HBRP PUBLICATION

Journal of Advances in Computational Intelligence Theory
Volume 1 Issue 3

Software Defect Prediction System using Machine Learning based Algorithms

***Sanusi B. A.*[1], *Olabiyisi S. O.* [2], *Olowoye A. O* [3], *Olatunji B. L.* [4].**
Department of Computer Science
Ladoke Akintola University of Technology, Ogbomoso, Nigeria.
Corresponding Author
E-mail Id:- *[1]sanusibashiradewale90@gmail.com*
[2]soolabiyisi@lautech.edu.ng
[3]aoolowoye@pgschool.lautech.edu.ng
[4]olatunji_tunde@yahoo.com

ABSTRACT

Measuring the performance, reliability or quality of a software simply describes the sequence of actions taken detecting bugs in a software product. The Bugs found during the development of software has made researchers develop different methods of bug prediction models. However, predicting the bugs in a concurrent software product reduces development time and cost. In this paper, experiments were conducted on public available bug prediction dataset which is a repository for most open source software. The Genetic algorithm was used to extract relevant features from the acquired datasets to eliminate the possibility of over-fitting. The extracted features are classified to defective or non-defective using random forest, decision tree and artificial neural network classification technique. Furthermore, the techniques were evaluated using accuracy, precision, recall and f-score. In completion of the conducted experiments, the random forest performs best among the algorithms in terms of accuracy, precision, and f-score with average score of 83.40%, 53.18%, and 52.04% respectively. Also, the results showed that neural network performs best in terms of recall with average score of 60% among the algorithms. Hence, the system helped software developers when developing a good quality software in order to check if the software system has a little or no defects before delivery to customers.

Keywords:-*Random Forest, Decision Tree, Artificial Neural Network, Software Defect Prediction, Software metrics, Genetic Algorithm.*

INTRODUCTION

A software defect is defined as a flaw, fault or failure in a computer system or program that gives an unexpected or incorrect result [9]. It gives either an incorrect, or unexpected result, and behaves in unintended ways. The unexpected result is identified during software testing and marked as a defect. The Software defect prediction approaches are much more cost effective to detect software defects as compared to software testing and reviews. According to recent studies, it is reported that the probability of detection of software bug prediction models may be higher than probability of detection of currently software reviews used in relating methods [8].Consequently, timely identification of software bugs facilitates the testing resources allocation in an efficient manner and enables developers to improve the architectural structure of a system by identifying the high-risk segments of the system [8]. It is relevant to identify fault-prone code at each stage of software testing and prediction of defects in the software product enhances good quality software. The Feature selection is a method for handling large metric sets, to identify which metrics contribute to the software defect prediction performance. By using the feature selection, redundant and hence, non-independent attributes are removed from the dataset [5].

In this paper the genetic algorithm was used for extracting relevant features from the raw datasets. Two approaches can be used to build a software defect prediction model like supervised learning and unsupervised learning. However, the Supervised learning has the problem that to train the software defect prediction model need the historical data or some known results.

There are many techniques or learning algorithm to select a vast variety of software metrics. However, The Random Forest, Decision Tree and Artificial Neural Network techniques were used in this paper for the prediction model with minimum set of metrics that can achieve the acceptable result. The performance of the techniques is evaluated using the accuracy, precision, recall and f-score. The accuracy is described as the number of the correctly classified instances. The precision measures the proportion of the identified files, classified as faulty and are actually faulty while recall measures the proportion of faulty files which are correctly identified as faulty.faulty.

RELATED WORK

Menzies et al. [7] make use of OneR, a classification rule algorithm to test thresholds of single attributes. They concluded that OneR is outperformed by the J48 decision tree most of times. Shafi et al. [10] used in addition to OneR another classification rule technique called ZeroR and was outperformed by OneR. ZeroR predicts the value of the majority class. Arisholm et al. has in two different studies [2] and [3], used the meta-learners Decorate and AdaBoost together with J48 decision tree. They assert that Decorate outperformed AdaBoost on small datasets and performed comparably well on large datasets. However, they did not disclose their definition of small and large datasets.

Grishma and Anjali investigated root cause for fault prediction by applying clustering techniques and identifies the defects occurs in various phases of SDLC. In this research they used COQUALMO prediction system to predict the defect in a software and applied various clustering algorithms like k-means, agglomerative clustering, density-based scan, COBWEB, expectation maximization and farthest first. The Implementation was done using WEKA tool. Finally, it was concluded that k-means technique works better when compared with other algorithms [4].

The studies in [11,6] both analyzed the applicability of various ML methods for fault prediction. Sharma and Chandra [11] then added to their study the most important previous researches about each ML techniques and the current trends in software bug prediction using machine learning. This study can be used as ground or step to prepare for future work in software defect prediction.

Agasta and Ramachandran [1] In Predicting the fault-proneness of program modules when the fault labels for modules are unavailable is a challenging task frequently raised in the software industry. They attempted to predict the fault–proneness of a program modules when fault labels for modules are not present. Supervised techniques like Genetic algorithm-based software defect prediction approach for classification has been proposed.

Yu *et al.* [12] then developed a model with a combination of derived metric sets to make better prediction of defect in concurrent software programs using deep learning technique which was named ConPredictor.

METHODOLOGY

Data Collection

The datasets were acquired from bug prediction dataset which is publicly available for use. This dataset is a repository for defect prediction for most open source software. In this paper, weighted entropy dataset codenamed “weighted-ent” was used out of the files in each repository. The Weighted entropy is the measure of information supplied by a probabilistic test whose elementary events are characterized by both their objective and qualitative weights.

Feature Selection

The Feature selection which may be referred to as attribute selection is simply the process of selecting subset of relevant features which is then used for the prediction model. However, this stage was achieved by using the genetic algorithm which was used to extract the features that most affect the outputs that is the number of bugs in a software product. In this paper, a typical genetic algorithm flowchart is represented in the Figure 1 below.

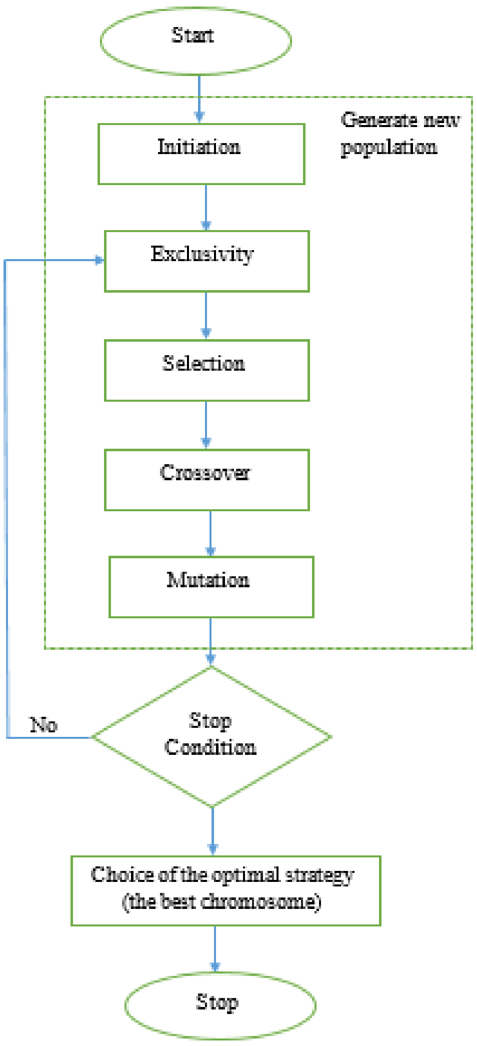

Fig.1:-*The Flowchart of a Typical Genetic Algorithm*

Machine Learning Algorithms

The Machine learning algorithms learn to predict outputs based on previous examples. However, in this paper as the experiment was conducted to test the extracted features, the learning algorithms were used with their standard settings in the MATLAB environment using statistical toolkit. The learning algorithms used in building the defect prediction model in this paper are Random Forest (RF), Decision Tree (DT) and Artificial Neural Network (ANN).

Random Forest

The premise of this techniques is building small decision tree with few features which is therefore computationally cheap. However, RF is an ensemble learning algorithm. Considering weak and small decision trees in parallel, the tress can then be combined in order to form a strong and single learner by taking the majority vote. Furthermore, random forests are often found to be the most accurate learning algorithms. Hence, the pseudocode used in this paper is given below in algorithm 1. Thus, building many more trees using the random forest learning algorithm is not only an option but these trees will also be less correlated which enables this algorithm to have good performance.

Algorithm 1: Pseudocode of Random Forest

```
Precondition: A training set $S := (x_1, y_1), \dots, (x_n, y_n)$, features $F$, and number of trees in forest $B$
FunctionRandomForest($S, F$)
$H \leftarrow \emptyset$
Fori $\in 1, \dots, B$ do
$S^{(i)} \leftarrow$ Abootstrap sample $S$
$h_i \leftarrow$Randomized Treelearn ($S^{(i)}, F$)
$H \leftarrow H \cup \{h_i\}$
End for
Return H
End function
Function Randomized Treelearn($S, F$)
At each node:
$F \leftarrow$ very small subset of $F$
Split on best feature in $F$
Return the learned tree
End function
```

Decision Tree

A decision tree can be described as one of the supervised learning algorithm that is widely used for classification and regression task). The cognitive procedure of acquiring knowledge and classification measure of decision tree are not complex. In this work, after the conducted experiments a decision tree was generated from the training samples and the defects were classified as represented in algorithm 2 below.

Algorithm 2: Pseudocode of Decision Tree Learning

```
Tree-Learning (TR, Target, Attr)
TR: training examples
Target: target attribute
Attr: set of descriptive attributes
{
Generate a Root node for the tree.
If TR have the same target attribute value $t_i$,
Then Return the single-node tree, that is. Root, with target attribute = $t_i$
IfAttr = empty (simply means no expressive attributes present),
Then Return the single-node tree, i.e. Root, with most common value of Target in TR
Otherwise
{
```

```
Select attribute A from Attr that classifies better TR depending on an entropy-based measure
Set A the attribute for Root
For each legal value of A, $v_i$, do
{
Add a branch below Root, corresponding to A = $v_i$
Let$TR_{vi}$be the subset of TR that have  A = $v_i$
If$TR_{vi}$is empty,
Then add a leaf node beneath the branch with target value = most common value of
Target in TR
Else below the branch, add the subtree learned by Tree-Learning( $TR_{vi}$, Target, Attr-{A})
}
}
Return (Root)
}
```

where
t_i = the value of the target attribute and
v_i =the value of descriptive attributes

Neural Network

Neural networks (NN) is simply an important tool for classification. The recent wide research activities in neural classification having existed that NN are a promising alternative to various conventional classification methods. In the classification stage, neural network is capable of producing an intended result with the use of labeled training segments. However, an Artificial Neural Network (ANN) is a structure built on the performance of biological neural networks. ANN is a learning algorithm based on a model that can simply be used for classification. Furthermore, some algorithms are in existence used in training neural network like Newton Method, Gradient Descent, Levenberg-Marquardt (LM) e. t. c. In this paper, LM was adopted which is used for training the ANN. Algorithm 3 shows the pseudocode of Levenberg-Marquardt used for the defects classification.

Algorithm 3: Pseudocode of Levenberg-Marquardt

```
Initialize Weights;
While not stop Criterion do
Calculates $C^P(w)$for each pattern
$e_1 = \sum_{P}^{P} = 1\, e^P(w)^T e^P(w)$
Calculates $J^P(w)$for each pattern
Repeat
Calculates $\Delta w$
$e_2 = \sum_{P}^{P} = e^P(w+\Delta w)^T e^P(w+\Delta w)$
If$e_1 \leq e_2$then
$\mu = \mu * \beta$
End If
Until$e_1 < e_2$
$\mu = \mu/\beta$
$w = w + \Delta w$
End while
```

where
$J^P(w)$ is the Jacobian matrix of the error vector
$e^P(w)$ is evaluated in w
I is the specification matrix.
Hence, the parameter μ is increased or decreased at each step.

Classification Stage

In the classification stage, the extracted features were classified to defect or non-defect using random forest, neural network and decision tree as highlighted in the previous section. Four times 4-fold cross-validation was performed when evaluating the prediction model. The datasets are separated into four equal parts. Three parts out of four are used for the extraction process and as training data while the forth part is used for testing. In order for every part of the datasets to be used as training and testing, this procedure was repeated four times. Cross validation was adopted since the number of data is limited and it has a merit over the existing technique called holdout method. In the holdout method, one part of the datasets is used for training and the other for testing. However, the solution to the bias idea was adopted using cross validation where all the instances were used one time for testing and training. This simply means that, instead of conducting four folds, a total of 16 folds is generated and the error estimate is therefore more reliable.

Performance Evaluation

The performance measures of the software defect prediction were achieved by this classification model with the availability and effectiveness of the metrics. True-Positive, False-Positive, False-Negative and True-Negative prediction outcomes was considered. Thereafter, the defect prediction performance was based on the following;

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative + False\ Negative} \quad (1)$$

This gives the quantitative relation of prediction that are correct.

$$Precision = \frac{True\ Postive}{True\ Positive + False\ Positive} \quad (2)$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (3)$$

$$F - measure = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \quad (4)$$

By collecting these performance measurements, future predictions on unseen files can be estimated. The block diagram of the defect prediction model is presented in Figure 2.

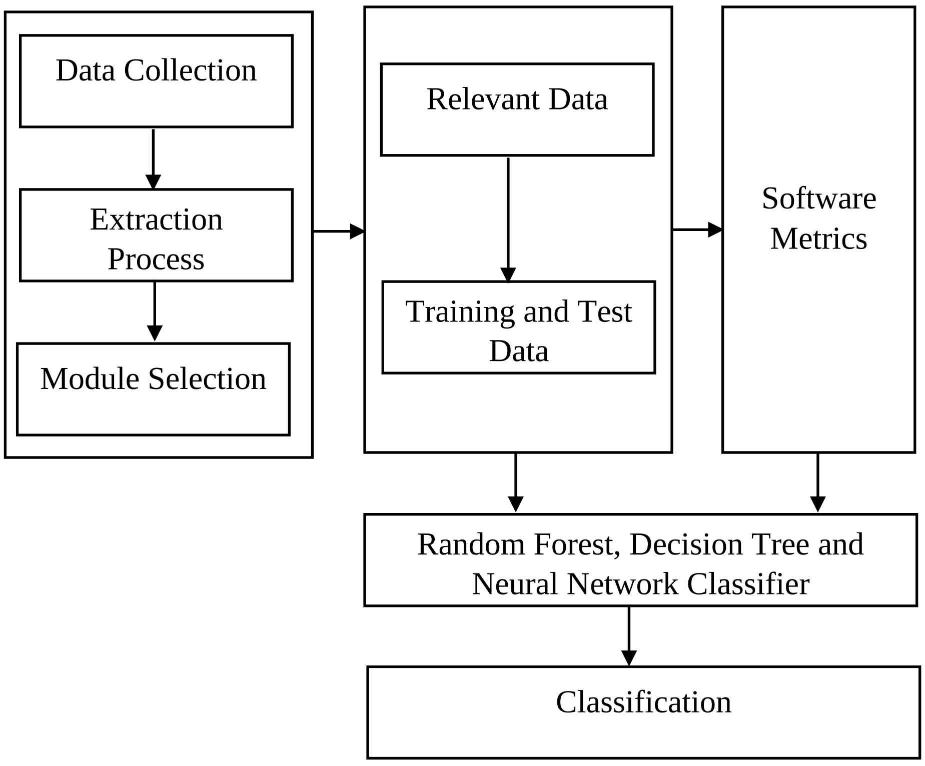

***Fig.2:**-Proposed Architecture.*

RESULTS AND DISCUSSION

The results achieved with the extracted feature set from the raw datasets are compared. Three techniques are then used for the defect prediction with cross validation. Cross validation tests exist in different ways but the method adopted in this paper is to divide the training data into a number of folds. The classifiers are evaluated by their classification using the accuracy, precision, recall and f-score, on one fold after having learned from the other folds. This procedure is then repeated until all the folds take part in the evaluation. The tables below show the performance evaluation of the techniques using the accuracy, precision, recall and f-score as highlighted in section 3 of this paper.

Accuracy

Table 1 represents the accuracy of the techniques for the set of data used in this paper. The average of the accuracy for each learning algorithms were calculated and the values are described in percentage. In table 1, one can see that the random forest algorithm outperformed the other classifiers. In summary, the random forest is the best algorithm for the overall datasets evaluated by accuracy.

***Table 1:**-The algorithm performance per dataset rated by accuracy*

Datasets	Artificial Neural Network	Random Forest	Decision Tree
ECLIPSE JDT CORE	86.93%	83.92%	75.88%
ECLIPSE PDE UI	83.28%	83.61%	81.81%
EQUINOX FRAMEWORK	70.77%	76.92%	73.85%
LUCENE	91.3%	89.13%	89.86%
AVERAGE	83.07%	83.40%	80.3%

Precision

The Precision is another performance evaluation which estimates how well the prediction model classifies faulty files that are actually faulty. Table 2 represents the individual score of all the learning algorithm per dataset as well as measuring the average of each classifier. In summary, the random forest is the best algorithm for the overall datasets rated by precision having average score of 53.18% followed by the ANN with 44.11%.

***Table 2:**-The algorithm performance per dataset rated by precision*

Datasets	Artificial Neural Network	Random Forest	Decision Tree
ECLIPSE JDT CORE	53.49%	76.74%	4.65%
ECLIPSE PDE UI	31.91%	34.04%	6.38%
EQUINOX FRAMEWORK	57.69%	76.92%	76.92%
LUCENE	33.33%	25%	0%
AVERAGE	44.11%	53.18%	21.99%

Recall

The Recall estimates how many of the faulty files the prediction model finds. As represented in Table 3 below, decision tree still estimates the dataset LUCENE at 0% recall. Also, the best algorithm is artificial neural network for the overall datasets rated by recall which is quiet a big gap to the other classifiers.

***Table 3:**-The algorithm performance per dataset rated by recall*

Datasets	Artificial Neural Network	Random Forest	Decision Tree
ECLIPSE JDT CORE	79.31%	60%	22.22%
ECLIPSE PDE UI	45.45%	47.06%	21.43%
EQUINOX FRAMEWORK	65.22%	68.97%	64.52%
LUCENE	50%	33.33%	0%
AVERAGE	60%	52.34%	27.04%

F-Score

F-Score is the last performance measure as highlighted in the section III above. This is a combination of recall and precision. Table 4 contains the values for all the datasets and the overall average value for each learning algorithms. Decision tree value for dataset LUCENE still stays at 0% f-score. However, the best algorithm is the random forest for overall rated by f-scorewith 52.04%.

***Table 4:**-The algorithm performance per dataset rated by f-score*

Datasets	Artificial Neural Network	Random Forest	Decision Tree
ECLIPSE JDT CORE	63.89%	67.35%	7.69%
ECLIPSE PDE UI	37.5%	39.5%	9.83%
EQUINOX FRAMEWORK	61.22%	72.73%	70.18%
LUCENE	40%	28.57%	0%
AVERAGE	50.65%	52.04%	21.93%

CONCLUSION

The increase in software development process gives rise to different defect prediction techniques and models which are useful in producing reliable and quality software. Experiments were carried out on publicly available bug prediction datasets in this paper, by extracting relevant features from the original sets so as to avoid overfitting and results reveal the performance evaluation measure of the techniques per datasets. However, the best algorithm overall is random forest which is clearly seen on the tables represented in section IV above. Also, it is observed that the use of decision tree technique does not provide better prediction performance as shown in the overall average of each performance measures. Furthermore, the results in this paper are compared and in some cases and better when compared to other same software defect prediction models.

REFERENCES

1. Agasta, A. and Ramachandran, M. (2014). *Predicting the Software Fault Using Genetic Algorithm technique.* The International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering.2014.3(2):390-398p.
2. Arisholm, E., Lionel, C. B and Magnus, F. (2007*). Data Mining Techniques for Building Fault-proneness Systems in Telecom Java Software*. In the 18th International Symposium on Software Reliability (ISSRE’07), 2007.215–224p.
3. Arisholm, E., Lionel, C. B. and Eivind, B. (2010). A Comprehensive and Systematic Investigation of Methods to Build and Evaluate Fault Prediction models. *Journal of Systems and Software,* 83(1):2–17p.
4. Grishma, B. R., and Anjali, C. (2015). Software root cause prediction using clustering methods: A review. Communication Technologies (GCCT), *2015 Global Conference on.IEEE.*
5. Eibe F., Ian H. W., and Mark A. H. (2011). *The Data Mining: Practical Machine Learning Tools and Methods*, Third Edition (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann.2011.
6. Malhotra, R. (2014). *Comparative analysis of statistical and machine learning techniques For predicting buggy modules.* Applied Soft Computing. 2014.21:286-297p.
7. Menzies, T., Greenwald, J. and Frank, A. (2007). *The Data mining static code features to learnbug predictors.* IEEE Trans. Softw. Eng. 2007.33:2–13p.
8. Menzies, T., Milton, Z., Turhan, B.,

Cukic, B., Jiang, Y., and Bener, A. (2010). *Bug prediction from static code attributes: current results, limitations, new techniques*. The Automated Software Engineering.2010.17(4):375–407p.

9. Parameswari, A. (2015). *Comparing Data Mining Techniques for the Software Defect Prediction*.
10. Shafi, S., Syed, M. H., Afsah, A., Malik, J. K. and Shafay, S. (2008). *The Software Quality Prediction Techniques: A Comparative Analysis.* In 2008 4th International Conference on Emerging Technologies.2008:242–246p.
11. Sharma, D. and Chandra, P. (2018). *Software Fault Prediction Using Machine-Learning Techniques"*. Smart Computing and Informatics.Springer, Singapore.2018:541-549p.
12. Yu, T., Wen, W., Han, X. and Hayes, J. (2018). *The Conpredictor model: The Concurrency Defect Prediction in Real-World Applications*. In the International Conference on Software Testing, Verification and Validation.2018:168-179p.