networks. Performance result: Compared to the SOTA baseline (DBNJIT), the best variant of DeepJIT (DeepJIT-Combined) achieves improvements of 10.50%, 10.36%, and 11.02% in the project QT and 9.51%, 13.69%, 12.22% in the project OPENSTACK in terms of the Area Under the Curve (AUC).
Hoang et al. (2020)	CC2Vec with Neural Network	Offline	No	No	2 open-source projects: QT and OPENSTACK	Feature result: Features extracted by deep networks. Performance result: Approaches using or augmented with CC2Vec embeddings outperform existing SOTA approaches that do not use the embeddings.
Tabassum et al. (2020)	ORB; the cross-project techniques	Online	Yes, prequential	Yes	The same 10 open-source projects as Cabral et al. (2019) plus another 3 proprietary projects	Feature result: N/A Performance result: CP approaches can improve predictive performance over WP approaches trained only with WP data. The All-in-One and Filtering CP approaches are particularly helpful during the initial phase of the project when there is not enough WP data. These approaches also helped to reduce sudden drops in performance of the predictive model after the initial phase of the project. They also improved overall predictive performance compared to the WP approach. Even though the ensemble approach was shown to perform well in offline learning, it was the worst approach when considering a realistic online learning scenario.
Pornprasit and Tantithamthavorn (2021)	A JIT Defect Prediction Approach at the Commit and Line Levels (JITLine)	Offline	Yes, holdout	No	2 open-source projects: QT and OPENSTACK	Feature result: N/A Performance result: The JITLine approach is better, more cost-effective, faster and more fine-grained than the SOTA JIT defect prediction approaches (EARL, DeepJIT, and CC2Vec).

Table 1 continued from previous page

Paper	ML Approach	Online or Offline	Respect Chronology?	Respect VL?	# Projects	Feature and Performance Results
Song and Minku (2023)	Oversampling Online Bagging (OOB) with Hoeffding trees	Online	Yes, prequential	Yes	13 open-source projects (Brackets, Broadleaf, Camel, Fabric8, JGroup, Nova, Django, Rails, Corefx, Rust, Tensorflow, VScode, wp-Calypso)	Feature result: N/A Performance result: Smaller evaluation waiting times were typically associated to considerably larger amount of label noise. Larger amount of evaluation label noise was associated to slightly worse validity of the continuous performance evaluation procedure. Smaller evaluation waiting times were themselves associated to better validity of the continuous performance evaluation procedure, probably due to concept drift. Different evaluation waiting times usually did not change the rankings of the JIT-SDP models.

References

Cabral GG, Minku LL, Shihab E, Mujahid S (2019) Class imbalance evolution and verification latency in just-in-time software defect prediction. In: International Conference on Software Engineering, Montreal, QC, Canada, pp 666–676

Chen TH, Nagappan M, Shihab E, Hassan AE (2014) An empirical study of dormant bugs. In: Conference on Mining Software Repositories, Hyderabad, India, pp 82–91

Eyolfson J, Tan L, Lam P (2011) Do time of day and developer experience affect commit bugginess? In: International Workshop on Mining Software Repositories, Waikiki, Honolulu, HI, USA, pp 153–162

Hoang T, Khanh Dam H, Kamei Y, Lo D, Ubayashi N (2019) DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction. In: International Conference on Mining Software Repositories, Montreal, QC, Canada, pp 34–45

Hoang T, Kang HJ, Lo D, Lawall J (2020) CC2Vec: Distributed representations of code changes. In: International Conference on Software Engineering, Seoul, South Korea, pp 518–529

Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N (2013) A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering 39(6):757–773

Kamei Y, Fukushima T, Mcintosh S, Yamashita K, Ubayashi N, Hassan AE (2016) Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering 21(5):2072–2106

Kim S, Whitehead EJ, Zhang Y (2008) Classifying software changes: Clean or buggy? IEEE Transactions on Software Engineering 34(2):181–196

McIntosh S, Kamei Y (2018) Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. IEEE Transactions on Software Engineering 44(5):412–428

Misirli AT, Shihab E, Kamei Y (2016) Studying high impact fix-inducing changes. Empirical Software Engineering Journal 21(2):605–641

Mockus A, Weiss DM (2000) Predicting risk of software change. Bell Labs Technical Journal 5(2):169–180

Pornprasit C, Tantithamthavorn CK (2021) JITLine: a simpler, better, faster, finer-grained just-in-time defect prediction. In: International Conference on Mining Software Repositories, Madrid, Spain, pp 369–379

Śliwerski J, Zimmermann T, Zeller A (2005) When do changes induce fixes? ACM sigsoft software engineering notes 30(4):1–5

Song L, Minku LL (2023) A procedure to continuously evaluate predictive performance of just-in-time software defect prediction models during software development. IEEE Transactions on Software Engineering 49(2):646–666

Tabassum S, Minku LL, Feng D, Cabral GG, Song L (2020) An investigation of cross-project learning in online just-in-time software defect prediction. In: International Conference on Software Engineering, Seoul, South Korea, p 554–565

Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalance data. In: International Conference on Software Engineering, Florence, Italy, pp 99–108