International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-5, March 2020

# Software Defect Prediction Via Deep Learning

**Rehan Ullah Khan, Saleh Albahli, Waleed Albattah, Mohammad Nazrul Islam Khan**

***Abstract:*** ***Existing models on defect prediction are trained on historical limited data which has been studied from a variety of pioneering and researchers. Cross-project defect prediction, which is often reuse data from other projects, works well when the data of training models is completely sufficient to meet the project demands. However, current studies on software defect prediction require some degree of heterogeneity of metric values that does not always lead to accurate predictions. Inspired by the current research studies, this paper takes the benefit with the state-of-the-art of deep learning and random forest to perform various experiments using five different datasets. Our model is ideal for predicting of defects with 90% accuracy using 10-fold cross-validation. The achieved results show that Random Forest and Deep learning are giving more accurate predictions with compared to Bayes network and SVM on all five datasets. We also derived Deep Learning that can be competitive classifiers and provide more robust for detecting defect prediction.***

***Keywords*****:** ***Defect prediction; Deep Learning; Software repository mining; Cross-Project; Class imbalance.***

## I.INTRODUCTION

Reducing defects and number of failures in software products is an important goal for software engineers. This is done in order to achieve maximum performance, build the trust of users and enhance the overall quality of the product. 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 Jist-In-Time (JIT) defect prediction is of more practical value compared with traditional defect predictions at module. The JIT was coined by the Kamei et al.[35] 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. 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 fresh in mind, help make the debug very easy.

Revised Manuscript Received on March 30, 2020.
***** Correspondence Author
**Rehan Ullah Khan*,** Department of Information Technology, College of Computer, Qassim University, Saudi Arabia
**Saleh Albahli,** Department of Information Technology, College of Computer, Qassim University, Saudi Arabia
**Waleed Albattah,** Department of Information Technology, College of Computer, Qassim University, Saudi Arabia
**Mohammad Nazrul Islam Khan,** Department of Computer Engineering, College of Computer, Qassim University, Saudi Arabia



Therefore, in JIT defect predictions, it is easy to find such a developer to inspect the predicted defect-prone change, as each change is associated with a particular developer. Kim et al. [36] used numerous features extracted from various sources such as change metadata, source code, and change log messages to build prediction models to predict defect-inducing changes. Their results showed that defect-inducing changes can be predicted at 60% recall and 61% precision on average.

However, there is much work available on the JIT effort aware system by using the traditional file, package or method level for the defect prediction [22-24].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. 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 performed various experiments using five different datasets for the prediction of defects using a state-of-the-art fusion approach of deep learning method with the Random Forest algorithm, which help predicting of defect with 90% accuracy using 10-fold cross-validation. Thus, our model can reduce the error level in accuracy and avoid false prediction as the data grows.

The rest of the paper is organized as follows. Section 2 reviews up-to-date literature on Just-in-time software defect prediction as well as the benefit of using deep learning in software engineering. In section 3, we have reported the approach used in our experiments. Section 4 presents our proposed model with its experimental analysis, evaluation results and experimental discussions. Finally, conclusions and perspectives are given in Section 5.

## II.BACKGROUND AND RELATED WORK

Just-in-time software defect prediction (JIT-SDP) has valuable vicinity in software defect prediction because it provides to identify defect-inducing changes. Yang at el. [1] have compared the performance of local and global models through a large-scale empirical study based on six open-source projects with 227417 changes in the context of JIT-SDP. Local models have significantly better effort-aware prediction performance than global models in the cross-validation and cross-project-validation scenarios. Therefore, local models are promising for effort-aware JIT-SDP. Xiang at el. have proposed a multi-objective optimization based supervised method MULTI to build JIT-SDP models [4]. MULTI can perform significantly better than the state-of-

the-art supervised and unsupervised methods in the three performance evaluation scenarios. The results confirm that supervised methods are still promising in effort-aware JIT-SDP. Hoang at el. have deployed an end-to-end deep learning framework, named DeepJIT that automatically extracts features from commit messages and code changes and use them to identify defects using QT and OPENSTACK software [7]. Yasutaka at el. [10] have investigated JIT models learned using other projects are a viable solution for projects with limited historical data. JIT models tend to perform best in a cross-project context when the data used to learn them are carefully selected.

Fu. at el. [2] have reported that supervised predictors did not perform outstandingly better than unsupervised ones for effort-aware just-in-time defect prediction on the basis of their experiments. Recently, Yang et al. have proposed an unsupervised model and applied it to projects with rich historical bug data. Supervised models that benefit from historical data are expected to perform better than unsupervised ones [3]. Meng at el. [5] have compared the effectiveness of unsupervised and supervised prediction models for effort-aware file-level defect prediction and suggested that unsupervised models do not perform statistically significantly better than state-of-art supervised model under within-project setting. Chen at el. [6] recommended that researchers need to use the unsupervised method LOC_D as the baseline method, which is used for comparing their proposed novel methods for Software defect number prediction (SDNP) problem in the future. Duksan at el [11] have proposed a transfer cost-sensitive boosting method that considers both knowledge transfer and class imbalance for cross-project defect prediction (CPDP) when given a small amount of labeled target data.

Thomas at el. [12] have studied cross-project defect prediction models on a large scale and obtained results indicate that cross-project prediction is a serious challenge, i.e., simply using models from projects in the same domain or with the same process does not lead to accurate predictions.

Feng at el. have examined two types of unsupervised classifiers: a) distance-based classifiers (e.g., k-means); and b) connectivity-based classifiers. They compared the performance of unsupervised classifiers versus supervised classifiers using three publicly available datasets (i.e., AEEEM, NASA, and PROMISE). Steffen at el. [14] have provided a comparative study to benchmark cross-project defect prediction approaches.

Panichella at el [15] have presented an empirical study aiming at statistically analyzing the equivalence of different defect predictors. They proposed a combined approach, coined as CODEP (COmbined DEfect Predictor) that employs the classification provided by different machine learning techniques to improve the detection of defect-prone entities. This is also confirmed by the superior prediction accuracy achieved by CODEP when compared to stand-alone defect predictors. Nam at el. have proposed [16] a prediction model with defect data collected from a software project and predict defects in the same project, i.e. within-project defect prediction (WPDP) and cross project defect prediction (CPDP) to predict defects for new projects lacking in defect data by using prediction models built by other projects.

Ryu at el. have proposed [17] a transfer cost-sensitive boosting method that considers both knowledge transfer and class imbalance for CPDP when given a small amount of labeled target data. However, Their analyses also yield defect predictors learned from NN-filtered CC data [18], with performance close to, but still not better than, WC data. Therefore, they perform a final analysis for determining the minimum number of local defect reports in order to learn WC defect predictors.

Y. Ma at el. have [19] considered the cross-company defect prediction scenario where source and target data are drawn from different companies. In order to harness cross company data, they try to exploit the transfer learning method to build faster and highly effective prediction model.

Xiao at el. have provided effective solutions for both within-project and cross-project class-imbalance problems [20]..

### 2.1 Deep Learning in Software Engineering

Machine learning, especially deep learning, has got much interest in the literature recently. It has got good promising results in the field of software engineering research. Machine learning of software engineering has the opportunity of using learning algorithms which is not available in traditional software engineering.

A great interest has been on software engineering on one side, and also for machine learning testing on the other, but as far as we know, the intersection of software engineering and machine learning has not received the same interest [22-24].

Arpteg et. al. [21] has discussed the challenges of software engineering in machine learning, namely deep learning. They defined a set of challenges related to the intersection of software engineering and machine learning. seven empirically validate projects were used in the study, they ended up with twelve challenges. the main goal was to grab the attention of the field researchers to these kinds of challenges in their future work.

Ma et. al. [25] studied the safety and security issues in deep learning systems from the software quality perspective. They believe that although such systems have gained success, they still suffer from different defects and vulnerabilities when they are deployed to serious security-related applications.

Lee et. al. [26] used deep learning method to discover software weaknesses. the method is based on learning the assembly code to define software defects. the results were promising and showed a high level of accuracy, which encourage for further investigations in this regard.

Corley et. al. [27] deployed deep learning models for feature location. Although the study is introductory, it shows promising results that encourage for further investigations in future. Similarly, Mani et. al. [28] proposed a new algorithm using deep learning model to report software bugs. the proposed method presents great improvements to the bug representation, which leads to a better understanding of the bug and consequently the bug classification.

## III.APPROACH

Figure 1 shows the generic flow of the evaluation approach. As in Figure 1, the defect data is divided into two parts. The training set, and the test set. The training set is used to learn the class distribution and the correlation of the class with the rest of the features. The training process is a supervised process. Once, the inherent structures in the data are learned, the output is predictive model. This predictive model can be used to predict new defect cases. The model is either a flow of steps or a mathematical representation for new un-seen cases. The model performance is checked by testing the model on the test data. The correct and incorrect predictions are noted. The approach then loads another set of the training data for next iteration on the dataset.

For defect analysis, and performance evaluation, we use the 10-folds cross-validation as a training and testing paradigm. The 10-folds cross-validation not only reliably learns and tests the performance of the classifier, but is also a standard approach for classifier comparison in the state of the art. The 10-folds cross-validation uses 90% data for training and 10% data for testing. For 90% of training data, the model is created and tested on the 10% testing data. The testing data is un-seen data for creating the model and therefore represents the real-world scenario for new queries for defect analysis. The prediction performance of a particular classifier is noted and stored. Then, the process is repeated 10 times as shown in Figure 1, and the average of performance is calculated. This removes the bias of results, and the performance can be generalized for practical applications.

Deep learning algorithm is the state-of-the-art model. Deep learning can be used as a simple Neural Network classifier where the features are extracted by non-Deep approaches and Deep learning only learns the class separation. This approach is not as robust as the one where deep learning is used to extract features. Deep learning in other the form where features are learned and classified by the Deep learning algorithm itself is used extensively in image classification and regression problems. In the comparative experiments when we use Deep learning, we mean the usage of Deep learning for classifying features only and is not used for features extraction. Our model trained on the following parameters, one input layer, 2 fully connected hidden layers and 1 out layer, a dropout rate of 0.9, the learning rate of 0.01, and error epsilon of 1.0E-4.

### 3.1 EVALUATION

### 3.2. Dataset

We use the PROMISE data [37] set for defect prediction made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering. This dataset consists of different modules and repositories that satisfy our requirements. There are five repositories; which are CM1, JM1, KC1, KC2, and PC1.

The CM1 is a NASA spacecraft instrument written in "C". Data comes from McCabe and Halstead features extractors of source code. These features were defined in an attempt to objectively characterize code features that are associated with software quality. The McCabe and Halstead measures are "module"-based where a "module" is the smallest unit of functionality. In C, "modules" would be called "function" or "method" respectively. The JM 1is written in "C" and is a real-time predictive ground system. The KC1 is a "C++" system implementing storage management for receiving and processing ground data. KC 2 is Data from C++ functions. Science data processing; another part of the same project as KC1; different personnel than KC1. Shared some third-party software libraries with KC1, but no other software overlap. The PC 1 is Data from C functions from flight software for earth orbiting satellite.

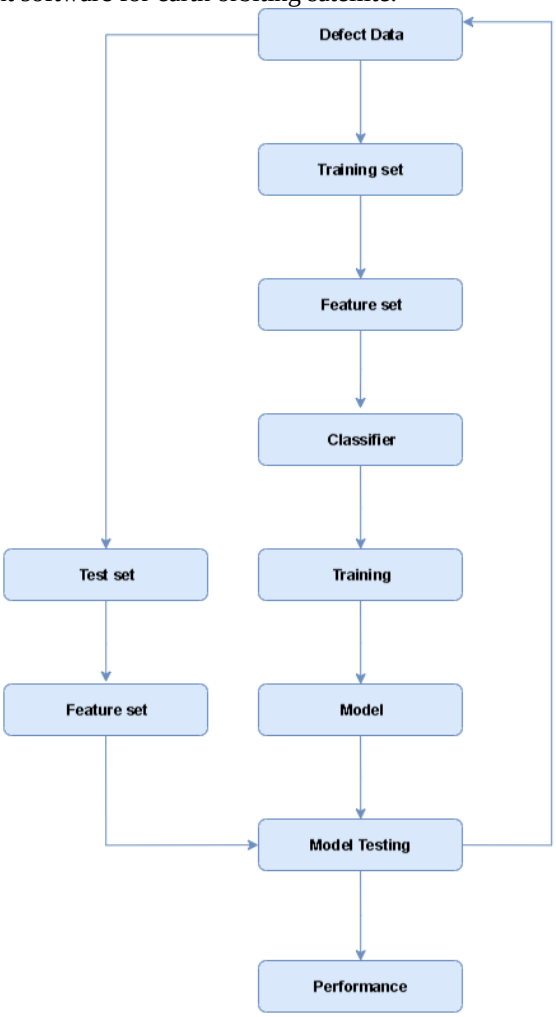


**Figure 1: The generic flow diagram of the evaluation of machine learning algorithms for the defect analysis.**

### 3.3. Analysis

For evaluation, we use the Precision, Recall in the form of an F-measure. The precision and recall are favored over the Accuracy when the classes in data are unbalanced. Since we have unbalanced classes in the dataset, we therefore, use these parameters for evaluation. The F-measure takes both the Precision and Recall for calculation and is reliable in the state of the art for similar applications. However, since the F-measure takes into account both the Precision and Recall, for the evaluation, we report only the F-measure for all the approaches. This simplifies the comparison discussion.. The Precision is calculated as:

The Precision is calculated as:

Precision=TP/(TP+FP) (1)
Where TP is True Positive and FP is False Positive.
Recall=TP/(TP+FN) (2)
Where FN is False Negative.
F-measure takes both the precision and recall into consideration and is calculated as:
Fmeasure=2*(Precision*Recall)/(Precision+Recall)
(3)
Figure 2 shows performance evaluation of the Bayes network, Random forest, SVM, and the Deep Learning based on F-measure. In Figure 2, for the "Deep learning", features are not extracted by Deep learning but only classified by Deep learning. From Figure 2, the Bayes Network has an F-measure of 0.718. Generally, the F-measure can be converted to % correct detections. This means that Bayesian network model can learn the defect and non-defect separation upto 71.8% in the CM1 dataset. This is considerably low detection performance. Compared to the Bayes network, the Random forest has increased F-measure of 0.854. This means that Random forest F-measure has a combined Precision and Recall of 85.4%. In other words, the model of Figure 1 generated by training a Random forest makes only 15% errors. The correct predictions will be 85 out of 100. This is generally acceptable range in the machine learning paradigm. In Figure 2, the SVM gets an F-measure of 0.852. This is almost similar to the F-measure of the Random forest (0.854). The SVM and Random forest has also show good overall performance in the state of the art for classification tasks. We find similar results for the CM1 dataset as that of the State of the art for SVM. The SVM reports 85% correct detections out of 100 queries. In Figure 2, the evaluation of the Deep learning gets an F-measure of 0.861. The F-measure of the Deep learning is higher than the F-measure of 0.718 for Bayes Network, 0.854 of the Random forest, and 0.852 of the SVM. The Deep learning on CM1 dataset outperforms the Bayes network, Random forest, SVM. The Deep learning has almost 16% correct defect/non-defect detections compared to the Bayes network. The correct detections of Deep learning compared to the Random forest and the SVM are not that significant, though slightly higher.

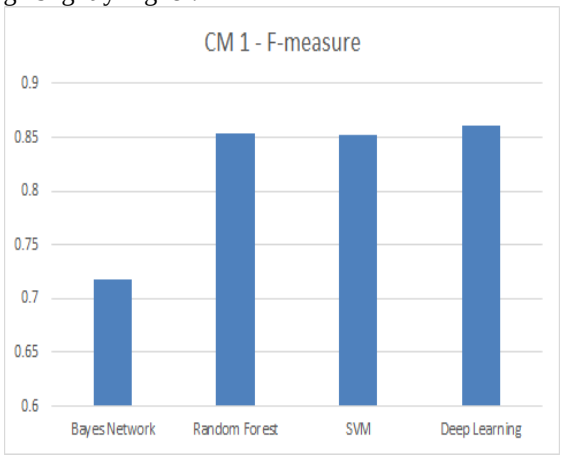


**Figure 2: The performance evaluation of Bayes network, Random forest, SVM, and the Deep learning for the CM1 defect dataset. Y-Axis shows the F-measure values. "Deep learning" represents the approach where features are classified by the Deep learning. The features are not extracted by the Deep Learning.**

Figure 3 shows the performance evaluation of the Bayes network, Random forest, SVM, and the Deep Learning based on F-measure using the JM1 dataset. From Figure 3, the Bayes Network has an F-measure of 0.71. This means that the Bayesian network model can learn the defect and non-defect separation upto 71% in the JM1 dataset. This is considerably low detection performance. Compared to the Bayes network, the Random forest has increased F-measure of 0.787. This means that the Random forest F-measure has a combined Precision and Recall of 78.7%. The correct predictions will be approximately 79 out of 100. In Figure 3, the SVM gets an F-measure of 0.722. This is almost similar to the F-measure of the Bayes network. The SVM reports 72.2% correct detections out of 100 queries. For JM1, as the features are only numerical and extracted external, the Deep learning is used as the normal Neural Network. This Neural model of the Deep learning is sometimes referred to as the dense/fully connected setup of layers. In Figure 3, the evaluation of the Deep learning gets an F-measure of 0.755. The F-measure of the Deep learning is higher than the F-measure of Bayes Network, and the SVM. The Deep learning on JM1 dataset does not outperforms the Random forest. However, it shows good detection performance compared to the Bayes network and the SVM. The correct detections of Deep learning compared to the Random forest are reduced as compared to the dataset CM1.

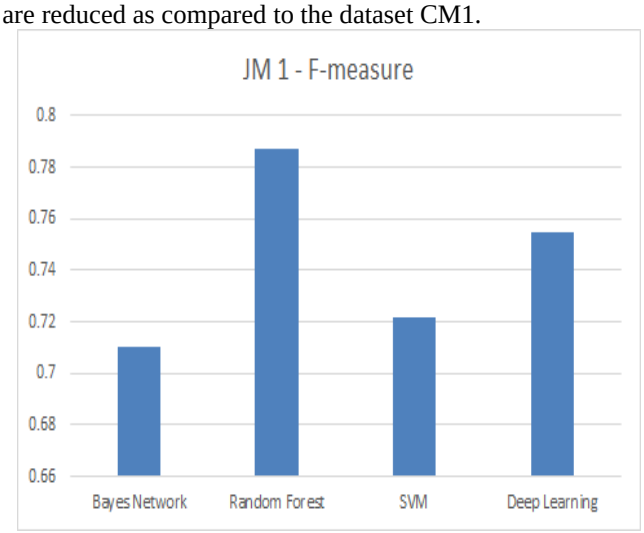


**Figure 3: The performance evaluation of Bayes network, Random forest, SVM, and the Deep learning for the JM1 defect dataset.**

Figure 4 shows the performance evaluation of the Bayes network, Random forest, SVM, and the Deep Learning based on F-measure using the KC1 dataset. From Figure 4, the Bayes Network has an F-measure of 0.73. Thus the Bayesian network can learn the defect and non-defect separation upto 73% using the KC1 dataset. As for the datasets of CM1 and JM1, the Bayes network shows considerably low detection performance. Compared to the Bayes network, the Random forest has increased F-measure of 0.848. Thus the Random forest F-measure has a combined Precision and Recall performance of 84.8%, thus accounting to 79 out of 100 correct detections. In Figure 4, the SVM gets an F-measure of 0.786. The SVM thus reports 78.6% correct detections out of 100 testing cases.

For KC1, the Deep learning is used as the dense/fully connected setup of layers. In Figure 4, the evaluation of the Deep learning gets an F-measure of 0.826. The F-measure of the Deep learning is higher than the F-measure of Bayes Network, and the SVM. This coincides with the results of the JM1 dataset. However, the Deep learning on KC1 dataset does not outperforms the Random forest. The Random forest provides almost 2.5% correction detections compared to the Deep learning. The results are in synch with the results of the dataset JM1.

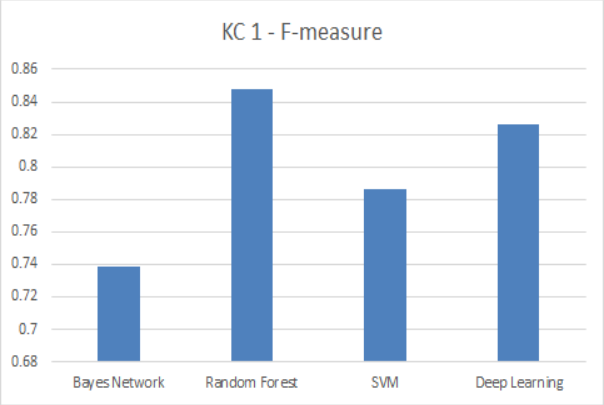


**Figure 4: The performance evaluation of Bayes network, Random forest, SVM, and the Deep learning for the KC1 defect dataset.**

Figure 5 shows the performance evaluation of the Bayes network, Random forest, SVM, and the Deep Learning based on F-measure using the KC2 dataset. From Figure 5, the Bayes Network has an F-measure of 0.797. Thus the Bayesian network can learn the defect and non-defect separation upto maximum of 80%. As similar to the datasets of CM1, JM1, and KC1, the Bayes network also shows considerably low detection performance for the KC2 dataset. The Random forest has increased F-measure of 0.825. Thus the Random forest F-measure has a combined Precision and Recall performance of 82.5%. In Figure 5, the SVM gets an F-measure of 0.784. The SVM thus reports 78.4% correct detections out of 100 testing cases. For KC2, the Deep learning is used as the dense/fully connected setup of layers. In Figure 5, the evaluation of the Deep learning gets an F-measure of 0.818. The F-measure of the Deep learning is higher than the F-measure of Bayes Network, and the SVM. This coincides with the results of the JM1, and KC1 datasets. However, the Deep learning on KC2 dataset does not outperforms the Random forest. The results are almost in synch with the results of the datasets JM1 and KC1.

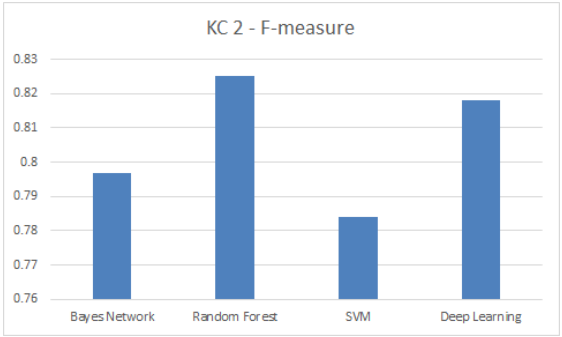


**Figure 5: The performance evaluation of Bayes network, Random forest, SVM, and the Deep learning for the KC2 defect dataset.**

Figure 6 shows the performance evaluation of the Bayes network, Random forest, SVM, and the Deep Learning based on F-measure using the PC1 dataset. From Figure 6, the Bayes Network has an F-measure of 0.803, learning the defect and non-defect separation upto 80%. As similar to the datasets of CM1, JM1, KC1, and KC1, the Bayes network also shows considerably low detection performance for the PC1 dataset. The Random forest has an F-measure of 0.927 with.almost 93% correct detections. In Figure 6, the SVM gets an F-measure of 0.897. The SVM thus reports almost 90% correct detections out of 100 testing cases. For PC1, the Deep learning is used as the dense/fully connected setup of layers. In Figure 6, the evaluation of the Deep learning gets an increased F-measure of 0.909. The F-measure of the Deep learning is higher than the F-measure of Bayes Network, and the SVM. This coincides with the results of the other three previously discussed datasets. The Deep learning on PC1 dataset does not outperform the Random forest. We observe that in 4 out of 5 cases, the Random forest outperforms the deep learning. In dataset CM1 only, the Deep learning outperforms the Random forest.

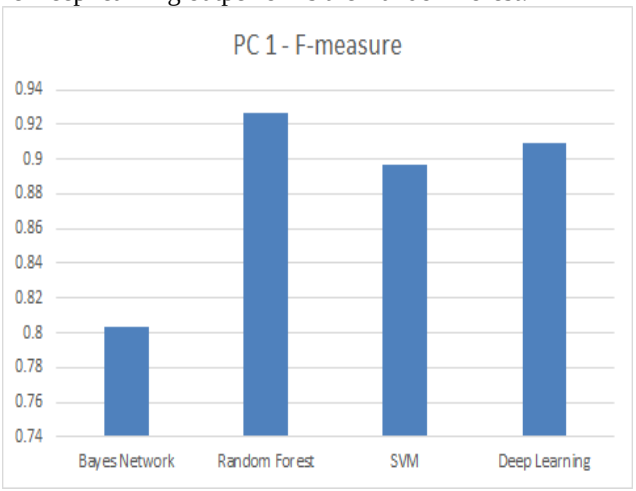


**Figure 6: The performance evaluation of Bayes network, Random forest, SVM, and the Deep learning for the PC1 defect dataset.**

### 3.4. Discussion

Figure 7 shows the average of the F-measure for the Bayes network, Random forest, the SVM, and the Deep learning for the five datasets of CM1, JM1, KC1, KC2, and the PC1 datasets. Figure 7 shows that the average F-measure for the Deep learning for the 5 datasets is higher than Bayes network, SVM, however, slightly less than the F-measure of the Random forest.

Deep learning is normally used in two settings; feature extraction and/or feature learning [38]. Deep learning can be used as a simple Neural Network classifier where the features are extracted by non-Deep approaches and Deep learning only learns the class separation. This approach is not as robust as the one where deep learning is used to extract features. In the evaluation of Deep learning for defect analysis, it is not possible to use it in the features extraction setup. As the features are only numerical and extracted external, the Deep learning be

Neural model of the Deep learning is sometimes referred to as the dense/fully connected setup of layers. In Figure 7, the evaluation of the Deep learning gets an F-measure of 0.833. The F-measure of the Deep learning is higher than the F-measure of 0.753 for Bayes Network, 0.808 of the SVM. The F-measure of the Random forest is 0.848. Thus, this on average is higher than all the other 3 classification approaches. The Deep learning on all 5 datasets outperforms the Bayes network and SVM only. The Deep learning has almost 8.5% correct defect/non-defect detections compared to the Bayes network. The correct detections of Deep learning compared to the SVM are 3%. The random forest outperforms all the three classifiers on the 5 datasets. The Random forest has 9.5% more correction detections compared to the Bayes network, 4% compared to the SVM and 1.5% compared to the Deep learning. These results coincide with the state of the art of Random forest for different applications.

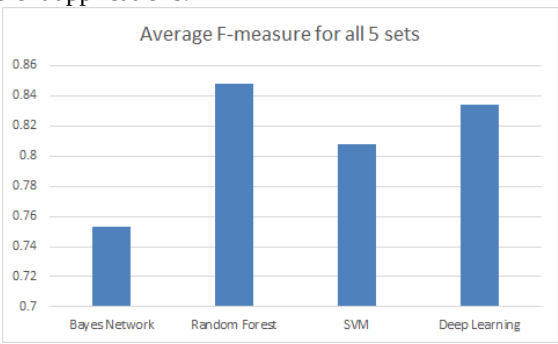


**Figure 7: Average of the F-measure for the Bayes network, Random forest, SVM, and the Deep learning for all five defect dataset.**

## IV.CONCLUSION AND FUTURE WORK

Different machine learning algorithms are used to detect defect prediction inducting changes. Nevertheless, , the performance of these learning mechanism is highly dependent on the data that is used to train the model. In this paper, we have modeled the outcomes using PROMISE dataset in five different modules and repositories: CM1, JM1, KC1, KC2, and PC1. We implemented the dataset using four different classifiers: Bayes network, Random forest, SVM, and the Deep Learning based on F-measure, making it more robust and outperform all the models available. On performing various experiments, Random Forest and Deep learning work better than Bayes network and SVM (on all five datasets). Specifically, deep learning gains competitive results comparing it with random forest making the deep learning more robust for detecting defect prediction.

Moving forward, we expect that further research could improve our model by investigating more parameters tuning and constraints. Additionally, studying different classifiers such as: neural networks will be other dimensions of extension.

### Data Availability

The experiment uses public dataset shared by Kamei et al [35], and they have already published the download address of the dataset in their paper.

## ACKNOWLEDGMENTS

This work is supported by Qassim University, represented by the Deanship of Scientific Research. The authors gratefully acknowledge them on the material support for this research under the number 3601-coc-2018-1-14-S during the academic year 1439 AH / 2018 AD.

## REFRENCES:

1. Yang, Xingguang, Huiqun Yu, Guisheng Fan, Kai Shi, and Liqiong Chen. "Local versus Global Models for Just-In-Time Software Defect Prediction." Scientific Programming 2019 (2019).
2. Fu, Wei, and Tim Menzies. "Revisiting unsupervised learning for defect prediction." In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 72-83. ACM, 2017.
3. Huang, Qiao, Xin Xia, and David Lo. "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction." In 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 159-170. IEEE, 2017.
4. Chen, Xiang, Yingquan Zhao, Qiuping Wang, and Zhidan Yuan. "MULTI: Multi-objective effort-aware just-in-time software defect prediction." Information and Software Technology 93 (2018): 1-13.
5. Yan, Meng, Yicheng Fang, David Lo, Xin Xia, and Xiaohong Zhang. "File-level defect prediction: Unsupervised vs. supervised models." In 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 344-353. IEEE, 2017.
6. Chen, Xiang, Dun Zhang, Yingquan Zhao, Zhanqi Cui, and Chao Ni. "Software defect number prediction: Unsupervised vs supervised methods." Information and Software Technology 106 (2019): 161-181.
7. Hoang, Thong, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. "DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction." In Proceedings of the 16th International Conference on Mining Software Repositories, pp. 34-45. IEEE Press, 2019.
8. Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, Qing Wang, An investigation on the feasibility of cross-project defect Prediction, Autom Softw Eng, Vol. 19, pp. 167–199, 2012
9. Duksan Ryu, Jong-In Jang, and Jongmoon Baik, A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction, JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(5): 969–980 Sept. 2015.
10. Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, Ahmed E. Hassan, Studying just-in-time defect prediction using cross-project models, Empir Software Eng, DOI 10.1007/s10664-015-9400-x., 2016
11. Duksan Ryu, Jong-In Jang, Jongmoon Bai, A transfer cost-sensitive boosting approach for cross-project defect prediction, Software Qual J., DOI 10.1007/s11219-015-9287-1, 2017
12. Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Brendan Murphy, Cross Project defect prediction: a large scale experiment on data vs. domain vs. process, Proceedings of the 7th joint meeting of the European software engineering conference, Netherlands, pp. 91-100, 2009
13. Feng Zhang, Quan Zheng, Ying Zou and Ahmad E. Hassan, Cross-Project defect prediction using a connectivity-based unsupervised classifier, 38th IEEE Software Engineering conference, USA, 2016.
14. Steffen Herbold, Alexander Trautsch, Jens Graboski, A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches, IEEE Transaction on Software Engineering, Vol. pp, No. 99, pp. 1-10,2017
15. Panichella, R. Oliveto, and A. De Lucia. Cross-project defect prediction models: L'Union fait la force. In Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), pages 164-173, 2014.
16. Nam, J., & Kim, S. Heterogeneous Defect Prediction. Proceeding ESEC/FSE 2015 Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 508-519). New York, NY, USA: ACM. , doi:10.1145/2786805.2786814, 2015
17. Ryu, D., Jang, J.-I., & Baik, J. A transfer cost-sensitive boosting approach for cross-project defect prediction. Software Quality Journal, 1-38, doi:10.1007/s11219-015-9287-1, 2015.

18. B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14(5):540-578, 2009.
19. Y. Ma, G. Luo, X. Zeng, and A. Chen. Transfer learning for cross company software defect prediction. Information and Software Technology, 54(3):248-256, 2012.
20. Xiao-Yuan Jing, Fei Wu, Xiwei Dong, Baowen Xu, An Improved SDA based Defect Prediction Framework for both Within-project and Cross project Class-imbalance Problems, IEEE Transactions on Software Engineering, DOI 10.1109/TSE.2016.2597849, 2016.
21. Arpteg, A., Brinne, B., Crnkovic-Friis, L., & Bosch, J. (2018, August). Software engineering challenges of deep learning. In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 50-59). IEEE.
22. U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review," Information and software technology, vol. 56, no. 10, pp. 1219–1232, 2014.
23. E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "What's your ml test score? a rubric for ml production systems," in Reliable Machine Learning in the Wild-NIPS 2016 Workshop, 2016.
24. C. Murphy, G. E. Kaiser, and M. Arias, "An approach to software testing of machine learning applications," in SEKE, 2007, p. 167.
25. Ma, L., Juefei-Xu, F., Xue, M., Hu, Q., Chen, S., Li, B., ... & See, S. (2018). Secure Deep Learning Engineering: A Software Quality Assurance Perspective. arXiv preprint arXiv:1810.04538.
26. Lee, Y. J., Choi, S. H., Kim, C., Lim, S. H., & Park, K. W. (2017, December). Learning binary code with deep learning to detect software weakness. In KSII The 9th International Conference on Internet (ICONI) 2017 Symposium.
27. Corley, C. S., Damevski, K., & Kraft, N. A. (2015, September). Exploring the use of deep learning for feature location. In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 556-560). IEEE.
28. Mani, S., Sankaran, A., & Aralikatte, R. (2019, January). Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (pp. 171-179). ACM.
29. C. M. Wang, J. N. Reddy and K. H. Lee, Shear Deformable Beams (Elsevier, Oxford, 2000).
30. Barandiaran, I. (1998). The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell, 20(8), 1-22.
31. James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning. Vol. 112. New York: springer, 2013.
32. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436.
33. Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends® in Machine Learning 2, no. 1 (2009): 1-127.
34. Hinton, Geoffrey E. "Learning multiple layers of representation." Trends in cognitive sciences 11, no. 10 (2007): 428-434.
35. Y. Kamei, E. Shihab, B. Adams, A. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 39(6):757–773, June 2013.
36. S. Kim, E. Whitehead, and Y. Zhang. Classifying Software Changes: Clean or Buggy? IEEE Transactions on Software Engineering, 34(2):181–196, Mar. 2008.
37. Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada
38. Albahli, Saleh. "A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction." Future Internet 11, no. 12 (2019): 246.

## AUTHORS PROFILE

**Rehan Ullah Khan** graduated from the University of Engineering and Technology Peshawar, with a BSc. degree (Information Systems) in 2004 and MSc (Information Systems) in 2006. He obtained PhD degree in 2011 from the Vienna University of Technology, Austria. He is currently an Assistant Professor at the IT Department, CoC, Qassim University, KSA. His current research interests include, segmentation, machine learning and recognition and security.

**Saleh Albahli** graduated from Qassim University with a BSc. degree (Computer Science) in 2005, the University of Newcastle in Australia with a MIT degree (Computer Science) in 2010. He obtained PhD degree (Computer Science) in 2016 from Kent State University, USA. He is currently an Assistant Professor at College of Computer, Qassim University, Saudi Arabia. His research interests include databases technology especially as they relate to Semantic web technologies and the integration of Semantic Web in database systems, big data analytics, data science and machine learning. He has attended various conferences, symposiums, workshops, training and presented many seminars. He has over 15th yeas experience in both IT industry and academia in Saudi Arabia, Australia, and USA.

**Waleed Albattah** received his Ph.D. from Kent State University, Ohio, USA. Dr. Albattah is a faculty member at Department of Information Technology , Qassim University, Saudi Arabia. His research interests are software engineering and machine learning. He is a member in ACM Society SIGSOFT and Intelligent Analytic research group at CoC.

**Mohammad Nazrul Islam Khan** is currently working as an Assistant Professor, Department of Computer Engineering, College of Computer at Qassim University, Saudi Arabia. Prior to joining Qassim University, he was teaching in Salalah College of Technology, Oman. He also worked at A.I.E.T. and A.I.D.C., India. He obtained his Ph.D. degree from University of Lucknow, India. He has published 2 books and several research papers in various journals of national and international repute.