Advanced Science Letters
Vol. 20, 1945-1950, 2014

A Comparison Framework of Classification Models for Software Defect Prediction

Romi Satria Wahono[1,2], Nanna Suryana Herman[2], Sabrina Ahmad[2]
[1]Faculty of Computer Science, Dian Nuswantoro University, Indonesia
[2]Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka

Software defects are expensive in quality and cost. The accurate prediction of defect-prone software modules can help direct test effort, reduce costs, and improve the quality of software. Machine learning classification algorithms is a popular approach for predicting software defect. Various types of classification algorithms have been applied for software defect prediction. However, no clear consensus on which algorithm perform best when individual studies are looked at separately. In this research, a comparison framework is proposed, which aims to benchmark the performance of a wide range of classification models within the field of software defect prediction. For the purpose of this study, 10 classifiers are selected and applied to build classification models and test their performance in 9 NASA MDP datasets. Area under curve (AUC) is employed as an accuracy indicator in our framework to evaluate the performance of classifiers. Friedman and Nemenyi post hoc tests are used to test for significance of AUC differences between classifiers. The results show that the logistic regression perform best in most NASA MDP datasets. Naïve bayes, neural network, support vector machine and k* classifiers also perform well. Decision tree based classifiers tend to underperform, as well as linear discriminant analysis and k-nearest neighbor.

Keywords: Software Defect Prediction, Machine Learning, Classification Model, Comparison Framework

1. INTRODUCTION

A software defect is an error, failure, or fault in a software [1], that produces an incorrect or unexpected result, or causes it to behave in unintended ways. It is a deficiency in a software product that causes it to perform unexpectedly [2]. Software defects or software faults are expensive in quality and cost. Moreover, the cost of capturing and correcting defects is one of the most expensive software development activities [3]. Recent studies show that the probability of detection through defect prediction models may be higher than the probability of detection through software reviews [4]. The accurate prediction of defect-prone software modules can certainly assist testing effort, reduce costs and improve the quality of software [5].

Classification algorithm is a popular machine learning approach for software defect prediction. It categorizes the software code attributes into defective or not defective, which is collected from previous development projects. Classification algorithm is also able to predict which components are more likely to be defect-prone, supports better targeted testing resources and therefore, improved efficiency. For prediction modeling, software metrics are used as independent variables and fault data is used as the dependent variable [6]. A wide range of classification techniques have already been proposed in the predicting software defect. Since 1990, more than 20 classification algorithms have been applied and proposed as the best method for predicting the software defect, including

[1] Email: romi@brainmatics.com

logistic regression (LR) [7], decision trees (DT) [8], neural network (NN) [9], naive bayes (NB) [10], and etc.

While many studies individually report the comparative performance of the modelling techniques they have used, no clear consensus on which perform best emerges when individual studies are looked at separately. Bibi et al. [11] report that regression via classification (RvC) works well. Hall et al. [5] suggests that studies using support vector machine (SVM) techniques perform less well. These may be underperforming as they require parameter optimization for best performance. Models based on C4.5 seem to underperform if they use imbalanced data [12] [13], as the algorithm seems to be sensitive to this. NB and LR, in particular, seem to be the techniques used in models that are performing relatively well overall [10] [14]. NB is a well understood algorithm that is in common use. Studies using random forests (RF) have not performed as well as might be expected [5], although many studies using NASA dataset use RF and report good performances [15]. However, models seem to have performed best where the right technique has been selected for the right set of data. No particular classifiers that performs the best for all the datasets [14].

However, we need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models [15] [14] [4]. In this research, we propose a comparison framework, which aims to benchmark the performance of a wide range of classification models within the field of software defect prediction.

This paper is organized as follows. In section 2, the proposed comparison framework are explained. The experimental results of classification models comparison are presented in section 3. Finally, our work of this paper is summarized in the last section.

2. PROPOSED COMPARISON FRAMEWORK

The proposed framework is shown in Figure 1. The framework is comprised of 1) a dataset 2) a classification algorithms, 3) a model validation, 4) a model evaluation and 5) a model comparison.

2.1 Dataset

One of the most important problems for software fault prediction studies is the usage of nonpublic (private) datasets. Several companies developed fault prediction models using proprietary data and presented these models in conferences. However, it is not possible to compare results of such studies with results of our own models because their datasets cannot be reached. The use of public datasets makes our research repeatable, refutable, and verifiable [16]. Recently, state-of-the-art public datasets used for software defect prediction research is available in NASA Metrics Data (MDP) repository [17].

The data used in the proposed framework are collected from the NASA MDP repository. NASA MDP repository is a database that stores problem, product, and metrics data [17]. Each NASA dataset is comprised of several software modules, together with their number of faults and characteristic code attributes. After preprocessing, modules that contain one or more errors were labeled as fault-prone, whereas error-free modules were categorized as not-fault-prone. Besides line of codes (LOC) counts, the NASA MDP datasets include several Halstead attributes [18] as well as McCabe complexity measures [19]. The former estimates reading complexity by counting operators and operands in a module, whereas the latter is derived from a module's flow graph. Some researchers endorse the static code attributes defined by McCabe and Halstead as defect predictors in the software defect prediction. McCabe and Halstead are module-based metrics, where a module is the smallest unit of functionality. Static code attributes are used as defect predictors, since they are useful, generalizable, easy to use, and widely used.

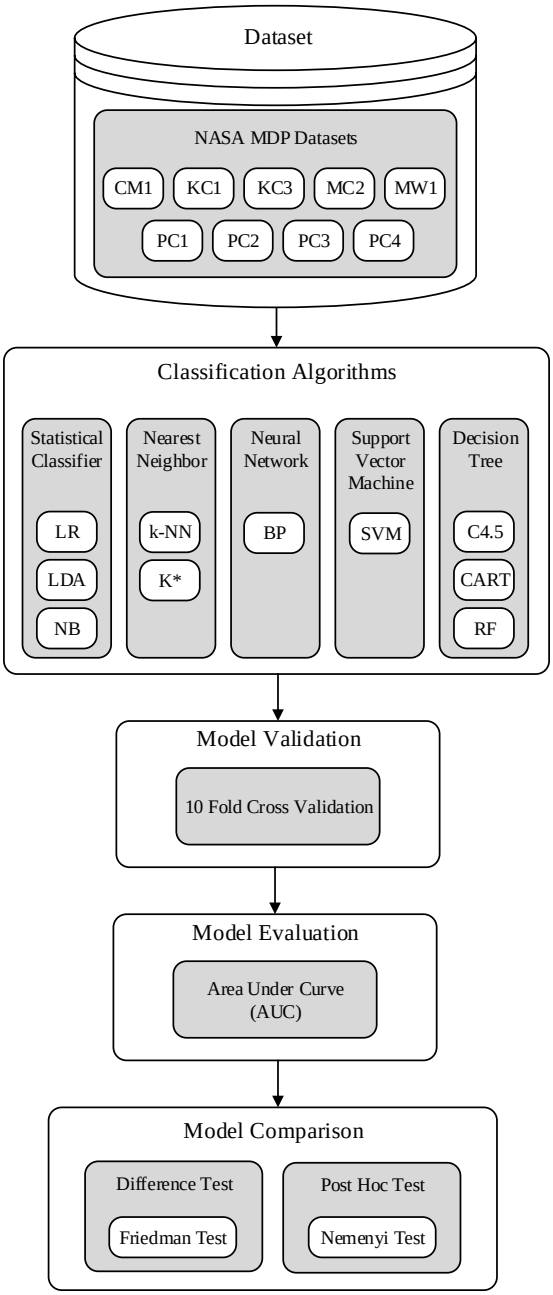

Fig. 1. Proposed Comparison Framework of Classification Models for Software Defect Prediction

In this research, we use nine software defect datasets from NASA MDP. Individual attributes per dataset, together with some general statistics and descriptions, are given in Table 1. These datasets have various scales of LOC, various software modules coded by several different programming languages including C, C++ and Java, and various types of code metrics including code size, Halstead's complexity and McCabe's cyclomatic complexity.

Table 1. Characteristics of NASA MDP Datasets

Code Attributes		NASA MDP Dataset								
		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
LOC counts	LOC_total	√	√	√	√	√	√	√	√	√
	LOC_blank	√	√	√	√	√	√		√	√
	LOC_code_and_comment	√	√	√	√	√	√	√	√	√
	LOC_comments	√	√	√	√	√	√	√	√	√
	LOC_executable	√	√	√	√	√	√	√	√	√
	number_of_lines	√		√	√	√	√	√	√	√
Halstead	content	√	√	√	√	√	√	√	√	√
	difficulty	√	√	√	√	√	√	√	√	√
	effort	√	√	√	√	√	√	√	√	√
	error_est	√	√	√	√	√	√	√	√	√
	length	√	√	√	√	√	√	√	√	√
	level	√	√	√	√	√	√	√	√	√
	prog_time	√	√	√	√	√	√	√	√	√
	volume	√	√	√	√	√	√	√	√	√
	num_operands	√	√	√	√	√	√	√	√	√
	num_operators	√	√	√	√	√	√	√	√	√
	num_unique_operands	√	√	√	√	√	√	√	√	√
	num_unique_operators	√	√	√	√	√	√	√	√	√
McCabe	cyclomatic_complexity	√	√	√	√	√	√	√	√	√
	cyclomatic_density	√		√	√	√	√	√	√	√
	design_complexity	√	√	√	√	√	√	√	√	√
	essential_complexity	√	√	√	√	√	√	√	√	√
Misc.	branch_count	√	√	√	√	√	√	√	√	√
	call_pairs	√		√	√	√	√	√	√	√
	condition_count	√		√	√	√	√	√	√	√
	decision_count	√		√	√	√	√	√	√	√
	decision_density	√		√	√	√	√	√	√	√
	edge_count	√		√	√	√	√	√	√	√
	essential_density	√		√	√	√	√	√	√	√
	parameter_count	√		√	√	√	√	√	√	√
	maintenance_severity	√		√	√	√	√	√	√	√
	modified_condition_count	√		√	√	√	√	√	√	√
	multiple_condition_count	√		√	√	√	√	√	√	√
	global_data_complexity			√	√					
	global_data_density			√	√					
	normalized_cyclomatic_complexity	√		√	√	√	√	√	√	√
	percent_comments	√		√	√	√	√	√	√	√
	node_count	√		√	√	√	√	√	√	√
Programming Language		*C*	*C++*	*Java*	*C*	*C*	*C*	*C*	*C*	*C*
Number of Code Attributes		*37*	*21*	*39*	*39*	*37*	*37*	*36*	*37*	*37*
Number of Modules		*344*	*2096*	*200*	*127*	*264*	*759*	*1585*	*1125*	*1399*
Number of fp Modules		*42*	*325*	*36*	*44*	*27*	*61*	*16*	*140*	*178*
Percentage of fp Modules		*12.21*	*15.51*	*18*	*34.65*	*10.23*	*8.04*	*1.01*	*12.44*	*12.72*

2.2 Classification Algorithms

The proposed classification framework aims to compare the performance of a wide range of classification models within the field of software defect prediction. For the purpose of this study, 10 classifiers have been selected, which may be grouped into the categories of traditional statistical classifiers (LR, LDA, and NB), nearest neighbors (k-NN and K*), NN, SVM, and decision tree (C4.5, CART, and RF). The selection aims at achieving a balance between established classification algorithms used in software defect prediction.

2.3 Model Validation

We use a stratified 10-fold cross-validation for learning and testing data. This means that we divide the training data into 10 equal parts and then perform the learning process 10 times. As shown in Table 2, each time, we chose another part of dataset for testing and used the remaining nine parts for learning. After, we calculated the average values and the deviation values from the ten different testing results. We employ the stratified 10-fold cross validation, because this method has become the standard and state-of-the-art validation method in practical terms. Some tests have also shown that the use of stratification improves results slightly [20].

Table 2. Stratified 10 Fold Cross Validation

n-validation	Dataset's Partition
1	■
2	■
3	■
4	■
5	■
6	■
7	■
8	■
9	■
10	■

2.4 Model Evaluation

We apply area under curve (AUC) as an accuracy indicator in our experiments to evaluate the performance of classifiers. AUC is area under ROC curve. Lessmann et al. [15] advocated the use of the AUC to improve cross-study comparability. The AUC has the potential to significantly improve convergence across empirical experiments in software defect prediction, because it separates predictive performance from operating conditions, and represents a general measure of predictiveness. Furthermore, the AUC has a clear statistical interpretation. It measures the probability that a classifier ranks a randomly chosen fault-prone module higher than a randomly chosen non-fault-prone module. Consequently, any classifier achieving AUC well above 0.6 is demonstrably effective for identifying fault-prone modules and gives valuable advice as to which modules should receive particular attention in software testing.

A rough guide for classifying the accuracy of a diagnostic test using AUC is the traditional system, presented by Gorunescu [21]. In the proposed framework, we added the symbols for easier interpretation and understanding of AUC (Table 3).

Table 3. AUC value, Its Meaning and Symbols

AUC	Meaning	Symbol
0.90 - 1.00	excellent classification	↑
0.80 - 0.90	good classification	↗
0.70 - 0.80	fair classification	→
0.60 - 0.70	poor classification	↘
< 0.60	failure	↓

2.5 Model Comparison

There are three families of statistical tests that can be used for comparing two or more classifiers over multiple datasets: parametric tests (the paired t-test and ANOVA), non-parametric tests (the Wilcoxon and the Friedman test) and the non-parametric test that assumes no

commensurability of the results (sign test). Demsar recommends the Friedman test for classifier comparisons, which relies on less restrictive assumptions [22]. Based on this recommendation, in our framework Friedman test is employed to compare the AUCs of the different classifiers. The Friedman test is based on the average ranked (R) performances of the classification algorithms on each dataset.

Let r_i^j be the rank of the j-th of c algorithms on the i-th of D datasets. The Friedman test compares the average ranks of algorithm $R_j = \frac{1}{D}\sum_{i-1}^{D} r_i^j$. Under the null-hypothesis, which states that all the algorithms are equivalent and so their ranks R_j should be equal. The Friedman statistic is calculated as follows, and distributed according to χ_F^2 with $C-1$ degrees of freedom, when D and C are big enough.

$$\chi_F^2 = \frac{12D}{C(C+1)}\left[\sum_j^D R_j^2 - \frac{C(C+1)^2}{4}\right]$$

If the null-hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test is used when all classifiers are compared to each other. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference, given by

$$CD = q_\alpha \sqrt{\frac{C(C+1)}{D}}$$

where critical values q_α are based on the Studentized range statistic.

3. EXPERIMENTAL RESULTS

The experiments were conducted using a computing platform based on Intel Core i7 2.2 GHz CPU, 16 GB RAM, and Microsoft Windows 7 Professional 64-bit with SP1 operating system. The development environment is Netbeans 7 IDE, Java programming language, and RapidMiner 5.2 library. We used the default parameter settings provide my RapidMiner 5.2 library.

We conducted experiments on 9 NASA MDP datasets by using 10 classification algorithms. Table 4 reports the AUCs of all classification algorithms. The last column of Table 4 reports the mean rank R_j of each classifier over all datasets, which constitutes the basis of the Friedman test.

Table 4. AUC of 10 Classification Models on 9 Datasets

	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4	*M*	*R*
LR	**0.763**	**0.801**	**0.713**	**0.766**	0.726	**0.852**	0.849	**0.81**	0.894	**0.797**	**1.44**
LDA	0.471	0.536	0.447	0.503	0.58	0.454	0.577	0.524	0.61	0.522	8.33
NB	0.734	0.786	0.67	0.739	0.732	0.781	0.811	0.756	0.838	0.761	3
k-NN	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	8.778
K*	0.6	0.678	0.562	0.585	0.63	0.652	0.754	0.697	0.76	0.658	5.33
BP	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.883	0.762	3.22
SVM	0.753	0.752	0.642	0.761	0.714	0.79	0.534	0.75	**0.899**	0.733	3.33
C4.5	0.565	0.515	0.497	0.455	0.543	0.601	0.493	0.715	0.723	0.567	7.78
CART	0.604	0.648	0.637	0.482	0.656	0.574	0.491	0.68	0.623	0.599	6.89
RF	0.573	0.485	0.477	0.525	**0.74**	0.618	0.649	0.678	0.2	0.549	6.89

The best classification model on each dataset is highlighted with boldfaced print. Table 4 shows that LR algorithm has the highest Friedman score (R). In statistical significance testing the P-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. One often "rejects the null hypothesis" when the P-value is less than the predetermined significance level (α), indicating that the observed result would be highly unlikely under the null hypothesis. In this case, we set the statistical significance level (α) to be 0.05. It means that there is a statistically significant difference if P-value < 0.05. From the experimental result, P-value is 0.0001, this is lower than the significance level α=0.05, thus one should reject the null hypothesis, and it means that there is a statistically significant difference. Consequently, one may proceed with a Nemenyi post hoc test to detect which particular classifiers differ significantly.

Nemenyi post hoc test calculates all pairwise comparisons between different classifiers and checks which models' performance differences exceed the critical difference. The results of the pairwise comparisons are shown in Table 5, which critical difference (CD) value is 4.5154.

Table 5. Pairwise Comparisons of Nemenyi Post Hoc Test

	LR	LDA	NB	k-NN	K*	BP	SVM	C4.5	CART	RF
LR	0	6.889	1.556	7.333	3.889	1.778	1.889	6.333	5.444	5.444
LDA	-6.889	0	-5.333	0.444	-3.000	-5.111	-5.000	-0.556	-1.444	-1.444
NB	-1.556	5.333	0	5.778	2.333	0.222	0.333	4.778	3.889	3.889
k-NN	-7.333	-0.444	-5.778	0	-3.444	-5.556	-5.444	-1.000	-1.889	-1.889
K*	-3.889	3.000	-2.333	3.444	0	-2.111	-2.000	2.444	1.556	1.556
BP	-1.778	5.111	-0.222	5.556	2.111	0	0.111	4.556	3.667	3.667
SVM	-1.889	5.000	-0.333	5.444	2.000	-0.111	0	4.444	3.556	3.556
C4.5	-6.333	0.556	-4.778	1.000	-2.444	-4.556	-4.444	0	-0.889	-0.889
CART	-5.444	1.444	-3.889	1.889	-1.556	-3.667	-3.556	0.889	0	0.000
RF	-5.444	1.444	-3.889	1.889	-1.556	-3.667	-3.556	0.889	0.000	0

P-value results of Nemenyi post hoc test are shown in Table 6. P-value < 0.05 results are highlighted with boldfaced print, which mean that there is a statistically significant difference between two classifiers, in a column and a row.

Table 6. P-value of Nemenyi Post Hoc Test

	LR	LDA	NB	k-NN	K*	BP	SVM	C4.5	CART	RF
LR	1	**< 0.0001**	0.986	**< 0.0001**	0.164	0.965	0.949	**0.000**	**0.005**	**0.005**
LDA	**< 0.0001**	1	**0.007**	1.000	0.526	**0.013**	**0.017**	1.000	0.992	0.992
NB	0.986	**0.007**	1	**0.002**	0.831	1.000	1.000	**0.028**	0.164	0.164
k-NN	**< 0.0001**	1.000	**0.002**	1	0.318	**0.004**	**0.005**	1.000	0.949	0.949
K*	0.164	0.526	0.831	0.318	1	0.901	0.927	0.789	0.986	0.986
BP	0.965	**0.013**	1.000	**0.004**	0.901	1	1.000	**0.046**	0.232	0.232
SVM	0.949	**0.017**	1.000	**0.005**	0.927	1.000	1	0.058	0.273	0.273
C4.5	**0.000**	1.000	**0.028**	1.000	0.789	**0.046**	0.058	1	1.000	1.000
CART	**0.005**	0.992	0.164	0.949	0.986	0.232	0.273	1.000	1	1.000
RF	**0.005**	0.992	0.164	0.949	0.986	0.232	0.273	1.000	1.000	1

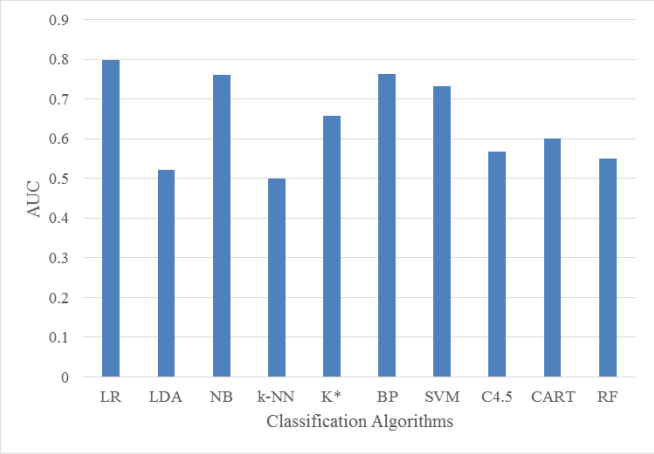

Fig.2. AUC Mean (*M*) Comparison of 10 Classification Models on 9 Datasets

As shown in Table 6, LR outperforms other models in most datasets. In terms of *R* value (Table 4) and AUC mean (*M*) (Figure 2), LR also has the highest value, followed by BP, NB and SVM in the second, third and fourth rank. Based on *P*-value results (Table 6), actually there is no significant difference between LR, NB, BP, and SVM models. This result confirmed Hall et al. [5] result that NB and LR, in particular, seem to be the techniques used in models that are performing relatively well in software defect prediction. SVM actually has excellent generalization ability in the situation of small sample data like NASA MDP dataset, but in this experiment SVM perform less well, as they require parameter optimization for best performance.

On the other hand, models based on decision tree approach (C4.5, CART and RF) seem to underperform. From *P*-value analysis, there is a significant difference between LR and the all decision tree based models. This is may be due to the imbalanced class distribution problem on software defect datasets. As we know decision tree learners create biased trees if some classes dominate. LDA and k-NN models also performing badly and to be failure in the most datasets. Significant difference table resulted by Nemenyi post hoc test is shown in Table 7.

Table 7. Significant Differences of Nemenyi Post Hoc Test

	LR	LDA	NB	k-NN	K*	BP	SVM	C4.5	CART	RF
LR	No	**Yes**	No	**Yes**	No	No	No	**Yes**	**Yes**	**Yes**
LDA	**Yes**	No	**Yes**	No	No	**Yes**	**Yes**	No	No	No
NB	No	**Yes**	No	**Yes**	No	No	No	**Yes**	No	No
k-NN	**Yes**	No	**Yes**	No	No	**Yes**	**Yes**	No	No	No
K*	No	No	No	No	No	No	No	No	No	No
BP	No	**Yes**	No	**Yes**	No	No	No	**Yes**	No	No
SVM	No	**Yes**	No	**Yes**	No	No	No	No	No	No
C4.5	**Yes**	No	**Yes**	No	No	**Yes**	No	No	No	No
CART	**Yes**	No	No	No	No	No	No	No	No	No
RF	**Yes**	No	No	No	No	No	No	No	No	No

4. CONCLUSION

A comparison framework is proposed for comparing the performance of classification algorithms in the software defect prediction. The framework is comprised of 9 NASA MDP datasets, 10 classification algorithms, 10 fold cross validation model, and AUC accuracy indicator. Friedman and Nemenyi are used to test the significance of AUC differences between models. The experimental results show that the LR perform best in most NASA MDP datasets. NB, NN, SVM and k* also perform well, and actually there is no statistically significant different between them. Decision tree based classifiers tend to underperform, as well as LDA and k-NN.

REFERENCES

[1] K. Naik and P. Tripathy, *Software Testing and Quality Assurance*. John Wiley & Sons, Inc., 2008.

[2] M. McDonald, R. Musson, and R. Smith, "The practical guide to defect prevention," *Control*, pp. 260–272, 2007.

[3] C. Jones, *Applied Software Measurement: Global Analysis of Productivity and Quality*, vol. 38, no. 1. McGraw-Hill Inc., 2008, p. 662.

[4] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, May 2010.

[5] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012.

[6] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, Apr. 2011.

[7] G. Denaro, "Estimating software fault-proneness for tuning testing activities," in *Proceedings of the 22nd International Conference on Software engineering - ICSE '00*, 2000, pp. 704–706.

[8] T. M. Khoshgoftaar, N. Seliya, and K. Gao, "Assessment of a New Three-Group Software Quality Classification Technique: An Empirical Case Study," *Empir. Softw. Eng.*, vol. 10, no. 2, pp. 183–218, Apr. 2005.

[9] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Syst. Appl.*, vol. 37, no. 6, pp. 4537–4543, Jun. 2010.

[10] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[11] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahavas, "Regression via Classification applied on software defect estimation," *Expert Syst. Appl.*, vol. 34, no. 3, pp. 2091–2101, Apr. 2008.

[12] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software," *Proc. 18th IEEE Int. Symp. Softw. Reliab.*, pp. 215–224, 2007.

[13] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Syst. Softw.*, vol. 83, no. 1, pp. 2–17, Jan. 2010.

[14] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A General Software Defect-Proneness Prediction Framework," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, May 2011.

[15] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008.

[16] C. Catal and B. Diri, “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem,” *Inf. Sci. (Ny).*, vol. 179, no. 8, pp. 1040–1058, Mar. 2009.
[17] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, “Reflections on the NASA MDP data sets,” *IET Softw.*, vol. 6, no. 6, p. 549, 2012.
[18] M. H. Halstead, *Elements of Software Science*, vol. 7. Elsevier, 1977, p. 127.
[19] T. J. McCabe, “A Complexity Measure,” *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, 1976.
[20] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining Third Edition*. Elsevier Inc., 2011.
[21] F. Gorunescu, *Data Mining: Concepts, Models and Techniques*, vol. 12. Springer-Verlag Berlin Heidelberg, 2011.
[22] J. Demsar, “Statistical Comparisons of Classifiers over Multiple Data Sets,” *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.