Article

A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction

Saleh Albahli

Department of Information Technology, College of Computer, Qassim University, Buraidah 51452, Saudi Arabia; salbahli@qu.edu.sa

Received: 1 November 2019; Accepted: 18 November 2019; Published: 20 November 2019

Abstract: Since the introduction of just-in-time effort aware defect prediction, many researchers are focusing on evaluating the different learning methods, which can predict the defect inducing changes in a software product. In order to predict these changes, it is important for a learning model to consider the nature of the dataset, its unbalancing properties and the correlation between different attributes. In this paper, we evaluated the importance of these properties for a specific dataset and proposed a novel methodology for learning the effort aware just-in-time prediction of defect inducing changes. Moreover, we devised an ensemble classifier, which fuses the output of three individual classifiers (Random forest, XGBoost, Multi-layer perceptron) to build an efficient state-of-the-art prediction model. The experimental analysis of the proposed methodology showed significant performance with 77% accuracy on the sample dataset and 81% accuracy on different datasets. Furthermore, we proposed a highly competent reinforcement learning technique to avoid false alarms in real time predictions.

Keywords: deep neural network; unlabeled dataset; just-in-time defect prediction; unsupervised prediction

1. Introduction

Reducing the defects and number of failures in a software product is an important goal for software engineers. This is done in order to achieve maximum performance, build the trust of user and enhance the overall quality of the product [1]. During the life cycle of a product, a software goes through several feature changes, quality iterations and reassembling. Ideally, all these changes are perfectly merged, should cause no defect and are free of error. However, technically these changes sometimes induce the defect in an already working product, known as defect inducing changes. So, a “defect-inducing-change” can be described as a type of software change (single commit or multiple iterations in a specific period of time), which may cause one or numerous faults or defects in the software’s source code [2]. The defects induced by changes are often hard to track, difficult to resolve and cause issues for developers also. This problem led the research community to come up with some techniques identify and predict the upcoming possible defects in software so the conflict can be resolved in a timely manner and the debugging effort can be saved. Since then, a lot of researchers introduced the techniques to predict the possible defects. These techniques are traditional (module level at module, e.g., package, file, or class) as well as current techniques called just-in-time (JIT) prediction [3]. After the introduction of effort aware JIT prediction by Mockus et al. [4], it became a highly studied model as it consisted of a better error prediction and detection mechanism. The authors used numerous change metrics for the calculation of the probability of a fact that “a certain change may or may not induce the defect”, instead of going through the lines of codes.

The JIT defect prediction method has major practical value when compared with conventional defect predictions at a module level [5]. The JIT was also coined by the Kamei et al. [6] who put

forward a method of checking the error based on raw metric, which not only predicted the error 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 methods [7]. Although these methods were game changing in the dimension of defect prediction, they lack the results in terms of accuracy and massive false prediction. In addition, these methods failed to reduce the tedious task of finding the author of the code, as many people are involved in each module and inspecting the code while the change details are still being investigated.

As described earlier, there is much work available on the JIT effort aware system by using the traditional file, package or method level for the defect prediction [8–10] as well as supervised machine learning methods and unsupervised learning methods. Still, there is a huge gap in accuracy, and false prediction. Therefore, it is necessary to have state-of-the-art supervised, unsupervised or deep learning methods that can reduce the accuracy gap and can provide efficient predictions, which are precise and timely [11]. Hence, the basic objective of this work is to cope with the challenges of JIT prediction, and propose a technique, which is highly efficient in terms of results and preciseness. In this paper, we proposed a novel methodology for the prediction of defects using a state-of-the-art fusion approach of deep learning method with the Rainbow Reinforcement Strategy, which can reduce the error level in accuracy and avoid false prediction as the data grows. We have provided following contribution in the domain of effort aware JIT prediction:

- We have provided a state-of-art fusion methodology for effort aware JIT prediction of defects, which combines the deep neural network, random forest and XGBoost classifier.
- The proposed method has provided efficient results on a publicly available standard dataset.
- For comparison purposes, we have provided results on individual classifiers such as Random Forest and XGboost model as well.
- We have proposed a reinforcement strategy for the model, so it can learn with the growth of data.
- The detailed comparison of the proposed method shows that our model is state-of-the-art and has excellent results.

The rest of the paper is organized in the following manner. Section 2 presents the review of existing techniques related to the effort aware JIT defect prediction, the shortcoming and detailed analysis of used methodologies. The proposed technique is explained with detailed elaboration in Section 3 followed by experimental evaluation and results with the discussion in Section 4. Section 5 concludes the paper and states the possible future directions.

2. Literature Review

As discussed earlier, the effort-aware JIT defect prediction ranks the source code based on the probability of 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. In the domain of defect prediction, there are works available which utilize the traditional techniques of feature selections alongside the newer methodologies. In the traditional works, Ohlsson et al. [8] proposed the implementation of linear discriminant analysis in order to predict the defect in the source code of software. Meanwhile, Xu et al. [9] exploited the power of decision trees to solve the problem. Similarly Xing et al. [10] shifted the focus towards solving the problem by utilization of an SVM. However, the results were inaccurate with the excess amounts of false prediction and low accuracy. In the meantime, Mockus et al. [4] introduced the concept of JIT based prediction model for the better error prediction and detection mechanism. The JIT was also coined by the Kamei et al. [6] who put forward a method of checking the error based on a raw metric which not only predicts the error 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 methods [7]. Although these methods were game changing in the dimension of defect prediction, they lacked the results in terms of accuracy and

amounts of false predictions. Similarly, Qiao et al. [12] proposed a deep learning approach for the effort-aware JIT defect prediction. They used neural network-based techniques for the useful feature selection. They used ten numeric metrics of code changes and fed them to neural network to predict the presence of a bug in the code change, under review. They ranked the code changes according to the benefit cost ratio, which is calculated beforehand by dividing the likelihood of each code change by its size. This technique provided the comparable result, but calculation overdose and complexity of the memory made it impossible to use in real-time environment.

Yang et al. [1] stated that many unsupervised effort-aware JIT prediction models perform better than the state-of-the-art supervisor modes. They used only prediction models that have a good scalability and low application cost (i.e., metrics modelling cost and collection cost). They put forward the idea of using the unsupervised learning technique and highlighted that when building the prediction models, they did not need defected data for unsupervised model and, as a consequence, incur a low building cost and a high application range [13]. 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 perform the modeling. Thus, the model has to develop and train effectively to automatically identify on its own to figure out information. In addition, this model primarily deals with the unlabeled data. Their assumption was quite true, as the unsupervised methodology provided good performance without extra cost but the false prediction error was not resolved. The unsupervised techniques also introduced extra classes, which can neither be considered as error nor as unharmed cases.

There are many defect prediction models built with the historical metrics as well. Many studies have targeted coarse-grained (file and packages level) prediction. Hata et al. [14] stated that fine-grained prediction is challenging because it requires method level histories of existing version control systems. They tackled the mentioned problem and developed a fine-grained prediction version control system and proved that fine-grained performs better than coarse-grained predictions. On the other hand, Kamei et al. [6] claimed that the common finding in literature described that the package level prediction normally outperforms fine-level predictions, does not hold true when the effort is considered. They showed that package level prediction could be improved when a file level prediction is performed, then the matrices are lifted on the package level instead of just collecting the matrices at the package level.

Kamie et al. [15] 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 the file and package level for defect prediction, we should identify defect-prone software changes. They conducted a large-scale study on six open source projects and six closed source projects, showing 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 provided an effort-reducing way to handle the risky changes and minimize the cost of the development of high-quality software. The existing literature mainly focuses on 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 modeling. This led to inaccurate results and high false predictions. In contrast to previous work, we focus on using the deep learning fusion-based techniques for the defect prediction and achieved state-of-the-art results with good accuracy and the lowest numbers of false predictions.

3. Proposed Methodology

As we have already discussed, our main concern is to develop an accurate classification model, which can predict the upcoming possible defects in software's source code. We interpreted this as a classification problem and proposed a methodology for effort aware just-in-time prediction using the fusion based method. Technically, instead of using only supervised or unsupervised method for classification, we proposed the use of a combination of a deep learning technique with Random Forest

and XGBoost and then corrected the prediction accuracy by employing a Rainbow based reinforcement method. Figure 1 presents the architecture diagram of proposed methodology.

Figure 1. Proposed methodology: Effort aware just-in-time prediction using deep ensemble method.

3.1. Deep Neural Network Inspired by DeepCrowd

For the prediction of the defects, we employed a state-of-the-art fully connected Deep Neural Network (DNN; multilayer perceptron) and derived the idea of residual short-connections from DeepCrowd [16]. This model can map the input data to the given labelled dataset representing the non-linear relationship. Unlike most conventional machine learning algorithms, DNN can detect the feature automatically without human intervention. The short connections help the model to overcome the Vanishing Gradient Problem [17]. Figure 2 represents the basic architecture of a DNN, a multilayer perceptron. It can be seen that a neural network is a linear combination of the weights, bias and inputs. The input u_i is processed by the function $fW(u_i)$ to produce an output u_k . The processing function $fW(u_i)$ can be any type of non-linear function, which updates the weights W , such as sigmoid [18], ReLU [19], SoftMax [20] or Tanh [21]. The proposed DNN has 10 hidden layers with a constant learning rate of 0.001, decay rate of 0.9 and exponential decay rate of 2.099. The model was trained using Keras a Tensor Flow based library. For the optimization of DNN, we used ADAM (adaptive moment estimation) optimization algorithm [22]. It updates model weight iteratively based on the training data.

Figure 2. The architecture representation of a deep neural network [23].

3.2. XGBoost Classifier

XGBoost [24] was initially developed with deep consideration of system optimization and principles in machine learning. As XGBoost is an enactment of gradient boosted decision-trees, which is designed for its speed and performance quality, it provides a memory efficient solution for the classification tasks. It has salient features such as Parallelization, Distributed Computing, Out-of-Core Computing and Cache Optimization. Furthermore, it has Sparse Aware employment so it can automatically handle the missing values of data. The objective function obj can be expressed as:

$$Obj^p = \sum_{k=0}^n M[l_k, l_k^{\hat{p}-1} + f_p(g_k)] + \Omega(f_p) \quad (1)$$

where p is iteration, l_k is real value/targeted of known label from the training dataset, $f_p(g_k)$ is partial addition of derivative where the value of x is $l_k^{\hat{p}-1}$. Here M is a differential loss function which computes the deference between the predicted label $l_k^{\hat{p}-1}$ and the target label l_k . The term Ω penalizes the complexity of the model. We have employed an XGBoost classifier with the following settings. *learning_rate* = 0.01, *n_estimators* = 5000, *max_depth* = 4, *min_child_weight* = 6, *gamma* = 0, *subsample* = 0.8, *colsample_bytree* = 0.8, *reg_alpha* = 0.005, *objective* = 'binary : logistic', *nthread* = 4, *scale_pos_weight* = 1 and *seed* = 27.

3.3. Random Forest

Random forest [25] contains several decision trees, the prediction of each decision tree is considered, and the class with the most votes is selected as the predicted class. It exploits the feature randomness and the bagging, while constructing each tree, in order to compute an uncorrelated forest. The prediction of forest by fusing the decision is more accurate than the prediction of individual tree, which means that inaccurate tree's decision is neglected to some extent and the results are based on the votes of the majority. In order to evaluate the decision of a tree, each tree is considered as a node and its importance is calculated with the help of Gini index [26] as:

$$imp_node_k = SW_{L(k)} \cdot Im_{L(k)} - SW_{R(k)} \cdot Im_{R(k)} \quad (2)$$

where imp_node_k is the importance of k^{th} node, $SW_{L(k)}$ presents the weighted samples, which are reaching to k^{th} node. Similarly, $Im_{L(k)}$ shows the impurity value. $L(k)$ and $R(k)$ denotes the left and right nodes after the split. We utilize following parameters for the implementation of Random Forest classifier. '*bootstrap*' = *True*, '*max_depth*' = 80, '*max_features*' = 3, '*min_samples_leaf*' = 5, '*min_samples_split*' = 12, and '*n_estimators*' = 100.

3.4. Fusion/Ensemble of Classifiers

The final step of the proposed methodology is to combine the classification of all three models in order to form a single heterogeneous model, which can provide the results efficiently in terms of accuracy and low false alerts. In order to achieve that, we implemented all three models in a single platform and combined the results by using an averaging mechanism. The results provided by the model are state-of-the-art in terms of lowest false prediction. The accuracy is also comparable. In order to enhance the prediction ability of the model we further proposed to employ a reinforcement technique, so the model can learn while the data grows. Further details of this strategy are elaborated upon in the following section.

3.5. Reinforcement Learning for Hyperparameters Optimization

Despite the accuracy in understanding of data, machines are often highly unpredictable in nature, which cause them to change to patterns all the time [27]. Even if we introduced a fusion based approach, which provides an accurate prediction of defects, the results can still be unpredictable and variable in nature. Hence, the methods should be optimized constantly when they are needed to be employed in real-time prediction. This helps to reduce the uncertainty of results, so the user will never have to suffer with inaccurate results. For this the possible strategies include:

- Change of the machine dependency towards more reliable features, including only the most correlated features for the prediction.
- Conduct a regressive amount of alpha and beta testing, and fine tune the models as much as possible, and choose the most optimal values for the hyper parameters.
- Exploit the values of current predictions and the real time values in order to add the reward/punishment mechanism (reinforcement learning).

As we have already implemented the first two strategies, we will employ the last strategy of reinforcement learning (RL). Since we have a limited number of pages, instead of explaining what RL actually is, we have decided to elaborate the implementation detail of RL. The only technical knowledge required to understand the RL based techniques is that they work similar to the learning approach of a human being. Consider a game in which for a good play (move), the player gets a +1 score, but for a bad play (move) the player gets a punishment of a -1 score. So the player remembers the moves, which cost it a +1 or -1 score and tries to minimize the error of that move. It always tries to achieve the +1 score and the game continues. Similarly, the RL based model gets a reward for an accurate prediction and deduction for a false prediction. By utilizing this technique, the model keeps improving itself until it reaches a conclusion. These types of RL based models are quite common in cloud computing applications, where maximum accuracy is the requirement. Technically, we propose to exploit the Q-learning technique [28]. In this methodology, the actions values from the state "X" to the state "Y" are designed. In order to achieve these action values, we exploited the idea of the Rainbow technique (seven Q-learning algorithms) [29]. As described, the whole RL method is based on reward mechanism, so we employed the reward policy as expressed in Equation (3):

$$Reward = Accuracy_{pre} + 2 \times loss_{pre} \quad (3)$$

while the Reward presents the actual reward, $Accuracy_{pre}$ shows the accuracy of prediction and $loss_{pre}$ presents the loss which is calculated by the model.

3.6. Rainbow Technique

The Rainbow technique is a state-of-the-art Q-learning methodology presented by Hessel et al. [29] for the purpose of deep RL. This methodology merges the seven Q-learning algorithms and provides robust predictions. The technical summary of these seven algorithms can be described as follows:

- To utilize a DQN, which computes the Q value with the help of a neural network, in order to minimize the objective function. A DQN trains itself exactly like the training of a deep neural network.
- In order to handle the overestimation problem, the double Q Learning is utilized.
- Prioritized replay algorithm provides the highest priority to those samples, which are responsible for the higher Q loss in the last iterations.
- The duel network architecture, which has two network streams, is employed. One network is used for the optimization of value and second is for the calculation of advantage calculation.
- To calculate the value in N-steps instead of just relaying on next single step, we utilized the power of multi learning.
- Instead of averaging distribution mechanisms, we utilized distributional RL which can approximate the better distribution of Q-values.
- Finally, we exploited the power of noisy nets to estimate the targeted values and neglect the noise values.

To be precise, the prediction output of the proposed methodology is fed to the Rainbow network and the reward/punishment values are estimated for effective real time learning. As proposed by the original author, we exploited the Bellman's equation [30] to estimate the final predication of the model at time t as described in the Equation (4).

$$Error_{pre}(t) = Val_q(t) + \frac{R(t) + \gamma val_q(t+1) - Val_q(t)}{\beta} - [A - dv(q(t), action(t))], \quad (4)$$

where the q denotes the current state, t denotes the time, $action(t)$ presents the current action in the q state. Where ρ shows the scaling constant for scaling which is exploited to scale the difference between required and the predicted values. Finally, the Val_q shows the value of the current-state q the at timestamp t .

4. Experimental Evaluation

4.1. Dataset

The proposed methodology is evaluated on the publicly available dataset [6] that 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. The open source projects were Bugzilla, Mozilla, Eclipse JDT, Columba, PostgreSQL and the Eclipse platform. The data for the Bugzilla and Mozilla were obtained from the MSR 2007 Mine Challenge. The data for Columba was gathered from the official CVS repository. Table 1 summarizes the statistics of the dataset.

Table 1. Statistic of the projects included in the dataset [15].

Project	Period	No. Changes	Avg LOC File	Change	No. Modified Files/Change	No. Changes/Day	No. Dev Per File Max	Avg.
Bugzilla	08/1998–12/2006	4620	389.8	37.5	2.3	1.5	37	8.4
Columba	11/2002–7/2006	4455	125.0	149.4	6.2	3.3	10	1.6
Eclipse JDT	5/2001–12/2007	35,386	260.1	71.4	4.3	14.7	19	4.0
Eclipse Platform	5/2001–12/2007	64,250	231.6	72.2	4.3	26.7	28	2.8
Mozilla	1/200–12/2006	98,275	360.2	106.5	5.3	38.9	155	6.4
PostgreSQL	7/1996–5/2010	20,431	563.0	101.3	4.5	4.0	20	4.0
OSS-Median	–	27,909	310.1	86.7	4.4	9.4	24	4.0
C-1	10/2000–12/2009	4096	–	16.4	2.0	1.2	–	–
C-2	10/2000–12/2009	9277	–	19.2	2.4	2.8	–	–
C-3	7/2000–12/2009	3586	–	16.6	2.0	1.3	–	–
C-4	12/2003–12/2009	5182	–	12.9	1.8	2.4	–	–
C-5	10/1982–12/1995	10,961	303.0	39.0	4.8	2.3	–	–
Com-Median	–	5182	–	16.6	2.0	2.3	–	–

This dataset is unbalanced, hence it greatly affects the sensitivity of the model. The learning pattern is also distributed. If the unbalanced data is not dealt perfectly, then the model will only learn to distinguish one class with a major number of instances. Hence, the model predicts one class over another so that the bias will be reduced. It will exhibit a higher accuracy but will not be the right model. It should also be noted that accuracy metrics did 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 were the right metric to measure the performance.

4.2. Result and Discussion

For evaluation of this model, we used the 10 cross fold validation for the performance testing of different unsupervised models. As shown in Figure 3, the correlation between different attributes is unbalanced, in order to handle the unbalanced data; we obtain the balance between two classes. Furthermore, the correlation between the balance attributes is shown in Figure 4. The improved correlation between attributes is obtained by applying normalization to the attributes.

Figure 3. Imbalance correlation matrix of attributes.

Figure 4. Subsample correlation matrix of attributes.

Distribution in the dataset is extremely important when the pattern learning is considered. In our case, as shown in Figure 5 there are many outliers, present in the current dataset, which can create bottleneck at the time of learning. These outliers need to be dealt with correctly, or must be discarded to make the distribution in the dataset even. When checking the dataset distribution with the target it is evident that there are too many outliers, these can create a bottleneck when it comes to pattern learning. So, they must be dealt with properly or they should be discarded.

Figure 5. Distribution of dataset.

Figure 6 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.

Figure 6. 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 outperformed unsupervised models when we have labelled data and the dataset is large enough for model training. This table also reports results on the whole project cross validation by using different datasets. In both cases high sensitivity and specificity is observed for all learning methods.

Table 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
LR	0.6550	0.6714	0.6714	0.6368	0.6714	0.6303	0.4237	0.4911	0.7805	0.4237
SVM	0.6550	0.6551	0.6877	0.6547	0.6551	0.5833	0.3897	0.4768	0.7727	0.3897
linear										
SVM	0.4725	0.4000	0.0186	0.4734	0.4000	0.6942	0.4000	0.0029	0.6946	0.4000
RBF										
Naïve-B	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
RF	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
K-Means	0.4775	1.000	0.0094	0.4761	1.0	0.6962	0.5178	0.1463	0.7057	0.5178

Yang et al., [1] state the importance of using the unsupervised model instead of the supervised model for learning. In their paper, they proposed that the removal of highly correlated attributes by computing the reciprocal of raw metrics 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 attributes with target values, it becomes clear that supervised learning is the better option. On the other hand, when the labelled dataset is not available, choosing the unsupervised learning is the preferred option. In our proposed approach, we used abstract features in contrast to the use of LA and LD as proposed by Yang et al., [1], 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 state-of-the-art results. Table 3 shows the performance of our proposed methodology on samples and across different datasets. 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%.

Table 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

4.3. Discussion

In practicality, there is no optimal approach that can meet all potential scenarios. Therefore, the unsupervised learning is a great way of learning when the labelled data is not applied, however the effort-aware JIT seems to be outperformed by the state-of-the-art supervised model. The model used by Yang et al. [1] 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 thus outperforms all the models available.

Yang [1] greatly validates the importance of having the unsupervised model, as well as their methodology of computing the reciprocal of the raw metric and excluding the LA and LD and then

removing the highly correlated data, as this helped in ranking the values obtained in descending order [31]. However, whilst observing the correlation with the target value and availability of sufficient data, it became obvious to opt for the supervised model. The supervised model XGBoost outperformed all the models in all aspects with a highly sensitive score.

Whilst using the model for cross validation cross project, we observed the same trend for the classification model performance. The accuracy and all other metrics performed well but XGBoost performed across all the models in both local and global scenarios.

5. Conclusions and Future work

Effort-aware JIT defect prediction helps project 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 performances of these learning mechanisms are 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 datasets and different datasets using different supervised and unsupervised learning methods. Our experiment concluded that unsupervised models have a very high degree of specificity and sensitivity both on current data and across the project's dataset. An unsupervised model is sufficient but a state-of-the-art supervised model outperforms the unsupervised model on two contexts: when the data is labeled, and the data is sufficient 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 fusing the output of three classifiers, i.e., Random Forest, XGBoost and Deep Neural Network for the sample dataset. We evaluated the proposed methodology only on the project that are publicly available; in future we can evaluate the result of our proposed methodology on closed source real-time software projects as well.

Funding: This research received no external funding.

Conflicts of Interest: The author declares that they have no conflict of interest.

References

1. Yang, Y.; Zhou, Y.; Liu, J.; Zhao, Y.; Lu, H.; Xu, L.; Leung, H. 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, Seattle, WA, USA, 13–18 November 2016; pp. 157–168.
2. Jacekliwerski, T.Z.; Zeller, A. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes* **2005**, *30*, 15.
3. Zheng, Q.; Kimura, H. Just-in-Time Modeling for Function Prediction and Its Applications. *Asian J. Control* **2001**, *3*, 35–44. [[CrossRef](#)]
4. Mockus, A.; Weiss, D.M. Predicting risk of software changes. *Bell Labs Tech. J.* **2000**, *5*, 169–180. [[CrossRef](#)]
5. Fukushima, T.; Kamei, Y.; McIntosh, S.; Yamashita, K.; Ubayashi, N. An empirical study of just-in-time defect prediction using cross-project models. In Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, 31 May–1 June 2014; pp. 172–181.
6. Kamei, Y.; Shihab, E.; Adams, B.; Hassan, A.E.; Mockus, A.; Sinha, A.; Ubayashi, N. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. Softw. Eng.* **2012**, *39*, 757–773. [[CrossRef](#)]
7. Zhou, T.; Sun, X.; Xia, X.; Li, B.; Chen, X. Improving defect prediction with deep forest. *Inf. Softw. Technol.* **2019**, *114*, 204–216. [[CrossRef](#)]
8. Ohlsson, N.; Alberg, H. Predicting fault-prone software modules in telephone switches. *IEEE Trans. Softw. Eng.* **1996**, *22*, 886–894. [[CrossRef](#)]
9. Xu, Z.; Liu, J.; Luo, X.; Yang, Z.; Zhang, Y.; Yuan, P.; Tang, Y.; Zhang, T. Software defect prediction based on kernel PCA and weighted extreme learning machine. *Inf. Softw. Technol.* **2019**, *106*, 182–200.
10. Xing, F.; Guo, P.; Lyu, M.R. A novel method for early software quality prediction based on support vector machine. In Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Chicago, IL, USA, 8–11 November 2005.

11. Yu, T.; Wen, W.; Han, X.; Hayes, J. ConPredictor: Concurrency Defect Prediction in Real-World Applications. *IEEE Trans. Softw. Eng.* **2018**, *45*, 558–575. [CrossRef]
12. Qiao, L.; Wang, Y. Effort-aware and just-in-time defect prediction with neural network. *PLoS ONE* **2019**, *14*, e0211359. [CrossRef] [PubMed]
13. Liu, C.; Yang, D.; Xia, X.; Yan, M.; Zhang, X. Cross-Project Change-Proneness Prediction. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 1, pp. 64–73.
14. Hata, H.; Mizuno, O.; Kikuno, T. Bug prediction based on fine-grained module histories. In Proceedings of the 34th International Conference on Software Engineering, Zurich, Switzerland, 2–9 June 2012; pp. 200–210.
15. Kamei, Y.; Matsumoto, S.; Monden, A.; Matsumoto, K.I.; Adams, B.; Hassan, A.E. Revisiting common bug prediction findings using effort-aware models. In Proceedings of the 2010 IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010; pp. 1–10.
16. Khan, G.; Farooq, M.A.; Hussain, J.; Tariq, Z.; Khan, M.U.G. Categorization of Crowd Varieties using Deep Concurrent Convolution Neural Network. In Proceedings of the 2019 2nd International Conference on Advancements in Computational Sciences (ICACS), Lahore, Pakistan, 18–20 February 2019; pp. 1–6.
17. Wang, X.; Qin, Y.; Wang, Y.; Xiang, S.; Chen, H. ReLTanh: An activation function with vanishing gradient resistance for SAE-based DNNs and its application to rotating machinery fault diagnosis. *Neurocomputing* **2019**, *363*, 88–98. [CrossRef]
18. Kwan, H.K. Simple sigmoid-like activation function suitable for digital hardware implementation. *Electron. Lett.* **1992**, *28*, 1379–1380. [CrossRef]
19. Schmidt-Hieber, J. Nonparametric regression using deep neural networks with ReLU activation function. *arXiv* **2017**, arXiv:1708.06633. Available online: <https://arxiv.org/abs/1708.06633> (accessed on 20 November 2019).
20. Grave, E.; Joulin, A.; Cissé, M.; Jégou, H. Efficient softmax approximation for GPUs. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2019; pp. 1302–1310.
21. Claudio, G.; Micheli, A.; Pedrelli, L. Deep Echo State Networks for Diagnosis of Parkinson's Disease. *arXiv* **2018**, arXiv: 1802.06708. Available online: <https://arxiv.org/abs/1802.06708> (accessed on 20 November 2019).
22. Fan, E. Extended tanh-function method and its applications to nonlinear equations. *Phys. Lett. A* **2000**, *277*, 212–218. [CrossRef]
23. Pedrelli, L. Analysis of Deep Learning Models using Deep Echo State Networks (DeepESNs). Available online: <https://www.deeplearningitalia.com/analysis-of-deep-learning-models-using-deep-echo-state-networks-deepesns/> (accessed on 20 November 2019).
24. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
25. Liaw, A.; Wiener, M. Classification and regression by randomForest. Available online: <http://cogms.northwestern.edu/cbm/LiawAndWiener2002.pdf> (accessed on 20 November 2019).
26. Yitzhaki, S. On an extension of the Gini inequality index. *Int. Econ. Rev.* **1983**, *24*, 617–628. [CrossRef]
27. Khan, G.; Tariq, Z.; Hussain, J.; Farooq, M.A. Segmentation of Crowd into Multiple Constituents Using Modified Mask R-CNN Based on Mutual Positioning of Human. In Proceedings of the 2019 International Conference on Communication Technologies (ComTech), Rawalpindi, Pakistan, 20–21 March 2019; pp. 19–25.
28. Nie, J.; Haykin, S. A Q-learning-based dynamic channel assignment technique for mobile communication systems. *IEEE Trans. Veh. Technol.* **1999**, *48*, 1676–1687.
29. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2019.

30. Pesch, H.J.; Bulirsch, R. The maximum principle, Bellman's equation, and Carathéodory's work. *J. Optim. Theory Appl.* **1994**, *80*, 199–225. [[CrossRef](#)]
31. Huang, Q.; Xia, X.; Lo, D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empir. Softw. Eng.* **2019**, *24*, 2823–2862. [[CrossRef](#)]

© 2019 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).