ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 7, Issue 5, May 2018

CLASSIFICATION OF SOFTWARE RELIABILITY MODELS USING NAÏVE BYES TECHNIQUE

Nishi, Dr. Dinesh Kumar
Maharishi Dayanand University, Rohtak

Abstract

Creating or making software with high quality has become challenging these days with the fact that size and complexity of the developed software is high. Predicting the quality of software in early phases helps to reduce testing resources. Various statistical and machine learning techniques are used for prediction of the quality of the software. In this paper, six machine learning models have been used for software quality prediction on five open source software. Varieties of metrics have been evaluated for the software includingNaïve bayes and NHPP model.

Keywords: NHPP model, Software reliability,Naïve byes prediction, interlocking etc.

I. INTRODUCTION

Different Software metrics which are used to find the software faults before the process of testing are class level, method-level metrics etc. Methods used for finding the software faults are machine learning, statistical method and expert estimation. But machine learning method is best method for finding the software faults because all the work is done by Naïve bayes In context of software engineering, software quality refers to software functional quality and software structural quality. Software functional quality reflects functional requirements whereas structural quality highlights non-functional requirements. Software metrics focus on the quality aspect of the product, process and project. In this paper the main emphasis is on software product. The objective of software product quality engineering is to achieve the required quality of the product through the definition of quality requirements and their implementation, measurement of appropriate quality attributes and evaluation of the resulting quality. Software quality measurement [15] is about quantifying to what extent a system or software possesses desirable characteristics namely Reliability, Efficiency, Security, Maintainability and (adequate) Size. This can be performed through qualitative or quantitative means or a mix of both. In both cases, for each desirable characteristic, there are a set of measurable attributes like Application Architecture Standards, Coding Practices, Complexity, Documentation, Portability and Technical & Functional volumes. The existence of these attributes in a piece of software or system tends to be correlated and associated with this characteristic.

Defects in system software lead to foremost difficulty in the software. A lot of software systems are sent to the clients with unnecessary defects. Testing is one of the most important approaches for finding the defect prone parts of the system. Software quality can be measured with various attributes like fault thickness, normalized rework, reusability, portability and maintainability etc.

II. Review Of Literature

Norman E.Fenton [5] describes that software metrics and statistical models have been developed to find the number of defects in the software system and the majority of the prediction models use size and complexity metrics to find faults.To find a single complexity metrics, a large complex multivariate statistical model has been introduced. The limitations are that by using size and complexity metrics, accessible models cannot find the faults successfully. Atchara Mahaweer awatetal describes that software fault prediction technique is the superlative approach for finding the software faults to enhance the quality and reliability of the software [10]. They used Method level metrics. The concept of neural networks is importantly used. Neural networks provide an important technique called Radial Basis Function (RBF) [10].The main function of RBF is to find the faults in the software and provide better accuracy. The object Oriented software systems are used for predicting the number of faults in the software [8, 10]. Inheritance and Polymorphism are the important features of Object Oriented systems. For finding the software faults, a large amount of data is required. Two important networks are used:- Multilayer Perceptron (MLP) is used for ruling the defective modules whereas Radial Basis functions Network are used to classify the defects according to a number of different types of faults 8 .Xing et al. [16] describes the importance of Support Vector Machine (SVM) model. This SVM model is used when only a little amount of data is obtainable. Data categorization is a significant use of SVM technique. SVM provides better accuracy than other techniques for the prediction of quality of the software but in public datasets, the performance of SVM is poor. The early lifecycle metrics play significant role in the software project management [7]. Early lifecycle metrics can be used to identify faulty modules. Method level metrics are widely used for software fault prediction. The authors used three NASA projects are: PC1, CM1 and JM1.After comparison of these different projects, they concluded that the requirement metrics have significant role in for software fault prediction.In another paper [4] the authors illustratedthe potential of SVM for finding the defects in the software and compared the performance with different machine learning models. The models developed by them with the help of SVM, provide better accuracy than the other models. In the context of four NASA datasets, they calculate the ability of SVM in predicting defect-prone software modules and contrast the performance of the software fault prediction against eight statistical and machine learning models .Gondra et al. [6] used Artificial Neural Networks (ANN’s) and Support Vector Machines (SVM’s) to reduce the price and progress for the effectiveness of the software testing process. Data is taken from the publicdataset that is freely available from the Promise repository. Researchers use different software metrics like Lines of Code (LOC), McCabe (1976), and Halstead (1977) metrics. Jun Zheng17described that the software fault prediction model can be built with the help of threshold-moving technique. The motive of the software developer is to develop the better quality software on time and inside the financial plan. Software fault prediction model classifies the modules into two classes: faulty modules and non – faulty modules. They discussed the use of different cost sensitive boosting algorithms for software fault prediction. The accuracy of the cost sensitive boosting algorithms is quite good than the other algorithms. R.Shatnawi [13] states that the majority of the modules for finding the prediction performance are correct whereas some modules are defective. They applied technique to find the number of faults in the particular module. This technique is called Eclipse. This technique works well on real world objects called Object Oriented systems. In this Object Oriented System, they used the existing defected data for eliminating the defective modules [13]. Singh et al. [14] describes that Levenberg- Marquardt (LM) algorithm based neural network tool is used for prediction of software defects at an early stage of SDLC. They used the class level metrics. The Defected data are collected from the NASA promise repository. LM Algorithm is based upon machine learning approach The accuracy of LM Algorithm based neural network is better than the Polynomial function -based neural network for detection of software defects.

III. Software Reliability

Software Reliability has been defined as the probability that a software fault, which causes deviation from required output by more than specified tolerances in a specified environment, does not occur during a specified exposure period. Thus reliability can be formally defined as:

R (i) = P [No failures in i runs] (1)

Or

R (t) = P [No failures in interval (0,t)] (2)

Assuming that inputs are selected independently according to some probability distribution function, we have

R (i) = [R (1)] i = (R) I (3)

Where R=R (1). We can define the reliability R as follows:

R=1-lim (nf/n) (4)

Where

n = number of runs,

nf = number of failures in n runs.

This is the operational definition of software reliability. Software reliability is a function of many factors like software development methodology, validation methods and also the languages in which the program is written. Failure intensity is an alternative way of expressing reliability. Let R be the reliability, λ the failure intensity and "t" the execution time. Then as shown in [Fig-1]

$R(t) = \exp(-\lambda t)$ (5)

The failure intensity statement is more economical since only one number is needed. Failure intensity like reliability is defined with respect to a specified operational profile. The relationship between failure intensity and reliability depends on the reliability model employed if these values are changing. [Fig-2] shows that as faults are removed, failure intensity tends to drop and the reliability tends to increase.

Software Error, Fault and Failure

The following definitions are commonly used in the software engineering literature [4]:

- Error: Human action which results in software containing a fault.

- Fault: A manifestation of an error in software; a fault if encountered may cause a failure.

- Failure: An unacceptable result produced when fault is encountered.

Even though these three things have different meanings, they are often used interchangeably in the literature.

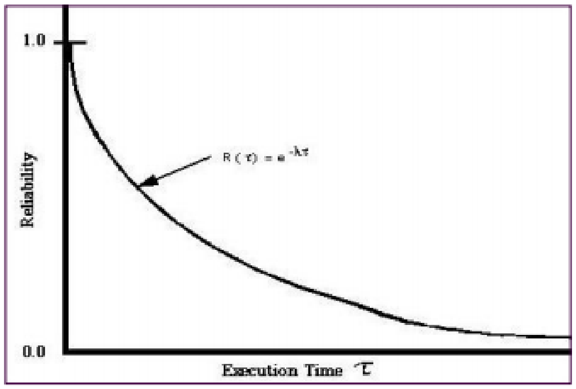

Figure 1: Reliability Function in Relation to Failure Intensity

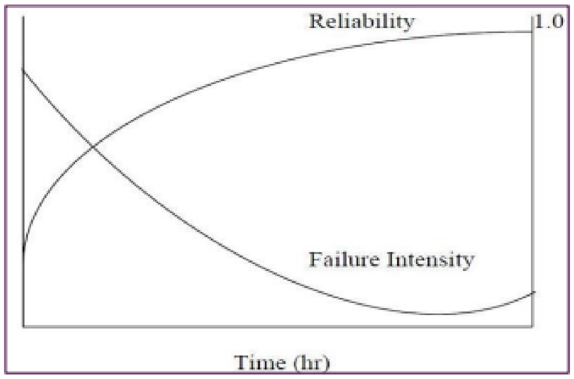

Figure 2: Decreases in Failure Intensity Increases Reliability

Time Relevant to Software Reliability

Three kinds of time are relevant to software reliability:

- The execution time for the program is the time required by a processor to execute the instructions of the program.

- Calendar time is the regular time we are familiar with.

- Clock time, used occasionally, represents the elapsed time from start to end of the program execution on a running computer. It includes wait time and execution time of other programs.

IV. Research approach

Naïve Bayes Classifiers (NB)

The Naïve Bayes classifier [3] is based on Bayes rule of conditional probability. It analysis each attribute individually and assumes that all of them are independent and important.

It is a classifier based on Bayes theorem used in software fault prediction. It resolves the several difficulties like spam classification (to predict whether email is spam or not), medical diagnosis (given list of symptoms, predict whether patient has cancer or not) and so on. This method can be used to predict faulty and non-faulty modules.

The naive Bayes is a simple probabilistic classifier model which is based on the Bayesian theory. The naive Bayes classifiers can be trained very smoothly in a supervised learning dataset. The Naive Bayes algorithm is a classification algorithm based on Bayes rule that assumes the attributes of one category are all conditionally independent of one another from other category. Thus, Naive Bayes is a hybrid approach between decision-tree classifiers and Naive Bayes classifiers. The structure of Naive Bayes classifiers represents knowledge in the form of a tree which is constructed recursively. The leaf nodes are Naive Bayes categorizers for predicting a single class. We employed 10-fold cross-validation on all sixteen life-cycle failure datasets using Naive Bayes at the node to evaluate the utility of a node. The utility of the split is the weighted sum of utility of the nodes which depends on the number of instances that go through that node. The Naive Bayesalgorithm tries to approximate whether the generalization accuracy of Naive Bayes at each leaf is higher than a single Naive Bayes classifierat the current node. Naive Bayes classifiers are generally easy to understand and the induction of these classifiers is extremely fast that require a single pass through the data. We found that Naive Bayes works well on real life failure datasets and thus it can be scaled up for accurate reliability prediction in real life industrial projects.

V. Result

In this section we developed a new model to improve the software reliability and error fault prediction using Naïve bayes prediction model.

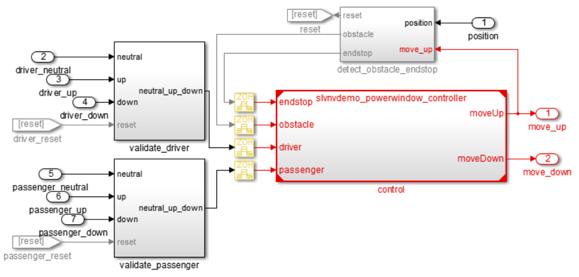

Figure 3: A slvnvdemo power window controller with validated passenger

Above figure shoes the a model which indicate about validated passenger in railway system.

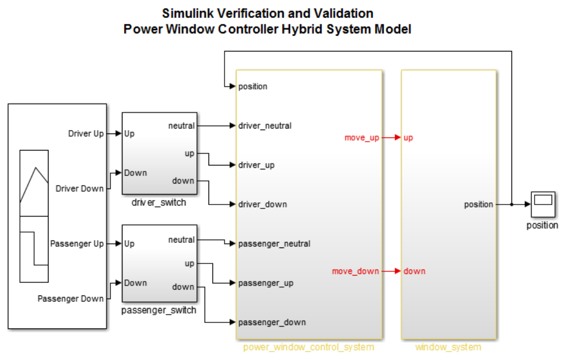

Figure 4: A slvnvdemo power window controller hybrid system model with passenger Up and down

Figure 4 indicate the power window controller for passenger Up and down position.

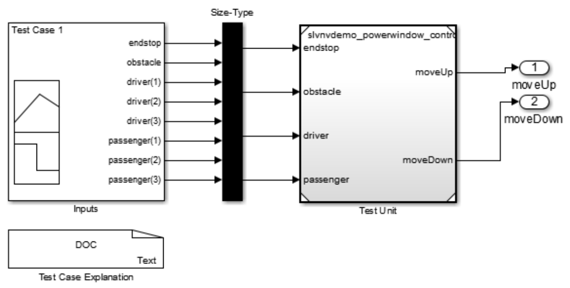

Figure 5: Test case explanation and unit testing of passenger for test case 1

In presented above figure we shows the Test case for 1. It also indicates the Move up and down position of passenger.

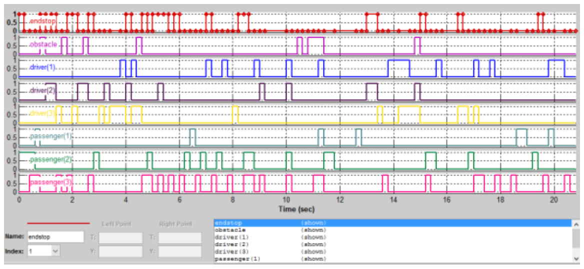

Figure 6: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

Now the very precious part of the Railway system, it indicated the signal of end stop, obstacle and passenger position.

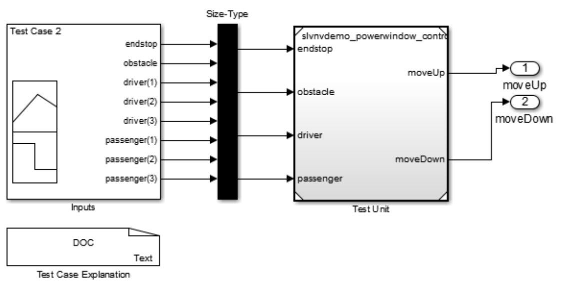

Figure 7: Test case explanation and unit testing of passenger for test case 2

As also discussed in figure 7, figure indicate the test case for 2 in software.

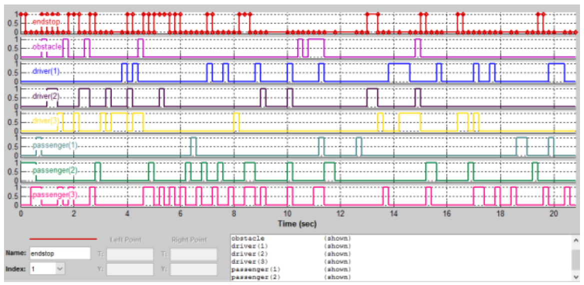

Figure 8: Signal presentation of end stop, obstacle, driver 1, 2 and passenger 1, 2

The model starts by logging input signals to the component implementing the controller in its parent model and creating harness model for the controller from that logged data. A new test case in the harness model it captures all test cases and simulates the controller model for model coverage. Finally, it executes the controller with those test cases in simulation mode and Software-In-the-Loop (SIL) mode.

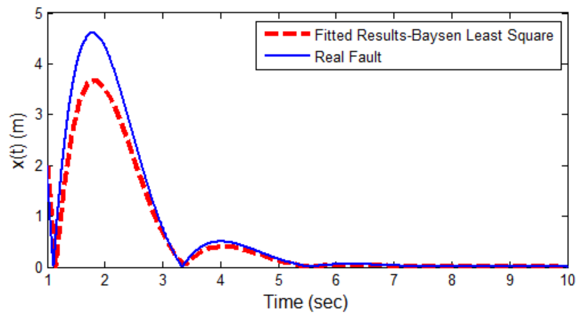

Figure 9: Real and fitted data

Above figure shows the fitted data and real fault. During the fitted data it also get some fault as shown in 9. Measures the percentage of the total variation about the mean accounted for by the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model does not fit the data well. The larger, the better the model explains the variation in the data

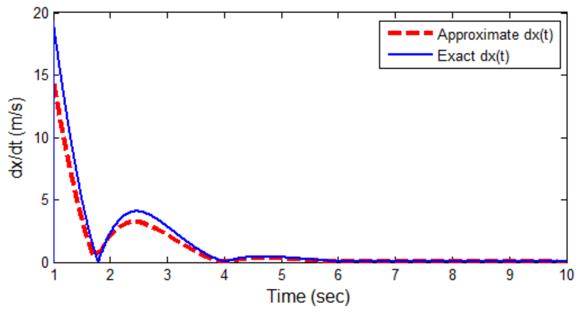

Figure 10: Approximate dx (t) and Exact dx (t)

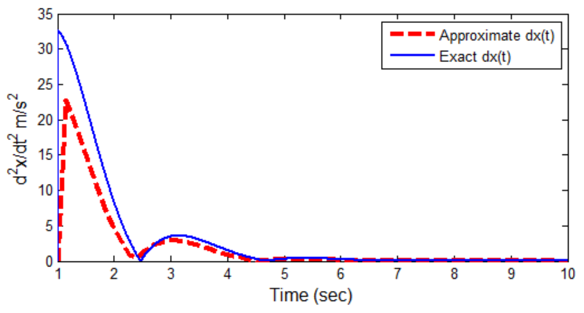

Figure 11: Approximate dx (t) and exact dx (t) on time

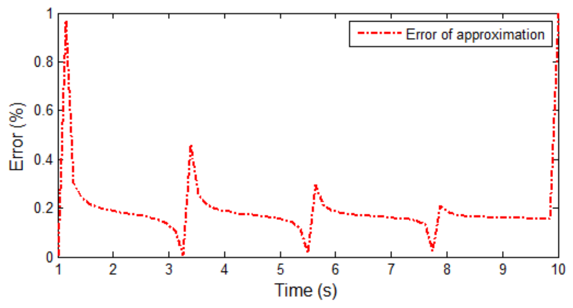

Figure 12: Error of approximation of software prediction

The error approximation incorporating the effect of time and covariates simultaneously and Sometimes, the effect of covariates are insignificant and the reduced form of the model may prove to be a better fit for the data and this can easily be obtained by setting f = 0, which gives us the better model. Similarly, when both covariate effect and time trend are insignificant, the model reduces to a NHPP model, with a constant recurrence rate, a. The number of intervals is always less or equal to number of failures that we observed because there can be more than one failure in any time interval.

VI. Conclusion

After research of various researches related to data mining techniques for software defect prediction, we got that data mining is an emerging approach for defect prediction. Machine Learning Classifiers have emerged as a way to predict the fault in the software system. Since most of these studies have been performed using different data sets, reflecting different software development environment and processes, it is difficult to conclude the best software prediction model. Various models and techniques are studied which have their associated merits and demerits. The objective of this study is to analyze the performance of various data mining techniques used in software defect prediction models.

References

[1.] W. Burkhart, Z. Fatiha "Testing Software and Systems" // 23rd IfipWg 6.1 International Conference (2011), 236.

[2.] K. Goseva-Popstojanova, A.P. Mathur, K.S. Trivedi "Comparison of architecture–based software reliability models" // 12th International Symposium on Software Reliability Engineering (2001), 22-31.
[3] H. Okamura, Y. Etani, T. Dohi. A multi-factors software reliabilitymodel based on logistic regressionProc. of the 21stInternational Symposium on Software Reliability Engineering, 2010: 31–40.

[3] A. Mccallum& K. Nigam (1998) “A Comparison of Event Models for Naive Bayes TextClassification”, Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)-Workshop on Learning for Text Categorization, pp. 41-48.

[4] .Elish, K. O., &Elish, M. O. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, ACM, Vol. 81(5), 2008; 649-660.

[5] Fenton, N. E., & Neil, M., A critique of software defect prediction models. IEEE Transactions on software Engineering, Vol. 25(5); 675- 689

[6] Gondra, I. Applying machine learning to software fault-proneness prediction. ACM Journal of Systems and Software, Vol. 81(2), 2008; 186- 195.

[7] Jiang, Y., Cukic, B., &Menzies, T. Fault prediction using early lifecycle data. Published in 18th IEEE international symposium on software reliability, 2007; 237-246.

[8] Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., &Musilek, P. Fault prediction in object-oriented software using neural network techniques. Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand, 2004; 1-8.

[9] N. Ahmad, M. U. Bokhari, S. M. K. Quadri, et al. The exponentiatedWeibull software reliability growth model with various testing-efforts and optimal release policyInternationalJournal of Quality and Reliability Management, 2008,25(2): 211–235.

[10] Mahaweerawat A., Sophasathit P., Lursinsap, C. Software fault prediction using fuzzy clustering and

radial basis function network .In proceedings of International conference on intelligent technologies, 2002;304-313.

[11] C. T. Lin, C. Y. Huang. Software reliability modeling with Weibull-type testing-effort and multiple change-pointsProc.of the IEEE Region 10 Conference, 2005 (in CD format).

[12] C. Y. Huang. Performance analysis of software reliability growth models with testing-effort and change-point. Journalof Systems and Software, 2005, 76(2): 181−194.

[13] Shatnawi, R. Improving software fault-prediction for imbalanced data.IEEE Proceedings of International Conference on Innovations in Information Technology, 2012; 54-59.

[14] Singh, M., &Salaria, D.S. Software Defect Prediction Tool based on Neural Network. International Journal of Computer Applications, Vol. 70(22),2013; 22-27.

[15] The Global Conference for Wikimedia,(2014); London

[16] Xing, F., Guo, P., &Lyu, M. R.A novel method for early software quality prediction based on support vector machine.16th IEEE international symposium on Software Reliability, Vol. 37(6), 4537-4543.