An Analysis of Software Bug Prediction Techniques

Jai Bhagwan
Assistant Professor
Department of Computer Science & Engineering,
Guru Jambheshwar University of Science & Technology, Hisar, India

Abstract: In the technological era, the demand of software is increasing unconditionally. Various social media platforms have been designed for entertainment and business purposes. Software development is not an easy task because we need to keep in mind the budget, development time, efficiency and other parameters too. To deliver successful software, the developers need to go through various phases of software development. To reduce the development efforts and cost of software, software bug detection may play a vital role. In past, various machine learning models have been designed to predict the software fault at any level of software development life cycle. In this research, various techniques have been examined and a comparative analysis has been presented.

IndexTerms **– Software Quality, Bug Prediction, Software Fault, Object-Oriented, ML (Machine Learning), ML Techniques.**

I. INTRODUCTION

Computer systems have received a great deal of attention and research because of the critical challenges of software quality and reliability. In order to prevent poor software design, which would result in a low-quality output, it is highly desired for project managers to have some early understanding about the target software product's quality. Early on, it could be identified and prevented [4]. Defective software modules can be found and the number of defects that occur during system operations can be decreased with targeted software quality inspection and improvement. A flaw in an executable product that results in system problems while it is being used is called a software fault [10].

Early detection of defective modules contributes to increased software process control, lower work required to rectify defects, lower costs, and increased software reliability. Planning resources for testing is made easier by identifying modules that are prone to defects, as managing resources while screening is seen as a challenging effort. Defect prediction techniques, which aid in resource and test plan development and software project control, are used for this process of detection [1]. Predicting software errors is a crucial task that must be completed in order to lower maintenance costs and increase software quality prior to system deployment. Early defect identification could result in fast defect rectification and the release of sustainable software [2]. Currently, the object-oriented paradigm forms the basis of most software development. Software metrics are the most effective means of evaluating object-oriented software quality. Researchers and practitioners have developed a variety of criteria to assess software quality. These measurements support the validation of software quality attributes including effort and error proneness [5]. C. Catal (2012) said, one of the quality assurance tasks in Software Quality Engineering, along with formal verification, fault tolerance, inspection, and testing, is software fault prediction. The prediction model is constructed using software metrics and fault data from an earlier software version [7].

In this paper, the literature of various software fault prediction techniques have been presented in detail. It has also been presented various techniques, their limitations and practical effects. Likewise, the paper is systematized as: Section II represents the related work. In section III, comparative analysis is presented. Finally, section IV summaries this research work.

II. RELATED WORK

A deep study of various software bug detection techniques is presented in this section. In [1], a model for predicting software bugs is introduced. The researchers employed an empirical investigation to examine the impact of utilizing two distinct solution algorithms and three distinct similarity functions on our CBR system's prediction accuracy. It is also investigated how changing the quantity of nearest neighbor cases affects performance accuracy. Additionally, the advantages of employing metric-selection processes for the CBR system are assessed. We base our study on case studies of a sizable legacy telecommunications system. The best fault prediction was found to be produced by the CBR system when it used the inverse distance weighted solution technique and the Mahalanobis-distance similarity function. Furthermore, the CBR models outperform models that rely on multiple linear regressions. Seven kinds of machine learning approaches and 64 primary researches have been identified by the experts for this research [2]. The outcomes demonstrate how well machine learning algorithms can predict whether a module or class will have faults or not. The methods that estimate software fault proneness using machine learning approaches perform better than the conventional statistical models.

In [3], the scientists' method for predicting fault densities in this work uses a decision tree learner applied to evolution data taken from the Mozilla open source web browser project. The evolution data consists of various defect, modification, and source code measurements that were calculated from seven current Mozilla releases. We also include the change correlation, which quantifies the quantity of change-dependencies among source files, in our analysis of modification metrics. Finding underlying rules that are simple for humans to understand was the primary motivation behind selecting decision tree learners over other models, such neural nets. The authors set up several experiments to test popular theories about flaws in software entities in order to determine these criteria. The experiments produced noteworthy findings. The suggested model [4] combines the best features of both Artificial Neural Networks (ANN) and Fuzzy Logic (FL) while removing their drawbacks. The model shows some positive characteristics, including the ability to handle objective data gathered during the software development process and knowledge/experiences gained from experts or from projects similar to the one being worked on, which is the primary information available during the early stages of a software development process. By using this technique, design flaws can be found and costly

rework can be avoided by using early software quality prediction. The suggested model can be practically applied and is easy to train, according to the experimental results.

The study [5] uses Chidamber and Kemerer (CK) metrics to investigate the use of logistic regression, artificial neural network, and linear regression techniques for software fault prediction. In this case, the CK metric suite is regarded as an independent variable and fault as a dependent variable. To find errors related to the classes, machine learning techniques like neural networks and statistical techniques like logistic and linear regression are being used. The Apache Integration Framework (AIF) version 1.6, case study was used to apply the comparative approach. In addition to highlighting the importance of the weighted method per class metric for fault categorization, the research demonstrates that, when compared to the other three neural network models, the hybrid approach of the radial basis function network produced a better fault prediction rate. By giving current faults with the Bayesian Interference a means of recourse, the fault prediction model aids in software development. This work [6] uses a combined Bayesian inference and Logistic Regression model design to aid all fault prediction strategies. Additionally, it is stated as true that a probabilistic approach to the faults that are reported and found for the next release can be represented by a Bayesian inference graph. The purpose of Bayesian inference for probabilistic reliability analysis is to assess risk-related data. These results imply a connection between object-oriented metrics and defective classes. This study serves as an example of a software performance evaluation method.

Based on performance evaluation criteria, the scientist [7] examined 85 fault prediction articles in this study and divided the metrics into two major categories. Only assessment metrics are looked at because evaluation techniques like cross validation and stratification of samples are outside the purview of this paper. This study demonstrates that several assessment factors have been employed by academics up to this point for the prediction of software faults, and more research on metrics to assess performance for imbalanced datasets is warranted. An entire industrial software system is the subject of the case study [8] that is being presented. The data from hypothesis testing and model selection methods were found to be inconsistent with the count models' ability to predict outcomes. Furthermore, the results of the comparison study based on one of the criteria did not correspond with those of another. Nonetheless, a count model's performance remains comparable with both the fit and test data sets with regard to a particular criterion. This guarantees that the model will produce a good prediction using the same criterion if a fitted model is judged fair based on a certain criterion.

The cost curve analysis of fault prediction model was presented and discussed by the authors in this study [9]. When a software module is categorized as fault-prone, verification processes must be applied, which raises the cost of development. Declaring a module to be fault-free when it isn't increases the likelihood of system failure and its related expenses. We find that software quality does not always increase with the prediction of fault-prone components through the examination of 16 projects from public repositories. The addition of misclassification cost in model assessment could suggest that even the "best" models perform no better than triangular classification, which uses cost curves as a typical technique for evaluating the performance of software quality models. The research [10] examines the effects of applying two alternative solution methods and three distinct similarity functions on our CBR system's prediction accuracy empirically. Additionally, the impact of altering the quantity of closest neighbor cases upon the accuracy of performance is investigated. Also assessed are the advantages of applying metric-selection processes for our CBR system. Our analysis is based on investigations of a sizable legacy telecom network. The CBR system that employed the inverse distance weighted solution technique and the Mahalanobis distance similarity function produced the best fault prediction, according to observations. Furthermore, the CBR models outperform models that use multiple-linear regression.

The researchers [12] examine a neural network approach in this paper to create models that forecast the quantity of errors in software modules. Using the data gathered during the construction of two commercial software systems, the scientists employ this process to create neural network models. We use a variety of linear regression techniques to create regression models using the exact same set of data after creating neural network models. The neural network strategy yielded improved predictive models for the investigated data sets in terms of both predictive quality and quality of fit. The scientists [13] take into account the cross-company forecasting of defects scenario in this study, in which the target and source data are obtained from various companies. The authors attempt to use the transfer learning approach to develop a prediction model that is both extremely efficient and quick in order to utilize cross-company data. The results of the trials on data sets from various organizations are presented, together with a theoretical analysis of the comparative methodologies. It shows that compared to state-of-the-art techniques, TNB is more accurate in terms of the area under the receiver operating characteristic curve in less runtime.

III. Comparative Analysis

A comparative analysis of various software bug detection techniques is shown below in Table 1.

Table 1. Comparative Analysis of various Techniques

Ref. No.	Methods Used	Limitations	Practical Implications
[1]	Pre-preprocessing, Equifrequency bins for results, Over sampling and under sampling.	Not Mentioned	Enhanced Naive Bayes classifier performance for predicting software defects. Pre-processing method to investigate relationships between faults and software metrics. Modules with a higher likelihood of defects were predicted when bins were labeled with missing data. Potential advantages for the software sector if the recommended strategy is

			implemented. Examining several binning strategies and varying the number of bins for every variable.
[2]	Data extraction process and data synthesis process. AUC, RF, NB, MLP, C4.5, and SVM.	Out of 64 primary research, only 19 compare LR and ML methods. Various experimental setups are employed to compare machine learning methods. Potential omission of a relevant study. Excluding research studies that have not been published. Presumption that every study is objective.	ML approaches can predict software error proneness with a reasonable degree of accuracy. ML models fared better in software fault prediction than logistic regression methods. In every study, the Random Forest approach fared better than other ML models. There has to be more research done contrasting ML methods with logistic regression. Predicting software faults has never been done using some ML approaches.
[3]	Decision tree learner, Data mining and ML techniques.	Metric relationships are intricate and challenging to comprehend intuitively. Understanding classifier decisions is impossible because to the vast volume of data. Insufficient direction to proactively enhance the quality of software in projects.	Give resources a proactive allocation of tools to raise the caliber of software. Estimate the number of defects in software modules. Recognize the causes of the problems in software entities. Utilize decision tree learners to anticipate defects from a range of input data. Dismiss theories concerning change couplings and size measurements. Subsequent research will integrate precise measurements of alterations and flaws.
[4]	Fuzzy neural network.	Not Mentioned	The proposed approach can accommodate several aspects that affect the quality of software. The model is capable of handling both objective data and expert knowledge and experience. It becomes possible to anticipate software quality in advance. Aids in locating design flaws and prevents costly rework. For small- to medium-sized problems, the model is easily trainable.
[5]	linear regression, logistic regression, neural network	Not Mentioned	Prediction models are a useful tool for system analysts to classify faults. For defect prediction, statistical and machine learning techniques can be used. When predicting faults, the WMC metric is helpful. Better fault prediction was obtained using

			the hybrid technique of RBFN.
[6]	Logistic regression model and various metrics, Bayesian Inference Model.	Not Mentioned	To forecast the occurrence of a failure, design Bayesian inference for each of the metrics. To determine the posterior probability of errors, apply Bayesian inference. Reception operational characteristic curves can be used to determine software metric threshold values. Establish a process for allocating resources to the development of dependable software.
[7]	Examination of different evaluation parameters used for software fault prediction.	Further study is required on metrics for performance assessment related to software fault prediction. Misclassification cost ratio calculation and cost curve application are challenging. Inherent variations in software failure forecasting in contrast to other difficulties with imbalanced datasets.	Research results cannot be compared because of disparate evaluation criteria. AUC, or area under the ROC curve, is a widely used statistic in fault prediction. One can anticipate the number of defects using R2 and AAE ARE. Further research is required on performance evaluation criteria related to fault prediction. For ease of comparison, commonly used measures for performance evaluation should be employed.
[8]	Newton-Raphson iteration technique, EM algorithm	Comparable outcomes were obtained by other mixing models and count models. A few mixture models weren't appropriate for the software platform under investigation. Other mixing models' modeling outcomes are not shown.	Software quality modeling can benefit from the use of count models such as Poisson regression. The hurdle negative binomial model and zero-inflated negative binomial model both function effectively. Models produced by various choice factors may differ. Regarding the same metric, the best fitted model produces the best prediction accuracy. Further case studies and research on noise and outliers are planned for the future.
[9]	Cost curve analysis	There is no agreement on the process for choosing the "best model" for a certain project. Predicting which components are likely to have errors does not always improve software quality. The performance of the "best" models is not better than that of trivial categorization.	Models of software quality have real advantages for software developers. A defect prediction model's evaluation needs to take the business environment into account. Evaluation includes the cost of component misclassification through the use of cost curve analysis. Cost analysis for misclassification is project-specific. Fault prediction models are especially useful for high-risk projects. Various risk projects benefit from the use

			of different modeling algorithms.
[10]	Case-based reasoning (CBR), City Block, Euclidean, and Mahalanobis, nearest-neighbor, cross – validation, average and inverse distance weighted average.	Not Mentioned	For software components that are still being developed, the CBR system can offer timely fault forecasts. Forecasts can assist in allocating resources for increased dependability. The impacts of various functions of similarity and solution techniques were investigated empirically. The CBR system's accuracy was increased via the metric selection process. Scalability and automatic estimation interpretation are features of the CBR system.
[11]	Regression tree modeling technique, Fault density technique	Not Mentioned	Analyzing software data effectively can be achieved by regression tree modeling. It facilitates the identification of problematic modules and the comprehension of links between data elements. The method can deal with missing values and is reliable and stable. It offers a reliable method for estimating program quality. Its misclassification rate and deviation are lower than those of the fault density procedure.
[12]	Neural network methodology, Multiple linear regression methods.	When there are significant assumptions broken, multiple regression models get unstable. For modeling software defects, neural network methods are better suitable than regression models. The prediction models generated by neural network methods were superior in terms of fit and quality.	Regression models are inferior to neural network models in terms of predictive quality. Data on software complexity is better suited for modeling using neural network models. Regression techniques can be effectively substituted by neural network modeling. To determine the usefulness of neural network representations in software engineering applications, more investigation is needed.
[13]	Transfer Naive Bayes	Not Mentioned	With cross-company data, the Transfer Naive Bayes (TNB) method may forecast software problems. On test data sets, TNB has excellent efficiency and runtime cost characteristic. TNB can direct the best resource allocation plans to lower the cost of software testing. TNB has the potential to improve software testing process efficacy.

IV. CONCLUSION

The need for software is always growing in the age of technology. Different social media networks have been created with both business and fun in mind. The budget, development time, efficiency, and other factors must all be considered, making software development an exhausting task. The developers must go through several software development phases in order to produce successful software. Software bug identification may be essential to lowering software development costs and effort. Multiple models based on machine learning have been developed in the past to forecast software errors at any stage of the software development life cycle.

This study has looked at a number of approaches and provided a comparative analysis. The limitations of the various techniques encourage us to develop a new machine learning technique for software bug prediction in future.

References

[1] Rana, Z. A., Mian, M., & Shamail, S. 2015. Improving Recall of Software Defect Prediction Models using Association Mining. Knowledge-Based Systems, 90: 1-13.

[2] Malhotra, R. 2015. A Systematic Review of Machine Learning Techniques for Software Fault Prediction. Applied Soft Computing, 27: 504–518.

[3] Knab, P., Pinzger, M., & Bernstein, A. 2006. Predicting Defect Densities in Source Code Files with Decision Tree Learners. MSR, ACM, 119-125.

[4] Yang, B., Yao, L., & Huang, H. 2007. Early Software Quality Prediction Based on a Fuzzy Neural Network Model. 3rd International Conference on Natural Computation, IEEE.

[5] Suresh, Y., Kumar, L., & Rath, S. K. 2014. Statistical and Machine Learning Methods for Software Fault Prediction using CK Metric Suite: A Comparative Analysis. ISRN Software Engineering, 2014: 1–15.

[6] Kapila, H., & Singh, S. 2013. Analysis of CK Metrics to Predict Software Fault-Proneness using Bayesian Inference. International Journal of Computer Applications, 74(2): 1–4.

[7] Catal, C. 2012. Performance Evaluation Metrics for Software Fault Prediction Studies. Acta Polytechnica Hungarica, 9(4): 193-206.

[8] Gao, K., & Khoshgoftaar, T. M. 2007. A Comprehensive Empirical Study of Count Models for Software Fault Prediction. IEEE Transactions on Reliability, 56(2): 223–236.

[9] Yue Jiang, Bojan Cukic, Tim Menzies, 2008. Cost Curve Evaluation of Fault Prediction Models. International Symposium on Software Reliability Engineering, 197-206.

[10] Khoshgoftaar, T. M., Seliya, N., & Sundaresh, N. 2006. An Empirical Study of Predicting Software Faults with Case-Based Reasoning. Software Quality Journal, 14(2): 85–111.

[11] Gokhale, S. S., & Lyu, M. R. 1997. Regression Tree Modelling for The Prediction of Software Quality. Third ISSAT International Conference on Reliability. 1-6.

[12] Khoshgoftaar, T. M., Pandya, A. S., & Lanning, D. L. 1995. Application of Neural Networks for Predicting Program Faults. Annals of Software Engineering, 1(1): 141–154.

[13] Ma, Y., Luo, G., Zeng, X., & Chen, A. 2012. Transfer Learning for Cross-Company Software Defect Prediction. Information and Software Technology, 54(3): 248–256.