International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Comparison of Machine Learning Models for Effective Software Fault Detection

[1]Shikha Gautam, [2]Ajay Khunteta, [3]Debolina Ghosh

Submitted: 03/02/2024 **Revised**: 11/03/2024 **Accepted**: 17/03/2024

Abstract: Software defect prediction is a field in software engineering that aims to identify and anticipate defects or bugs in software systems before they occur. The goal is to develop techniques and models that can help software development teams prioritize their testing efforts and allocate resources more effectively. For this purpose, various Machine learning techniques used and these algorithms can utilize various features, such as code metrics, historical defect data, and developer information, to build predictive models. This paper aims to develop a model for software defect prediction using various ML algorithms. Experiments were conducted using the proposed model on KC2 dataset from the NASA PROMISE repository. The Decision tree algorithm achieved 73.28%, Naïve Bayes 83.97%, KNN 80.15%, Support vector Machine 82.44% and Random Forest 80.92% for KC2 dataset. The results demonstrated that the different model succeeded in effectively predicting the defects in PROMISE datasets KC2.

Keywords: *Machine learning, Software Defect Prediction, Debugging, PROMISE dataset*

1 Introduction

Software defect prediction is of paramount importance in ensuring the delivery of high-quality software products. Early identification and prediction of defects during the software development process play a pivotal role in mitigating risks, enhancing software reliability, and minimizing the time and cost involved in resolving issues. Although manual testing remains valuable, it has inherent limitations in terms of efficiency and coverage. To address these challenges, software firms have increasingly adopted prediction models, especially those based on machine learning (ML), to bolster defect detection and reduce associated risks. ML-based defect prediction models capitalize on historical software datasets to categorize software instances as either defective or non-defective. ML-based defect prediction models leverage historical software datasets to classify software instances as defective or non-defective.

Machine Learning algorithms train and test the model using different software dataset. The ML classifier is trained by the training set, whereas the trained model's performance is assessed using the testing set.

By using ML techniques for defect prediction, software developers can benefit from improved accuracy, efficiency, and scalability compared to manual testing. ML models can evaluate huge volumes of data and classify difficult patterns that may not be readily apparent to human testers. This approach helps developers allocate their testing efforts more effectively and prioritize areas that are more likely to contain defects (He et.al., 2015)

The defect prediction models' accuracy can vary depending on factors such as the quality plus relevance of the dataset used for training Dataset, the choice of ML algorithms, and features used for prediction. Ongoing research and development in the field of software defect prediction aim to enhance the accuracy and applicability of these models, ultimately leading to more reliable and maintainable software products.

In this paper, we have implemented different machine learning algorithms to train the dataset and then test the model to find out the accuracy of each model. In section 2, we have discussed some closely related work. Basics of machine learning models are discussed in Section 3. In Section 4, we have implemented different machine learning algorithms to find the accuracy of each model for fault detection. Finally, Section 5 concludes the paper.

2 Related work

(Zhang et al.,2009) The reason of this study was to examine the connections between fault and lines of code. The author used nine NASA projects, three Eclipse system versions (2.0, 2.1, and 3.0) for this purpose. Numerous classifier techniques, including Multilayer Perceptron, Decision Tree, Spearman Correlation, Naive Bayes, and Logistic Regression have been used and performance measurement techniques like recall and precision for analysing the relationship. The findings indicated that there is a marginally favourable connection between defects and LOC.

[1]*School of Computer Engineering Poornima University, Jaipur*
[2]*School of Computer Engineering Poornima University, Jaipur*
[3]*Department of Information Technology Manipal University Jaipur*
shikha683@gmail.com, dean.fce@poornima.edu.in, debolina442@gmail.com

(Stuckman et al.,2013) The objective of this paper was considering product metrics for predicting software defects. For this purpose, author has taken dataset from 19 Apache projects and applied Correlation analysis algorithms used for evaluation. For the performance measurement of the above experiment used statistical methods. The result shows that different metrics appeared important under the different situation.

(Xia et al.,2014) The objective of this paper was outlined a method for choosing effective software metrics for fault prediction. It is suggested to use a new approach that combines correlation analysis and the ReliefF feature selection algorithm. Three distinct classifiers are tested against two other well-known feature selection methods on the PROMISE repository's historical data sets. The ANOVA analysis demonstrates that a novel feature selection technique known as ReliefF-LC (a fusion algorithm based on ReliefF and linear correlation analysis) can enhance fault prediction effectiveness.

(Ma et al., 2014) The objective of this paper was to evaluate requirement. metrics for fault prediction. For this purpose, author has taken dataset named CM1, PC1 and utilized numerous classifiers including the Logistic Regression, Random Forest, Naive Bayes, Bagging and AdaBoost algorithms. For the performance measurement of the above experiment used recall, precision, F-measure, and AUC techniques. Outcomes showed that using both requirement and design metrics together enhanced the results for fault prediction.

(Kamei et al.,2011) The objective of this paper was to look into the capabilities of code clone metrics for fault prediction. For this purpose, author has done experiment on 3 versions of the Eclipse tool (3.0, 3.1 and 3.2) by using classifier called Logistic Regression as evaluation method and performance measurement done by Recall, Precision and F1-measure techniques. The results showed that clone measurements did not enhance fault prediction and that relationship between clone metrics and fault prediction varied with dissimilar module sizes.

(Krishnan et al.,2011) This study's goal was to assess change indicators for fault prediction across several software project versions. For this purpose, author has done experiment on three releases of Eclipse by using classifier called J48 algorithm as evaluation method and performance measurement done by Accuracy, Recall, FPR techniques. It was discovered that all change measures were reliable fault predictors.

(Rahman et al.,2013) The objective of this paper was to do investigation of the defect prediction potential of process and code metrics. For this purpose, author has done experiment on 12 projects developed by Apache by using various classifier named Logistic regression, J48, SVM, and Naive Bayes as evaluation method and performance measurement done by Accuracy, Precision and Recall techniques. It was discovered that process metrics consistently outperformed code metrics.

3 Basic Concepts

The various machine learning models for foretelling software system flaws are briefly discussed in the preceding section. Software engineers and academics continue to face a difficult challenge with forecast accuracy (Dada et.al.,2021) and overall performance. For this purpose, the different machine-learning algorithms are covered in this section.

3.1 Decision Tree

Among the most well-liked and frequently utilized machine learning algorithms is the decision tree that is part of the supervised learning family. It is a versatile and interpretable method that can be applied to applications requiring regression and classification. Each internal node represents a characteristic or attribute, each branch represents a decision rule, and each leaf node represents a class label or a predicted value. Decision trees are built utilizing this tree-like flowchart structure. The construction procedure of a decision tree involves partitioning recursively the data based on the values of the features. At each step, the algorithm selects the feature that provides the most significant information gain or decrease in impurity, depending on the specific criteria used. This allows the decision tree to learn complex decision boundaries and capture nonlinear relationships between the features and the target variable. Decision trees interpretability is one of their key benefits. The resulting tree structure can be easily visualized and understood, making it useful for explaining the reasoning behind the predictions (Malhotra. 2015). Over fitting is a problem with decision trees, especially when the tree is overly deep and complicated. To address this, various techniques have been developed, such as pruning, setting a maximum depth, or limiting the number of samples required to split a node. Additionally, ensemble methods like Gradient Boosting and Random Forests can be used to combine multiple decision trees and improve overall prediction performance (Phuong Ha.,2019)

3.2 Naive Bayes

An efficient machine learning approach for classification tasks is called Naive Bayes. It is founded on the ideas of the Bayes theorem and this presupposes that, given the class variable, the features are conditionally independent of one another. The algorithm gets its name from the Bayes' theorem, which defines how to update the likelihood of a hypothesis constructed on new evidence. In the context of classification, Naive Based on the feature values of an instance, Bayes determines the likelihood that

it belongs to a specific class (Dejaeger et.al.,2012) Here's a high-level overview of how Naive Bayes works:

Training Phase

Naive Bayes requires a labelled training dataset where the class labels are known. The procedure determines each class's prior probability, which is the probability of encountering a data point belonging to that class in the training dataset. For each feature, Naive Bayes calculates the conditional probability of that feature given each class. This involves estimating the probability distribution of each feature for each class. The choice of probability distribution (e.g., Gaussian, Bernoulli, and Multinomial) depends on the type of feature (continuous, binary, and discrete).

Prediction Phase:

Given a new, unlabelled instance, Naive Bayes calculates the posterior probability of each class using Bayes' theorem. The category with the maximum posterior probability is predicted as the output class for the instance. It is computationally efficient and scales well with large datasets. It performs well even with a small amount of training data (Prabha et.al. 2020).

3.3 K-Nearest Neighbor

KNN (K-Nearest Neighbours) is an easy and powerful algorithm for machine learning algorithm utilized for both regression and classification problem (Malhotra et.al., 2015). Being non-parametric, it makes no assumptions regarding the distribution of the underlying data. KNN is predicated on the concept that similar cases frequently share a class or have comparable target values. An overview of how KNN functions is given below:

Training Phase:

In the training phase, KNN simply stores the feature vectors and corresponding class labels (in the case of classification) or target values (in the case of regression) of the training instances. It does not perform any explicit model building or parameter estimation (Koru et.al., 2005).

Prediction Phase:

Given a new, unlabelled instance that needs to be classified or predicted, KNN finds the k nearest neighbors to that instance according to Euclidean distance or any. The number of neighbors to take into account is determined by the value of the user-defined hyper parameter k. The category label with the maximum count is chosen as the projected category for the new instance via KNN, which uses a majority vote among the class labels of the k nearest neighbors.

3.4 Support Vector Machine

SVMs are effective and flexible machine learning algorithms that are frequently used for both types of applications like Regression and Classification. SVMs remain effective when dealing with complex datasets with clear margin or separation between classes. Here's a brief introduction to how Support Vector Machines work: SVMs aim to find an optimal hyper plane in a high-dimensional feature space that maximally separates the instances of different classes. In a binary classification setting, instances are divided into two classes by the hyper plane, which serves as a decision boundary (Shanthini, A.,2012). Finding the hyper plane with the highest margin is the objective. The instances adjoining to the decision boundary are named support vectors and play a critical role in defining the hyper plane.

Using a kernel function, SVMs convert the initial feature set into a higher-dimensional space. This allows SVMs to learn nonlinear decision boundaries in the original feature space. Nonlinear SVMs employ kernel functions, such as RBF (radial basis function), Sigmoid, or polynomial, in order to linearly separate the instances by mapping them onto a higher-dimensional space. The kernel function calculates the similarity or distance between instances in the transformed space. By using different kernel functions, SVMs can capture different types of nonlinear relationships between features.

Training and Prediction:

During the training phase, SVMs learn the optimal hyper plane by solving the optimization problem. This involves finding the support vectors and determining the parameters that define the decision boundary. In the prediction phase, SVMs use the learned model to classify new instances. The new instances are mapped into the feature space, and their position relative to the decision boundary determines their class assignment (Rathore et.al.2019).

3.5 Random Forest

A well-liked and effective machine learning technique called RF is used for both regression and classification problems. Multiple decision trees were combined in this ensemble method to produce predictions. The robustness, precision, and capability of Random Forests to handle high-dimensional data are well established (Catal et.al.,2009). Here's an introduction to how Random Forest works: Ensemble of Decision Trees:An ensemble of decision trees, each trained tree using a random subset of the training data and the features, makes up a Random Forest. The addition of randomization during training decreases over fitting and improves the generalizability of the model.

Random Subsampling:

During the training phase, Random Forest randomly selects subsets of the original training data through a process called bootstrapping or sampling with replacement. This means that every tree is trained on a dissimilar data subset, allowing for diversity in the ensemble (Hammouri et.al.,2018)

Additionally, only a random subset of the features is taken into account for every split in the decision tree. This advancement enhances the diversity and reduces correlation among the trees.

Majority Voting or Averaging:For classification tasks, each tree in the Random Forest predicts the class label of a given input. The class that receives the most votes from the trees is chosen as the final prediction by majority vote.

4. Experimental Setup

As seen in Figure 1, the Software repository dataset is split into two categories: a training set for creating learners using the provided learning schemes, and a test set for assessing the learners' performances. The performance report states which learning scheme is chosen and utilised to create a prediction model and forecast software defect (P. Kumudha et.al.,2016).

There is open access given to the datasets used in many software projects so that researchers can utilize them for experiments and study. The capacity of a certain dataset to serve the desired function is one of the elements that may influence the prediction accuracy of a model. A model's capacity to forecast defects is influenced by the design process. The effectiveness of the predictive outcomes might not be the similar at all times. Different kind of performance measures metrics were used like F Score, Accuracy, Recall and Precision. Table-1: Explanation of Metrics used in Software Fault Prediction.

1	LOC	Number of McCabe's line count of code
2	v(g)	Number of McCabe “Cyclomatic Complexity”
3	ev(g)	Number of McCabe “ essential complexity”
4	iv(g)	Number of McCabe “design complexity"
5	n	Number of Halstead total operators + operands
6	v	Number of Halstead "volume"
7	l	Number of Halstead "program length"
8	d	Number of Halstead "difficulty"
9	i	Number of Halstead "intelligence"
10	e	Number of Halstead "effort"
11	b	Number of Halstead
12	t	Number of Halstead's time estimator
13	LOCode	Number of Halstead's line count
14	LOComment	Number of Halstead's count of lines of comments
15	LOBlank	Number of Halstead's count of blank lines
16	LOCodeAndComment	Number
17	uniq_Op	Number of unique operators
18	uniq_Opnd	Number of unique operands
19	total_Op	Number of total operators
20	total_Opnd	Number of total operands
21	branchCount	Number of the flow graph
22	Problems	module has/has not one or more reported %Defects

Table-2: Different models with their performance measure

Model	Accuracy	Precision	Recall	F Measure
Decision Tree	73.28	36	32	33.9
Navie Bayes	83.97	70.58	42.85	53.33
KNN	80.15	54.54	42.85	48
SVM	82.44	85.71	21.42	34.28
RF	80.92	57.14	42.85	48.98

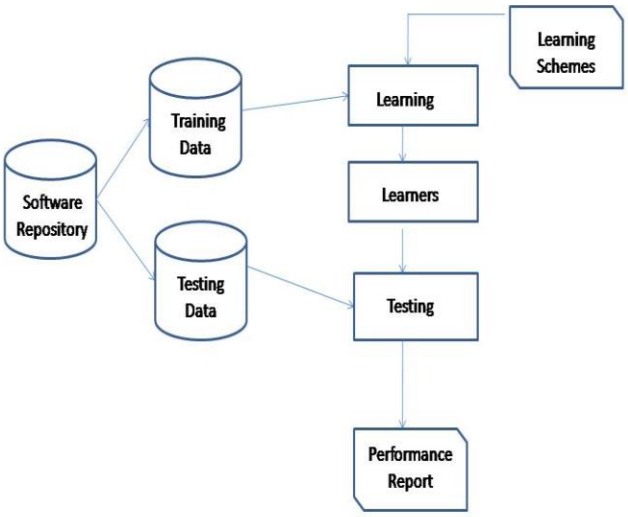

Fig-1 Model of Software Defect Prediction

4.1 Dataset Description

In this section, we have considered an open-source Promise Repository Dataset named KC2 which includes total 522 instances and 22 attributes to evaluate the accuracy of each algorithm. The dataset with attributes is presented in Table 1. Attributes include 8 derived Halstead metrics, 5 unlike LOC measure, 4 base Halsted measures, 3 McCabe metrics and 1 branch count with 1 target feature (Singh et.al.,2015)

4.2 Result analysis

The dataset's software faults were categorized using the experiment's findings. The training set (75%) and test set (25%), respectively, of the dataset utilized for the study were separated. The dataset was trained using a Random Forest, Support Vector machine, Decision Tree, Naïve Bayes and KNN Classifier.(Jing et.al.2017) Classification accuracy was used as the primary metric to evaluate the performance of the algorithms, which is calculated by dividing the number of correct prediction by the total number of predictions made. We have also calculated Precision, Recall and F1-Score of each model. The formula for Accuracy Precision, Recall and F1 Score are presented in Equation 1-4. Table 2 represents the performance measure of each model. The implementation results are also represented in graphical format, as shown in Figure 2. In Figure 3, we represent the accuracy of each model. From Figure 3, it is clear that Naïve Bayes outperforms other four machine learning models.

$$\text{Accuracy} = \frac{TP+TN}{\text{Total Sample}} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3)$$

$$F1 = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

Where, TP represents True Positive.

TN represents True Negative

FP represents False Positive

FN represents False Negative

5. Conclusion & Future Scope

The outcomes demonstrate that the model is capable of properly handling PROMISE datasets, which are renowned for their noisy features and high dimensions. In terms of Accuracy, the performance of the dataset KC2 was evaluated with the use of five different machine learning algorithms. The experimental results show that Naive Bayes and SVM performed better than KNN, Decision Tree and Random forest in predicting software defects(Jiang et.al.,2008). This experiment was very helpful for me to know how the machine learning algorithm will be implemented and what kind of dataset have to prepare for further experiments. In conclusion, it is anticipated that this research will have advanced knowledge in the area of software fault prediction. It is thought that this will make it easier and more accurate for software engineers to find faults in software systems. This could facilitate the creation of a high-quality software bundle. Future investigations can draw some interesting conclusions from this work based on the generated results (Madeyski et.al.,2015).

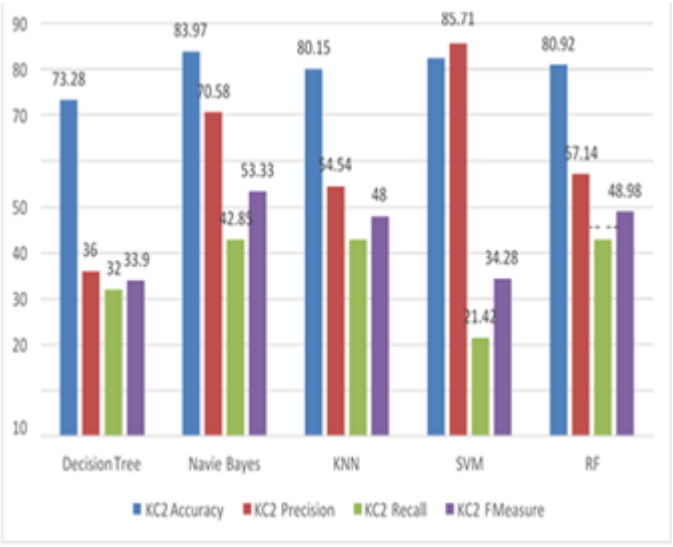

Fig 2: Performance Comparison of various Model for Dataset KC2

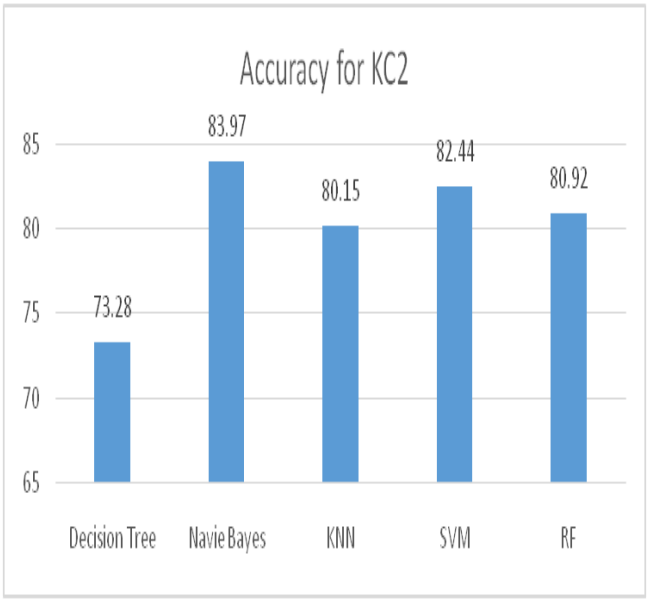

Fig 3: Accuracy Graph for different Model

References

[1] Kamei, Y., Sato, H., Monden, A., Kawaguchi, S., Uwano, H., Nagura, M., ... & Ubayashi, N. (2011, November). An empirical study of fault prediction with code clone metrics. In 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement (pp. 55-61). IEEE.

[2] Krishnan, S., Strasburg, C., Lutz, R. R., & Goševa-Popstojanova, K. (2011, September). Are change metrics good predictors for an evolving software product line?. In Proceedings of the 7th international conference on predictive models in software engineering (pp. 1-10).

[3] Rahman, F., & Devanbu, P. (2013, May). How, and why, process metrics are better. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 432-441). IEEE.

[4] Ma, Y., Zhu, S., Qin, K., & Luo, G. (2014). Combining the requirement information for software defect estimation in design time. Information Processing Letters, 114(9), 469-474.

[5] Xia, Y., Yan, G., Jiang, X., & Yang, Y. (2014, May). A new metrics selection method for software defect prediction. In 2014 IEEE International Conference on Progress in Informatics and Computing (pp. 433-436). IEEE.

[6] Stuckman, J., Wills, K., & Purtilo, J. (2013, October). Evaluating software product metrics with synthetic defect data. In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 259-262). IEEE.

[7] Zhang, H. (2009, September). An investigation of the relationships between lines of code and defects.

In 2009 IEEE international conference on software maintenance (pp. 274-283). IEEE.

[8] Dada, E. G., Oyewola, D. O., Joseph, S. B., & Duada, A. B. (2021). Ensemble machine learning model for software defect prediction. Adv. Mach. Learn. Artif. Intell, 2, 11-21.

[9] Dejaeger, K., Dejaeger, K., Verbraken, T., & Baesens, B. (2012). Toward comprehensible software fault prediction models using bayesian network classifiers. IEEE Transactions on Software Engineering, 39(2), 237-257.

[10] Singh, P., & Verma, S. (2015). S22-Cross Project Software Fault Prediction at Design Phase. International Journal of Computer, Electrical, Automation, Control and Information Engineering, 9(3), 800–8005.

[11] Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing Journal, 27, 504–518.

[12] Phuong Ha, T. M., Hung Tran, D., Le, M. H., & Thanh Binh, N. (2019). Experimental study on software fault prediction using machine learning model. In Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019. Institute of Electrical and Electronics Engineers Inc.

[13] Shanthini, A. (2012). Applying Machine Learning for Fault Prediction Using Software Metrics. International Journal of Advanced Research in Computer Science and Software Engineering, 2(6), 274–278.

[14] Prabha, C. L., & Shivakumar, N. (2020). Software Defect Prediction Using Machine Learning Techniques. In Proceedings of the 4th International Conference on Trends in Electronics and Informatics, ICOEI 2020 (pp. 728–733).

[15] Jing, X. Y., Wu, F., Dong, X., & Xu, B. (2017). An Improved SDA Based Defect Prediction Framework for Both Within-Project and Cross-Project Class-Imbalance Problems. IEEE Transactions on Software Engineering, 43(4), 321–339.

[16] Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. EmpiricalSoftwareEngineering, 13(5), 561–595.

[17] Catal, C., & Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences,179(8), 1040-1058.

[18] P. Kumudha and R. Venkatesan, "Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction," vol. 2016, 2016.

[19] Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software Bug Prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2), 78–83.

[20] Rathore, S.S., Kumar, S. A study on software fault prediction techniques. Artif Intell Rev 51, 255–327 (2019).

[21] Malhotra, R., & Jain, A. (2012). Fault prediction using statistical and machine learning methods for improving software quality. Journal of Information Processing Systems, 8(2), 241-262.

[22] Koru, A. G., & Liu, H. (2005, May). An investigation of the effect of module size on defect prediction using static measures. In Proceedings of the 2005 workshop on Predictor models in software engineering (pp. 1-5).

[23] Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal, 23(3), 393-422.

[24] He, P., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59, 170-190.