Utilizing Machine Learning For Predicting Software Defects

Dr. Jitendra Sheetlani[1*], Prashant Keswani[2]

[1*]Professor of Computer Application, Medicaps University, Indore
[2]Research Scholar, Sri Satya Sai University of Technology and Medical Sciences, Sehore

***Corresponding Author:** Dr. Jitendra Sheetlani
*Professor of Computer Application, Medicaps University, Indore

Abstract
Assessing software effectiveness, reliability, and quality involves a systematic approach to identifying bugs within the product. The detection of bugs during software development has spurred the development of various prediction methods to address them. Predicting bugs in concurrent software products is crucial for reducing development time and costs. This study delves into experiments conducted on a publicly available bug prediction dataset, which encompasses numerous open-source software projects. Employing the Genetic algorithm, relevant features were extracted from the datasets to mitigate overfitting risks. These features were then categorized as defective or non-defective using classification techniques such as random forest, decision tree, and artificial neural networks. Evaluation of these techniques included metrics like accuracy, precision, recall, and F-score. Results revealed that random forest outperformed other algorithms in accuracy, precision, and F-score, with average scores of 83.40%, 53.18%, and 52.04%, respectively. Additionally, the neural network demonstrated superior recall, achieving an average score of 60% among the algorithms. Consequently, this system offers valuable support to software developers, aiding them in delivering high-quality software with minimal defects to customers.

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

1. Introduction

A software defect refers to a flaw, fault, or failure within a computer system or program, resulting in unexpected or incorrect behaviors [9]. These discrepancies are typically discovered during software testing and categorized as defects. Utilizing software defect prediction methods proves to be more cost-effective in detecting such issues compared to traditional testing and reviews. Recent studies suggest that the probability of detecting software bugs through prediction models may exceed that of current software review methods [8]. Prompt identification of software bugs allows for the efficient allocation of testing resources and aids in enhancing a system's architectural structure by identifying high-risk segments [8]. Recognizing fault-prone code at each stage of software testing contributes to the development of high-quality software.

Feature selection serves as a crucial technique for managing extensive metric sets by identifying which metrics significantly influence software defect prediction performance. By employing feature selection, redundant and nonindependent attributes are eliminated from the dataset [5]. In this study, the genetic algorithm was employed to extract relevant features from raw datasets. Two primary approaches for constructing a software defect prediction model are supervised learning and unsupervised learning. However, supervised learning necessitates historical data or known results to train the model, presenting challenges.

While a multitude of techniques and learning algorithms are available for selecting software metrics, this paper utilized the Random Forest, Decision Tree, and Artificial Neural Network techniques for the prediction model, utilizing a minimal set of metrics to achieve acceptable results. The performance of these techniques was assessed using accuracy, precision, recall, and F-score. Accuracy reflects the number of correctly classified instances, precision measures the proportion of identified faulty files that are genuinely faulty, and recall evaluates the proportion of faulty files correctly identified as such.

2. Related Work

Menzies et al. [7] utilized OneR, a classification rule algorithm, to evaluate thresholds of single attributes, concluding that OneR is often outperformed by the J48 decision tree. Shafi et al. [10], in addition to OneR, employed another classification rule technique called ZeroR, which was found to be surpassed by OneR. ZeroR predicts the value of the majority class. Arisholm et al. [2, 3] conducted two separate studies using the meta-learners Decorate and AdaBoost along with J48 decision tree. They reported that Decorate outperformed AdaBoost on small datasets and performed comparably well on large datasets; however, they did not specify their definition of small and large datasets.

Grishma and Anjali investigated the root cause for fault prediction by applying clustering techniques and identifying defects occurring in various phases of the SDLC. They utilized the COQUALMO prediction system to predict defects in software and applied various clustering algorithms such as k-means, agglomerative clustering, density-based scan, COBWEB, expectation maximization, and farthest first. The implementation was carried out using the WEKA tool. Ultimately, they concluded that the k-means technique exhibited superior performance compared to other algorithms [4].

Studies in [11, 6] analyzed the applicability of various machine learning methods for fault prediction. Sharma and Chandra [11] expanded their study to incorporate important previous research on each machine learning technique and current trends in software bug prediction using machine learning. This study serves as a foundation or stepping stone for future work in software defect prediction. Agasta and Ramachandran [1] addressed the challenging task of predicting the fault-proneness of program modules when fault labels are unavailable in the software industry. They attempted to predict the fault-proneness of program modules in the absence of fault labels, proposing supervised techniques like the Genetic algorithm-based software defect prediction approach for classification.

Yu et al. [12] developed a model named ConPredictor, utilizing a combination of derived metric sets to improve the prediction of defects in concurrent software programs using deep learning techniques.

3. Methodology

Data Collection

The datasets used in this study were obtained from a publicly available bug prediction dataset, which serves as a repository for defect prediction in numerous open-source software projects. Specifically, the dataset selected for analysis in this paper was the "weighted-ent" dataset, derived from files within each repository. Weighted entropy, referred to as "weighted-ent" in this context, quantifies the information provided by a probabilistic test, considering both the objective and qualitative weights of its elementary events.

Feature Selection

Feature selection, also known as attribute selection, is the process of choosing a subset of relevant features for use in a prediction model. In this study, this process was accomplished using the genetic algorithm, which extracted the features that have the greatest impact on the outputs, namely the number of bugs in a software product. A typical genetic algorithm flowchart is depicted in Figure 1 below.

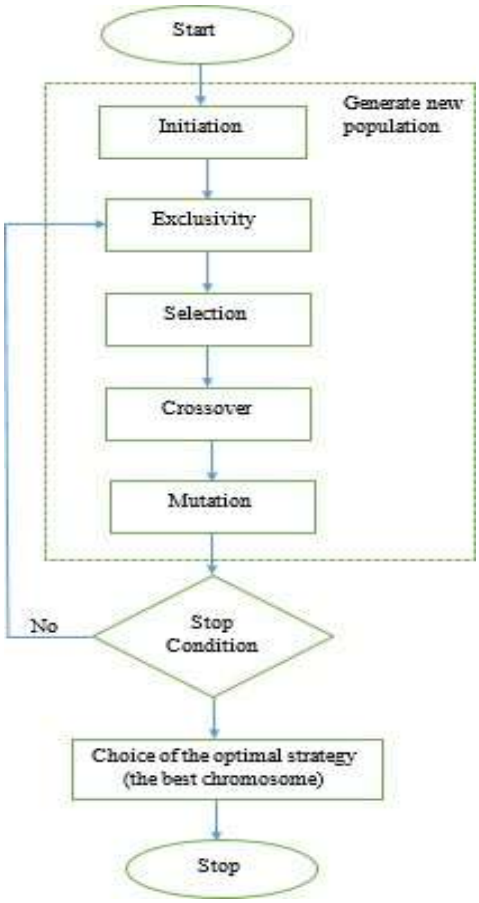

Fig.1:-The Flowchart of a Typical Genetic Algorithm

Machine Learning Algorithms

While machine learning algorithms typically learn to predict outputs based on previous examples, in this paper, the experiment aimed to test the extracted features, employing learning algorithms with their standard settings in the MATLAB environment using the statistical toolkit. The learning algorithms utilized for constructing the defect prediction model in this study include Random Forest (RF), Decision Tree (DT), and Artificial Neural Network (ANN).

Random Forest

The essence of this technique lies in constructing small decision trees with only a few features, making it computationally inexpensive.

However, Random Forest (RF) operates as an ensemble learning algorithm. By concurrently considering weak and small decision trees, these trees can be amalgamated to create a robust and singular learner through majority voting. Moreover, random forests are frequently noted for their high accuracy in learning algorithms. Therefore, the pseudo code employed in this paper is provided below in Algorithm 1. Consequently, generating a larger number of trees using the random forest learning algorithm not only remains a viable option, but these trees also exhibit lower correlation, enhancing the algorithm's overall performance.

Algorithm1: Pseudocode ofRandomForest

Precondition: Atrainingset$S$➞(x1,y1),…,(xn,yn),features$F$,andn umbero ftreesinforest B FunctionRandomForest(S,F)

H➛$Ø$

Fori∈1,…,do (i)➛Abootstrapsample$S$ hi➛ Randomized Treelearn ((i), F)

➛*hi+ Endfor Return$H$

Endfunction

Function Randomized Treelearn (S,F) Ateachnode:

F➛verysmallsubseto fF

Split on best feature in FReturn the learned tree Endfunction

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.

Algorithm2: Pseudocode of Decision Tree Learning

Tree-Learning (TR,Target,Attr)TR: trainingexamples

Target: targetattribute

Attr: setofdescriptiveattributes

{

Generatea Rootnode for thetree.

If TR have the sametargetattributevalue$t$i,

Then Returnthe single-nodetree,thatis. Root, withtarget attribute

=ti

IfAttr= empty (simpl ymeans no expressi veattri butespresent), Then Return thesingle -nodetree, i.e. Root, withmostcommon valueofTargetin TR

Otherwise { Selectattribute A from At trthat classifiesbetter TR dependingonan entropy-basedmeasure Set A the attribute for Root Foreac hlega lvalue of A, i, do { Addabranch below Root, correspondingto A=vi Let TR $v$ibethesubset of TR thathaveA=vi

IfTRviisempty,

Then add a leaf node beneath the branch with target value = most

common value ofTargetinTR Elsebe low thebranch, addthesu btreelearnedbyTreeLearning(TRvi, Target,Attr-{A})

Return (Root) where ti= the value of the target attribute and vi=the value of descriptive attributes

Neural Network

Neural networks(NN) is simply an important tool for classification. There cent 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. Algorithm3 shows the pseudo code of Levenberg-Marquardt used for the defects classification.

Algorithm3:PseudocodeofLevenberg-Marquardt
InitializeWeights;
While not stop Criterion doCalculates(w)foreachpattern
P
$e1=\sum=1(w)(w)$
P
Calculates(w)foreachpattern Repeat
Calculates Δw
P
$e2=\sum=(w+\Delta w)TeP(w+\Delta w)$
P
If$e1\leq e2$then
$\mu=\mu$*QEnd IfUntil$e1<e2$
$\mu=\mu$/Q
w=w+Δw
Endwhile where
(w)is the Jacobian matrix of the error vector (w)ise valuated in w *I*s the specification matrix.
Hence, the parameter μ is increased or decreased at each step.

Classification Stage
In the classification phase, the extracted features underwent classification into defect or non-defect categories using random forest, neural network, and decision tree methods, as outlined in the preceding section. To evaluate the prediction model, a fourfold cross-validation was conducted four times. The dataset was divided into four equal parts, with three parts utilized for the extraction process and training, and the remaining part used for testing. This process was repeated four times to ensure that every part of the dataset served as both training and testing data. Cross-validation was chosen due to the limited number of data, providing an advantage over traditional performance evaluation techniques. By adopting cross-validation, all instances were utilized once for both testing and training, addressing potential bias concerns. This approach generated a total of 16 folds, resulting in a more reliable error estimate.

The performance of the software defect prediction was evaluated using various metrics, including True Positive, False Positive, False Negative, and True Negative prediction outcomes. Subsequently, the defect prediction performance was assessed based on the following criteria: [Include specific criteria as needed].

$$Accuracy=\frac{TruePositive+TrueNegative}{TruePositive+FalsePositive+TrueNegative+FalseNegative} \quad (1)$$

This gives the quantitative relation of predictionthat are correct.

$$Precision=\frac{TruePostive}{TruePositive+FalsePositive} \quad (2)$$

Called holdout method. In the hold out method, one part of the datasets is used for

$$Recall=\frac{TruePosiTrstiveit+iveFalseNe}{gative} \quad (3)$$

$$F-measure=2\times(Precision\times Recall)/Precision+Recall \quad (4)$$

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.

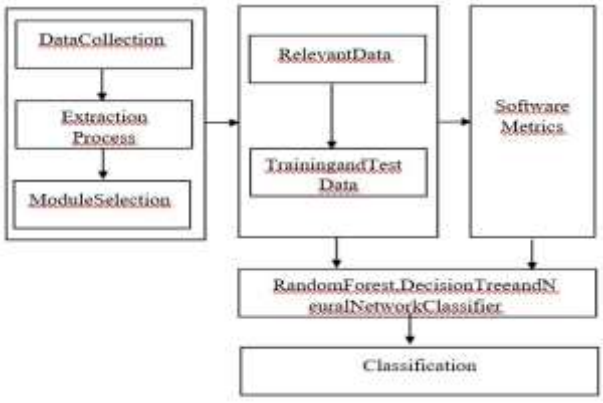

Fig.2:-Proposed Architecture

4. Results and Discussion

The outcomes obtained from the feature set extracted from the raw datasets are compared, followed by the utilization of three techniques for defect prediction with cross-validation. Cross-validation tests are implemented in various manners, but the approach adopted in this paper involves dividing the training data into multiple folds. The classifiers are assessed based on their classification performance, including accuracy, precision, recall, and F-score, on one fold after being trained on the other folds. This process is iterated until all folds contribute to the evaluation. The tables presented below illustrate the performance evaluation of the techniques, focusing on accuracy, precision, recall, and F-score, as outlined in Section 3 of this paper.

Accuracy

Table 1 displays the accuracy of the techniques applied to the dataset utilized in this study. The average accuracy for each learning algorithm was computed and is presented as a percentage. From Table 1, it is evident that the random forest algorithm surpassed the other classifiers. In conclusion, random forest emerges as the top-performing algorithm for the overall datasets assessed in terms of accuracy

Table1:-*The algorithm performance per dataset rated by accuracy*

Datasets	Artificial NeuralNetwork	Random Forest	Decision Tree
ECLIPSEJDTCORE	86.93%	83.92%	75.88%
ECLIPSEPDE UI	83.28%	83.61%	81.81%
EQUINOXFRAMEWORK	70.77%	76.92%	73.85%
LUCENE	91.3%	89.13%	89.86%
AVERAGE	83.07%	83.40%	80.3%

Precision

Precision is another performance metric that assesses the accuracy of the prediction model in classifying faulty files that are genuinely faulty. Table 2 showcases the individual scores of each learning algorithm per dataset, along with the average for each classifier. In summary, random forest emerges as the top algorithm for the overall datasets based on precision, with an average score of 53.18%, followed by ANN with 44.11%.

Table2:-*The algorithm performance per dataset rated by precision*

Datasets	ArtificialNeuralNet work	RandomFo rest	DecisionT ree
ECLIPSEJDTCORE	53.49%	76.74%	4.65%
ECLIPSEPDE UI	31.91%	34.04%	6.38%
EQUINOXFRAMEW ORK	57.69%	76.92%	76.92%
LUCENE	33.33%	25%	0%
AVERAGE	44.11%	53.18%	21.99%

Recall

The Recall metric indicates the proportion of faulty files that the prediction model successfully identifies. As shown in Table 3 below, decision trees estimate the recall for the LUCENE dataset at 0%. Additionally, the artificial neural network emerges as the top algorithm for the overall datasets in terms of recall, exhibiting a notable gap compared to the other classifiers.

Table3:-*The algorithm performance per dataset rated by recall*

Datasets	ArtificialNeuralN etwork	RandomFo rest	Decision Tree
ECLIPSEJDT CORE	79.31%	60%	22.22%
ECLIPSEPDEUI	45.45%	47.06%	21.43%
EQUINOXFRAME WORK	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. Table4 contains the values for all the datasets and the overall average value for Each learning algorithms.

Decision tree value for dataset LUCENEstill stays at 0% fscore. However, the best algorithm isthe random forest for overall rated by f-score with 52.04%.

Table4:-*The algorithm performance per dataset rated by f-score*

Datasets	ArtificialNeuralNetwork	RandomForest	DecisionTree
ECLIPSEJDT CORE	63.89%	67.35%	7.69%
ECLIPSEPDEUI	37.5%	39.5%	9.83%
EQUINOXFRAMEWORK	61.22%	72.73%	70.18%
LUCENE	40%	28.57%	0%
AVERAGE	50.65%	52.04%	21.93%

5. Conclusion

The evolution of the software development process has led to the emergence of various defect prediction techniques and models aimed at enhancing the reliability and quality of software products. This paper conducts experiments on publicly available bug prediction datasets, extracting relevant features from the original sets to prevent overfitting. The results unveil the performance evaluation of the techniques across different datasets. Notably, the random forest algorithm emerges as the best performer overall, as clearly demonstrated in the tables presented in Section IV. Conversely, the utilization of decision tree technique does not yield superior prediction performance, as evidenced by the overall average of each performance measure. Furthermore, the results presented in this paper are compared to other software defect prediction models, demonstrating superior performance in certain cases.

References

1. 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 TelecomJavaSoftware.Inthe18thInternationalSymposiu monSoftwareReliability(ISSRE'07),2007.215–224p.
3. Arisholm,E.,Lionel,C.B.andEivind, B.(2010). A Comprehensive and Systematic Investigation of Methods to Build and Evaluate FaultPrediction models. Journal of SystemsandSoftware,83(1):2–17p.
4. Grishma, B. R., and Anjali, C. (2015).Software root cause prediction using clustering methods: Areview. Communication Technologies (GCCT), 2015 Global Conferenceon. IEEE.
5. Eibe F., Ian H. W., and Mark A. H.(2011). The Data Mining: Practical Machine LearningToolsandMethods, Third Edition (The Morgan Kaufmann Series in Data Management Systems).MorganKaufmann.2011.
6. Malhotra,R.(2014).Comparativeanalysisofstatisticaland machinelearningtechniquesForpredictingbuggymodules .AppliedSoftComputing.2014.21:286-297p.
7. Menzies,T.,Greenwald,J.andFrank, A.(2007).TheDataminingstaticcode features to learnbug predictors.IEEE Trans. Softw. Eng. 2007.33:2–13p.
8. Menzies,T.,Milton,Z.,Turhan,B., Cukic,B.,Jiang,Y.,andBener,A.(2010).Bugpredictionfro mstaticcodeattributes:currentresults,limitations,newtec hniques.TheAutomate SoftwareEngineering.2010.17(4):375–407p.
9. Parameswari,A.(2015).Comparing DataMiningTechniq ues for the So ft ware Defect Prediction.
10. Shafi,S.,Syed,M.H.,Afsah,A.,Malik,J.K.andShafay,S.(2 008).The Software Quality Prediction Techniques: A Comparative Analysis. In 2008
11. 4thInternationalConferenceonEmergingTechnologies.2 008:242–246p.
12. Sharma,D. and Chandra, P.(2018).Software Fault Prediction Using Machine-Learning Techniques". Smart Computing and Informatics. Springer, Singapore.2018:541-549p.
13. [Yu,T.,Wen,W.,Han,X.andHayes, J.(2018).The Conpredictor model: The Concurrency Defect Prediction inReal-World Applications.Inthe International Conference on Software Testing,Verification andValidation.2018:168-179p.