International Journal of Advanced Research in Engineering Technology & Science
ISSN: 2349-2819
Email: editor@ijarets.org Volume-8 Issue-5 May-2021 www.ijarets.org

SOFTWARE DEFECT PREDICTION ANALYSIS TO IMPROVE SOFTWARE RELIABILITY USING MACHINE LEARNING ALGORITHM AND ARTIFICIAL NEURAL NETWORK

Dr. Vikas Jain,

Assistant Professor, SCRIET-DCA,

Ch. Charan Singh University, Meerut, Uttar Pradesh, India

ABSTRACT

One of the most significant characteristics of a program is its level of quality. Over the last several years, there has been a dramatic surge in popularity of software defect prediction, which may have a direct impact on quality. Software that has faulty components may result in a number of negative results, including higher maintenance expenses, missed deadlines, and increased expenditure. The use of machine learning algorithms to forecast software problems is one of the most effective ways in this area. In this inquiry, many machine learning approaches were used. These techniques include artificial neural networks (ANNs), random forests (RFs), random trees (RTs), decision tables (DTs), linear regression (LRs), gaussian processes (GPs), SMOreg, and M5P. The purpose of this discussion is to suggest a new approach for predicting software faults with the intention of avoiding such issues in the future. The defect prediction is based on the past, which helps to ensure accuracy. It was shown that a mix of machine learning methodologies could be more accurate in forecasting software problems than other methods. This was demonstrated by the findings.

Keywords: - *Software Quality, Defect prediction, Machine Learning, learning algorithms.*

INTRODUCTION

Software defect prediction (SDP) is a strategy that enhances software quality while simultaneously reducing the costs associated with software testing. This is accomplished by constructing several classification or categorization models via the use of various machine learning techniques. Many organizations that specialize in software development strive to foresee future problems in order to maintain a level of software quality that is good enough to please consumers and to keep testing expenses to a relatively low level. When it comes to predicting the mistake that will occur during the Software Development Life Cycle (SDP), we use a Machine Learning (ML) technique that makes use of historical data. By using this well-organized method, the objective is to meet the expectations of the client while simultaneously producing software of a high quality at a low cost.

The goal of SDP is to produce dependable software of a high quality while making the most of the resources that are available. From the beginning of the software development lifecycle forward, this will make it possible for programmers to prioritize the use of computer resources. Many software development businesses would want to be able to anticipate software problems in order to reduce the amount of money spent on testing and to maintain a high level of software quality for the purpose of ensuring customer happiness. With the help of SDP, we are able to construct a large number of categorization models by using

a variety of machine learning strategies. This not only enables us to carry out testing more quickly but also enhances the overall quality of the program. To this point, researchers have investigated a wide range of machine learning approaches with the intention of enhancing software quality while simultaneously reducing testing expenses. This is accomplished by predicting when software modules may include errors. The SDP Decision tree, Naïve Bayes (NB), Radial basis function, Support Vector Machine (SVM), K-nearest neighbor, Multi-layer perceptron, and Random Forest (RF) are a few examples of the machine learning techniques that are used in this context.
New developments in machine learning include the combination of feature selection techniques (PCA, etc.) with assembling procedures and other machine learning methods. Within the body of research that is now available, a number of software metrics have been established and used for SDP. A more viable approach would be to deal with the software metrics that are most important and to concentrate on those measures in order to forecast software faults. By evaluating historical data obtained from software repositories, SDP is possible to determine the degree to which software modules are dependable and of a high quality. The literature that is now available on software development process (SDP) makes use of a tremendous number of different software metrics. In order to develop SDP models, software measurements are used in combination with data gathered from pre-existing systems or software projects that are analogous to those previously developed.

In order to make an accurate forecast of software bugs, it would be more practical to investigate and focus on the software metrics that are considered to be the most important. Having said that, the dataset that was used for the paper has been available to the general public on the Promise Repository ever since the year 2005. Particulars on a wide range of applications that have been investigated by NASA are included in this document. The output is classified using K-means clustering for the purpose of this research. This classification takes place after feature selection (FS) and dataset pre-processing have been completed. Following that, we use machine learning techniques such as support vector machines (SVM), neural networks (NB), and random forests (RF), as well as PSO. A method known as an ensemble is then used in order to combine the data. Lastly, but certainly not least, each and every machine learning model is reviewed and compared to previous studies. A number of different metrics, including precision, accuracy, recall, F-measure, confusion matrix, and performance error metrics, are used in order to evaluate the performance of the models.

OBJECTIVE

1. The paper examines software fault prediction using machine learning methods.
2. Early defect prediction helps developers prioritize testing on error-prone modules, enhancing software quality.

METHODOLOGY

In order to accomplish the objective of the comparative analysis investigation, a variety of algorithms were investigated. The list contains the machine learning algorithms that are generally considered to be the most popular and commonly utilized. The following is a list of the algorithms that were chosen, along with short explanations of each one:

Artificial neural network

The term "Artificial Neural Network" (ANN) is used to define artificial neural networks, which are models that are patterned after their biological counterparts. Neural networks are used in the fields of mathematics and computers. Its fundamental components consist of a network of artificial neurons that are communicated with one another and work together to perform data analysis. It is able to demonstrate its flexibility by undergoing structural adjustments throughout its learning phase. These modifications are a response to both internal and external information flows. In some respects, it is analogous to the functioning of the biological brain. The components that comprise an artificial neural network (ANN) are the processing units, also known as nodes, and the connections, also known as links, that connect them. A network of neurons that are linked to one another and have the ability to transmit and receive information from other neurons is known as a neural network. A description of the operation of an ANN is provided below. Initially, the nodes that are located in the input layer are responsible for introducing the values of the variables into the neural network. Weights are assigned to the connections that connect the nodes in the network. These neural networks have the capacity to learn and adapt due to the fact that the numerical weights are updated based on the experience of the network. The process of traversing the network and its nodes is what causes the values of the variables to be determined. One of the factors that influences the result of the variable value is the weight that each link carries.

Random forest

Learning supervised classification and regression is accomplished via the use of a method known as Random Forest (RF). However, rather of utilizing just one decision tree, it builds a forest of decision trees, and then the committee votes on each of them to determine which one is the best option. A combination of bagging and random selection are the two machine learning approaches that are used in RF. By allowing the trees to vote on the predictions using a bootstrap sample of the training data, bagging is able to create predictions effectively. A collection of characteristics at each node is divided up in the most effective manner possible via the use of random feature selection. It's possible that the outcomes will be favorable if the inputs and qualities are fully arbitrary. Through the process of splitting at each node, RF constructs a tree using a subset of characteristics that is chosen at random. One of the most notable features of RF is that it results in a faster performance when compared to boosting and bagging. In spite of outliers and background noise, RF is resistant to both. Reportedly, RF performs better than the most advanced classification algorithms currently available. The usage of radio frequency (RF) is beneficial for large data collections. If there is a lack of data, the RF technique may be able to fill it in.

Random Tree Random

Under a certain set of circumstances, a tree may be used to graphically depict each and every response that is technically conceivable. We refer to it as a tree because, at its foundation, it is a decision tree that has a root and several branches that yield various data sets. Both Adele Cutler and Leo Breiman were the ones who came up with the concept of random trees. The method is capable of solving issues involving classification as well as regression. Random trees are a subset of a wider group of tree predictors that are collectively referred to as a forest. After being provided with an input feature vector, the random trees classifier will first conduct a poll of all of the forest trees to determine their categorization suggestions. After this, it will then provide the label for the category that obtained the most number of "votes." Within the context of a regression analysis, the output of the classifier is the average of the outputs from each individual tree in the forest.

Random Trees is a technique of machine learning that may be created by combining the characteristics of random forests with those of single model trees without any further steps being required. Each leaf of a model tree defines a local subspace that has been optimized using a linear model. Model trees are examples of decision trees. It is possible to randomize trees in two different ways, and one of these methods, known as Random Forests, is known to significantly boost the effectiveness of individual decision trees.

RESULTS AND DISCUSSION

For the purpose of this experiment, the ISTQB V 2.2 platform was used to apply the eight machine learning algorithms that were discussed before in order to foresee software problems. Using crossvalidation (10-fold) and percentage split (55%, 65%, 75%, and 85%), the dataset that was analyzed was put through a series of tests. The results of these tests were summarized below. The statistical characteristics that were investigated in order to determine how accurate the classifiers were in predicting software failures in the dataset that was chosen are shown in Table 1.

The LR technique and the SMOreg approach both produced high values for the correlation coefficient (R2) in the K1 dataset, as can be shown in Table II. Both of these methods' results are presented here. On the other hand, when it came to the other criteria, the SMOreg technique demonstrated the highest level of predictability. It was clear from this that the SMOreg classifier performed better than the other seven machine learning algorithms. The artificial neural network (ANN) algorithm, on the other hand, was deemed to have the poorest performance. Furthermore, the algorithm was unable to accurately forecast the values for both RS and RMSE, which were greater than 100%. This was owing to the fact that the numbers were constant throughout all of the input data.

Table. 1 results obtained from a 10-fold cross validation test of the ML algorithms

Classifier	AUC	Measures	Recall	Precession	Accuracy
ANN	0.1584	6.9869	14.7296	115.6751	134.899
R a ndo m Forest (RF)	0.4995	4.4208	9.3905	73.1899	86.0011
Random Tree (RT)	0.1317	5.5088	12.2059	91.2029	111.7856
Decision Table (DT)	0.2663	4.9232	11.2294	81.5082	102.8431
Linear regression (LR)	0.7594	6.5889	I I .1759	109.0851	102.3526
Gaussian Processes (GP)	0.6794	4.6329	7.9437	76.7025	72.7514
SMOreg	0.7383	4.3159	7.4378	73.1085	68.1179
M5P	0.6989	4.3618	7.7834	72.2135	71.2 1 :.

The CFS method that was used in the selection of the subset of qualities is shown in Table 1. After considering eight different measurements, we decided to go with DIT, CBO, RFC, NPM, and LOC. Accuracy, precision, recall, F-measure, and area under the curve (AUC) are the measurements that are shown in Table 3, along with their respective percentages and standard deviations. Using the projected model, we applied it to 274 different classes. As a result, the area under the curve (AUC) is high (0.81), and the model has a good level of accuracy (74.24%), recall (79%), and F-measure (75%).

In Figure 1, the standard deviation of the accuracy measure as well as the percentage of accuracy are shown separately. A visual representation of the value of additional metrics in relation to their standard deviation (precision, recall, and F-measure) is comparable to the representations shown in Figures 1 and 2. In light of these findings, the hypothesis that machine learning algorithms may be beneficial in the process of constructing prediction models for flawed courses is given more weight. It is necessary to do more research of a similar kind in order to validate the correctness of the model that was projected.

Table 2. Measuring Performance via analyzing standard deviation

Method	Measuring Performance	Standard deviation
Accuracy	74.24	699
Precession	72	0.08
Recall	79	0.10
F-Measures	75	0.07
AUC	0.81	0.07

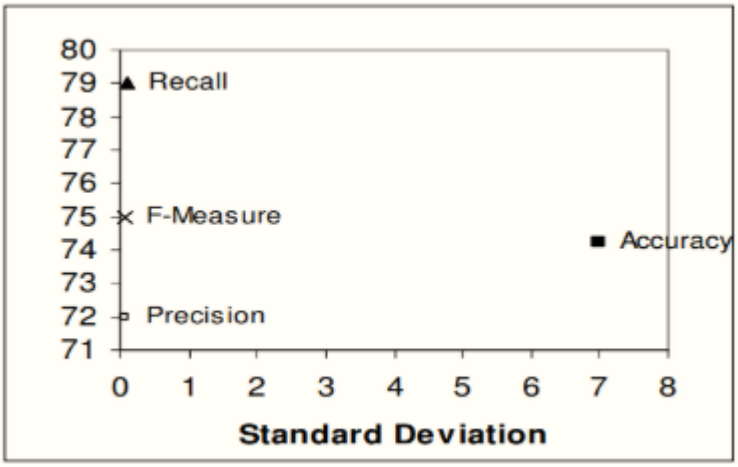

Fig 1. Efficiency metrics compared to dispersion

Table 3. The outcomes of statistical analyses conducted on ML algorithms employing the % split mode

Classifier	60%	70%	80%	90%

ANN	6.3554	5.097	5.7436	0.8393
Random Forest (RF)	3.9995	3.604	1.9424	0.2115
Random Tree (RT)	4.3654	4.2973	1.3014	0.0303
Decision Table (DT)	6.2844	5.5632	2.5614	2.5614
Linear regression (LR)	5.9445	5.5295	4.2484	2.6998
Gaussian Processes (GP)	5.1449	5.0989	3.4081	1.9759
SMOreg	4.434	3.9551	2.7383	1.1206
M5P	4.0811	5.9991	3.0258	0.5304

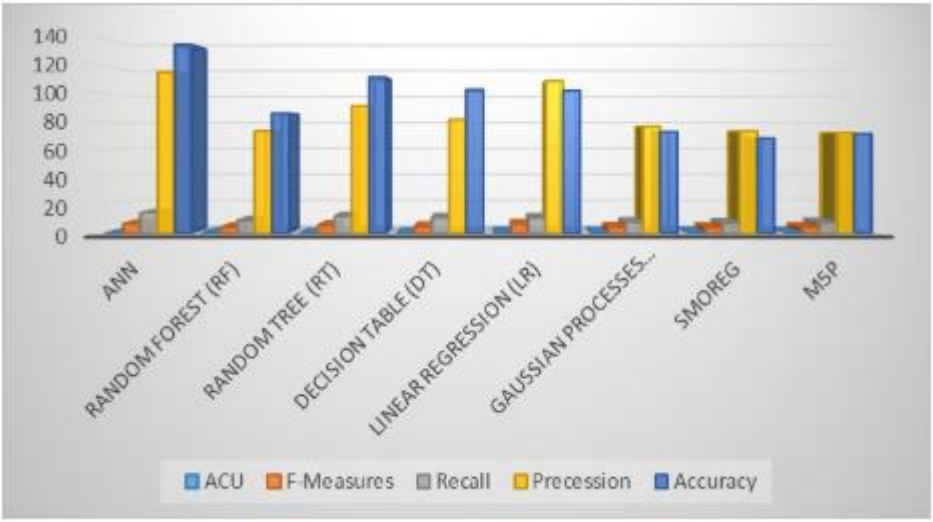

Fig. 2. The algorithm's standard deviation error

CONCLUSION

In this study, seven different machine learning algorithms are compared using software quality metrics such as accuracy, precision, recall, and F-measure. The purpose of this study is to determine which category of algorithms has the greatest capacity to predict software defects prior to the release of software systems to the real environment or to the delivery of software to customers. The following four datasets, which were donated by NASA, are used throughout the course of this experimental investigation: PC1, CM1, KC1, and

KC2. These datasets were obtained from the public PROMISE repository, which is the source. When it comes to defect prediction, tree-structured classifiers, also known as ensemble learners such as Random Forests and Bagging, perform much better than their rivals, as shown by the findings of this experimental inquiry. The ability of Bagging to accurately predict the number of software defects is very impressive. With regard to all datasets, the overall accuracy ranges from 83.7-94.1%, the precision ranges from 81.3% to 93.1%, the recall ranges from 83.7% to 94.1 %, and the FMeasure of Bagging ranges from 82.4 to 92.8 %.When compared to other machine learning algorithms used on the PC1 dataset, bagging yields the most favorable outcomes across the board for all quality criteria.

REFERENCE

1. Victor R Basili, Lionel C. Briand, and Walcelio L Melo. ´ A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10):751–761, 1996.

2. Evren Ceylan, F Onur Kutlubay, and Ayse B Bener. Software defect identification using machine learning techniques. In 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), pages 240–247. IEEE, 2006.

3. Karim O Elish and Mahmoud O Elish. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 81(5):649– 660, 2008.

4. Norman Fenton, Paul Krause, and Martin Neil. Software measurement: Uncertainty and causal modeling. IEEE software, 19(4):116–122, 2002.

5. Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In 15th International Symposium on Software Reliability Engineering, pages 417–428. IEEE, 2004.

6. Taghi M Khoshgoftaar, Edward B Allen, and Jianyu Deng. Using regression trees to classify fault-prone software modules. IEEE Transactions on reliability, 51(4):455–462, 2002.

7. Taghi M Khoshgoftaar, Edward B Allen, John P Hudepohl, and Stephen J Aud. Application of neural networks to software quality modeling of a very large telecommunications system. IEEE Transactions on Neural Networks, 8(4):902–909, 1997.

8. Sunghun Kim, Hongyu Zhang, Rongxin Wu, and Liang Gong. Dealing with noise in defect prediction. In 2011 33rd International Conference on Software Engineering (ICSE), pages 481–490. IEEE, 2011.

9. Yan Ma, Lan Guo, and Bojan Cukic. A statistical framework for the prediction of fault-proneness. In Advances in Machine Learning Applications in Software Engineering, pages 237–263. IGI Global, 2007.

10. Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27:504–518, 2015.

11. Jinsheng Ren, Ke Qin, Ying Ma, and Guangchun Luo. On software defect prediction using machine learning. Journal of Applied Mathematics, 2014, 2014.

12. 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.

13. Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability, 62(2):434–443, 2013.

14. Robert Andrew Weaver. The safety of software: Constructing and assuring arguments. University of York, Department of Computer Science, 2003.

15. N. Seliya, T. M. Khoshgoftaar, and J. van Hulse, "Predicting faults in high assurance software," in Proceedings of IEEE International Symposium on High Assurance Systems Engineering, 2010, pp. 26–34. doi: 10.1109/HASE.2010.29.

16. S. Chatterjee and A. Roy, "Web software fault prediction under fuzzy environment using MODULO-M multivariate overlapping fuzzy clustering algorithm and newly proposed revised prediction algorithm," Applied Soft Computing Journal, vol. 22, pp. 372–396, 2014, doi: 10.1016/j.asoc.2014.03.030.

17. G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," Vietnam Journal of Computer Science, vol. 1, no. 2, pp. 79–95, May 2014, doi: 10.1007/s40595-013-0008-z.

18. C. Catal, "A comparison of semi-supervised classification approaches for software defect prediction," Journal of Intelligent Systems, vol. 23, no. 1, pp. 75–82, Jan. 2014, doi: 10.1515/jisys-2013-0030.

19. Z. Li, X. Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "On the Multiple Sources and Privacy Preservation Issues for Heterogeneous Defect Prediction," IEEE Transactions on Software Engineering, vol. 45, no. 4, pp. 391–411, Apr. 2019, doi: 10.1109/TSE.2017.2780222.

20. J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: A survey," Knowl Based Syst, vol. 80, pp. 14–23, May 2015, doi: 10.1016/j.knosys.2015.01.010.

21. D. Sharma and P. Chandra, "Software fault prediction using machine-learning techniques," in Smart Innovation, Systems and Technologies, 2018, vol. 78, pp. 541–549. doi: 10.1007/978-981-10-5547-8_56.

22. R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing Journal, vol. 27, pp. 504–518, 2015, doi: 10.1016/j.asoc.2014.11.023

23. A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software Bug Prediction using machine learning approach," International Journal of Advanced Computer Science and Applications, vol. 9, no. 2, pp. 78–83, 2018, doi: 10.14569/IJACSA.2018.090212.

24. R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," Applied Soft Computing Journal, vol. 21, pp. 286–297, 2014, doi: 10.1016/j.asoc.2014.03.032.

25. S. K. Niranjan, Chirala Engineering College, IEEE Computational Intelligence Society, and Institute of Electrical and Electronics Engineers, Proceedings of the 2017 International Conference on Big Data Analytics and Computational Intelligence : ICBDACI 2017 : 23-25 March 2017, Chirala Engineering College, Chirala, Andhra Pradesh, India.