SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM
TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
Volume 8 No 5
ISSN 2054-7390

Evaluation of Machine Learning Classification Techniques in Predicting Software Defects

Olorunshola, Oluwaseyi Ezekiel[1] , Martins E. Irhebhude[2], Abraham E. Evwiekpaefe[3] and Ogwueleka, Francisca Nonyelum[4]

[1,2,3,4]*Computer Science Department, Nigerian Defence Academy, Kaduna-Nigeria.*

seyisola25@yahoo.com; mirhebhude@nda.edu.ng; contact_abraham@yahoo.com; ogwuelekafn@gmail.com

ABSTRACT

The advances in technology has brought about a significant rise in the number of software being developed and deployed on daily basis. This has brought about more dependencies on software and any defect in the software can lead to calamitous issues due to the type of data stored on the software or the type and importance of the functions the software perform. It is necessary to make sure that the all the defects are properly identified before deployment for use. The purpose of this study is to evaluate classifiers on software defects dataset and recommend appropriate classifier for defective software prediction. This will save the developer the stress and time of searching for the defects all through the program code, which will in turn lead to the software to be free from defects that can cause problems in the future of the use of the software. In this research, six categories of machine learning algorithms (two from each category) were tested in Waikato Environment for Knowledge Analysis (WEKA) which are; Bayes (Naives Bayes and Bayes Net), Functions (Multilayer Perceptron (MLP) and Sequential Minimal Optimization (SMO)), Lazy (IBK and KStar), Meta (Random Committee and Bagging), Rules (Decision Table and JRip) and Trees (J48 and Random Forest). The PROMISE dataset was used and the performance metric recorded were; accuracy, false positive rate, precision, recall, f-measure, Receiver Operating Curve (ROC), Kappa Statistics and Root Mean Square Error (RMSE). It was observed that Random Forest performs better under the 10 folds cross validation than the algorithms tested having an Accuracy of 0.818, a Recall of 0.818, a F-measure of 0.787, a ROC of 0.755 and a RMSE of 0.3669.

Keywords: software defects, software, defects, machine learning algorithms, classification algorithms

1 Introduction

Software is a complex entity composed in various modules with diverse range of defect occurrence possibility [1]. Software systems are becoming more and more complex and scalable, therefore, there is need to constantly and continuously detect and fix the design defects [2]. Defect in a software module occurred due to incorrect programming logic or incorrect code which further produces wrong output and leads to a poor-quality software product [1]. If a software defect can be predicted, the quality of that software is improved. The software metric is measured during the software development life cycle (SDLC) phases. SDLC is the total process involved from the inception of a system to the deployment of the system and software metric is the measure of some property of a piece of software during the SDLC [3]. A

DOI: 10.14738/tmlai.85.8733
Publication Date: 12th Aug, 2020
URL: http://dx.doi.org/10.14738/tmlai.85.8733

software defect is a condition in a software product that does not meet with the requirements of the software or the end user. In other words, a defect is an error in coding or logic that causes a program to malfunction or produce incorrect and unexpected results [2].

Software defect prediction is the process of using classifiers to predict and discover potentially defective parts or modules in the software. This will warn code reviewers and allocate efforts. If this is done efficiently, the final product will contain minimal defects which leads to the software having higher quality [4]. It is also worthy of note that the earlier defects are found, the more the development cost reduces. Machine learning techniques can be used to analyse data from different perspectives and enable developers to retrieve useful information, they can be used to predict defects in software [5]. Machine learning approaches have been used to predict software defects. Some include Navies Bayes, random tree, J48 and so on.

Software defects can cause programs to misbehave, leading to negative consequences for the tech industry. Fixed pre-release flaws will theoretically save enterprises from high cost of maintenance and a poor name. The software industry spends billions of pounds on finding and fixing defects every year [6]. The importance attached to the detection of software defects is a major concern for software industry as a result of the type of damage both to the reputation and finances of an organization. Different studies have been carried using different algorithm to predict software defects, therefore this research starts by evaluating these studies. This research evaluates the performances of classification algorithms to determine the best performing algorithm(s) in the detection of software defects, with concentration on algorithms that researchers have not paid much attention to and other algorithms for comparison. Defect prediction can precisely estimate the most defect-prone software components, and help software engineers allocate limited resources to those bits of the systems that are most likely to contain defects in testing and maintenance phases [7].

Accurate defect prediction is enormously important, because of the huge economic impact of faulty software. According to Jim Johnson, chairman of the Standish Group, “Faulty software costs businesses $78 billion per year”. Also, from the $42 billion that the USA Department of Defense spends for the development and maintenance of their computer systems, only 17% is spent buying hardware. It is generally believed that repairing failures after a software system is deployed is 100 times as expensive as repairing those faults during development [8].

The aim of this research is to evaluate different machine learning classification algorithms used in predicting software defects and to determine the best performing algorithm. To achieve this aim, the objectives include to;

I. Get a software defect dataset
II. Train and test software defects dataset using various algorithms
III. Record performance metrics of the classification

Defects in the information technology field are numerous ranging from software to hardware, however, this study is limited only to prediction of defects in software. These defects can be predicted using many techniques but this work is limited to twelve classification algorithms. There are also many datasets that

can be used, but this work is limited to PROMISE JM1 dataset. Many performance metrics can also be used, but only eight were used.

The numerous lines of codes, the bulkiness of a program and the numerous functionality call for a need to better predict defects before they are deployed and also ensure that the non-defect part of the software is not predicted as defect which will in turn waste the time of the software developer. Machine learning classification algorithms can be used to predict these defects accurately with low false negative rate.

This study is relevant in deploying a system or software that has no defect based on the proper prediction of the defect. It will be of utmost importance to searching of defects in software which will save time, efforts and achieve a deplorable software free of defects

The remaining sections of this work is arranged thus; following this section is the literature review; where review of past works is considered. Thereafter, the methodology explains the steps taken to achieve the final conclusion. Section four gives an elaborate explanation of the result gotten after the methodology was applied. Section five concluded and made recommendations based on findings from the research.

2 Literature Review

In this section, software defects prediction is examined and some related works are discussed

2.1 Software defects prediction

Software Quality demand have been on the high rise since software became part of human life and plays a significant role in the overall performance of the humans [9], with this, there is need to be able to properly predict the defects in software. Software defects prediction is the process of predicting code areas that can potentially contain defects before deployment can be done. It helps the developers to allocate their testing efforts towards the potentially defect area; this area can be a code, a file design structure and so on [4], [9], [10]. [11] did a systematic literature review of software defect prediction based on the research trends, datasets, methods and frameworks. The literatures reviewed are between year 2000 and 2013. [12] also carried out a systematic literature review investigating the use and performance of unsupervised learning techniques in software defect prediction. The literatures reviewed were published between January 2000 and March 2018.

The objective of a research by [13] was to help developers identify defects based on existing software metrics using data mining techniques and thereby improving the software quality. In this research, various classification techniques are revisited which are employed for software defect prediction using software metrics in the literature.

In [8] authors stated that it is of general opinion that repairing defects after the deployment of the software is 100 times as expensive as repairing the defects during the development therefore, the adverse effect of defects in software include financial loss. Authors in [14] stated that “Erroneous predictions of defect-free modules as defective (false alarms) force testers to inspect ‘safe’ modules and waste their precious time. On the other hand, missing defective modules (false negatives) would cause more expensive and hard-to-fix failures in the final software product.” With these adverse effects, it is therefore important that software defects are accurately predicted before it is deployed for public consumption

Copyright © Society for Science and Education United Kingdom

Authors in (J. Li et al., 2017) presented a method called Defect Prediction through the Convolutional Neural Network (DP-CNN), which leverages deep learning for the successful generation of apps. It includes system segment and structural elements. Specifically, the token vectors were first extracted based on Abstract Syntax Trees (ASTs) of the programs, which are then encoded as numerical vectors by mapping and word embedding. They feed numerical vectors into Convolutional Neural Network (CNN) to automatically learn system segment and structural characteristics. Afterwards, the studied features and the conventional hand-crafted features were mixed for detecting correct device defects. The method was tested on seven open source projects in terms of F-measure in defect prediction and the average value gotten from the seven was 0.608.

In [10] authors proposed a cost-sensitive (CS) classification strategy called CSForest which is a decision trees ensemble. A CS voting strategy known as CSVoting was also proposed to take advantage of the collection of decision trees and reduce the expense of classification. The potential solution was explored to class disparity inside the decision forest algorithm. The empirical test on the proposed strategies was carried out and equated them with six (6) classifier algorithms on six (6) publicly accessible clean datasets widely used in software-defect prediction study. The initial experimental findings show that authors proposed approaches were superior to existing techniques. The finding showed that CSForest had a Recall of 0.375 for MC2 dataset – a score that was better than existing techniques in previous studies.

In [15], authors compared some algorithms (sampling, cost sensitive, ensemble and hybrid approaches) which deal with imbalanced dataset in the domain of defect prediction. They used the National Aeronautics and Space Administration (NASA) datasets and compared the evaluation metrics. Some include the Matthew's Correlation Coefficient (MCC) and ROC curve. It was discovered from their study that there are differences in the values of the evaluation metrics based on the dataset used.

The study in [9] presented the first application of the Adaptive Neuro Fuzzy Inference System (ANFIS) for the software fault prediction problem. Also, Artificial Neural Network (ANN) and Support Vector Machine (SVM) methods, were added to understand the performance of ANFIS. A total of 5 training and test datasets were produced used in the study which was derived from the PROMISE Software Engineering Repository. Similarly, the McCabe metrics were nominated because they broadly addressed the programming efforts. ROC-Area Under Curve (AUC) was used as a performance measure. The results obtained were 0.7795, 0.8685, and 0.8573 for the SVM, ANN and ANFIS methods, respectively.

In [8], the study examined stratification, a widely used technique for learning imbalanced data that has received little attention in software defect prediction. The experiments focused on the Synthetic Minority Over-Sampling Technique SMOTE technique. SMOTE is a method of over-sampling minority-class examples. The main goal was to determine if SMOTE can improve recognition of defect-prone modules, and at what cost. It was discovered that there was at least 23% in the average geometric mean classification accuracy.

Authors in [16] used different parameter values for the classifiers and explored the usefulness of employing dimensionality reduction techniques, such as Principle Component Analysis (PCA), and Ensemble Learning techniques. The results show the effectiveness of the considered classifiers in detecting bugs. Additionally, using PCA did not have a noticeable impact on performance while parameter tuning positively impact classifies' accuracy, especially with ANN. The best results were obtained by using

Ensemble Learning methods such as Bagging (which achieves 95.1% accuracy with Mozilla dataset) and Voting (which achieves 93.79% accuracy with kc1 dataset).

In [17] authors used the ensemble based statistical models – Gradient Boosting and Random Forest for classification. An empirical study was carried out to determine the performance of multinomial classification for cross project defect prediction. The result outcome concludes that multinomial/multiclass classification is applicable on cross project defect prediction (CPDP) data and has comparable results to within project defect prediction (WPDP) data. The average accuracy gotten for CPDP and WPDP were 0.72 and 0.79 using the random forest classification algorithm.

Authors in [18] used Bagging, SVM, decision tree (DT), and random forest (RF) supervised and ensemble classifiers to predict defects on 10 NASA datasets. The results showed that RF was the best performing classifier compared to the others having the highest ROC-AUC of 0.84 with the PC4 dataset.

Authors in [19] used the K- Nearest Neighbor (KNN) to predict faulty software projects. Experimental studies was conducted on five public datasets with different similarity measures. Results showed that KNN can be used to predict software faults with accuracy rate of 87.2%.

In [20] authors used several machine learning classification techniques which are Naïve Bayes (NB), Multilayer Perceptron (MLP). Radial Basis Function (RBF), SVM, KNN, kStar (K*), One Rule (OneR), PART, DT, and RF to predict software defects in twelve widely used NASA datasets (CM1, JM1, KC1, K31, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5). The performance metric used were Precision, Recall, F-Measure, Accuracy, MCC, and ROC Area. The RF performs averagely above others having the highest ROC area of 0.945 with the PC4 dataset.

In [21], authors did a keen study analysis in order to compare the performance of RF, Naïve Bayes, Recursive Partitioning and Regression Trees (RPart) and SVM classifiers when predicting defects in some dataset. It was discovered that each classifier detects different set of defects. It was concluded that classifier ensembles with decision-making strategies not based on majority voting are likely to achieve better results in predicting defects.

Authors in [22] studied how class imbalance learning method can benefit software defect prediction with the aim of finding better solutions. They investigated different types of class imbalance learning methods, including resampling techniques, threshold moving, and ensemble algorithms of which AdaBoost.NC shows the best overall performance metric which include G-mean and AUC. To further improve the performance of the algorithm, and facilitate its use in software defect prediction, they propose a dynamic version of AdaBoost.NC, that can adjust its parameter automatically during training, with the goal of improving or at least maintaining the effectiveness of AdaBoost.NC without the exhaustive search for the best parameter, it is shown to be more effective and efficient than the original AdaBoost.NC. For the mw1 dataset, the mean and standard deviation of G-mean is 0.647 ± 0.272 with the dynamic AdaBoost.NC while for AdaBoost, the value is 0.525 ± 0.306.

Authors in [23] proposed a cost sensitive discriminative dictionary learning (CDDL) approach to classify and predict software defects. The use of dictionary learning technique to predict software defects. They made use of characteristics of the mined metrics from the open source software learning multiple dictionaries and sparse representation coefficients. The misclassification cost issue was also taken into consideration. It was shown from the experimental results that CDDL out performed several

Copyright © Society for Science and Education United Kingdom

representative state-of-the-art defect prediction methods Having a value of 0.79 as the probability of detection on the average while cost-sensitive boosting neural network (CBNN) which performs better than all the other algorithms tested (SVM, Naïve Bayes and so on) is 0.64.

Authors [1] in their paper proposed a feature selection approach: Linear Twin Support Vector Machine (LSTSVM) model to predict defect prone software modules. F-score, a feature selection technique, was used to determine the significant metrics set which are prominently affecting the defect prediction in a software module. The efficiency of predictive model could be enhanced with reduced metrics set obtained after feature selection and further used to identify defective modules in a given set of inputs. The paper evaluated the performance of proposed model and compared it against other existing machine learning models. The experiment was performed on four PROMISE software engineering repository datasets. The experimental results indicated the effectiveness of the proposed feature selection based LSTSVM predictive model on the basis standard performance evaluation parameter. The accuracy of the pc1 dataset was recorded to be 0.9463 with the model proposed while the other algorithms recorded 0.9359 (Multilayer Perceptron (MLP)), 0.9319 (Logistic Regression (LR)), 0.9358 (DT), 0.9310 (SVM), 0.9182 (KNN), 0.9284 (RBF) and 0.9044 (Bayesian Belief Network (BBN)).

3 Methodology

In this section, the environment and dataset is described, algorithm and performance metrics is also examined, this making up the totality of the methodology

3.1 Environment description

This research was carried out using Waikato Environment for Knowledge Analysis (WEKA) version 3.8.3 developed by [24]. Twelve algorithms in WEKA were used in order to evenly distribute the selection of two algorithms from each of the six categories in WEKA that have not been paid attention to. Some of the other algorithms chosen from these twelve were chosen so that there will be complete comparison. The classification was done on an HP laptop with windows 10 installed as the Operating System (OS) with a processor of intel(R) core(TM) i3-312M CPR @ 2.50GHz, installed RAM of 8.00GB and 64 bit OS

3.2 Dataset description

The PROMISE dataset [25] was used in this research. It was created by NASA metrics data program. The specific dataset used for the analysis is the JM1 donated by Tim Menzies. It was written in C and is a real-time predictive system. The metrics extracted from the source code are the McCabe [26] and Halstead [27] features. These metrics were module based smallest unit of functionality. The dataset was chosen because there has not been much concentration on using this dataset for classification with the selected classifiers.

The software metrics is fully discussed in section 3.3. In total, there are 10,885 instances with 22 attributes (5 different LOC, 3 McCabe metrics 4 base Halstead Measures, 8 Derived Halstead measures, a branch-count, and 1 goal field). The information is presented in table 3.1. The dataset has 2,106 (19.35%) instances of the defects dataset to be false and 8779 (80.65%) to be true.

3.3 Software Metric

Software metric is a measure of some property of a piece of software or its specifications [2]. In this work, the dataset used is based on McCabe and Halstead features These features were defined in order to

objectively characterize code features that were related with software quality The Halstead and McCabe measures are based on modules (this is also known as functions/methods) which is the smallest unit of functionality [25]. In [25] authors explained the features extracted as follows;

1. Halstead metric was developed by late Maurice Halstead who claimed that "code that is hard to read is more likely to be fault prone. Halstead estimates reading complexity by counting the number of concepts in a module which is the number of unique operators"[25]. This metric is used to determine a quantitative measure of complexity directly from the operators and operands in the module [2], [27]. This were grouped into three which are

   a. Base Measures:

      i. mu1 = number of unique operators

      ii. mu2 = number of unique operands

      iii. N1 = total occurrences of operators

      iv. N2 = total occurrences of operands

      v. length = N = N1 + N2

      vi. vocabulary = mu = mu1 + mu2

      vii. Constants set for each function:

         - mu1' = 2 = potential operator count (just the function name and the "return" operator)
         - mu2' = potential operand count. (the number of arguments to the module)

   b. Derived Measures:

      i. P = volume = V = N * log2(mu) (the number of mental comparisons needed to write a program of length N)

      ii. V* = volume on minimal implementation = (2 + mu2') * log2(2 + mu2')

      iii. L = program length = V*/N

      iv. D = difficulty = 1/L

      v. L' = 1/D

      vi. I = intelligence = L'*V'

      vii. E = effort to write program = V/L

      viii. T = time to write program = E/18 seconds

   c. Lines of Code (LOC)

Copyright © Society for Science and Education United Kingdom

2. McCabe metric was developed my Thomas McCabe and he stated that "code with complicated pathways are more error-prone." His metrics therefore reflect the pathways within a code module. The metric is a collection of four software metrics which are explained below [9], [25], [26].

    a. Cyclomatic Complexity (v(G)): this measures the number of linearly independent paths. A set of paths is said to be linearly independent if no path in the set is a linear combination of any other paths in the set through a program's flowgraph. Cyclomatic complexity changes according to decision nodes of the module structure. Testability decreases as long as cyclomatic complexity increases.

    b. Essential Complexity (ev(G)): this is the extent to which a flowgraph can be reduced by decomposing all the submodules of 'G' that are "D-structed primes" (this is also known as proper one-entry one-exit sub flowgraphs). Maintainability and modularity decrease as long as essential complexity increases.

    c. Design Complexity (iv(G)): this is cyclomatic complexity of a module's reduced flowgraph according to decisions and loops that do not impact the interrelationship between submodules. This complexity measurement reflects the modules calling patterns to its immediate subordinate modules.

    d. Line of code

Table 3.1: The attribute information [25]

S/N	Attributes	
1.	Loc	numeric % McCabe's line count of code
2.	v(g)	numeric % McCabe "cyclomatic complexity"
3.	ev(g)	numeric % McCabe "essential complexity"
4.	iv(g)	numeric % McCabe "design complexity"
5.	N	numeric % Halstead total operators + operands
6.	V	numeric % Halstead "volume"
7.	L	numeric % Halstead "program length"
8.	D	numeric % Halstead "difficulty"
9.	I	numeric % Halstead "intelligence"
10.	E	numeric % Halstead "effort"
11.	B	numeric % Halstead
12.	T	numeric % Halstead's time estimator
13.	lOCode	numeric % Halstead's line count
14.	lOComment	numeric % Halstead's count of lines of comments
15.	lOBlank	numeric % Halstead's count of blank lines
16.	lOCodeAndComment	numeric
17.	uniq_Op	numeric % unique operators
18.	uniq_Opnd	numeric % unique operands
19.	total_Op	numeric % total operators
20.	total_Opnd	numeric % total operands
21.	branchCount	numeric % of the flow graph
22.	Defects	{false,true} % module has/has not one or more reported defects

3.4 Classification Algorithms Application

A total of 12 algorithms was selected in the WEKA environment, 2 from each category which are Bayes (Naive Bayes and Bayes Net), Functions (Multilayer Perceptron (MLP) and Sequential Minimal Optimization (SMO)), Lazy (IBK and KStar), Meta (Random Committee and Bagging), Rules (Decision Table and JRip) and Trees (J48 and Random Forest). The selected algorithms were chosen because, from the literatures reviewed, little or no attention was paid to Random committee, Bagging, MLP, IBK, KStar, JRip. Researchers have paid attention to Decision Table. Naïve Bayes, Bayes Net, SMO, J48 and Random Forest [1], [9], [18], [20], [21], [23], these were also chosen to provide a means of comparison with the algorithm that have little or no attention. The 10 folds cross validation (which specify the number of parts/folds used to be 10) was primarily used in this research. It randomly segregates the dataset into 10 separate part in which each fold is revealed in nearly equivalent size as the full dataset. Each fold is used as the testing dataset while the remaining 9 folds are used for training; the error rate is then processed on the dataset used for testing. This process is repeated 10 times for each of the part acting as the testing dataset. After this is done for all the folds, the final error estimation is done which is the mean of all the 10 error estimates [28].

3.5 Performance Evaluation Metrics

The following metrics were used in the evaluation of the algorithms chosen.

1. Accuracy. This summarizes the performance of a classification model as the number of predictions divided by the total number of predictions. It measures the rate of correctly classified applications putting into consideration the true positive, true negative, false positive and false negative [29].

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

Where TP = True positive

TN = True Negative

FN = False Negative

FP = False Positive

2. False Positive rate (FPR). This measures the rate of wrongly classified as benign. A low FP-rate signifies the classifier is a good one [30].

$$FPR = \frac{FP}{FP + TN}$$

3. Precision, Recall and F-measure. Precision is the ratio of positively predicted instances among the retrieved instances. Recall is the ratio of positively predicted instances among all the instances and F-measure is the harmonic mean of recall and precision [31], [32].

Copyright © Society for Science and Education United Kingdom

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FP}$$

$$\text{F-Measure} = 2\,x\,\frac{\text{Precision}}{Precision + Recall}$$

4. Receiver Operating Characteristic (ROC) curve. The true positive rate is constructed against the false positive rate [30].

   ROC Curve is Plot of $\text{FPR}(x)$ vs TPR **where** TPR is True Positive Rate

5. Kappa statistic is used to measure inter of how closely the instances labelled by the classifier of machine learning followed the data classified as ground truth, influencing the accuracy of a random classifier as assessed by the predicted accuracy [33].

   Kappa = (observed accuracy - expected accuracy) / (1 - expected accuracy)

Root mean square error (RMSE). This is the standard deviation of the predicted error. Predicted error is the error between the training and testing dataset. A low RMSE indicates that the classifier is an excellent one [34], [35]

RMSE = $\sqrt{1 - r^2}\;x\;SD$ **Where** SD = Standard Deviation

r = Predicted error

4 Result and Discussion

In this section, the result gotten by using the methodology (section 3.0) is discussed.

4.1 Analysis of Result

The PROMISE dataset [25] used in this research consist of defect instances which can either be false or true. After the analysis was done in the WEKA environment using the 10 folds cross validation, the results are discussed in this section. Table 4.1 shows the classifier grouping and the result of the analysis which are accuracy, false positive rate, precision, recall, f-measure, ROC curve, kappa statistics and RMSE. Figures 4.1 to 4.8 show the data visualization of the analysis. Each of these performance metrics is discussed below in the following paragraphs

From Table 4.1 and Figure 4.1, in terms of accuracy, it can be seen that, RF performed best having an accuracy of 81.8%, however it is also clear that the false positive rate is high (62.1%) from figure 4.2 which means that the classifier performed poorly.

In figure 4.2 and table 4.1, it can be seen that Bayes Net have a low FP-rate of 43.1%, this therefore draws attention towards itself despite having a little above average accuracy of 69.2% as seen in figure 4.1.

Taking a close look at the precision in figure 4.3 and table 4.1, it is seen that SMO performs best with 84.4%, but very poorly in FP-rate with 80.3% (as seen in figure 4.2) and even though it has a high accuracy of with 80.7% (as seen in figure 4.1). This is makes it a bad for a classifier for this dataset.

In figure 4.4, for the recall metric, it was observed that random forest has the highest metric (81.8%). It is also observed in figure 4.5 that RF have the highest metric under the F-measure with a metric of 78.7%. In figure 4.6, it is also observed that random forest performed the best for ROC Curve with 75.5%.

A high Kappa statistic signifies that the classifier is a good one. This is seen in figure 4.7 in the performance of random committee classifier having the highest value of 0.2557. The lowest kappa

statistics is the SMO with 0.0061, this further proves SMO unsuitability for this dataset even though it has a high precision.

In figure 4.8, It must be noted that Bayes net performs poorly in the RMSE with 0.5323 proving it is not too suitable for this dataset although it was able to maintain a low FP-rate. Random forest performs best in the RMSE with the lowest metric of 0.3669. This further buttress that this algorithm is averagely suitable for this dataset because it is affected by the FP-rate.

Table 4.1: The Result of the Performance Metrics of the Algorithms

CLASSIFIERS		Accuracy	FP-Rate	Precision	Recall	F-Measure	ROC Area	Kappa Statistics	RMSE
		10 Folds-CV	10 Folds-CV	10 Folds-CV	10 Folds-CV	10 Folds-CV	10 Folds-CV	10 Folds-CV	10 Folds-CV
Bayes	BayesNet	0.681	**0.391**	0.769	0.681	0.71	0.701	0.2207	**0.5323**
	NaiveBayes	0.804	0.654	0.765	0.804	0.77	0.679	0.1932	0.4401
Functions	MLP	0.81	0.762	0.769	0.81	0.741	0.69	0.0716	0.3791
	SMO	0.807	0.803	**0.844**	0.807	0.722	0.502	**0.0061**	0.439
Lazy	IBK	0.77	0.539	0.762	0.77	0.766	0.64	0.2381	0.4794
	kStar	0.786	0.589	0.761	0.786	0.77	0.638	0.2228	0.4258
Meta	RandomCommittee	0.81	0.602	0.78	0.81	0.785	0.724	**0.2557**	0.3846
	Bagging	0.812	0.66	0.775	0.812	0.774	0.741	0.2016	0.3697
Rules	DecisionTable	0.809	0.708	0.766	0.809	0.759	0.703	0.1415	0.3767
	Jrip	0.81	0.68	0.771	0.81	0.768	0.57	0.1765	0.3859
Trees	J48	0.795	0.633	0.759	0.795	0.769	0.653	0.1981	0.4079
	RandomForest	**0.818**	0.621	0.787	**0.818**	**0.787**	**0.755**	0.2521	**0.3669**

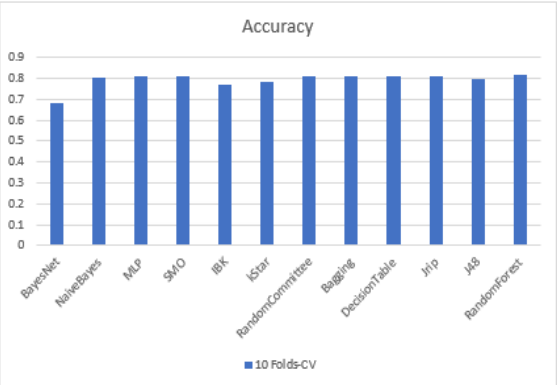


Figure 1: Comparison of the Accuracy

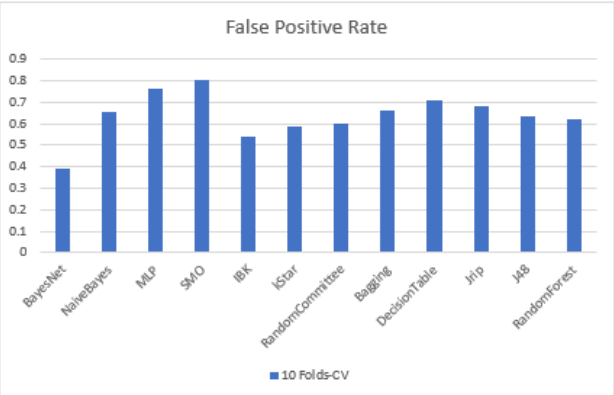


Figure 2: Comparison of the False Positive Rate

Copyright © Society for Science and Education United Kingdom

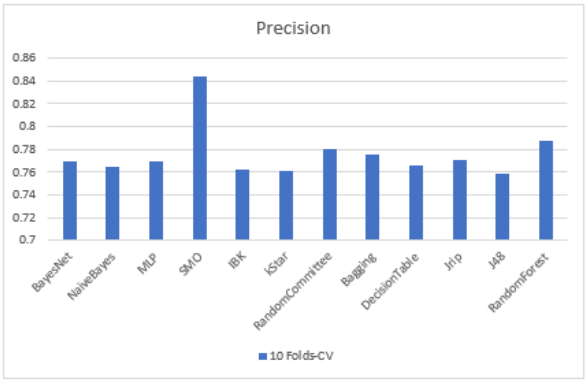


Figure 3: Comparison of the Precision

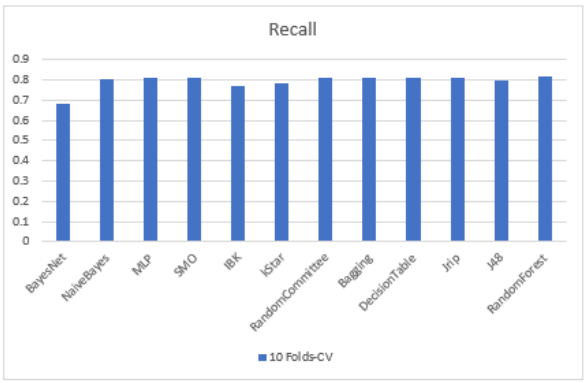


Figure 4: Comparison of the Recall

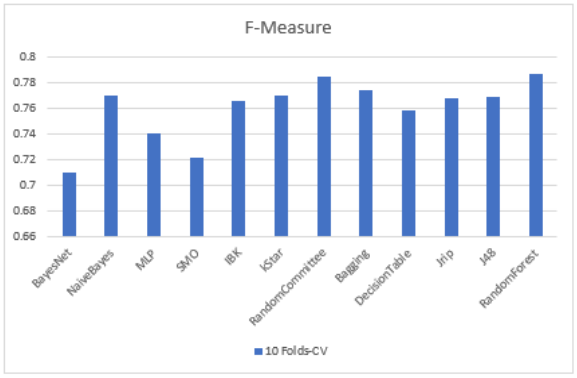


Figure 4.5: Comparison of the F-Measure

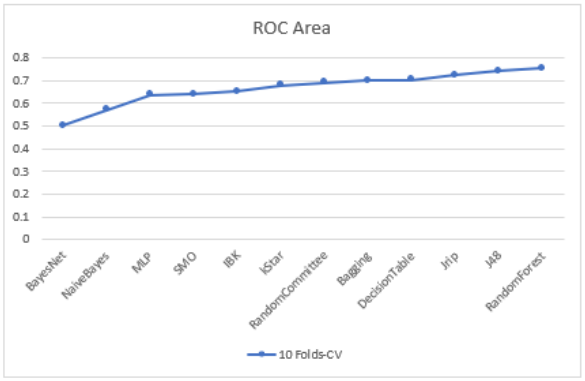


Figure 4.6: Comparison of the ROC Area

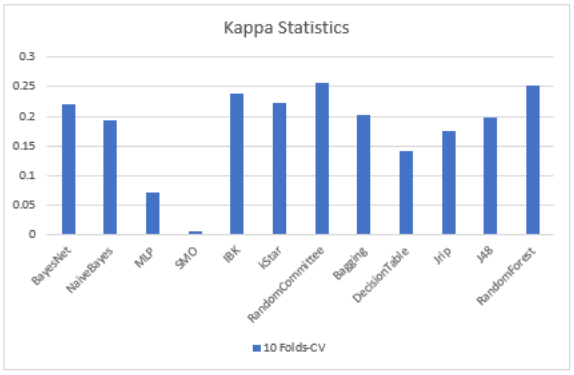


Figure 4.7: Comparison of the Kappa Statistics

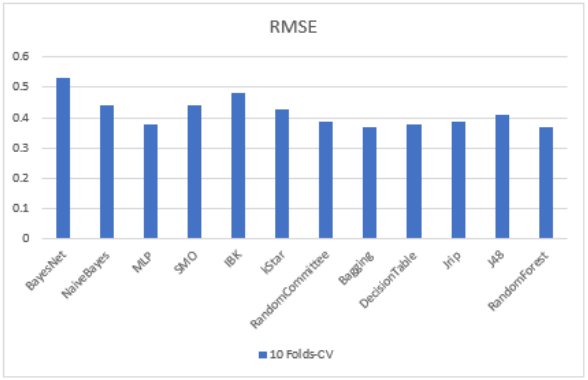


Figure 4.8: Comparison of the RMSE

4.2 Comparison with selected literatures

The result of the best performing algorithm (RF) was compared with the results from some selected literatures. Table 4.2 presents the comparison with selected state-of-art algorithms.

Table 4.2 Comparison of RF Performance Metric with Other Literatures

References	Techniques/Metric	Accuracy	Recall	F-Measure
This Research	**Random Forest**	**0.818**	**0.818**	**0.787**
[4]	CNN	-	-	0.608
[10]	CSForest	-	0.375	-
[17]	RF (CPDP and WPDP)	0.72 and 0.79	-	-

5 Conclusion

This study determined the best classification algorithms on software defects dataset. Although the result of the analysis is not distinctly varied between the classifier, it can be seen that random forest performs better than most of the algorithms from the performance metrics used. Accuracy (10 folds cross validation (0.818)), Recall (10 folds cross validation (0.818)), F-Measure (10 folds cross validation (0.787)), ROC Area (10 folds cross validation (0.755)) and RMSE (10 folds cross validation (0.3669)) were values obtained from the analysis. The Bayes Net was also a good classifier for this dataset having a relatively good performance in the metrics especially the FP-rate, having the lowest FP-rate of 0.391 (10 folds cross validation).

Further studies should consider using the random forest algorithm along with the Bayes Net on other software defects prediction dataset. This is to establish that the algorithm is not dataset-biased. Thus, further proving the performance of this algorithms or disproving it. There is also need to test more algorithms to ensure that other classifiers are put into consideration. Future works can also consider gathering dataset and extracting necessary features from the dataset instead of using the publicly available ones. This dataset should be tested and the result should be compared with the result of this paper.

REFERENCES

[1] D. Tomar and S. Agarwal, “Feature selection based least square twin support vector machine for diagnosis of heart disease,” *Int. J. Bio-Science Bio-Technology*, vol. 6, no. 2, pp. 69–82, 2014, doi: 10.14257/ijbsbt.2014.6.2.07.

[2] P. Paramshetti and D. A. Phalke, “Survey on Software Defect Prediction Using Machine Learning Techniques,” *Int. J. Sci. Res.*, vol. 3, no. 12, pp. 2012–2015, 2014.

[3] A. Mishra and D. Dubey, “A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios,” *Int. J. Adv. Res. Comput. Sci. Manag. Stud.*, vol. 1, no. 5, pp. 64–69, 2013.

[4] J. Li, P. He, J. Zhu, and M. R. Lyu, “Software defect prediction via convolutional neural network,” *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2017*, pp. 318–328, 2017, doi: 10.1109/QRS.2017.42.

[5] S. Aleem, L. F. Capretz, and F. Ahmed, “Benchmarking Machine Learning Techniques for Software Defect Detection,” *Int. J. Softw. Eng. Appl.*, vol. 6, no. 3, pp. 11–23, 2015, doi: 10.5121/ijsea.2015.6302.

[6] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, “Building an Ensemble for Software Defect Prediction Based on Diversity Selection,” *Int. Symp. Empir. Softw. Eng. Meas.*, vol. 08–09–Sept, 2016, doi: 10.1145/2961111.2962610.

[7] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,” *Inf. Softw. Technol.*, vol. 59, pp. 170–190, 2015, doi: 10.1016/j.infsof.2014.11.006.

[8] L. Pelayo and S. Dick, “Applying novel resampling strategies to software defect prediction,” *Annu. Conf.*

Copyright © Society for Science and Education United Kingdom

North Am. Fuzzy Inf. Process. Soc. - NAFIPS, no. July 2007, pp. 69–72, 2007, doi: 10.1109/NAFIPS.2007.383813.

[9] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Syst. Appl.*, vol. 42, no. 4, pp. 1872–1879, 2015, doi: 10.1016/j.eswa.2014.10.025.

[10] M. J. Siers and M. Z. Islam, "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," *Inf. Syst.*, vol. 51, no. March 2018, pp. 62–71, 2015, doi: 10.1016/j.is.2015.02.006.

[11] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2015.

[12] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Inf. Softw. Technol.*, vol. 122, 2020, doi: 10.1016/j.infsof.2020.106287.

[13] M. C. M. Prasad, L. F. Florence, and A. Arya, "A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques," *Int. J. Database Theory Appl.*, vol. 8, no. 3, pp. 179–190, 2015, doi: 10.14257/ijdta.2015.8.3.15.

[14] A. Tosun and A. Bener, "Reducing false alarms in software defect prediction by decision threshold optimization," *2009 3rd Int. Symp. Empir. Softw. Eng. Meas. ESEM 2009*, pp. 477–480, 2009, doi: 10.1109/ESEM.2009.5316006.

[15] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," *ACM Int. Conf. Proceeding Ser.*, 2014, doi: 10.1145/2601248.2601294.

[16] A. Al-Nusirat, F. Hanandeh, M. Kharabsheh, M. Al-Ayyoub, and N. Al-dhufairi, "Dynamic detection of software defects using supervised learning techniques," *Int. J. Commun. Networks Inf. Secur.*, vol. 11, no. 1, pp. 185–191, 2019.

[17] L. Goel, M. Sharma, S. Khatri, and D. Damodaran, "Prediction of Cross Project Defects using Ensemble based Multinomial Classifier," *ICST Trans. Scalable Inf. Syst.*, vol. 0, no. 0, p. 159974, 2018, doi: 10.4108/eai.13-7-2018.159974.

[18] A. Alsaeedi and M. Z. Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study," *J. Softw. Eng. Appl.*, vol. 12, no. 5, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.

[19] M. Hammad, A. Alqaddoumi, H. Al-Obaidy, and K. Almseidein, "Predicting software faults based on k-nearest neighbors classification," *Int. J. Comput. Digit. Syst.*, vol. 8, no. 5, pp. 461–467, 2019, doi: 10.12785/ijcds/080503.

[20] A. Iqbal *et al.*, "Performance analysis of machine learning techniques on software defect prediction using NASA datasets," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, pp. 300–308, 2019, doi: 10.14569/ijacsa.2019.0100538.

[21] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?," *Softw. Qual. J.*, vol. 26, no. 2, pp. 525–552, 2018, doi: 10.1007/s11219-016-9353-3.

[22] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Reliab.*, vol. 62, no. 2, pp. 434–443, 2013, doi: 10.1109/TR.2013.2259203.

[23] X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu, and J. Liu, "Dictionary learning based software defect prediction,"

Proc. - Int. Conf. Softw. Eng., no. 1, pp. 414–423, 2014, doi: 10.1145/2568225.2568320.

[24] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," pp. 357–361, 1994.

[25] J. Sayyad Shirabad and T. J. Menzies, "The {PROMISE} Repository of Software Engineering Databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005, [Online]. Available: http://promise.site.uottawa.ca/SERepository.

[26] T. J. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.*, vol. 2, no. 4, pp. 308–320, 1976.

[27] M. H. Halstead, "Elements of Software Science," *Elsevier*, vol. 17, p. 127, 1977, [Online]. Available: https://www.semanticscholar.org/paper/Elements-of-software-science-Halstead/69e18701cf756d8463f3dde24dcb84c5e2e6dcdf.

[28] J. Brownlee, "A Gentle Introduction to k-fold Cross-Validation," May 2018. https://machinelearningmastery.com/k-fold-cross-validation/ (accessed Jun. 13, 2020).

[29] J. Brownlee, "Failure of Classification Accuracy for Imbalanced Class Distributions," Jan. 2020. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/ (accessed Jun. 13, 2020).

[30] J. Brownlee, "ROC Curves and Precision-Recall Curves for Imbalanced Classification," Jan. 2020. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/ (accessed Jun. 13, 2020).

[31] Metacademy, "F measure," *2020*. https://metacademy.org/graphs/concepts/f_measure (accessed Jun. 13, 2020).

[32] Metacademy, "Precision and recall," 2020. https://metacademy.org/graphs/concepts/f_measure#focus=precision_recall&mode=learn (accessed Jun. 13, 2020).

[33] D. Parson, "Data Analytics, Kappa statistic," 2020. http://faculty.kutztown.edu/parson/fall2019/Fall2019Kappa.html (accessed Jun. 13, 2020).

[34] S. Glen, "RMSE: Root Mean Square Error," *StatisticsHowTo.com*, 2020. https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/ (accessed Jun. 13, 2020).

[35] S. Holmes, "RMS Error," 2000. http://statweb.stanford.edu/~susan/courses/s60/split/node60.html (accessed Jun. 13, 2020).

Copyright © Society for Science and Education United Kingdom