Applied Computer Science, vol. 16, no. 3, pp. 5–15
doi:10.23743/acs-2020-17

Submitted: 2019-10-12
Revised: 2019-11-25
Accepted: 2020-09-07

Deep Neural Network, unlabeled dataset,
Just-In-Time defect prediction, unsupervised prediction

Saleh ALBAHLI[*]

A DEEP ENSEMBLE LEARNING METHOD FOR EFFORT-AWARE JUST-IN-TIME DEFECT PREDICTION

Abstract

Since the introduction of Just-in-Time effort aware defect prediction, many researchers are focusing on evaluating the different learning methods for defect prediction. To predict the changes that are defect-inducing, it is important for learning model to consider the nature of the dataset, its imbalance properties and the correlation between different attributes. In this paper, we evaluated the importance of dataset properties, and proposed a novel methodology for learning the effort aware just-in-time defect prediction model. We form an ensemble classifier, which consider the output of three individuals classifier i.e. Random forest, XGBoost and Deep Neural Network. Our proposed methodology shows better performance with 77% accuracy on sample dataset and 81% accuracy on different dataset.

1. INTRODUCTION

Reducing the number of failures in a software is important in order to produce the quality product. During software development, a software goes through many changes, and these changes are needed to be defect and error free. Since the introduction of Effort aware Just-in-Time (JIT) prediction by Mockus et al. (Mockus & Weiss, 2000), it is highly studied model for the better error detection mechanism. Mockus and Weiss used numerous change metrics for the prediction of the probability that the change will induce any defect in software instead of going through the lines of codes.

[*] Qassim University, College of Computer, Department of Information Technology, Saudi Arabia, 51452, Qassim, salbahli@qu.edu.sa

JIT defect prediction is of major practical value compared with conventional defect predictions at module. The JIT was coined by the Kamei et al. (Kamei et al., 2012) who put forward a method of checking the error based on raw metric which not only predicts the error out from the line of code under inspection, but also highlights the latent defect which can be detected at the check in time unlike other effort aware detection method (Zhou, Sun, Xia, Li & Chen, 2019). This method also reduces the tedious task of finding the author of the code as many people are involved over a module and doing the inspection at the check in time, where the change details are still being investigated, help make the debug very easy.

Much work is available on the JIT effort aware system using the file, package or method level for the defect prediction. However, there is still the need to accurately predict the defect using supervised, unsupervised and deep learning methods (Yu, Wen, Han & Hayes, 2018). In this paper, we proposed a novel methodology for the prediction of defects using the publicly available dataset for training few learning methods, and later we ensemble the output of each classifier to provide the final target prediction. In our proposed model, we used ensemble method classification utilizing random forest, XGBoost and deep neural network for model training. Therefore, our model try to take into account of the abstract features by using the deep ensemble technique, XGBoost and Random Forest, making it more robust and outperform the models available.

The rest of the paper is organized as follows. Section 2 review the existing techniques related to the effort aware JIT defect prediction. Proposed methodology is explained in detail in Section 3 followed by Experimental evaluation and result with discussion in section 4. Section 5 conclude the paper and state the possible future direction.

2. LITERATURE REVIEW

Effort-aware JIT defect prediction ranks the source code based on the probability of the defects and the effort to examine such variations. Effective and efficient defect prediction and detection algorithms help to find the defects accurately and, in less time with small effort. Such effort-aware models often help to allocate the software quality assurance tasks like code reviews and testing. Qiao et al., (Qiao & Wang, 2019) proposed a deep learning approach for the effort-aware JIT defect prediction. They used neural network and deep learning approach for the useful feature selection. They used ten numerical metrices of code changes and feed them to neural network to predict the presence of bug in the code change under review. They rank the code changes according to the benefit cost ratio, which is calculate beforehand by diving the likelihood of each code change by its size.

Yang et al. (Yang et al., 2016), state that many unsupervised effort aware JIT prediction models performs better than the state-of-the-art supervisor modes. They used only those prediction model that have a good scalability and low application cost (i.e. metrices modelling cost and collection cost). It put forward the idea of using the unsupervised learning technique and highlighted that building the prediction models do not need defected data for unsupervised model, as a consequence, incur a low building cost and a high application range (Liu, Yang, Xia, Yan & Zhang, 2018). Therefore, it would be more fitting for users to use unsupervised models in effort-aware JIT defect prediction especially when defect-inducing changes can be predicted well. In general, unsupervised models aggregate similar data-points and performs the modelling. Thus, the model has to develop and train effectively to automatically identify on its own to figure out information. It primarily deals with the unlabeled data.

There are many bug prediction models built with the historical metrices. Many studies have targeted coarse-grained (file and packages level) prediction. Hata et al., (Hata, Mizuno & Kikuno, 2012) stated that fine-grained prediction is challenges because it needs method level histories of existing version control system. They tackle the mentioned problem and developed a fine-grained prediction version control system and proved that fine-grained performs better than the coarse-grained prediction. On the other hand, Kamei et al. (Kamei et al., 2012), claim that the common finding in literature say that package level prediction normally outperforms fine-level predictions, does not hold true when the effort is considered. They show that package level prediction can be improved when file level prediction is performed and then lifting them on the package level instead of just collecting all the metrices at the package level.

Kamie et al., (Kamei, Matsumoto, Monden, Matsumoto, Adams & Hassan, 2010) stated that defect detection from file or package level is very time consuming and it makes the approach very ineffective for large software systems. They proved that instead of using file and package level for defect prediction, we should identify defect-prone software changes. The conducted large-scale study on six open source projects and 6 open source projects, shows the 68% average accuracy with 64% average recall for the proposed system. Only 20% of effort is needed to review the changes, and 35% of all defect-inducing changes were identified. This proposed model provides effort-reducing way to handle the risky changes and minimize the cost for the development of high-quality software.

The existing literature mainly focuses effort-aware JIT prediction using the data extracted based on developer and using the unsupervised models that aggregated the similar data-point to perform the modelling. In contrast to previous work, we focus on using the deep learning ensembling techniques for the defect prediction. Later, implementing Deep ensembling using the XGboost, Random Forest and Deep neural network and then averaging their output.

3. PROPOSED METHODOLOGY

In this paper, we proposed a methodology for effort aware just-in-time prediction using the ensemble method. Instead of using only supervised or unsupervised method for classification, we proposed to use the combination of deep learning method, supervised methods for the classification task. As shown in Figure 1, for this task, we used various multiple learning algorithms including deep neural network, XGBoost and Random Forest. Deep neural network can map the input data to the given labelled dataset representing the non-linear relationship. Unlike most conventional machine learning algorithm, deep neural network can detect the feature automatically without the human intervention. For DNN optimization, we used ADAM (adaptive moment estimation) optimization algorithm. It updates model weight iteratively based on the training data. We used the following parameter setting for ADAM optimization:

- *Learning Rate* = 0.001,
- *Decay Rate* = 0.9,
- *Exponential Decay Rate* = 2.099.

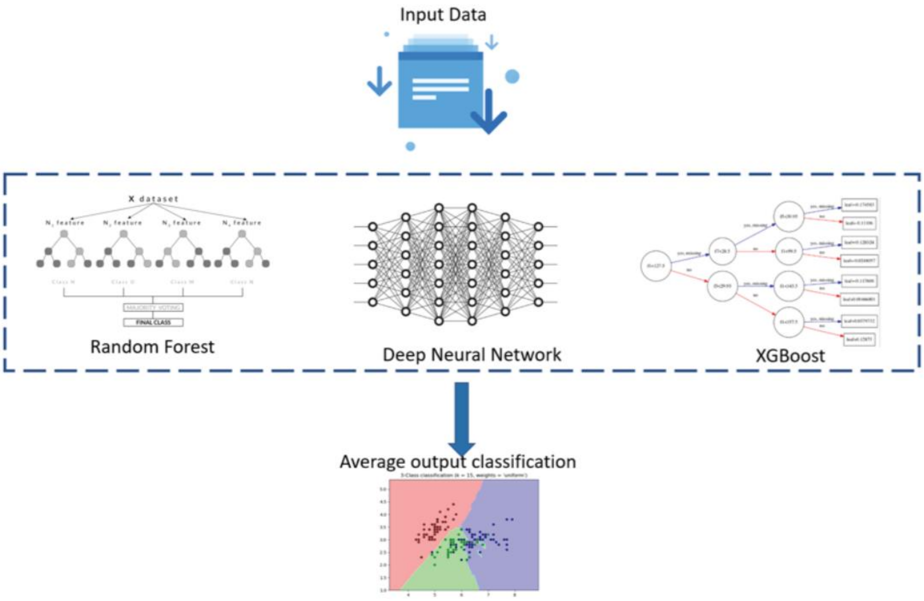

Fig. 1. Proposed Methodology: Effort aware just-in-time prediction using deep ensemble method

XGBoost (Chen & Guestrin, 2016) was initially developed with deep consideration of system optimization and principles in machine learning. The objective function *f* is:

$$\text{obj} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{i=1}^{t} \Omega(f_i) \tag{1}$$

Note that this objective function should contain training loss and regularization. XGBoost complexity is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2 \tag{2}$$

Random forest contains many decision tree, and the prediction of each decision tree is considered and the class with the most votes is selected as the predicted class.

For the proposed methodology, we implemented deep ensembling used the above mentioned three classification mechanism, and later average the given output to get the final output.

4. EXPERIMENTAL EVALUATION

4.1. Dataset

The proposed methodology is evaluated on the publicly available dataset (Kamei et al., 2012) which was created by collecting information from the csv repositories with the corresponding bug reports of six large open source projects and five large commercial projects. Open source projects were Bugzilla, Mozilla, Eclipse JDT, Columba, PostgresSQL, Eclipse platform. The data for the Bugzilla and Mozilla were obtained from the MSR 2007 Mine Challenge. The data for Columba were gathered from the official CVS repository. Table 1 summarizes the statistic of dataset.

Tab. 1. Statistic of the projects included in the dataset (Kamei et al., 2012)

	Period	The total number of changes		Average LOC		# of modified files per change	# of changes per day	# dev. per file	
				File	Change			Max.	Avg.
Bugzilla	08/1998 – 12/2006	4.62	(36%)	389.8	37.5	2.3	1.5	37	8.4
Columba	11/2002 – 07/2006	4.46	(31%)	125.0	149.4	6.2	3.3	10	1.6
Eclipse JDT	05/2001 – 12/2007	35.38	(14%)	260.1	71.4	4.3	14.7	19	4.0
Eclipse Platform	05/2001 – 12/2007	64.25	(14%)	231.6	72.2	4.3	26.7	28	2.8
Mozilla	01/2000 – 10/2006	98.28	(5%)	360.2	106.5	5.3	38.9	155	6.4
PostgresSQL	07/1996 – 05/2010	20.43	(25%)	563.0	101.3	4.5	4.0	20	4.0
OSS-Median	–	27.91	20%	310.1	86.7	4.4	9.4	24	4.0
C-1	10/2020 – 12/2009	4.10		–	6.4	2.0	1.2	–	–
C-2	10/2020 – 12/2009	9.28		–	19.2	2.4	2.8	–	–
C-3	07/2002 – 12/2009	3.59		–	16.6	2.0	1.3	–	–
C-4	12/2003 – 12/2009	5.18		–	12.9	1.8	2.4	–	–
C-5	10/1982 – 12/1995	10.96		303.0	39.0	4.8	2.3	–	–
COM-Median	–	5.18		–	16.6	2.0	2.3	–	–

* *The percentage in brackets shows the percentage of defect-inducting changes to all changes*

This dataset is imbalanced hence it greatly affects the sensitivity of the model. The learning pattern is also distributed. If the imbalanced data is not dealt perfectly, then the model will only learn to distinguish one class with major number of instances. Hence, it makes the model to predict output for one class more over other class and in that case the biasedness will follow up. It will exhibit higher accuracy but will not be a right model. It may also be noted that accuracy metric does not help us get the right metric for the measure of the accuracy but F1 score, p score and recall followed by specificity and sensitivity are the right metric to measure the performance.

4.2. Result and Discussion

For evaluation we used the 10 cross fold validation for the performance testing of different unsupervised models. As shown in Figure 2, the correlation between different attributes is imbalanced, in order to handle the imbalanced data, we obtain the balance between two classes. The improved correlation between attributes is obtained by applying normalization on the attributes.

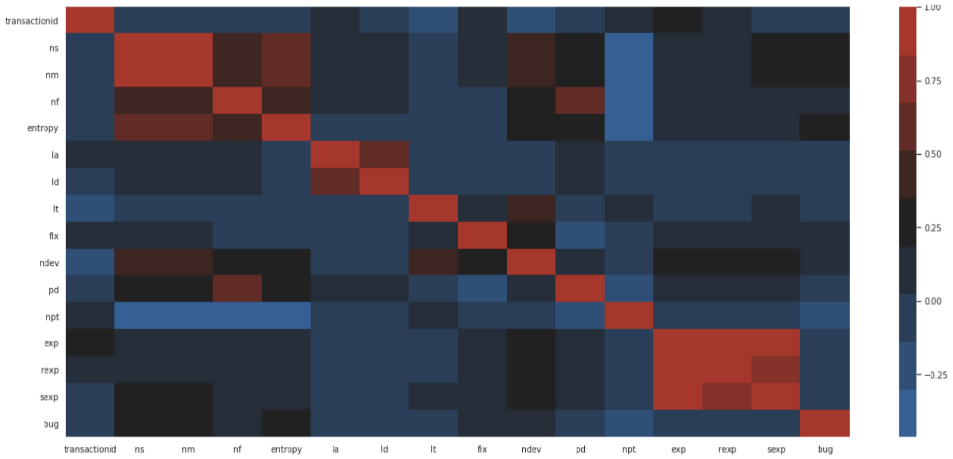

Fig. 2. Imbalance correlation matrix of attributes

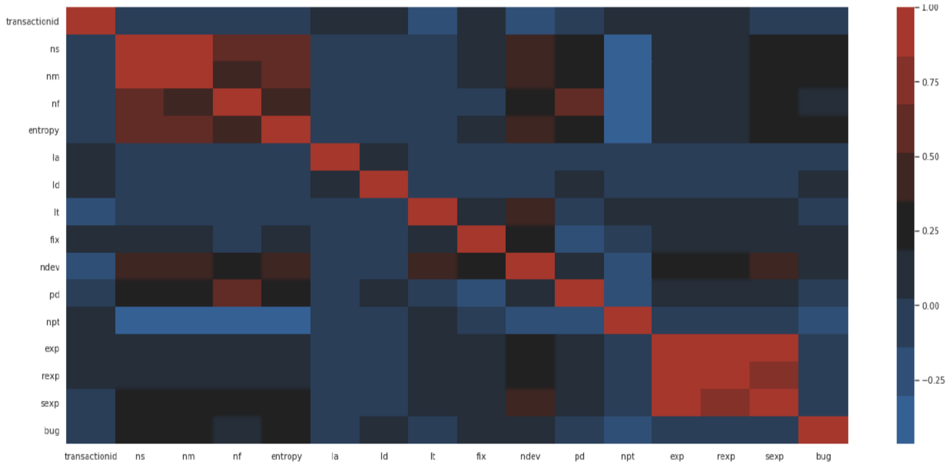

Fig. 3. Subsample correlation matrix of attributes

The current dataset which can create bottleneck at the time of learning. These outliers are needed to be dealt properly or we can discard them to make the distribution in the dataset even. Checking the dataset distribution with the target and we see that there are too many outliers and they can create a bottleneck when it comes to pattern learning. So, they must be dealt with properly else discarded.

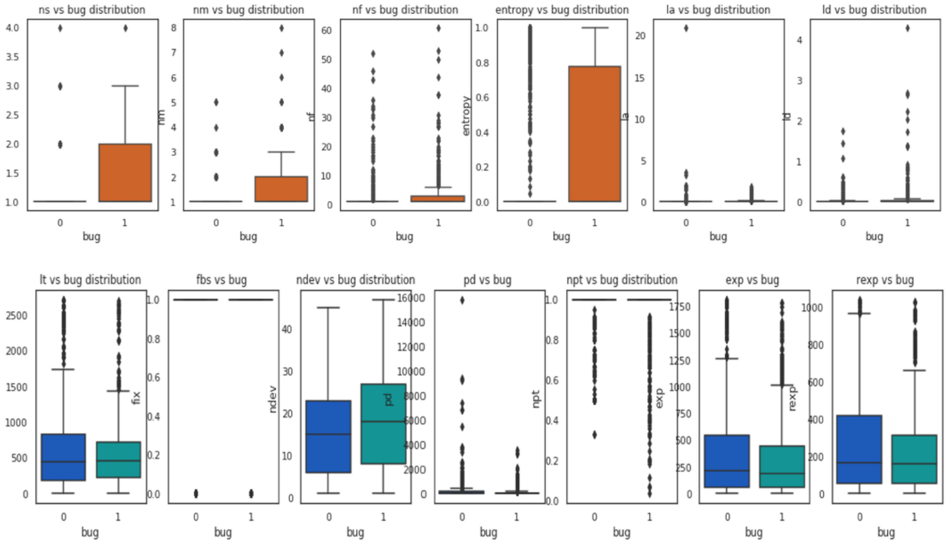

Fig. 4. Distribution of dataset

Figure 5 shows the distribution of three unsupervised model i.e. exp, rexp and sexp with the target. All these three attributes show maximum correlation with the target as compared to the rest of the attributes like npt, pd, ndev, fbs etc.

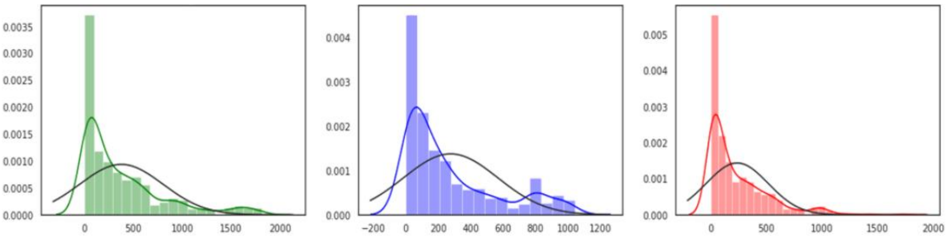

Fig. 5. Distribution of exp, rexp and sexp with the target

In Table 2, we report the performance of deep neural network along with other supervised and unsupervised learning methods. For performance reporting, we used the following measures: accuracy, p-score, f1-score, sensitivity and specificity. According to our evaluation, Random forest followed by XGBoost outperforms other classification methods and shows 71% accuracy. In general, supervised models outperforms unsupervised models when we have labelled data and the dataset is large for model training. This table also report results on across project cross validation using different dataset. In both cases high sensitivity and specificity is observed for all learning methods.

Tab. 2. Performance of different supervised, unsupervised and deep learning methods

MODEL	Acc	P-score	F1-score	Sensitivity	Specificity	Acc2	P-score2	F1-score2	Sensitivity2	Specificity2
Log. Reg.	0.6550	0.6714	0.6714	0.6368	0.6714	0.6303	0.4237	0.4911	0.7805	0.4237
SVM linear	0.6550	0.6551	0.6877	0.6547	0.6551	0.5833	0.3897	0.4768	0.7727	0.3897
SVM RBF	0.4725	0.4000	0.0186	0.4734	0.4000	0.6942	0.4000	0.0029	0.6946	0.4000
Naive Bayes	0.5775	0.5597	0.6943	0.6842	0.5597	0.6828	0.4726	0.3885	0.7398	0.4726
J48	0.6600	0.6813	0.6714	0.6377	0.6813	0.6823	0.4571	0.2893	0.7194	0.4571
Rand. Forest	0.7100	0.7701	0.6979	0.6637	0.7701	0.6754	0.4519	0.3561	0.7308	0.4519
AdaBoost	0.6900	0.7171	0.6633	0.6960	0.6633	0.6808	0.4760	0.4605	0.7628	0.4760
XGBoost	0.7000	0.7393	0.6984	0.6650	0.7393	0.6823	0.4571	0.2893	0.7194	0.4571
DNN	0.6275	0.6177	0.6823	0.6453	0.6177	0.5322	0.3289	0.4001	0.7156	0.3289
KMeans	0.4775	1.000	0.0094	0.4761	1.0000	0.6962	0.5178	0.1463	0.7057	0.5178

Yang et al. (Yang et al., 2016), state the importance of using unsupervised model instead of supervised model for learning. In their paper, they proposed the removal of highly correlated attributes by computing the reciprocal of raw metric and discarding the LA and LD, helps in ranking the values in descending order. However, considering the availability of enough training data and the presence of highly correlated attribute with target value supervised learning is the better option. On the other hand, when the labelled dataset is not available, choosing the unsupervised learning is the considerable option. In our proposed approach, we used abstract features in contrast to the use of LA and LD as proposed by Yang et al. (Yang et al., 2016), and the use of ensemble classification methods of three supervised learning method, deep Neural Network, XGBoost and Random forest performs better as compared to the stateof-the-art results. Table 3 shows the performance of our proposed methodology on sample and across different dataset. Our proposed methodology does not only perform better on the sample dataset but also shows better accuracy on different dataset with the accuracy of 81.85%.

Tab. 3. Performance on sample and different dataset

	Accuracy	P-score	F1-score	Sensitivity	Specificity
Sample Dataset	0.7739	0.7842	0.7546	0.7798	0.7842
Different Dataset	0.8185	0.7993	0.5860	0.8162	0.7993

Practically, there is no optimal approach which can meet all potential scenarios. Therefore, the unsupervised learning is a great way of learning when the labelled data is not applied but the Effort-aware JIT seems to be outperformed by the state of the art supervised model.

Yang et al. (Yang et al., 2016)model doesn't hold true for all the attributes as LA and LD had to be not taken into consideration. Our approach tries to take into account of the abstract features by using the deep ensemble technique and XGBoost and Random Forest, making it more robust and outperform all the models available. Yang paper (Yang et al., 2016) greatly highlights the use of unsupervised model and validates the importance of having the unsupervised model, their methodology of computing the reciprocal of raw metric and excluding the LA and LD and then removing the highly correlated help rank the values obtained in descending order (Huang, Xia & Lo, 2019). However, whilst observing the correlation with the target value and availability of sufficient data, it becomes obvious to opt for the supervised model. The supervised model XGBoost outperforms all the model in all aspects with high sensitive score.

Whilst using the model for Cross Validation Across Project, we observe the same trend for the classification model performance. The accuracy and all other metric performed well but XGBoost performed across all the models in both local and global models scenario.

5. CONCLUSIONS AND FUTURE WORK

Effort-aware Just-in-Time (JIT) defect prediction helps projects teams to allocate limited resources to the defect-prone software modules efficiently and accurately. Many machine learning and data mining approaches are used to detect and predict these defect inducing changes. However, the performance of these learning mechanism is highly dependent on the data that is used to train the model. In this paper, we proposed a novel methodology for effort aware just in time prediction for sample dataset and different dataset using different supervised and unsupervised learning methods. Our experiment concluded that unsupervised model have a very high degree of specificity and sensitivity both on current data and across project dataset. Unsupervised model are great but state of the art supervised model outperforms the unsupervised model on two context, when the data are labelled and the data is suffice enough. Specifically, Our results show that considering the performance of a single classifier, Random forest and XGBoost performs better than the other state-of-the-art methods. In addition, 77% accuracy is achieved by ensembling the output of three classifiers, i.e. Random Forest, XGBoost and Deep Neural Network for the sample dataset. We evaluate our proposed methodology only on the project that are publicly available, in future we can evaluate the result of our proposed methodology on closed source software projects.

Data Availability

The experiment uses public dataset shared by Kamei et al (Kamei et al., 2012), and they have already published the download address of the dataset in their paper.

REFERENCES

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785

Hata, H., Mizuno, O., & Kikuno, T. (2012). Bug prediction based on fine-grained module histories. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 200–210). IEEE Press.

Huang, Q., Xia, X., & Lo, D. (2019). Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering*, *24*(5), 2823–2862. https://doi.org/10.1007/s10664-018-9661-2

Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.I., Adams, B., & Hassan, A. E. (2010). Revisiting common bug prediction findings using effort-aware models. In *2010 IEEE International Conference on Software Maintenance* (pp. 1–10). IEEE. https://doi.org/10.1109/ICSM.2010.5609530

Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., & Ubayashi, N. (2012). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, *39*(6), 57–773. http://doi.org/10.1109/TSE.2012.70

Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2018). Cross-Project Change-Proneness Prediction. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* (Vol. 1, pp. 64–73). IEEE.

Mockus, A., & Weiss, D.M. (2000). Predicting risk of software changes. *Bell Labs Technical Journal*, *5*(2), 169–180.

Qiao, L., & Wang, Y. (2019). Effort-aware and just-in-time defect prediction with neural network. *PloS one*, *14*(2), e0211359. https://doi.org/10.1371/journal.pone.0211359

Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., ... & Leung, H. (2016). Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 157–168). ACM. https://doi.org/10.1145/2950290.2950353

Yu, T., Wen, W., Han, X., & Hayes, J. (2018). ConPredictor: Concurrency Defect Prediction in Real-World Applications. *IEEE Transactions on Software Engineering*, *45*(6), 558–575. https://doi.org/10.1109/TSE.2018.2791521

Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. *Information and Software Technology, 114*, 204–216. https://doi.org/10.1016/j.infsof.2019.07.003