International Journal of Knowledge and Systems Science
Volume 11 • Issue 2 • April-June 2020

Comparison of Machine Learning Techniques for Software Quality Prediction

Somya Goyal, Manipal University Jaipur, Jaipur, India & Guru Jambheshwar University of Science & Technology, Hisar, India

Pradeep Kumar Bhatia, Guru Jambheshwar University of Science & Technology, Hisar, India

ABSTRACT

Software quality prediction is one the most challenging tasks in the development and maintenance of software. Machine learning (ML) is widely being incorporated for the prediction of the quality of a final product in the early development stages of the software development life cycle (SDLC). An ML prediction model uses software metrics and faulty data from previous projects to detect high-risk modules for future projects, so that the testing efforts can be targeted to those specific 'risky' modules. Hence, ML-based predictors contribute to the detection of development anomalies early and inexpensively and ensure the timely delivery of a successful, failure-free and supreme quality software product within budget. This article has a comparison of 30 software quality prediction models (5 technique * 6 dataset) built on five ML techniques: artificial neural network (ANN); support vector machine (SVMs); Decision Tree (DTs); k-Nearest Neighbor (KNN); and Naïve Bayes Classifiers (NBC), using six datasets: CM1, KC1, KC2, PC1, JM1, and a combined one. These models exploit the predictive power of static code metrics, McCabe complexity metrics, for quality prediction. All thirty predictors are compared using a receiver operator curve (ROC), area under the curve (AUC), and accuracy as performance evaluation criteria. The results show that the ANN technique for software quality prediction is promising for accurate quality prediction irrespective of the dataset used.

KEYWORDS

Area Under the Curve (AUC), Artificial Neural Network (ANN), Classification Tree, Fault Prediction, KNN, Machine Learning (ML), Naïve-Bayes, Receiver Operator Curve (ROC), Software Quality, Support Vector Machine (SVM)

1. INTRODUCTION

Software with high quality which meets the user's needs, requirements and performs as per the expectation, are tempted in the entire world since always. The Software must ensure the failure-free performance; that is the Reliability of product. So many quality attributes and metrics with numerous Quality Assurance techniques are developed, but still the question: how to ensure that the resulting product will possess good quality is an open problem. The early detection of failure-prone modules directly correlates with the quality of end-product. The fault prediction involves the early detection of those "risky" modules of the software which prone to errors, impair the quality and will surely incur heavy development (testing) and maintenance cost. The early detection of faulty (buggy) modules improves the effectiveness of quality enhancement activities and ultimately improves the overall quality. The software fault prediction is used as an indicator for software quality for two reasons: (1) Quality is inversely proportional to the failures, which in turn caused by faults (development anomalies). To ensure high quality, failures must be minimized to zero. It can be achieved only by

DOI: 10.4018/IJKSS.2020040102

100% detection and removal of defects from the modules. Hence, the accuracy in defect prediction is the most crucial factor to judge the quality of software product. (2) The early fault detection provides decisive power to the entire development team to strategically allocate the testing resources. Quality improvement activities can intensively be applied to the detected risky modules (Kan, 2002). Scheduling can be done more effectively to avoid delays. Prioritization of failure-prone modules for testing can be done. Ultimately, the quality can be improved and ensured by prediction of faults in early phases of development cycle. In case, faulty modules are not detected in early development phases, then the cost of getting the defect fixed increases multifold. Along with the increased cost, the chances of getting the defect detected by the customer in the live environment also increase. In situ, defect is found in the live environment, may stop the operational procedures which can eventually result in fatal consequences too. Software industry already witnessed such failures like NASA Mass Climate Orbiter (MCO) spacecraft worth $125 million lost in the space due to small data conversion bug (NASA, 2015).

Till today, so many researchers have contributed in this problem domain, but the gap is that their results are not univocal. Their work is not generalized but biased to the specific datasets or the techniques used to solve the problem.

Hence, the more accurate fault prediction is done, the more precise quality prediction is achieved. The present work is focused on the following research goals:

R1: To transform the software quality prediction problem as a learning problem (classification problem).
R2: To create ML prediction models using static code metrics as predictor.
R3: To evaluate the accuracy of prediction models empirically.
R4: To find which ML technique outperforms other ML techniques.

The major contribution of this work, is to develop and validate a cross-platform generalized model, to accurately predict the faulty modules in the software during development, so that the defects cannot propagate to the final phases and ultimately, the quality of software can be improved. As the quality of the software is inversely proportional to the number of defects in the end-product (Kan, 2002).

In this Paper; (1) The Software quality prediction problem is formulated as a two-class classification problem. Following this approach, a few features (attributes) from the previous project dataset are used as predictors and the quality of the module is generated as response by the classifier in terms of the predicted class label. (2) In total, 30 models are built for software quality prediction using 5 ML techniques (ANN, SVM, Naïve-Bayes classifier, DT, and k-nearest neighbor) over 6 datasets. Each model is validated using 10-fold cross-validation. (3) An empirical comparison among the developed prediction models is made using ROC, AUC, and accuracy as performance evaluation criteria.

The rest of the paper is organized as follows. Section 2 provides the glimpses of the related work from the literature. Section 3 gives the detailed description for the experimental set-up. The details to datasets, metrics, methods and performance evaluation criteria as research methodology are included in Section 4. Section 5 covers the results and the discussions. The work is concluded in Section 6, with remarks on the future work.

2. RELATED WORK

In this section, a review of literature is presented on the use of machine learning techniques for the prediction of software quality as well as the work already carried out about the static code metrics to detect faulty modules. The study is tabulated as Table 1, and shows that the static code metrics have great influence on the software quality and hence have been selected as inputs to develop the software quality prediction models. From our observation artificial neural network and support vector machines are commonly machine learning techniques being used by the authors. In this work, we proposed prediction models for quality assessment of the software using five ML techniques namely artificial neural network, Decision Trees, Naïve Bayes Classifier, k-nearest neighbor, and support vector machine.

The study done reflects the popularity of AUC-ROC as performance evaluation criteria. We are using receiver operator characteristics (ROC) curves, area under the curve (AUC) measure and Accuracy as performance evaluation criterion to compare the models proposed. Earlier the researchers have utilized variety of datasets as NASA, Eclipse and others. For our work, we are choosing the NASA PROMISE repository for the reason that the NASA deploys contractors extensively for data-collection both from government and commercial organizations.

Table 1 emphasizes on the work carried out by the researchers in the field of Software quality prediction via ML techniques to compare the various approaches and selecting the performance measures to be considered for the comparison of the effectiveness of the proposed models for the prediction of quality using ML approach.

The State-of-the-Art of incorporating Machine Learning techniques for software quality prediction is summarized in Table 2. It is emphasized that Decision Trees, Artificial Neural Networks, Bayesian Learning, Support Vector Machines are the most popular ones among the researchers for developing the software quality prediction models. We also proposing models based on the application of the current state-of-the-practice of the ML techniques for software defect prediction.

3. EXPERIMENTAL SET-UP

This section covers the experimental set-up required (depicted as in Figure 1) for our work - the comparison of effectiveness of 30 proposed models is done. 5 ML techniques (ANN, SVM, DT, Naïve-Bayes, kNN) with 6 datasets from PROMISE repository are employed to develop 30 models that predict whether the module is “buggy” or “clean”.

The steps taken to carry the entire research include Data Collection, Feature Extraction & Selection, Building the Prediction Model, Validation of the developed Model, Predicting the software quality using the validated classifier and Evaluation of the classifiers to make a comparison among them.

The complete experimental flow is modeled in Figure 1. A total of 30 (6 datasets × 5 ML classification techniques) distinct prediction models are built, validated and compared in the current paper.

4. RESEARCH METHODOLOGY

This section covers the steps followed to carry this research work with necessary references to the previous related studies in literature.

4.1. Data Description

The work utilizes the Data collected from NASA projects using McCabe metrics which are made available in the PROMISE repository. This research is done with six fault prediction datasets named CM1, KC1, KC2, PC1, JM1 and ALL_DATA (combination of CM1, KC1, KC2, PC1, JM1). Data has been collected using McCabe and Halstead features extractors from the source code of multiple projects (Sayyad and Menzies, 2005), (PROMISE). Each instance in the dataset has 22 attributes, out of which 21 attributes can be used as features and 1 attribute serves as class-label for classification purpose. The class distribution of ‘Buggy’ and ‘Clean’ classes among the dataset is depicted in Table 3 and graphically shown in Figure 2. The PROMISE repository is widely used by researchers for quality prediction (Czibula et al, 2014), (Dejaeger et al, 2013), (Laradji et al, 2015) and (Rodríguez et al, 2013).

Table 1. Summary of literature on software quality prediction using ML techniques

Study Name	Technique Used	Dataset Used	Metrics Used	Performance Evaluation Criteria Used	Critical Comment [Goyal & Bhatia]
Gyimothy et al. (2005)	Decision tree, Linear and Logistic regression, Neural network	Source code of Mozilla with the use of Columbus framework	CK metrics suite, LCOM, and LOC	Precision, Correctness, Completeness	Precision of Models is Not Satisfactory
Zhou and Leung (2006)	Logistic regression, Naive Bayes, and Random forest	NASA consisting of 145 classes, 2107 methods and 40 KLOC.	CK metrics suite, and SLOC	Accuracy	Models are capable to find low-severity faults only
Gondra (2008)	ANNs, SVMs	NASA data sets	McCabe metrics	Accuracy	SVMs are advocated to be better than ANNs
Mishra and Shukla (2012)	SVM, Fuzzy, and Naive Bayes	Eclipse and Equinox data sets.	Complexity metrics (FOUT, MLOC) and Abstract syntax tress based metrics	Accuracy	Fuzzy Rules are emphasized more
Dejaeger et al. (2013)	Bayesian Classifier	NASA MDP Repository	McCabe and Halstead metrics and OO metrics	AUC, H-measure	A balance between the comprehensibility and predictive power of predictors is stressed more
Rodríguez et al. (2013)	Sub-group discovery	Promise repository - NASA projects (CM1, KC2, KC2, KC3, MC2, MW1 and PC1)	McCabe and Halstead metrics	Accuracy	Descriptive approach is introduced and experimented with 70% Accuracy
Malhotra (2013)	Decision Tree	AR1 and AR6	McCabe metrics	AUC-ROC	Comparative analysis is done biased with the datasets used
Czibula et al. (2014)	Association Rules	NASA data sets (CM1, KC2, KC2, KC3, MC2, MW1 and PC1)	McCabe metrics	Accuracy, AUC	Length and confidence of rules are still under investigation
Laradji et al. (2014)	SVM	NASA datasets: PC2, PC4, and MC1	McCabe metrics	AUC	Results are biased both with the dataset used and data preprocessing techniques employed
Erturk and Sezer (2015)	ANN, SVM and ANFIS	Promise Software Engineering repository data	McCabe metrics	AUC-ROC	Parameter reduction is done beautifully in a practical way
Abaei et al. (2015)	Semi-supervised Hybrid Self-Organizing Map (HySOM)	NASA data sets	McCabe and Halstead metrics	Accuracy	Unsupervised ML is experimented with not so good justifications
Erturk and Sezer (2016)	ANN, ANFIS	Promise Software Engineering repository data	McCabe metrics and OO metrics	AUC-ROC	Predictor is developed as a tool which can be used in real-time data collection for future projects
Boucher et al. (2018)	SVM and Clustering	Promise and Eclipse	OO metrics	ROC	ROC is advocated as the best measurement stick for software defect prediction

Table 2. State-of-the-Art for software quality prediction using ML techniques

Year	Study	ML Techniques
1988	(Selby and Porter, 1988)	Decision Trees
1993	(Briand et al., 1993)	Decision Trees
1997	(Khoshgoftaar et al., 1997)	Artificial Neural Networks
1997	(Lanubile and Visaggio, 1997)	Artificial Neural Networks
1999	(Fenton and Neil, 1999)	Bayesian Learning
2005	(Gyimothy et al, 2005)	Decision Trees, Artificial Neural Networks
2006	(Zhou and Leung, 2006)	Bayesian Learning, Decision Trees, k-Nearest Neighbor
2006	(Song et al., 2006)	Bayesian Learning, Association Rule based Learning, Decision Trees
2007	(Menzies et al., 2007)	Bayesian Learning- Naïve Bayes Classifiers
2008	(Gondra, 2008)	Artificial Neural Networks, Support Vector Machines
2011	(Diri et al., 2011)	Bayesian Learning
2012	(Mishra and Shukla, 2012)	Support Vector Machines, Naïve Bayes Classifiers
2013	(Chen et al., 2013)	Support Vector Machines
2013	(Malhotra, 2013)	Decision Trees
2013	(Rodríguez et al, 2013)	Sub-group discovery
2014	(Czibula et al, 2014)	Association Rules
2015	(Laradji et al, 2015)	Support Vector Machines
2015	(Erturk and Sezer, 2015)	Support Vector Machines, Artificial Neural Networks, ANFIS
2015	(Abaei et al, 2015)	Self-Organizing Maps
2017	(Moussa et al., 2017)	Particle Swarm Optimization, Genetic Algorithms
2017	(Li et al., 2017)	Convolution Neural Networks
2018	(Kumar et al, 2018)	Support Vector Machines
2018	(Zhang et al., 2018)	Decision Trees, Artificial Neural Networks, Bayesian Learning
2018	(Jayanthi et al., 2018)	Artificial Neural Networks
2018	(Diana et al, 2018)	Artificial Neural Networks with Association Rules
2018	(Boucher and Badri, 2018)	Decision Trees, Artificial Neural Networks, Bayesian Learning, Support Vector Machines

4.2. Effectiveness of Metrics

Each dataset has 22 metrics (attributes) which includes- 5 different lines of code measure, 3 McCabe metrics, 4 base Halstead measures, 8 derived Halstead measures, a branch-count, and 1 Class_Label. The Class_Label is used as response in our 2-class classification problem. The definition for attributes is given in Table 4 (Huda et al, 2017).

In our work, attribute #1, #2, #3 and #4 are selected as features to build the quality predictors. The selected features are the McCabe's Complexity metrics (Thomas, 1976). These are static code metrics and in direct association with the quality of the software. The reason of using static code metrics as predictors is two-fold. First, these attributes are successfully used and widely accepted to predict the software defects and quality (Czibula et al, 2014), (Dejaeger et al., 2013), (Laradji et

CHECK

Figure 1. Experimental set-up

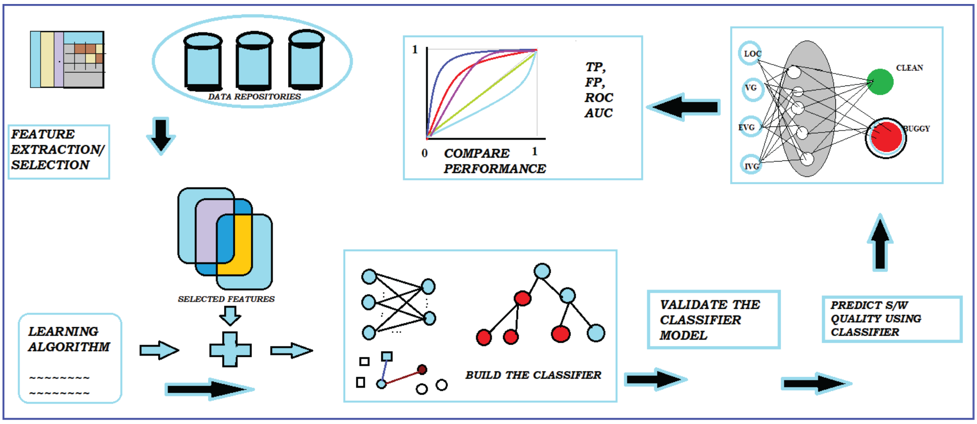

DATA REPOSITORIES
FEATURE EXTRACTION/ SELECTION
SELECTED FEATURES
LEARNING ALGORITHM
~~~~~~~~
~~~~~~~~
BUILD THE CLASSIFIER
VALIDATE THE CLASSIFIER MODEL
PREDICT S/W QUALITY USING CLASSIFIER
1
0
1
COMPARE PERFORMANCE
TP,
FP,
ROC
AUC
LOC
VG
EVG
IVG
CLEAN
BUGGY

Table 3. Description to data-set used

#	Data-Set Name	# Total Instances	#Buggy Instances	# Clean Instances
1.	CM1	498	49	449
2.	KC1	2109	326	1783
3.	KC2	522	107	415
4.	PC1	1109	77	1032
5.	JM1	10885	2106	8779
6.	ALL_DATA	15123	2665	12458

Figure 2. Class distribution among data- 2 class (Buggy-Clean)

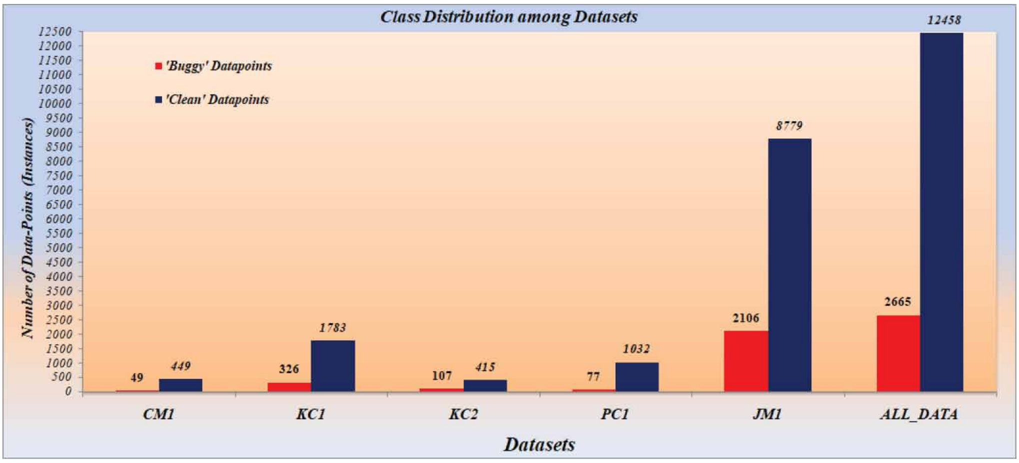


Table 4. Features/Attributes definition

#	Feature Name	Feature Definition
1	loc	LOC (McCabe's)
2	v_(g)	Cyclomatic Complexity (McCabe's)
3	ev_(g)	Essential Complexity (McCabe's)
4	iv_(g)	Design Complexity (McCabe's)
5	_n'	Operator and Operand total (Halstead)
6	_v' ∅	Volume (Halstead)
7	_l' ∅	Program Length (Halstead)
8	_d'	Difficulty (Halstead)
9	_i'	Intelligence (Halstead)
10	_e'	Effort (Halstead)
11	_b'	(Halstead)
12	_t'	Time (Halstead)
13	lOC_Code yes	LOC (Halstead)
14	lOC_Comment	Line of Comments (Halstead)
15	lOfBlank	Line of Blanks (Halstead)
16	lOfCode +Comment	Line of Code and Comment (Halstead)
17	Uniq-Op	No. of unique operator
18	Uniq-Opnd	No. of unique operand
19	Total-Op	Total operator
20	Total-Opnd	Total operands
21	Branch-Count	Count of Branch
22	**Class_Label**	{Buggy, Clean}

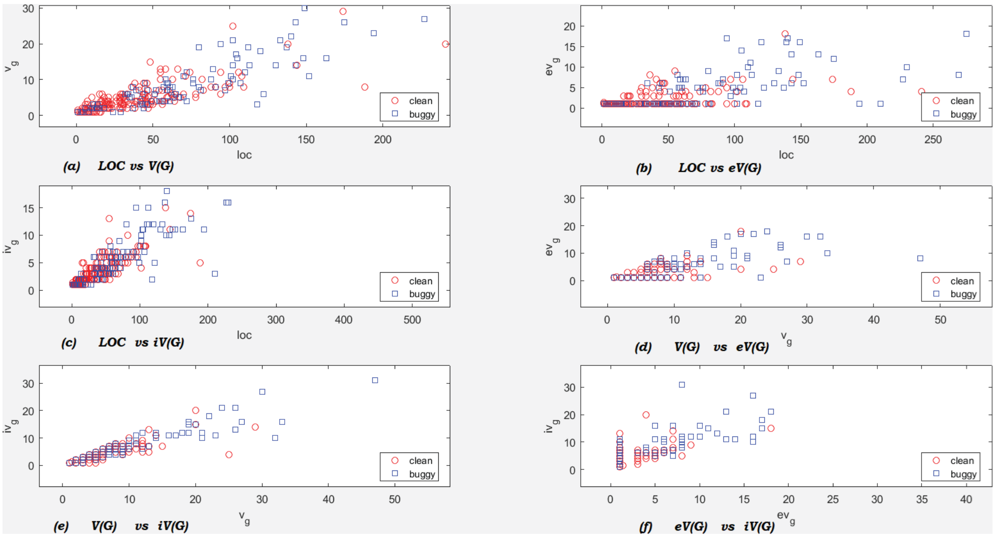
al., 2015), (Rodríguez et al., 2013). Second, these metrics can be computed very easily at low costs (Menzies et al., 2007).

The correlation among the selected features for the classes is shown in Figure 3 where red bubbles represent the 'clean' and blue squares represent the 'buggy' data-points (Goyal and Parashar, 2018). This figure shows the pair-wise correlation between the attributes selected for the prediction models. Figure 3(a) shows strong correlation for 'LOC' and 'cyclomatic complexity'. Figure 3(b) captures 'LOC' vs 'essential complexity'. Figure 3(c) is about 'LOC' vs 'design complexity'. Figure 3(d), Figure 3(e) and Figure 3(f) clearly show the separation of 'clean' and 'buggy' modules along with the interdependency of features 'cyclomatic complexity' vs 'essential complexity', 'cyclomatic complexity' vs 'design complexity' and 'essential complexity' vs 'design complexity' respectively.

4.3. Methods

The prediction models are built using Machine Learning approaches which are proven to be promising in this research field as emphasized in Table 2 as the state-of-the-art. The machine learning techniques used to build our prediction models are artificial neural network, support vector machine, naïve-bayes classifier, classification trees and k-nearest neighbor (Mitchell, 1997). Each of 5 ML classifiers is trained and tested for all 6 datasets (CM1, KC1, KC2, PC1, JM1 and All_Data) resulting in 30 prediction models:

Figure 3. Correlation among selected attributes with the classes (shown with red and blue)



(a) LOC vs V(G)
(b) LOC vs eV(G)
(c) LOC vs iV(G)
(d) V(G) vs eV(G)
(e) V(G) vs iV(G)
(f) eV(G) vs iV(G)

- *Artificial Neural Networks* are multilayer perceptrons used for classification and regression. In our work, we utilized MLP with one hidden layer. The size of input layer is 4 and output layer is 2. Algorithm used is Back-Propagation for learning;
- Decision tree is a hierarchical data structure which implements the divide-and-conquer strategy. It is nonparametric method widely being used for both classification and regression. In our prediction models, CART algorithm is used for classification;
- Support Vector Machines are maximum margin methods. It allows a variety of linear, radial basis function (RBF) and Gaussian kernels. We deployed SVM with RBF kernel (Kumar et al., 2018);
- Naïve-Bayes classifier is based on probability models. It assumes conditional independence among the features for the given the class label. The conditional density is given as a product of one dimensional densities:

$$p\left(x \mid y=c,\theta\right)=\prod_{j=1}^{D} p\left(x_j \mid y=c,\theta_{jc}\right)$$

- The nearest neighbor class of classifiers adapts the amount of smoothing to the local density of data. The degree of smoothing is controlled by k. k denotes the number of neighbors taken into account. In this paper, 10 nearest neighbors are taken into consideration for prediction by setting value of k as 10. For the selection of value of parameter k; we started with k=1 then keep on incrementing the value till the accuracy keeps optimizing and at k=10, the model got maximum accuracy. Further, increment will degrade the performance. Hence, the value of k is fixed as 10. (Nwe et al, 2020).

4.3.1. Performance Evaluation Criteria

The comparison among the prediction models is made by evaluating the performance of individual classifier over following criteria:

- Confusion matrix contains information about the Target (actual) and Output (predicted) classifications done by the fault prediction model (Kumar et al., 2018);
- The sensitivity of the model is defined as the percentage of the ‘buggy’ modules correctly predicted and the specificity of the model is defined as the percentage of the ‘clean’ modules correctly predicted;
- Receiver Operating Characteristics (ROC) curve is analyzed to evaluate the performance of the prediction model (Hanley and McNeil, 1982). During the development of the ROC curves, many cutoff points between 0 and 1 are selected and the sensitivity and specificity at each cut off point is calculated;
- Area Under the ROC Curve (AUC is a combined measure of the sensitivity and specificity (Hanley and McNeil, 1982);
- Accuracy is the measure of the correctness of prediction model. It is defined as the ratio of correctly classified instances to the total number of the instances (Hanley and McNeil, 1982).

The above mentioned criteria are widely used in the literature for the evaluation of predictor performance (Dejaeger et al., 2013), (Czibula et al., 2014), (Gondra, 2008), (Malhotra, 2013), (Boucher and Badri, 2018), (Diana et al., 2018), (Son et al., 2019), (Song et al., 2018), (Zhang, 2019).

5. RESULTS AND DISCUSSIONS

In this section, the relationship between the static code metrics i.e. McCabe’s complexity metrics and the fault proneness of software modules is analyzed and presented. The analysis is done using confusion matrix, ROC, AUC and accuracy criteria. The findings of the research work with empirical evaluations are discussed.

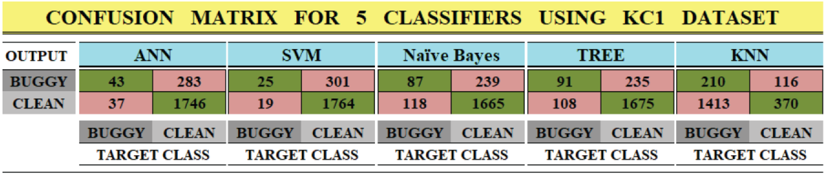
5.1. Confusion Matrix

The models are trained, tested and validated using 10-fold cross validation. The strength of accurate prediction is depicted in confusion matrices for all 30 classifiers grouped by the dataset in Figure 4 through Figure 9. Using confusion matrix, the prediction strength of the classifiers can easily be analyzed by counting the correctly ‘buggy’ classified and correctly ‘clean’ classified data-points. ACCURACY

While comparing the count of the correctly classified as ‘buggy’ and correctly classified as ‘clean’ modules, a rational approach is to be considered.

The cost of missing a ‘buggy’ instance is much higher than the misclassification of a ‘clean’ instance as ‘buggy’. In the consideration of this view of not missing “risky” modules, the percentage

Figure 4. Confusion Matrix for five classifiers over KC1 dataset



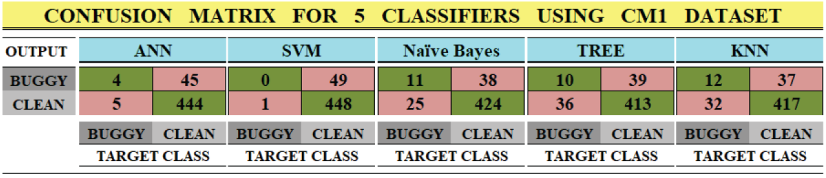
CONFUSION MATRIX FOR 5 CLASSIFIERS USING KC1 DATASET

TRUE

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	43	283	25	301	87	239	91	235	210	116
CLEAN	37	1746	19	1764	118	1665	108	1675	1413	370
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

PREDICTED

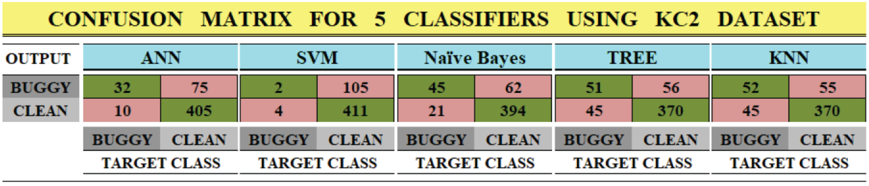
Figure 5. Confusion Matrix for five classifiers over CM1 dataset



CONFUSION MATRIX FOR 5 CLASSIFIERS USING CM1 DATASET

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	4	45	0	49	11	38	10	39	12	37
CLEAN	5	444	1	448	25	424	36	413	32	417
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

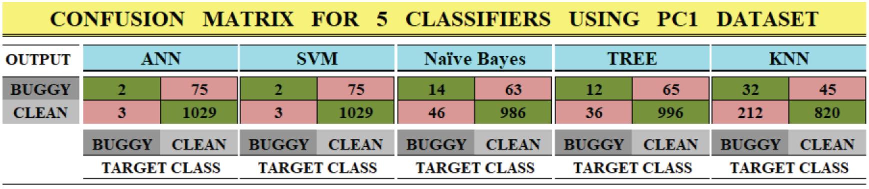
Figure 6. Confusion Matrix for five classifiers over KC2 dataset



CONFUSION MATRIX FOR 5 CLASSIFIERS USING KC2 DATASET

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	32	75	2	105	45	62	51	56	52	55
CLEAN	10	405	4	411	21	394	45	370	45	370
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

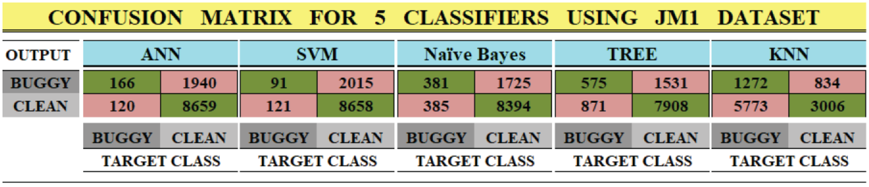
Figure 7. Confusion Matrix for five classifiers over PC1 dataset



CONFUSION MATRIX FOR 5 CLASSIFIERS USING PC1 DATASET

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	2	75	2	75	14	63	12	65	32	45
CLEAN	3	1029	3	1029	46	986	36	996	212	820
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

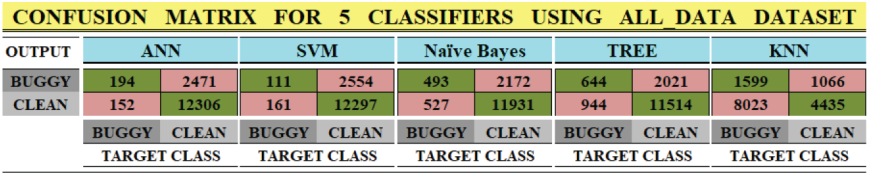
Figure 8. Confusion Matrix for five classifiers over JM1 dataset



CONFUSION MATRIX FOR 5 CLASSIFIERS USING JM1 DATASET

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	166	1940	91	2015	381	1725	575	1531	1272	834
CLEAN	120	8659	121	8658	385	8394	871	7908	5773	3006
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

Figure 9. Confusion Matrix for five classifiers over ALL_DATA dataset



CONFUSION MATRIX FOR 5 CLASSIFIERS USING ALL_DATA DATASET

OUTPUT	ANN		SVM		Naïve Bayes		TREE		KNN	
BUGGY	194	2471	111	2554	493	2172	644	2021	1599	1066
CLEAN	152	12306	161	12297	527	11931	944	11514	8023	4435
	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN	BUGGY	CLEAN
	TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS		TARGET CLASS	

of the ‘buggy’ modules correctly predicted is analyzed, which is shown in Table 5. This percentage is also known as sensitivity of the predictor. The kNN-KC1 model outperforms all other models in terms of sensitivity.

Figure 10 gives the graphical analysis for the sensitivity of all thirty classifiers, which are grouped by ML method (depicted by common color). It is seen that the sensitivity is high for all six classifiers built on k-nearest neighbor method. It is the most noticeable finding of this research work. Figure 11 represents this property of k-nearest neighbor based six classifiers in clearer manner using the box-plots.

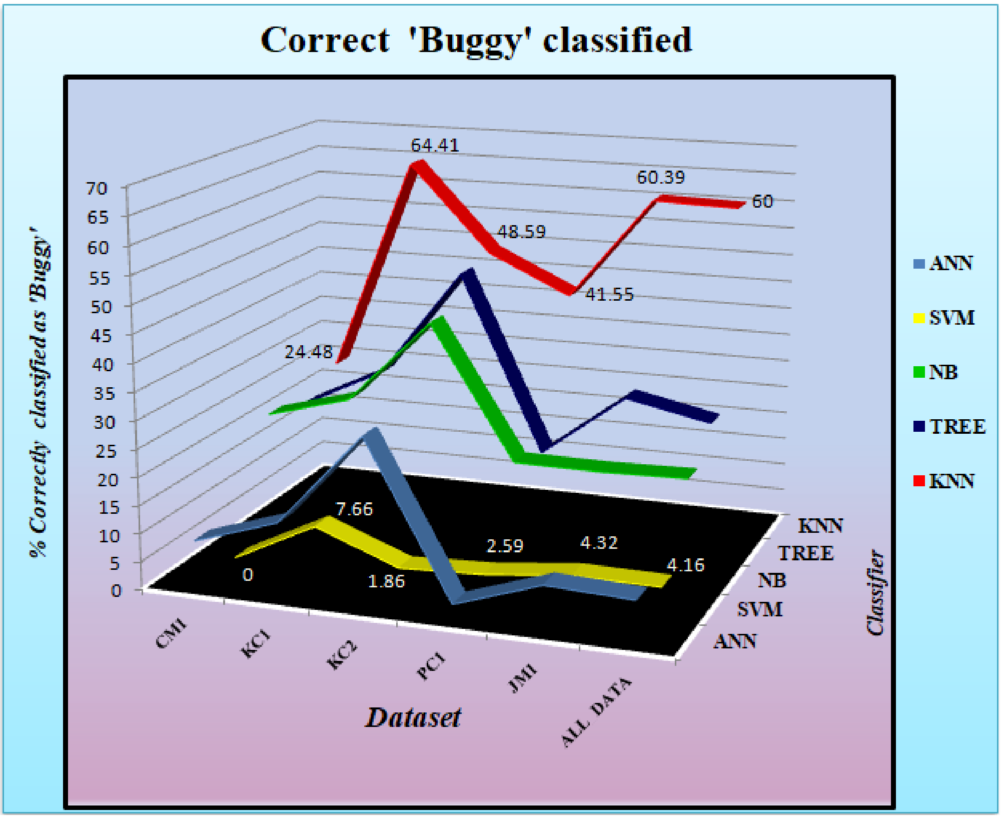
5.2. ROC Curve

The next evaluation criterion is ROC Curve analysis. It is the visual representation of prediction accuracy of classifier on the test data. We analyzed that which ROC curve is closer to the left-hand

Table 5. Correctly classified 'Buggy' instances (percentage)

% Correct 'Buggy'	*ANN*	*SVM*	*NB*	*TREE*	*KNN*
CM1	8.16	0	22.44	20.4	24.48
KC1	13.19	7.66	26.68	27.91	64.41
KC2	29.9	1.86	42.05	47.66	48.59
PC1	2.59	2.59	18.18	15.58	41.55
JM1	7.88	4.32	18.09	27.3	60.39
ALL_DATA	7.27	4.16	18.49	24.16	60

Figure 10. Correctly classified 'Buggy' instances (percentage) for all 30 classifiers grouped by ML method



border and top border as it represents the accuracy in prediction of a binary classifier. Figure 12 through Figure 17 show the ROC curves of all 30 classifiers grouped by the dataset used.

It is found from the ROC curve analysis that the classifiers built using ANN technique are promising to predict 'risky' modules with pretty good accuracy irrespective of the dataset used.

5.3. AUC Measure

The next performance measure used is Area under the ROC Curve. We observed that how close the AUC value is to '1'. Because the AUC value equal to '1', implies 100% prediction accuracy of the

Figure 11. Box-plots for correctly classified ‘Buggy’ instances (percentage) (grouped by ML method)

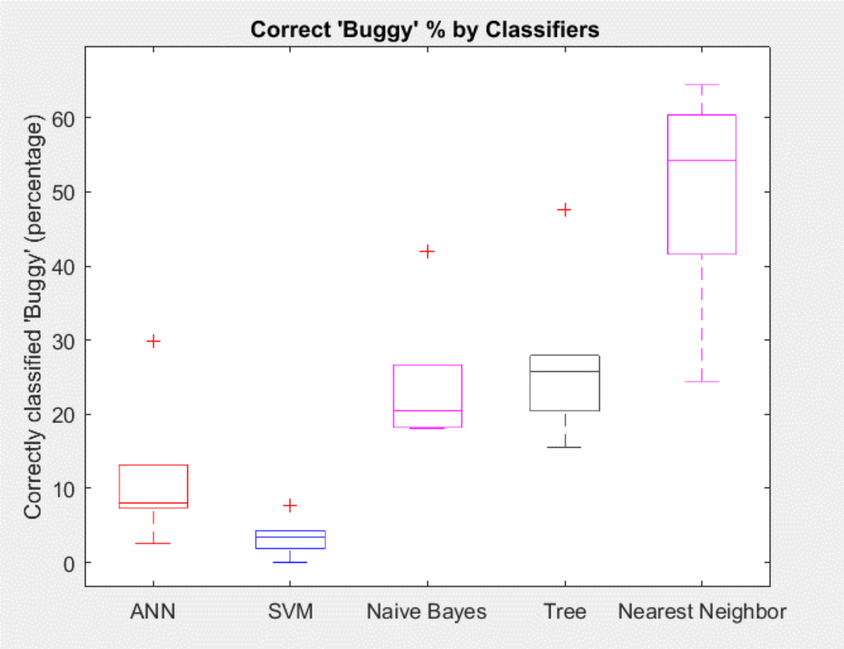


Figure 12. ROC curves for 5 classifiers built using CM1 dataset

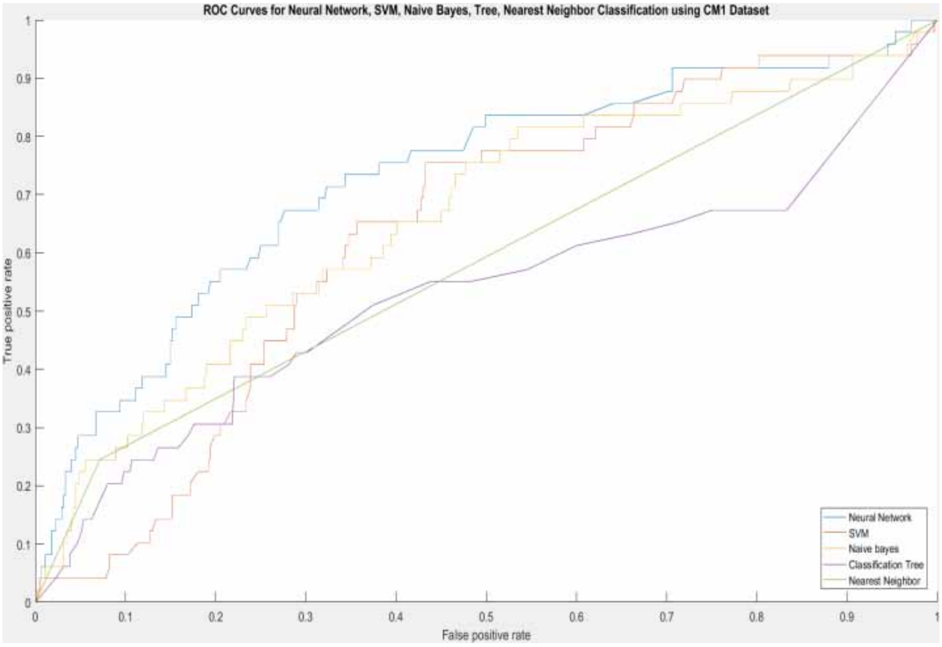


Figure 13. ROC curves for 5 classifiers built using JM1 dataset

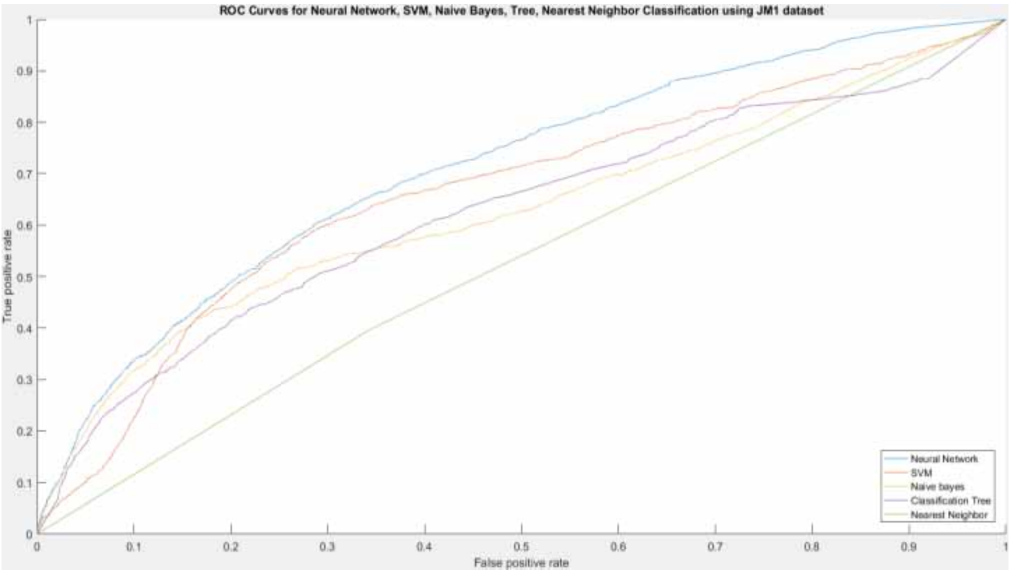
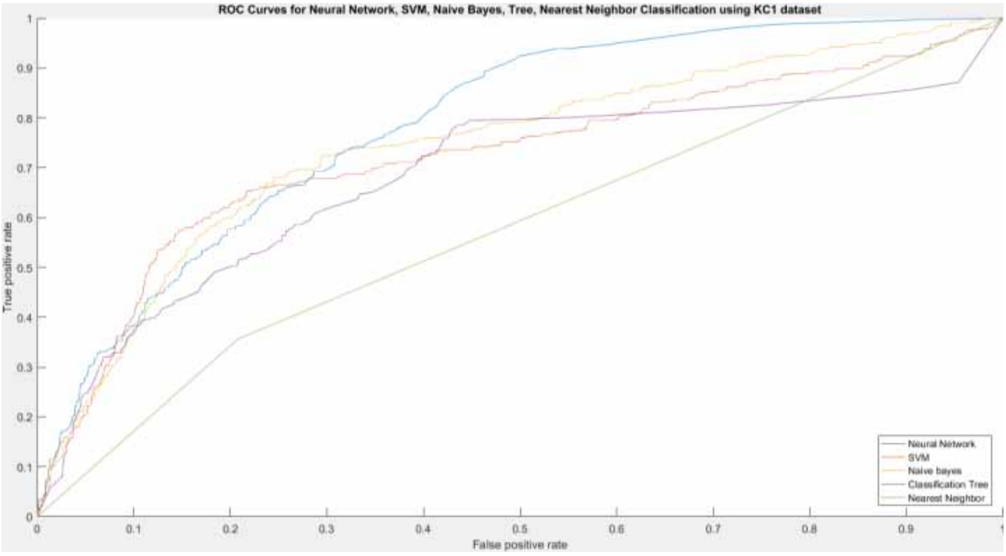


Figure 14. ROC curves for 5 classifiers built using KC1 dataset



classifier. Table 6 shows the AUC values for all thirty classifiers. It is found that the ANN classifier built using KC2 dataset is having maximum AUC with value 0.8315. Figure 18 graphically represents AUC measure for classifiers grouped by dataset and it is clear that ANN-KC2 is the best among all 30 classifiers with 0.8315 AUC measure.

Further, to analyze which ML technique outperforms in terms of AUC measure, box plots for classifiers drawn as in Figure 19. It is observed that ANN technique outperforms other ML techniques in terms of AUC measure.

Figure 15. ROC curves for 5 classifiers built using KC2 dataset

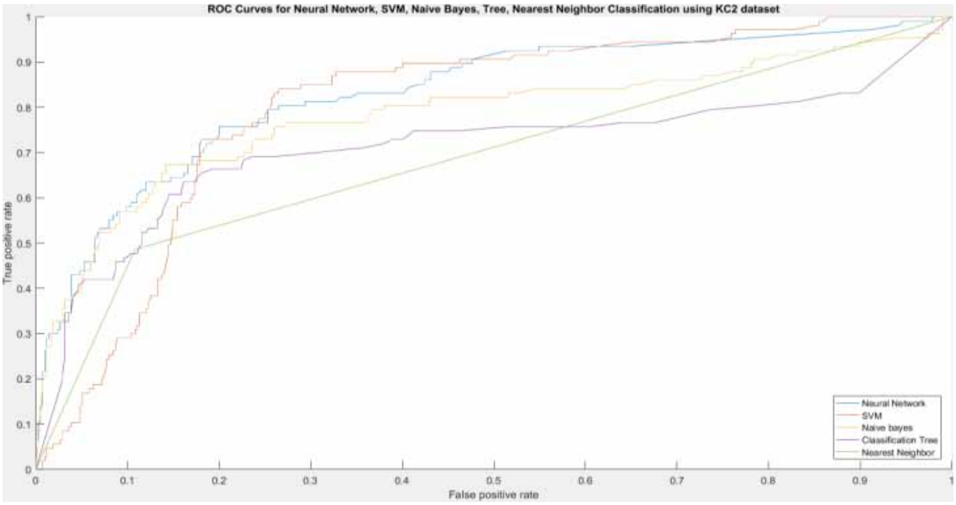
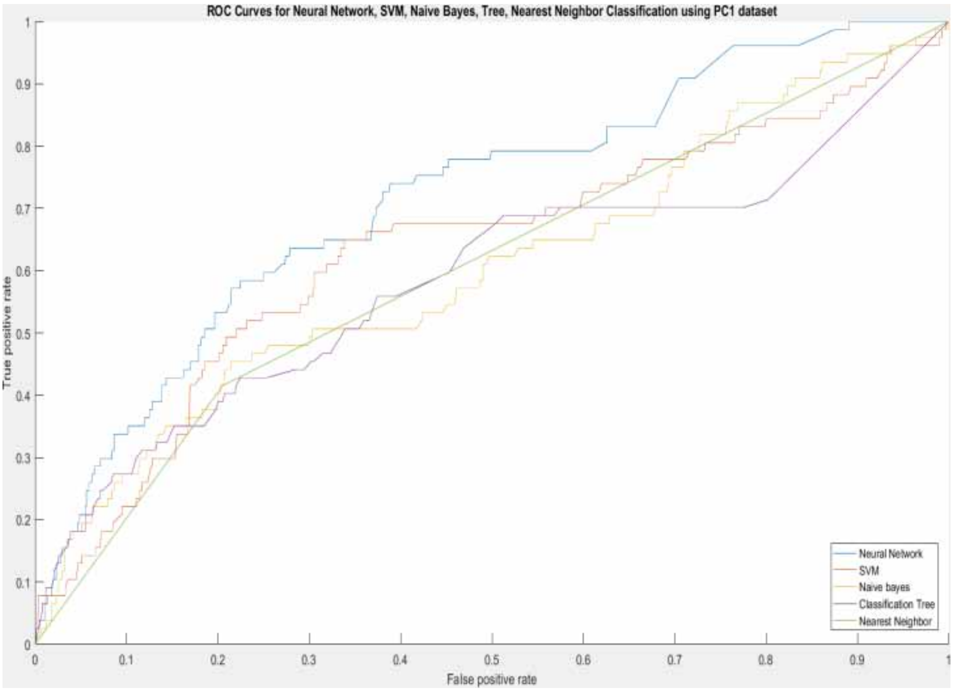


Figure 16. ROC curves for 5 classifiers built using PC1 dataset



5.4. Accuracy Measure

The performance for classifiers is also evaluated on the Accuracy evaluation criteria. The values of Accuracy measure for all 30 classifiers are tabulated as Table 7. The ANN-PC1 and SVM-PC1 show

Figure 17. ROC curves for 5 classifiers built using ALL_DATA dataset

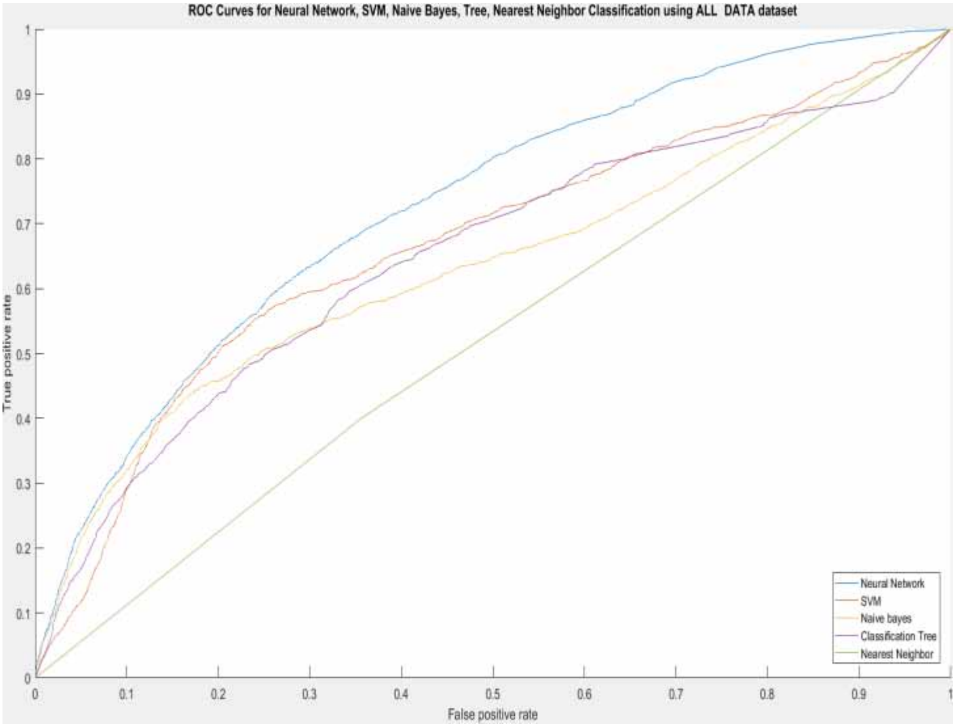


Table 6. AUC measure

AUC	*ANN*	*SVM*	*NB*	*TREE*	*KNN*
CM1	0.7286	0.6341	0.6592	0.5289	0.5868
KC1	0.7878	0.7220	0.7442	0.6828	0.5741
KC2	0.8315	0.8021	0.7816	0.7104	0.6887
PC1	0.7187	0.6348	0.6053	0.5863	0.6050
JM1	0.7102	0.6612	0.6262	0.6227	0.5268
ALL_DATA	0.7282	0.6664	0.6330	0.6479	0.522

similar performance with accuracy measure 0.9296. The graphical representation also analyzed for Accuracy measure as shown in Figure 20.

To analyze which ML technique is superior to others, over the Accuracy measure, Box-plots created for classifiers grouped by ML technique used as in Figure 21. It is observed that ANN based classifiers outperform the others.

6. CONCLUSION AND FUTURE SCOPE

This work is focused on the empirical comparison of Machine Learning techniques for software quality prediction task. In this paper, Software quality prediction task is modeled as a two-class classification

Figure 18. AUC measure for All 30 classifiers (grouped by dataset)

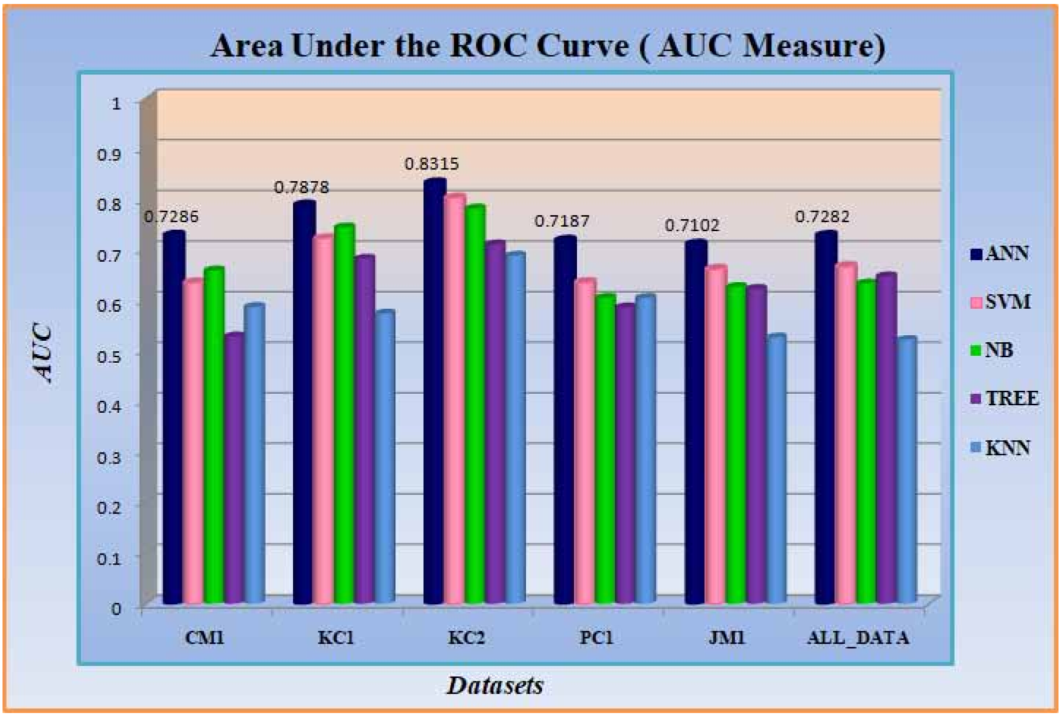


Figure 19. Box-plots for AUC measure (grouped by ML method)

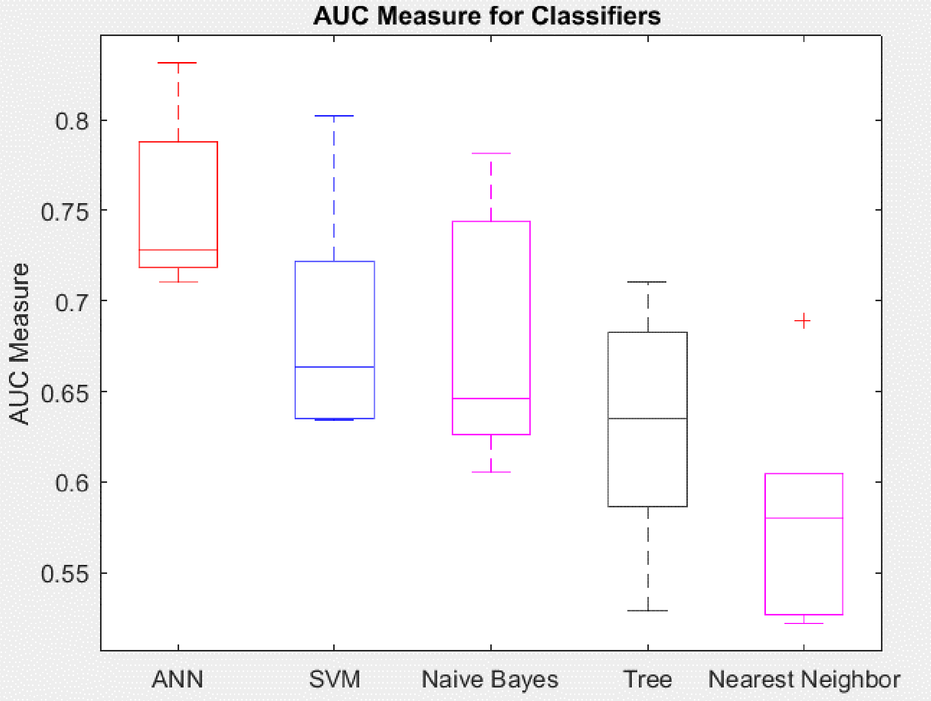
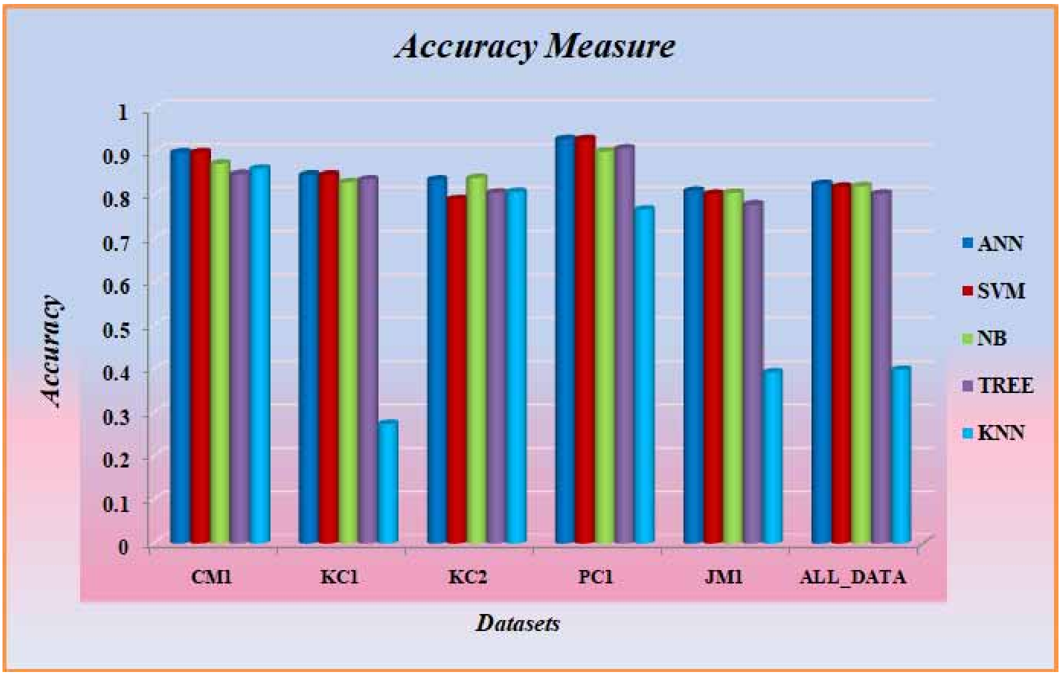


Table 7. Accuracy measure

Accuracy	ANN	SVM	NB	TREE	KNN
CM1	0.8996	0.8996	0.8735	0.8494	0.8614
KC1	0.8482	0.8482	0.8307	0.8373	0.275
KC2	0.8371	0.7911	0.8409	0.8065	0.8084
PC1	0.9296	0.9296	0.9017	0.9089	0.7682
JM1	0.8107	0.8037	0.8061	0.7793	0.3930
ALL_DATA	0.8265	0.8204	0.8215	0.8039	0.3989

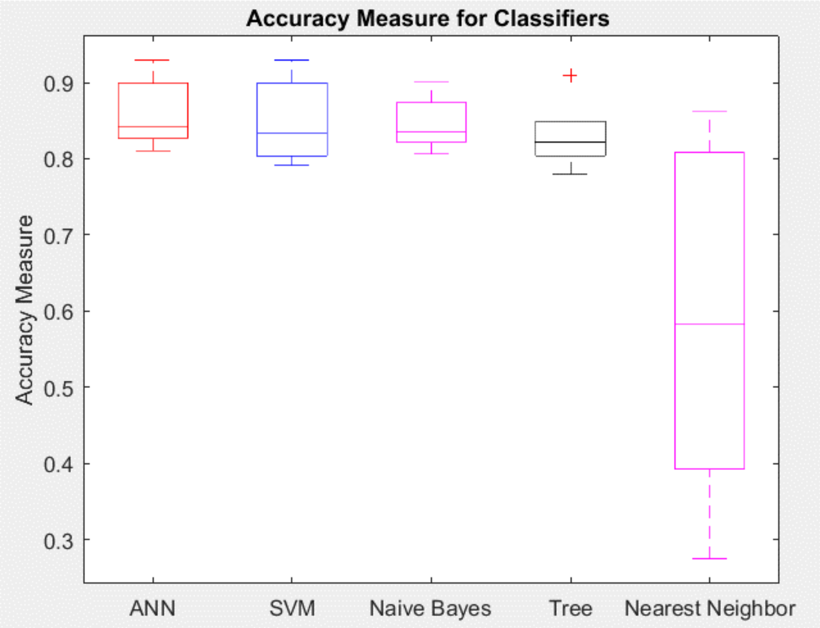
Figure 20. Accuracy measure for all 30 classifiers (grouped by dataset)



problem so that it can be solved using ML techniques. The evaluation of the prediction models which are developed using static code metrics is done using three criterion- ROC, AUC and Accuracy. The experiments conducted in MATLAB and the findings are:

- Using six datasets from PROMISE repository containing the fault prediction data, thirty quality predictors developed, trained and tested with five machine learning techniques. The observation is made that Static code metrics are powerful enough to predict the software quality;
- The performance of developed classifiers is evaluated and comparison is made with the help of necessary measures and plots. It is found that the models built using ANN method are promising for the software quality prediction with pretty good accuracy;

Figure 21. Box-plots for accuracy measure (grouped by ML method)

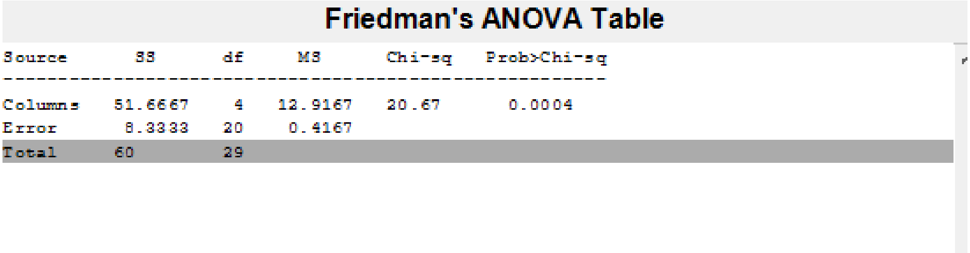


- It is observed that ANN method outperforms other ML techniques used (SVM, Naïve-Bayes, Decision tree and k-Nearest Neighbor) with the value of 0.8315 and 0.9296 as AUC measure and Accuracy measure respectively.

Further, the results are statistically validated using Friedman's test at the confidence level of 95% (Lehmann and Romano, 2008). We assume the null hypothesis H0: "The performance reported by ANN and other models are same". And, the alternate hypothesis H1: "The performance of these proposed models is significantly different". The value of test statistic comes to be very less than 0.05 shown in Figure 22. Hence, we can say that the results are statistically significant. The hypothesis H0 cannot be accepted and hence, the result that the ANN outperforms the prediction models is statistically significant.

The ANN based model is capable to find faulty (or risky) modules about 92.96% which implies that 92.96% faults will be detected and removed in the early phases. Further, the testing efforts can be focused on these highlighted faulty modules. Ultimately, the product delivered will be the better

Figure 22. Statictical test results for validation of experiment


Friedman's ANOVA Table

```
Source       SS       df      MS       Chi-sq    Prob>Chi-sq
-----------------------------------------------------------
Columns    51.6667     4    12.9167    20.67        0.0004
Error       8.3333    20     0.4167
Total      60         29
```

one with enhanced quality and error-free. So, we do conclude that ANN based prediction models ultimately improves the quality of our end-product to a great extent.

In future, this work can be enhanced with the transformation of software quality prediction task as multi-class classification problem. The work can be replicated with larger dataset. Deep architectures can also be incorporated for quality prediction of the software modules.

REFERENCES

Abaei, G., Selamat, A., & Fujita, H. (2015). An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. *Knowledge-Based Systems*, *74*, 28–39. doi:10.1016/j.knosys.2014.10.017

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Applications of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks*, *8*(4), 902–909. doi:10.1109/72.595888 PMID:18255693

Boucher, A., & Badri, M. (2018, April). Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology*, *96*, 38–67. doi:10.1016/j.infsof.2017.11.005

Briand, L., Basili, V., & Hetmanski, C. (1993, November). Developing interpretable models with optimized set reduction for identifying high-risk software components. IEEE Transactions on Software Engineering, *19*(11), 1028–1043. doi:10.1109/32.256851

Chen, N., Hoi, S. C. H., & Xiao, X. (2013). Software process evaluation: A machine learning framework with application to defect management process. *Empirical Software Engineering*.

Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*, *264*, 260–278. doi:10.1016/j.ins.2013.12.031

Mishra, B., & Shukla, K. (2012). Defect prediction for object oriented software using support vector based fuzzy classification model. *International Journal of Computers and Applications*, *60*.

Dejaeger, K., Verbraken, T., & Baesens, B. (2013). Toward comprehensible software fault prediction models using Bayesian network classifiers. *IEEE Transactions on Software Engineering*, *39*(2), 237–257. doi:10.1109/TSE.2012.20

Diri, B., Catal, C., & Sevim, U. (2011). Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. *Expert Systems with Applications*, *38*(3), 2347–2353. doi:10.1016/j.eswa.2010.08.022

Erturk, E., & Sezer, E. A. (2015). A comparison of some soft computing methods for software fault prediction. *Expert Systems with Applications*, *42*(4), 1872–1879. doi:10.1016/j.eswa.2014.10.025

Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software*, *81*(5), 186–195. doi:10.1016/j.jss.2007.05.035

Goyal, S., & Parashar, A. (2018). Machine Learning Application to Improve COCOMO Model using Neural Networks. *International Journal of Information Technology and Computer Science*, *10*(3), 35–51. doi:10.5815/ijitcs.2018.03.05

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, *31*(10), 897–910. doi:10.1109/TSE.2005.112

Hanley, J., & McNeil, B. J. (1982). The meaning and use of the area under a Receiver Operating Characteristic ROC curve. *Radiology*, *143*(1), 29–36. doi:10.1148/radiology.143.1.7063747 PMID:7063747

Huda, S., Alyahya, S., Mohsin Ali, M., Ahmad, S., Abawajy, J., Al-Dossari, H., & Yearwood, J. (2017). A Framework for Software Defect Prediction and Metric Selection. *IEEE Access*, *6*, 2844–2858. doi:10.1109/ACCESS.2017.2785445

Jayanthi, R., & Florence, L. (2019). Software defect prediction techniques using metrics based on neural network classifier. *Cluster Computing*, *22*(1), 77–88.

Kan, S. H. (2002). *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc.

Kumar, L., Sripada, S. K., Sureka, A., & Rath, S. K. (2018). Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM). *Journal of Systems and Software*, *137*, 686–712. doi:10.1016/j.jss.2017.04.016

Lanubile, F., & Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: Lessons learned. *Journal of Systems and Software*, *38*(3), 225–234. doi:10.1016/S0164-1212(96)00153-7

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

Lehmann, E. L., & Romano, J. P. (2008). *Testing Statistical Hypothesis:Springer Texts in Statistics*. New York: Springer.

Li, J., He, P., Zhu, J., & Michael, R. (2017). Lyu, Software Defect Prediction via Convolutional Neural Network. In *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security* (pp. 318-328). IEEE Press. doi:10.1109/QRS.2017.42

Malhotra, R. (2013). Prediction of High-, Medium-, and Low-Severity Faults Using Software Metrics. *Software Quality Professional, 15*(4).

Menzies, T., DiStefano, J., Orrego, A., & Chapman, R. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, *32*(11), 1–12.

Miholca, D.-L., Czibula, G., & Czibula, I. G. (2018, February). A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. *Journal of Information Science*, *441*, 152–170. doi:10.1016/j.ins.2018.02.027

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Moussa, R., & Azar, D. (2017, June). A PSO-GA Approach Targeting Fault-Prone Software Modules. *Journal of Systems and Software*, *132*, 41–49. doi:10.1016/j.jss.2017.06.059

NASA, Government of the United States. (2015). Space Math VI. Retrieved from https://www.nasa.gov/sites/default/files/files/Space_Math_VI_2015.pdf

Nwe, M. M., & Lynn, K. T. (2020). KNN-Based Overlapping Samples Filter Approach for Classification of Imbalanced Data. In Studies in Computational Intelligence (pp. 55–73). Springer Verlag. doi:10.1007/978-3-030-24344-9_4

Rodríguez, D., Ruiz, R., Riquelme, J. C., & Harrison, R. (2013). A study of subgroup discovery approaches for defect prediction. *Information and Software Technology*, *55*(10), 1810–1822. doi:10.1016/j.infsof.2013.05.002

Sayyad, S. & Menzies, T. (2005). The PROMISE Repository of Software Engineering Databases. University of Ottawa. Retrieved from http://promise.site.uottawa.ca/SERepository.

Selby, R., & Porter, A. (1988). Learning from examples: Generation and evaluation of decision trees for software resource analysis. IEEE Transactions on Software Engineering, *14*(12), 1743–1757. doi:10.1109/32.9061

Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019, February 1). Empirical study of software defect prediction: A systematic mapping. *Symmetry*, *11*(2). doi:10.3390/sym11020212

Song, Q., Guo, Y., & Shepperd, M. (2018). A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE Transactions on Software Engineering*. doi:10.1109/TSE.2018.2836442

Song, Q., Shepperd, M., Cartwright, M., & Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, *32*(2), 69–82. doi:10.1109/TSE.2006.1599417

Thomas, J. (1976). McCabe, a complexity measure. *IEEE Transactions on Software Engineering*, *2*(4), 308–320.

Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology*, *107*, 125-136. doi:10.1016/j.infsof.2018.11.005

Zhang, Y., Lo, D., Xia, X., & Sun, J. (2018, April). Combined classifier for cross-project defect prediction: An extended empirical study. *Frontiers of Computer Science*, *12*(2), 280–296. doi:10.1007/s11704-017-6015-y

Zhou, Y., & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering*, *32*(10), 771–789. doi:10.1109/TSE.2006.102