International Journal of Computational Intelligence Research
ISSN 0973-1873 Volume 19, Number 1 (2023), pp. 51-62

https://dx.doi.org/10.37622/IJCIR/19.1.2023.51-62

Learning based Software Defect Prediction

Divya Kumari Tankala and Dr. T. Venu Gopal

Computer Science and Engineering
G.Narayanamma Institute of Technology and Science, Hyderabad, India
JNTUH College of Engineering Jagityal, Telangana, India.
divyatankala@gnits.ac.in, dr.tvgopal@gmail.com

Abstract

Software Engineering is a Comprehensive domain since software system have become larger and complex than ever. Such software characteristics make it very complex to develop fault free software. Therefore, identifying defects automatically and fixing them is challenging task. Improper modelling, lack of requirement specifications, wrong coding, poor configuration management may cause defects which leads to failure of software system. Such defects must be detected and fixed at early stage of software development to minimize cost. Machine learning algorithms widely used in the software defect prediction also achieves good results in predicting software defects using deep learning techniques. In this paper, we are providing comparative study of various algorithms like convolutional neural networks, multi-layer Perceptrons in identifying defects in software. Moreover, the experiments conducted on NASA datasets.

Keywords— Code defect detection, software defects, multilayer perceptron, CNN, Decision Tree algorithm, Logistic Regression

INTRODUCTION

Now a days, deep learning – applied to code recommendations, code defect prediction, code clone detection in software engineering. Software defects usually occurs when there is incorrect collection of requirements from stakeholders, improper design specifications. The major intent of a software development process to implement a product that is effective, low cost-efficient and of high quality. A major drawback to having good quality and reliable software is the occurrences of faults, where faults degrade the software quality, leading to rework, increases in development and maintenance costs and can become unreliable end products, also not acquire customer satisfaction. A defect management approach should be employed to

software process in order to improve software quality by tracking and analyzing of these defects. The goal of software defect management is to amplify the quality of software by identifying, analyzing and fixing the defects in the early phase of Software Development Life Cycle. However, developing fault free software system is a difficult task. There are many existing techniques and tools to predict or estimate defects of software. Most of them implemented using machine learning algorithms. Moreover, Deep learning has the capability to handle large amounts of data, provides various models to exploiting the corrupted data and learn useful features. Hundreds of thousands of computer programs fall into major domains of software. They have become complex than ever. Such characteristics of software leads to a challengeable task, which is prevention of defects in software. Always prevention is better than cure so that we can reduce the maintenance cost.

RELATED WORK

Handling defects before to the delivery may cost less compared to after the product release which costs more. Therefore, software defect prediction or estimation is highly needed activity in software engineering domain. Various approaches have been proposed in last decades still it becomes challenging as the complexity of modern software is increasing. Predicting software defects can help developers in fixing bugs and reduce error propagation with less efforts using classification and regression models. [1] Hybrid approach is the combination of artificial neural network (ANN) and Simplified Swarm Optimization (SSO) for fault prediction. ANN helps in categorizing fault-prone and nonfault-prone segments in software. Lessmann et al. [3] proposed a framework to compare software defect classification predictions and conducted a large-scale empirical comparison of 22 classification models over 10 public datasets from the NASA Metrics Data repository. Ghotra et al. [4] replicated the research of Lessmann et al. [3], built different classification models using different classification techniques and tested them on three datasets. They found that defect prediction models vary significantly in performance and that significant differences exist among different classification techniques. Herbold et al. [5] replicated 24 approaches that used different classifiers (e.g., logistic regression, C4.5 Decision Tree, RF, etc.). They found that the logistic regression model proposed by Camargo Cruz et al. [7] was best at predicting fault-prone software. Jing et al. [11] proposed using a dictionary learning technique to predict software defects. They classified modules into buggy and non-buggy using the powerful classification capability of dictionary learning. Cross-project defect prediction requires some degree of homogeneity (i.e., different projects must be describable using the same metrics) between the projects used for training and those used for testing [10]. Zhang et al. [6] used a connectivitybased unsupervised classifier to solve this problem.

APPROACH

The basic idea of the project is to build a model that predicts the defects in the software system. The proposed framework consists of six major steps: Loading

dataset, preprocessing the data, applying Deep Learning Algorithm, Train and Test data, Predict defects and Analysis. First, the software defect dataset is taken as input. Then, Data Preprocessing and normalization is performed on the dataset. On the normalized dataset, Deep Learning Algorithms are applied. Further, the data is split, where 20% of the data is used for testing the data and rest 80% of the data is used for training the data based on the algorithm. The algorithm used will predict the defects and we can further analyze the performance of the algorithms using various metrics. The figure 2.1 gives workflow mechanism of fault prediction model.

The proposed methodology to predict defects using deep learning techniques that are multi-layer perceptron and convolutional neural networks. Multilayer Perceptron consists of multiple layers of computational units, usually inter connected in a feed-forward way. By changing hyperparameters we can get better performance in identifying defects of software. The figure 2.2 helps to understand the basic process involved in choosing dataset and then preprocess the dataset to train model by choosing respective neural network model. Hyperparameters like number of epochs, batch size, optimization function, activation functions like ReLU, Tanh, SWISH, learning rate etc can be modified while training the model to achieve better performance.

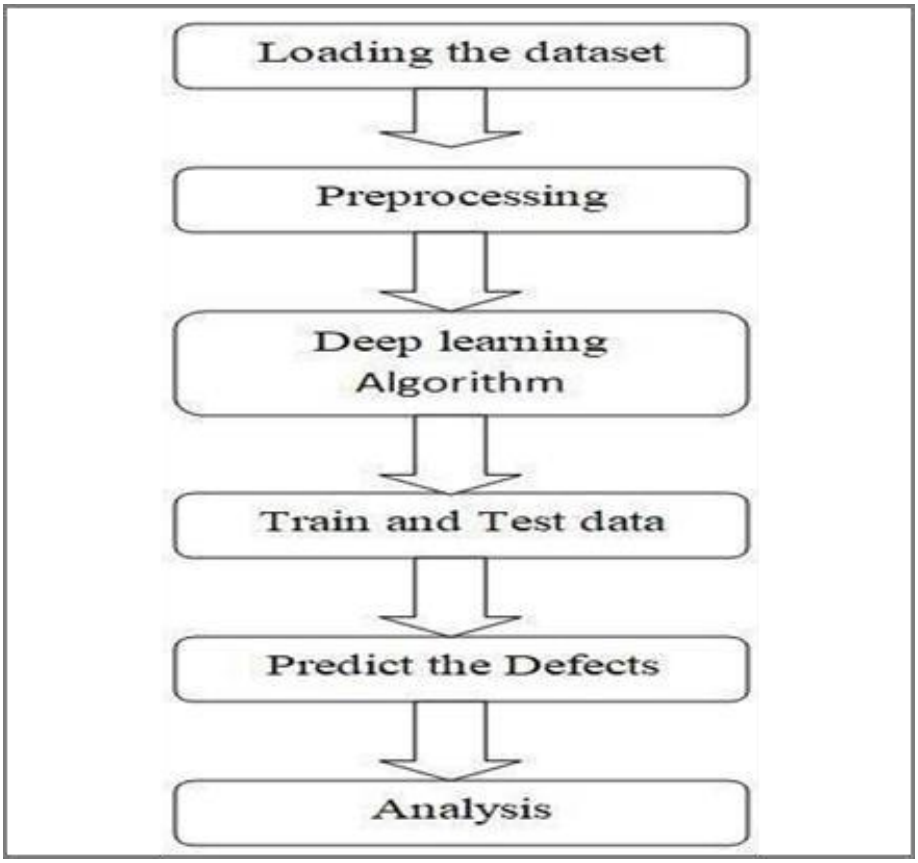

Fig 2.1: Workflow Mechanism

Convolutional neural networks model used to automatically learn the semantic and structural features of programs. The proposed approach, firstly, to extract syntactical features or tokens by constructing Abstract Syntax Trees (ASTs). But neural model can be trained with vectors, so that ASTs encoded into vectors. Such numerical vectors employed to CNN to predict defects using learned features with the help of Logistic Regression. They use a tree structured Long Short-Term Memory network (LSTM) that matches directly with the Abstract Syntax Tree representation of source code (ASTs), that help in obtaining syntactic and semantic features of source code. The model able to predict defectiveness of new code files automatically either within the project or in a different project.

Multi-layer Perceptron (MLP) is a class of Feed forward artificial neural network, applied to promise datasets. MLPs with many layers performs better in learning difficult tasks. Such neural networks have so many hyperparameters to be applied to evaluate classification model. The important factors of neural network model are number of epochs, number of neuron layers, batch size, activation function, learning rate and optimizer, which can be modified during the training process of neural model to achieve better results. Figure 2.2 helps in understanding research methodology framework and process and how CNN and MLP employed to the process of defect prediction. If the score for accuracy and defect rate are satisfied then compares both the obtained results otherwise with hyperparameter tuning we achieve best results. The NASA datasets (PC1, JM1, KC1, CM1) used to train neural model with respect to F1 score, Accuracy, defect rate.

A. Comparison of algorithms

A comparison will be conducted in order to determine the effectiveness of manipulating the studied parameters (hyperparameter, number of layers and neurons, activation function). Comparison of various models for fault prediction is possible with the help of python libraries (such as Keras, Numpy, Panda and Sklearn, Matplotlib).

Decision Tree algorithm:

This DT Algorithm belongs to the family of supervised machine learning algorithms. It can be used in prediction problems. The following pseudo code can help better to understand the prediction process.

Choose an initial (Number of layers, hyper parameters) randomly
Perform validation
Repeat
    Select the number of layer (1-N: N depends on experiments)
    Select Hyperparameter
    Perform training
    Perform validation (if prediction rate not best prediction rate)
    Replace hyperparameters
    Replace the number of neuron layer
Until matches criteria

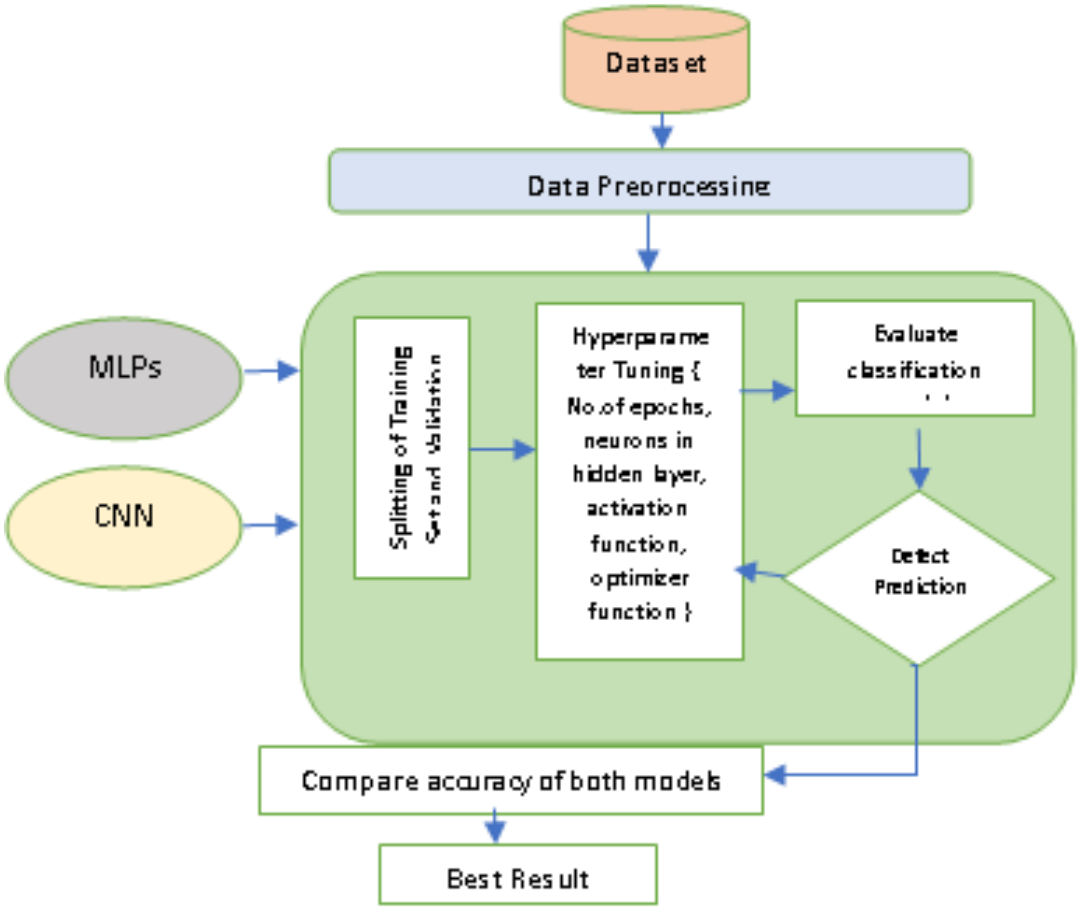

Figure 2.2: Research methodology for defect prediction

Extra tree classifier algorithm:
The Extra Trees algorithm (ET Classifier) works by creating a large number of

unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

Logistic Regression algorithm:
Logistic Regression (LR) is a machine learning algorithm based on supervised learning. Logistic regression performs the task to predict a dependent variable value (y) based on a given independent variable (x).

Gaussian naïve bayes algorithm:
Gaussian Naive Bayes(GNB) is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. First, Calculate the prior probability for given class labels, second, Find Likelihood probability with each attribute for each class then Put these values in Bayes Formula and calculate posterior probability.

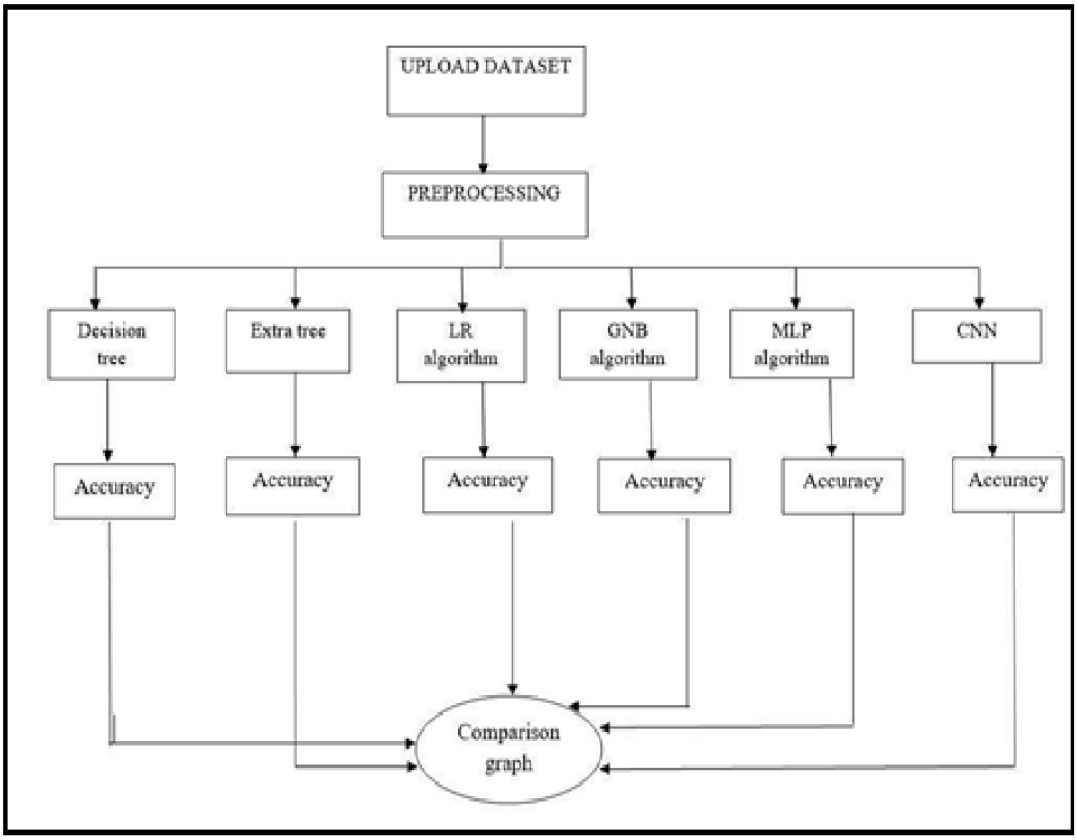

Figure 2.3: Process of comparison of classification algorithms for defect prediction

B. Dataset Description
The datasets are selected from the PROMISE Repository, include software measurement data and associated error data collected. CM1 is a NASA spacecraft

instrument written in "C". Data comes from McCabe and Halstead features extractors of source code, PC1 Data from C functions, flight software for earth orbiting satellite, JM1 is written in "C" and is a real-time predictive ground system, KC1 is a "C++" system implementing storage management for receiving and The graphs represent comparison of various models with respect to CM1, KC1, JM1 and PC1 datasets in Figure 2.4. The tables show the metrics to detect defects in software with respect to True Positive rate, True Negative rate, accuracy for the models mentioned in each table. The classification models and neural network models we used to compare to get best results in prediction of defects.

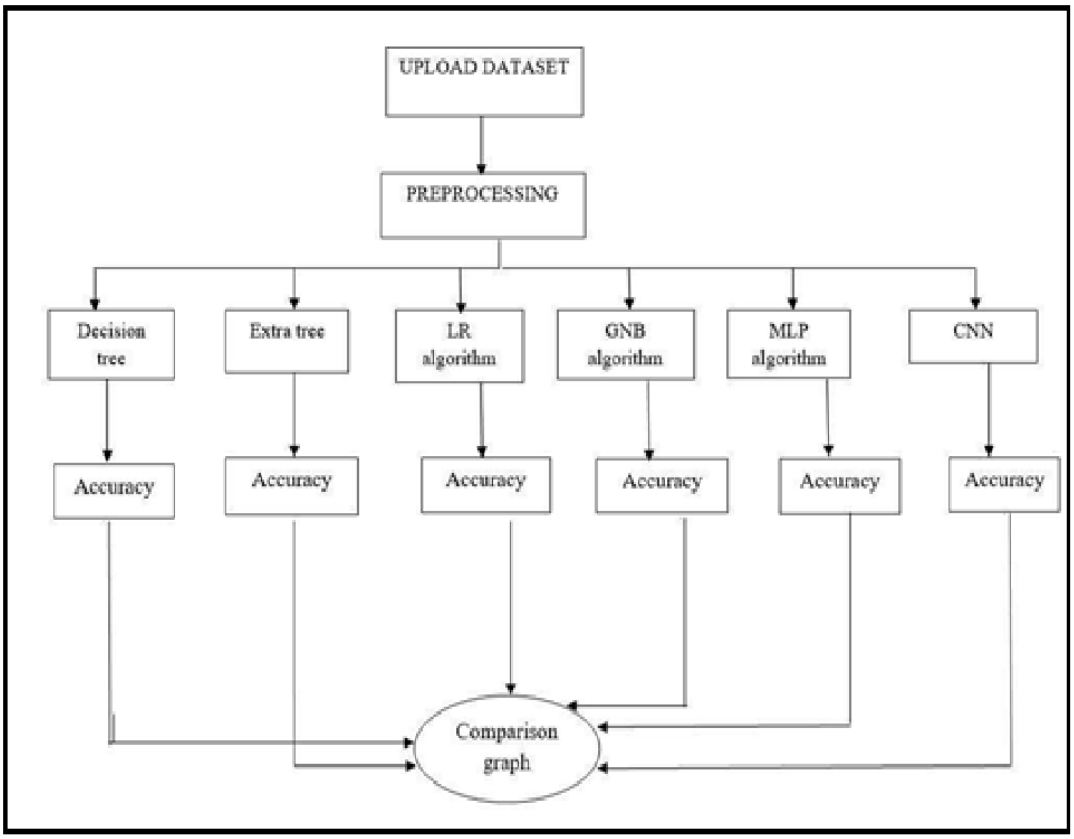

Figure 2.4: Graphs for Comparison of algorithms with CM1, KC1, JM1, PC1

Table 2.1: Comparison of different models with respect to JM1 dataset

Metric / Model	Accuracy	TPR	TNR	Defect Prediction
DT	75.18	84.81	30.38	14.99
LR	81.58	96.98	9.87	16.65
GNB	81.47	95.14	17.92	15.64
ET Classifier	82.72	98.65	8.57	16.61
MLP	82.44	97.76	11.16	16.34
CNN	82.62	97.76	12.21	16.17

The table 2.1 show the values of all metrics we use to understand the performance of software defect estimation with respect JM1 dataset. Neural network models give better results compared with classification and regression models. Similarly other tables 2.2,2.3,2.4 show case the values of metrics with respect to PC1, CM1 and KC1 datasets.

Table 2.2: Comparison of different models with respect to PCI dataset.

Metric / Model	Accuracy	TPR	TNR	Defect Prediction
DT	90.99	95.54	45	5.3
LR	91.89	99.51	15	7.7
GNB	87.38	93.06	30	6.9
ET Classifier	85.89	96.50	14	5.6
MLP	90.54	99	5	8.6
CNN	90.99	99.1	10	8.25

Table 2.3: Comparison of different models with respect to CM1 dataset

Metric / Model	Accuracy	TPR	TNR	Defect Prediction
DT	82.5	86.3	15.2	14.6
LR	81	93	7.14	13.9
GNB	80	89.53	21.42	12.5
ET Classifier	92	88.56	5.6	8.7
MLP	83	96.51	0	14.43
CNN	85	97.67	7.14	13.40

Table 2.4: Comparison of different models with respect to KC1 dataset.

Metric / Model	Accuracy	TPR	TNR	Defect Prediction
DT	86.18	92.01	44.06	7.76
LR	88.62	98.07	30.50	10.32
GNB	85.30	92.01	44.06	8.99
ET Classifier	86.96	97.24	23.72	11.30
MLP	81.75	85.95	55.93	7.6
CNN	83.88	90.08	45.76	8.91

We also implemented interface as shown in figure 2.5, using flask to choose dataset, pre-process dataset and select any of the mentioned algorithms to predict defects in

software and displays result with accuracy, TPR (True Positive Rate) and TNR (True Negative Rate) accuracies. Among these models, neural network models, gave better performance. Regression and Classification models gives moderate results in estimating defects in software.

Figure 2.5 Interface to upload dataset and predict defect in software using one of the models.

CONCLUSION AND FUTURE SCOPE

Deep learning is widely used in the area of prediction, which is one of the most promising domains. Deep learning achieves noticeable performance in terms of prediction. The goal of the project is to identify the defectiveness in the software which reduces the cost, time and effort that should be spent on software products. In this project, Defectiveness is obtained by implementing Deep Learning and Machine Learning Algorithms on the software defect dataset. We are also able to compute the performance of the Deep Learning and Machine Learning Algorithms. The experimental results show that the proposed method exhibits better performance than other compared methods. In this paper, we identified the defectiveness in software which decreases the maintenance and support effort for the project. This project can further be improved to utilize other data sets for the user interface and can be intended to address some other deep learning factors as well. Also, it would be worthwhile to investigate more about datasets and also their relationship with its fault ratio with the appropriate algorithm and its parameters. This study aims to determine the best deep learning algorithms for SFP and to reach the best possible results.

REFERENCES

[1] A.Pahal and R. S. Chillar, A hybrid approach for software fault prediction usingartificial neural network and simplified swarm optimization, IJARCCE, vol. 6, no. 3, pp. 601–605, Mar. 2017.

[2] Okutan, A., Yıldız, O.T. Software defect prediction using Bayesian networks. Empir Software Eng 19, 154–181 (2014). https://doi.org/10.1007/s10664-012-9218-8

[3] Lei Qiao, Xuesong Li, Qasim Umer, Ping Guo, Deep Learning Based Software Defect Prediction, Neurocomputing (2019)

[4] B. Ghotra, S. McIntosh, A. E. Hassan, Revisiting the impact of classification techniques on the performance of defect prediction models, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, ICSE '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 789–800. URL http://dl.acm.org/citation.cfm?id=2818754.2818850

[5] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Transactions on Software Engineering 44 (9) (2018) 811–833. doi:10.1109/TSE.2017. 2724538.

[6] F. Zhang, Q. Zheng, Y. Zou, A. E. Hassan, Cross-project defect prediction using a connectivity-based unsupervised classifier, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 309–320. doi:10.1145/2884781.2884839.

[7] E. C. Cruz, K. Ochimizu, Towards logistic regression models for predicting fault-prone code across software projects, in: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 460–463. doi:10.1109/ESEM.2009.5316002.

[8] H. Laradji, M. Alshayeb, L. Ghouti, Software defect prediction using ensemble learning on selected features, Information and Software Technology 58 (2015) 388–402. doi:{10.1016/j.infsof.2014.07.005}.

[9] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (4) (2008) 485–496. doi:10.1109/TSE.2008.35.

[10] Nam, W. Fu, S. Kim, T. Menzies, L. Tan, Heterogeneous defect prediction, IEEE Transactions on Software Engineering (2018) 874–896. doi:10.1109/TSE.2017.2720603.URL doi.ieeecomputersociety.org/10.1109/TSE.2017.2720603

[11] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, J. Liu, Dictionary learning based software defect prediction, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, ACM, New York, NY, USA, 2014, pp.

414–423. doi:10.1145/2568225.2568320. URL http://doi.acm.org/10.1145/2568225.2568320