International Journal of Advances in Applied Sciences (IJAAS)
Vol. 9, No. 4, December 2020, pp. 284~293
ISSN: 2252-8814, DOI: 10.11591/ijaas.v9.i4.pp284-293

Development of software defect prediction system using artificial neural network

Olatunji B. L.[1], Olabiyisi S. O.[2], Oyeleye C. A.[3], Sanusi B. A.[4], Olowoye A. O.[5], Ofem O. A.[6]
[1,2,4,5]Department of Computer Science, Ladoke Akintola University of Technology, Nigeria
[3]Department of Information Systems, Ladoke Akintola University of Technology, Nigeria
[6]Department of Computer Science, University of Calabar, Nigeria

Article Info

Article history:

Received Mar 23, 2020
Revised Jun 11, 2020
Accepted Jun 18, 2020

Keywords:

Artificial neural network
Genetic algorithm
Software defect prediction
Software metrics

ABSTRACT

Software testing is an activity to enable a system is bug free during execution process. The software bug prediction is one of the most encouraging exercises of the testing phase of the software improvement life cycle. In any case, in this paper, a framework was created to anticipate the modules that deformity inclined in order to be utilized to all the more likely organize software quality affirmation exertion. Genetic Algorithm was used to extract relevant features from the acquired datasets to eliminate the possibility of overfitting and the relevant features were classified to defective or otherwise modules using the Artificial Neural Network. The system was executed in MATLAB (R2018a) Runtime environment utilizing a statistical toolkit and the performance of the system was assessed dependent on the accuracy, precision, recall, and the f-score to check the effectiveness of the system. In the finish of the led explores, the outcome indicated that ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK and LUCENE has the accuracy, precision, recall and the f-score of 86.93, 53.49, 79.31 and 63.89% respectively, 83.28, 31.91, 45.45 and 37.50% respectively, 83.43, 57.69, 45.45 and 50.84% respectively and 91.30, 33.33, 50.00 and 40.00% respectively. This paper presents an improved software predictive system for the software defect detections.

Corresponding Author:

Olatunji B. L.,
Department of Computer Science,
Ladoke Akintola University of Technology,
Gra oke aafin area Ogbomoso north local government, 210214, Ogbomosho, Nigeria.
Email: olatunji_tunde@yahoo.com

1. INTRODUCTION

A software defect is a fault, blunder, or failure in a software system [1]. It creates either an off base, or unforeseen result, and acts in a unintended way [2]. It is a flaw in the software system that makes it perform out of the blue [3]. A software defect can be referred to as imperfection during the software improvement process that makes the software fail and not meets the ideal desire [4]. The defect prediction in software is the way toward deciding pieces of a software system that may contain bugs [5]. Use of Defect Prediction systems in the early software life-cycle permits the pro to focus their testing labor in a way that the parts identified as mistake inclined are tried inside and out in contrast with different pieces of the software system [6] This prompts the decrease of labor costs during improvement and furthermore loosens up the support effort [7]. Late investigations report that the chance of bug discovery by the software defect prediction systems might be higher than the chance of identification by as of now utilized software audits in mechanical strategies [8]. Thusly, the right prediction of defect-inclined software assists with

coordinating test effort, to decrease costs, to improve the software testing process by focusing on defect-inclined modules [9], lastly to make the nature of the software better [10].
That is the reason today's software defect prediction is a significant examination subject in the software engineering field [11]. Software defect prediction is a key procedure in software engineering to make the quality and affirmation of software better in less time and at least expense [12]. It is actualized before the testing phase of the software advancement life cycle. Software defect prediction systems give defects or various defects.

The software defect prediction has been roused by various analysts to give a different system inside a task or cross-undertaking to improve different quality and watching affirmation of software [12]. There are two ways to deal with builds a software defect prediction system like supervised learning and unsupervised learning. Supervised learning has an issue of requiring historical information to prepare the software defect prediction system while unsupervised learning doesn't require historical information or some known outcomes [2]. The improvement of software technology causes an expansion in the number of software items, and their support has become a difficult assignment. Besides, half of the life cycle cost for a software system incorporates upkeep exercises. With the ascent in complexity in software systems, the likelihood of having defective modules in the software systems is getting higher [13]. A key focus, defect prediction, has risen as a functioning examination zone for decades. Defect prediction methods build systems dependent on different sorts of metrics and foresee defects at different granularity levels, e.g., change, file, or module levels [14]. These procedures can be utilized to effectively apportion quality confirmation assets. In spite of various defects, prediction contemplates research on defect prediction despite everything increments exponentially.

Tending to this issue can give knowledge to the two experts and scientists. Experts can utilize observational proof on defect prediction to settle on informed choices about when to utilize defect prediction and how it would best fit into their advancement procedure. Specialists can improve defect prediction procedures dependent on the desires for professionals and appropriation challenges that they face. To pick up bits of knowledge into the reasonable estimation of defect prediction, a quantitative report was performed in this examination so as to help software designers with the errand of comprehension, assessing, and improving their software items. It is imperative to predict and fix the defects before it is conveyed to clients in light of the fact that the software quality confirmation is a tedious task and now and again doesn't take into consideration complete testing of the whole system because of spending issues. There are numerous open datasets that are accessible free for specialists like PROMISE, ECLIPSE, and APACHE to conquer the difficult issue when preparing performed on another project. Analysts have been creating enthusiasm to build a cross-project defect prediction system with various metrics set like class-level metrics, process metrics, static code metrics yet they couldn't build increasingly feasible systems [12]. There are numerous classifiers or learning algorithm to choose a wide assortment of software metrics like Naive Bayes, Support Vector Machine, K-Nearest Neighbor, Random Forest, Decision Tree, Neural Network and Logistic Regression. Hence, in this paper a software defect prediction system was developed using Artificial Neural Network as the classifying algorithm and with the use of Genetic Algorithm the possibility of overfitting was eliminated by extracting the relevant features from the original datasets which the outcomes give best predictive performance.

2. RELATED WORK

Fenton and Neil [15], make utilization of Bayesian networks for forecasting of unwavering quality and defectiveness of software. It makes utilization of casual process factors and qualitative and quantitative measures, in this manner taking into account the constraints of traditional software impediments. The utilization of a powerful discretization method brings about a better prediction system for software defects. Jie et al. [16], make utilization of different statistical procedures, and machine learning methods were utilized to verify the validity of software defect prediction systems. In this investigation, the neuro-fuzzy method was thought of. The data from ISBSG were taken to achieve the research. Manu [17], make utilization of another computational insight sequential hybrid design including Genetic Programming (GP) and Group Method of Data Handling (GMDH) viz. The GPGMDH has been contemplated. Be that as it may, the GP and GMDH, a large group of methods on the ISBSG dataset have been tried.

The GP-GMDH and GMDH-GP hybrids surpass all other independent and hybrid procedures. It is presumed that the GPGMDH or GMDH-GP system is the greatest system among all different methods for software cost estimation. Puneet and Pallavi [18] utilized different data mining strategies for software mistake prediction, like affiliation mining, classification, and clustering methods. This has helped the software engineers in growing better systems. For a situation where defect marks are absent, unsupervised procedures can be utilized for system advancement.

In 2014, Mattias and Alexander worked on software defect prediction utilizing machine learning (Random Forest and J46) on test and source code metrics. The goal of the proposal was to explore whether a test, combined with a source code file contained enough information to upgrade the software defect performance if metrics from both source files and test files are joined. Gray et al. [19] proposed an investigation utilizing the static code metrics for a group of modules contained inside eleven NASA data sets and make utilization of a Support Vector Machine classifier. A careful progression of the pre-processing stage was applied to the data before classification, including the balancing of the two classes (defective or something else) and the dismissal of countless rehashing events. The Support Vector Machine in this trial yields a normal accuracy of 70% on previously inconspicuous data. According to the reviewed related works, it is observed that the previously developed software prediction systems have a limitation of overfitting which happens when the system acquire the detail in the training data to the extent that it negatively effects the performance of the system on new data.

3. RESEARCH METHOD

The architecture of the developed system in this paper is presented in Figure 1. The following are the stages that were adopted in this paper:

i. The first stage is acquisition of data. This stage involves gathering necessary datasets which were used in this paper. However, the datasets were acquired from http://bug.inf.usi.ch/download.php which is publicly available for use.
ii. The next stage is the feature selection stage which was achieved by using Genetic Algorithm so as to extract the relevant features from the datasets acquired in the first stage.
iii. In the classification stage, the extracted features were classified using Artificial Neural Network.
iv. Finally, the results of this work were evaluated using accuracy, precision, recall and f -score.

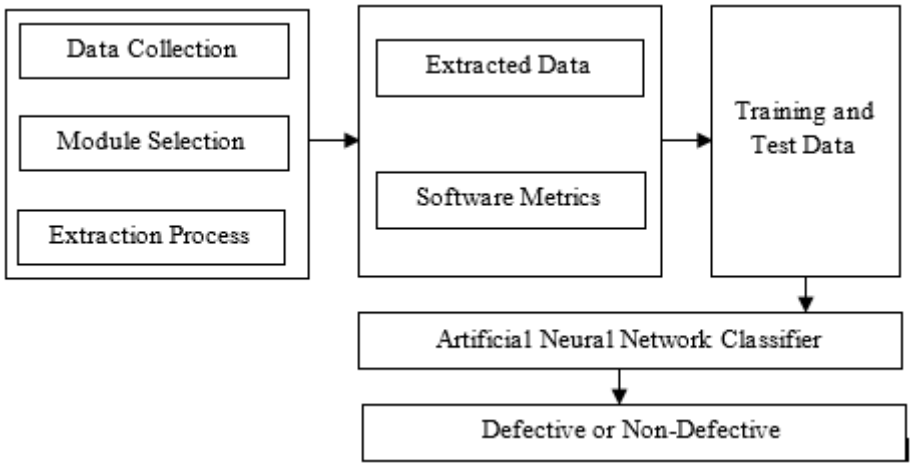

Figure 1. Architecture of the developed system.

3.1. Data collection

Software defect prediction research depends on data that must be gathered from in any case separate stores. In this paper, the datasets were acquired from http://bug.inf.usi.ch/download.php which is a store for the bug prediction dataset for most open-source software. “The Eclipse Jdt Core, Eclipse Pde Ui, Equinox Framework and Lucene” are the software systems that were considered in this paper. However, each software systems includes different pieces of information but in this paper weighted entropy module codenamed “weighted.ent” was selected because it has most familiar parameters like lines of code which suites the aim of defect prediction system. Weighted entropy is the proportion of data provided by a probabilistic test whose basic occasions are described by both their target probabilities and by some subjective loads.

3.2. Feature selection

The computational complexity of some of the previously mentioned machine learning algorithms makes the building of the system infeasible to use if all of the features in the dataset is used. Along these lines, feature selection was utilized to remove a lot of most significant free factors contained in the first

dataset to dispense with factors that won't add to the presentation of prediction, at that point improve learning proficiency and increment prediction accuracy. However, in this paper Genetic Algorithm (GA) was used for extracting the relevant features in eliminating the possibility of overfitting. GA is a versatile heuristic technique for worldwide advancement looking through used to create valuable answers for machine learning applications and it reenacts the conduct of the development procedure in nature. Figure 2 depicts the flowchart of a typical GA. The feature was ultimately reduced using the fitness function;

$$\sum_{i=1}^{m} \frac{1}{|(\sum_{j=1}^{n} g(j)) - R(i)|} \quad (1)$$

where
g is a $m \times n$ matrix of feature and
R is the corresponding output.

3.3. Classification stage

The extracted relevant feature was divided into folds and ensure that each fold was used as testing set at some point and used to train the classifier. K-fold cross validation was adopted where the acquired datasets was divided into a k number of folds. However, since four open source software were considered in this paper the datasets was divided into 4 folds. In the primary cycle, the principal fold was utilized to test the framework and the rest was utilized to prepare the framework. In the subsequent emphasis, the subsequent fold was utilized as the testing set while the rest fill in as the preparation set. This process was repeated until each fold of the 4 folds are been used as the testing set. The system has a flow in which every user can follow. This also can be used in software engineering field when measuring the flow and quality of a software according to software metrics. Cross validation was adopted since the amount 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. In this paper, 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. Hence, Artificial Neural Network (ANN) was adopted in the classification stage using Levenberg-Marquardt (LM) Algorithm to train the ANN. The choice of the LM Algorithm in this paper is that it is not that memory efficient but faster than other algorithms. It approximates the blunder of the network with a second-order articulation which diverges from the back-propagation algorithm that does it with a first-order articulation. LM refreshes the ANN loads as follows:

$$\Delta w = [\mu I + \sum_{P=1}^{P} J^P(w)^T J^P(w)]^{-1} \nabla E(w) \quad (2)$$

where
$J^P(w)$ is the Jacobian matrix of the error vector;
$e^P(w)$ evaluated in w and
I is the identity matrix.
The vector error $e^P(w)$ is the error of the network for patter p, that is

$$e^P(w) = t^p - o^p(w) \quad (3)$$

The parameter μ is increased or decreased at each step. If the error is reduced, then μ is divided by a factor β and it is multiplied by β in other case. LM performs the steps detailed in Algorithm 1. It calculates the network output, the error vectors and the Jacobian matrix for each pattern. Then, it computes Δw using equation 2 and recalculates the error with $w + \Delta w$ as network weights. If the error has decreased, μ is divided by β, the new weights are maintained and the process starts again; otherwise, μ is multiplied by β. Δw is calculated with a new value and it iterates again [20].

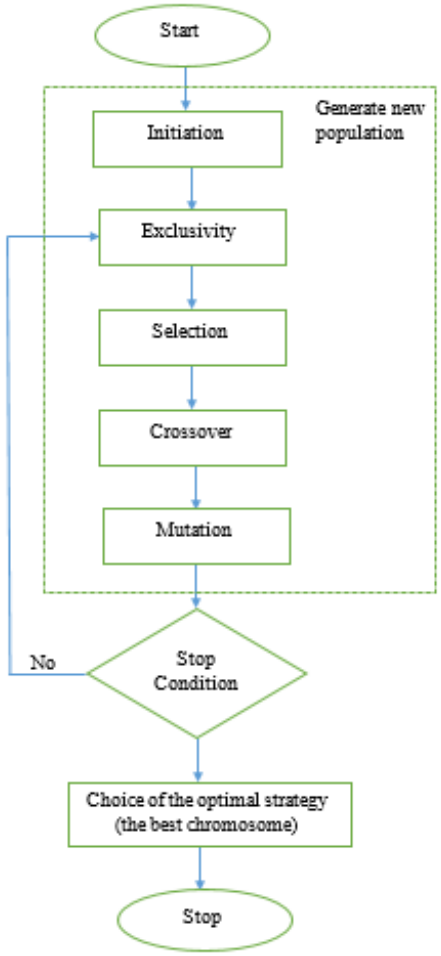

Figure 2. The flowchart of a typical genetic algorithm

Algorithm 1: 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 Δw
```

$$e_2 = \sum_{P}^{P} = e^P(w + \Delta w)^T e^P(w + \Delta w)$$

```
If e1 ≤ e2 then
```

$\mu = \mu * \beta$

```
End If
Until e1 < e2
```

$\mu = \mu/\beta$

$w = w + \Delta w$

```
End while
```

3.4. Performance metrics

In order to measure defect prediction results by classification models, different performance measures are available for effectiveness. In this paper, the following prediction outcomes were considered:

i. True positive (TP): buggy instances predicted as buggy
ii. False positive (FP): clean instances predicted as buggy
iii. True negative (TN): clean instances predicted as clean
iv. False negative (FN): buggy instances predicted as clean

With these outcomes, the following measures which are mostly used in the software defect prediction literature are defined:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (4)$$

Accuracy thinks about both true positives and true negatives over all occurrences. As it were, accuracy shows the proportion of all accurately classified cases.

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

$$Recall = \frac{TP}{TP+FN} \quad (6)$$

Recall measures correctly predicted buggy instances among all buggy instances.

$$F - measure = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \quad (7)$$

F-measure is a harmonic mean of precision and recall. By collecting these performance measurements, future predictions on unseen files can be estimated. The calculation of accuracy, precision and recall makes use of the confusion matrix.

4. RESULTS AND DISCUSSION

The experiment was conducted by first extracting the relevant features from the datasets used in this research as discussed in section 3.2 using GA. However, weighted-ent dataset has 17 features excluding the class names and with the adoption of the GA, the features are reduced to 13 using the fitness function discussed in section 3.2. Figure 3 shows the graphical user interface of the GA at the feature selection stage. Using the mathematical formulas discussed in section 3.4, the values in Table 1 are calculated and by collecting these performance measurements, future predictions on unseen files can be estimated.

According to the conducted experiments the percentage of the True Positive Rate (TPR) and True Negative Rate (TNR) of the datasets used in this research work; ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK and LUCENE are (79.31% and 88.24%), (45.45% and 87.97%), (45.45% and 73.81%) and (50.00% and 93.85%) respectively. The training and validation for the datasets ECLIPSE JDT CORE, ECLIPSE PDE UI, EQUINOX FRAMEWORK and LUCENE was conducted. However, the best validation performance is 0.52482 at epoch 5, 0.21032 at epoch 5, 0.67527 at epoch 9 and 0.01356 at epoch 10 respectively. Figures 4, Figure 5, Figure 6 and Figure 7 shows the chart representation of the training and validation for each dataset respectively.

Summarily, K-fold validation method was used to validate the dataset where all the datasets partakes in both training and testing process as discussed in Section 3.3. More so, as shown in Table 1 throughout the performance measures the dataset LUCENE has the highest accuracy of 91.30% while EQUINOX FRAMEWORK has the highest precision of 57.69% which measures how good the prediction system is at identifying actual faulty files. Furthermore, recall used in this research measures the proportion of faulty files which are correctly identified as faulty where ECLIPSE JDT CORE has the highest recall of 79.31% and highest F-Score of 63.89%.

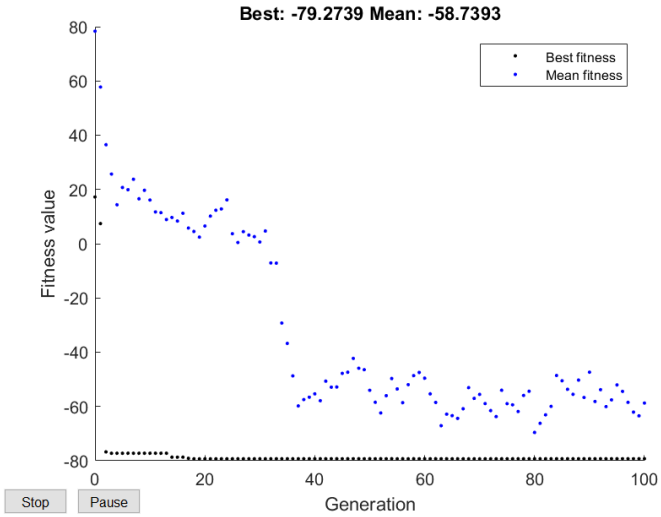

Figure 3. Desktop interface of the feature selection stage

Table 1. System results of the accuracy, precision, recall, and F-Score

Datasets	Accuracy	Precision	Recall	F-Score
ECLIPSE JDT CORE	86.93%	53.49%	79.31%	63.89%
ECLIPSE PDE UI	83.28%	31.91%	45.45%	37.50%
EQUINOX FRAMEWORK	83.43%	57.69%	45.45%	50.84%
LUCENE	91.30%	33.33%	50.00%	40.00%
AVERAGE	86.24%	44.11%	55.05%	48.06%

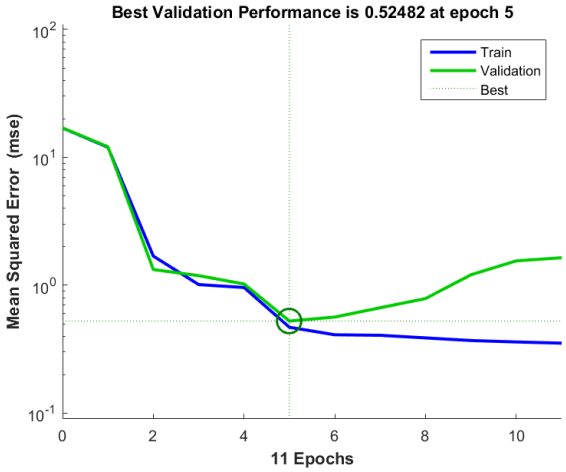

Figure 4. Training and validation for ECLIPSE JDT CORE

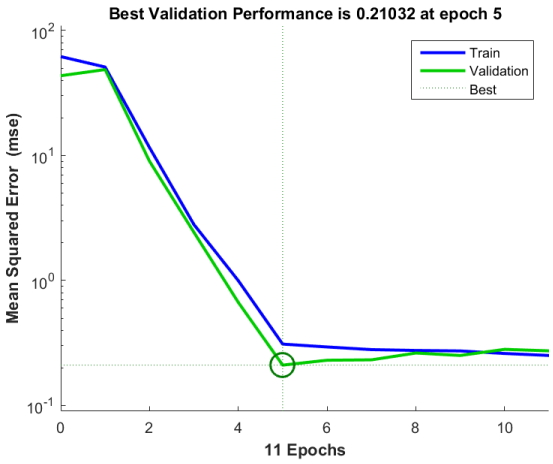

Figure 5. Training and validation for ECLIPSE PDE UI

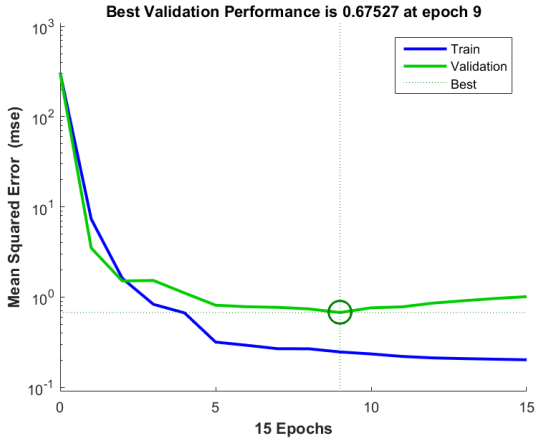

Figure 6. Training and validation for EQUINOX FRAMEWORK

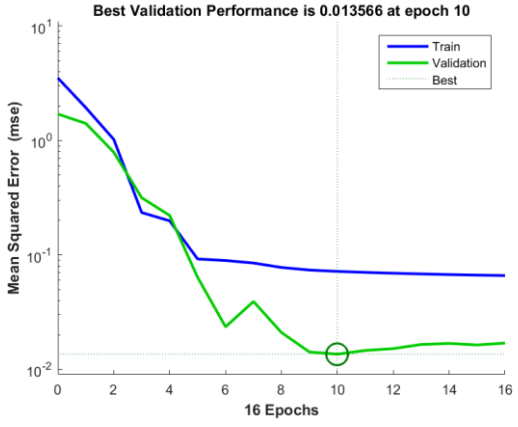

Figure 7. Training and validation for LUCENE

To ease the comparison to the related study, the average of the results for all the datasets and performance measures are presented in Figure 8. As accuracy is dependent on the balance of the underlying dataset, it is further compared to the average accuracy result of the related study. [21] Proposed

a ConPredictor system to predict defects specific to concurrent programs by combining both static and dynamic program metrics. As this research is conducted using the same performance measures as [21] and as they summarize many studies, the results of this study are compared to ones compiled by [21].

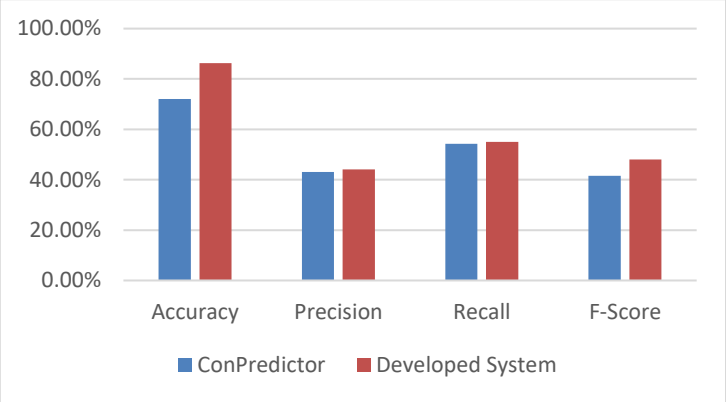

Figure 8. ConPredictor comparison results against the developed system

5. CONCLUSION

The development of software product is increasing exponentially due to their benefits and occurrence of defects in the software products is inevitable. In other words, this defect needs to be reduced to minimum count. Software defect prediction effectively improve the quality and efficiency of software which enhances the procedure of following defective parts in software preceding the beginning of the testing stage. However, some classification techniques such as Naïve Bayes, random forest and decision tree has been adopted for software defect prediction according to literature. Hence, in this paper GA was successfully used for feature selection alongside ANN in predicting the defective modules in a software system. This developed system was compared with existing system which at the completion of the conducted experiments it outshines the existing system by giving a best predictive performance.

REFERENCES

[1] Naik, K. and Tripathy, P., "Software Testing and Quality Assurance," *John Wiley & Sons, Inc*. pp. 29-32, 2008.
[2] Sanusi B. A., Olabiyisi S. O., Olowoye A. O. and Olatunji B. L., "Software Defect Prediction System using Machine Learning based Algorithms," *Journal of Advances in Computational Intelligence Theory*, vol. 1, no. 3, pp. 1-9, 2019.
[3] McDonald M., Musson, R. and Smith, R., "The practical guide to defect prevention," *Control*, pp. 260-272, 2007.
[4] Kumaresh, S. and Baskaran, R., "Defect analysis and prevention for software process quality improvements," *International Journal of Computer Applications*, vol. 8, no. 9, pp. 250-254, 2010.
[5] Kitchenham, B. A.," Guidelines for Performing Systematic Literature Review in Software Engineering," *Technical Report EBSE-2007-001, Keele University and Durham University, Staffordshire*, pp. 184-210, 2007.
[6] Catal, C. and Diri, B., "A Systematic Review of Software Fault Prediction studies," *Expert Syst. Appl.*, vol. 36, pp. 7346–7354, 2009.
[7] Radjenovic, D., Hericko, M., Torkar, R. and Zivkovic, A., "Software fault prediction metrics: A Systematic literature review," *Inf. Softw. Technol*, vol. 55, pp. 1397-1418, 2013.
[8] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., and Bener, A., "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375-407, 2010.
[9] Catal, C., Sevim, U., and Diri, B., "Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2347-2353, 2011.
[10] Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S., "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276-1304, 2012.
[11] Song, Q., Jia, Z., Shepperd, M., Ying, S., and Liu, J., "A General Software Defect-Proneness Prediction Framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356-370, 2011.

[12] Rajesh, K. and Gupta, D. L., "Software Fault Prediction," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 9, pp. 250-254, 2015.
[13] Xu, J. Ho, D. and Carpret, L. F., "An empirical study on the procedure to derive software quality estimation models," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 2, no. 4, pp. 1-16, 2010.
[14] Hata, H., Mizuno, O. and Kikuno, T. "Bug prediction based on fine-grained module histories," *In Proceedings of the 34th International Conference on Software Engineering, ICSE '12*, pp. 200-210, 2012.
[15] Fenton, N. and Neil, M., "Using Bayesian networks to predict software defects and reliability," *Proc. IMechE vol. 222 Part O: J. Risk and Reliability*, pp. 702-703, 2008.
[16] Jie Xu, Danny Ho and Luiz Fernando, "An Empirical Study on The Procedure Drive Software Quality Estimation Models," *International journal of computer science & information Technology (IJCSIT)*, vol. 2, no. 4, pp. 12-15, 2010.
[17] Manu, B., "Computational Hybrids Towards Software Defect Predictions," *International Journal of Scientific Engineering and Technology*, vo. 2, no. 5, pp. 311-316, 2013.
[18] Puneet, J. K. and Pallavi, "Data Mining Techniques for Software Defect Prediction," *International Journal of Software and Web Sciences*, vol. 3, no. 1, pp. 54-57, 2013.
[19] Gray, D., Bowes, D., Davey, N. and Sun, Y., "Bruce Christianson, Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics," *11th International Conference, EANN 2019*, pp. 21-25, 2019.
[20] Khan, K. and Sahai, A., "Comparison of BA, GA, PSO, BP and LM for Training Feed Forward Neural Networks in E-Learning Context," *Int J Intel Syst App*, vol. l7, pp. 23-29, 2012.
[21] Yu, T., Wen, W., Han, X. and Hayes, J., "Conpredictor: Concurrency Defect Prediction in Real-World Applications," *In IEEE International Conference on Software Testing, Verification and Validation*, pp. 168-179, 2018.