

Modeling Of Fault Prediction Using Machine Learning Techniques

Deepali Gupta*, Amanpreet Singh Brar, Parvinder Singh Sandhu*****

***G.I.M.T, Kurukshetra, Haryana, **G.N.D.E.C, Ludhiana, Punjab, *** R.B.I.E.B.T, Mohali, Punjab**

***deepali_gupta2000@yahoo.com, ** appibrar@yahoo.com, *** pss@gndec.ac.in**

Abstract-- Predicting faults early in the software life cycle can be used to improve software process control and achieve high software reliability. Quality of software is increasingly important and testing related issues are becoming crucial for software. Methodologies and techniques for predicting the testing effort, monitoring process costs, and measuring results can help in increasing efficiency of software testing. Being able to measure the fault-proneness of software can be a key step towards steering the software testing and improving the effectiveness of the whole process. To estimate software faultiness before and during testing and analysis activities would greatly help software testing and analysis. Many systems are delivered to users with excessive faults. It has long been recognized that seeking out fault-prone parts of the system and targeting those parts for increased quality control and testing is an effective approach to fault reduction. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. Prediction models based on software metrics, can estimate number of faults in software modules. In this paper different machine learning algorithms and neural network techniques on two different real-time software defect datasets are evaluated. The results show that when all the prediction techniques are evaluated then best algorithm for classification of the software components into faulty/fault-free systems is found to be Generalized Regression Neural Networks.

Index Terms -- Fault proneness, Software metrics, Software reliability and Software quality.

I. INTRODUCTION

Faults in software systems continue to be a major problem and many systems are delivered to users with excessive faults. The source of these faults is many and varied. This is been recognized that seeking out fault-prone parts of the system and targeting those parts for increased quality control and testing is an effective approach to fault reduction. It has been long recognized that a limited amount of valuable work in that area has been carried out previously. Timely predictions of faults in software modules can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults.

Despite this it is very difficult to identify a reliable approach to identifying fault-prone software components.

Metrics is defined as “The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products”. Software metrics can be classified as either product metrics or process metrics.

Product metrics are measures of the software product at any stage of its development, from requirements to installed System. Product metrics may measure the complexity of the software design, the size of the final Program or the number of documentation produced. Process metrics on the other hand, are measures of the software development process, such as overall development time, type of methodology used, or the average level of experience of the programming staff.

The aim of software metrics is to predict the quality of the object oriented software products. This paper presents models of fault prediction in software systems. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. The modeling techniques include principal component analysis, discriminate analysis, logistic regression, logical classification models, and layered neural networks.

The rest of the paper is organized as follows. Section 2 gives overview of the problem formulation. Related works is presented in Section 3. A detail about public domain data set is given in Section 4. Section 5 focuses on present work done and conclusion of this work is given in Section 6.

II. PROBLEM FORMULATION

Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Early detection of fault-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development [6]. The ability of software quality models to accurately identify critical components allows for the application of focused verification activities ranging from manual inspection to automated formal analysis methods. Software quality models, thus, help ensure the reliability of the delivered products. It has become an imperative to develop and apply good software quality models early in the software development life cycle, especially for large-scale development efforts.

The basic hypothesis of software quality prediction is that a module currently under development is fault prone if a module with the similar product or process metrics in an earlier project (or release) developed in the same environment was fault prone. Therefore, the information available early within the current project or from the previous project can be used in making predictions. This methodology is very useful

for the large-scale projects and specially for the projects with multiple releases.

Accurate prediction of fault-prone modules enables the verification and validation activities focused on the critical software components [7]. Therefore, software developers have a keen interest in software quality models. It is required that fault-prone prediction models should be efficient and accurate.

Fault-proneness models are models that are built from information about the code and its faults, and that relate code to faults. The existence of such classes of software would allow deriving fault-proneness models from historical data, i.e., data available for old applications of a given class, and then using such models for predicting fault-proneness of new software applications of the same class. Such models could be useful during both planning and executing testing activities. Planning testing activities could take advantage of information about software fault-proneness for anticipating costs and allocating activities, while test execution can use this information to evaluate the quality of the results.

Prediction of fault-prone modules:

- Supports software quality engineering through improved scheduling and project control.
- Can be a key step towards steering the software testing and improving the effectiveness of the whole process by planning and executing testing activities.
- Enables effective discovery and identification of defects.
- Enables the verification and validation activities focused on critical software components.
- Used to improve software process control and achieve high software reliability.
- Can be used to direct cost-effective quality enhancement efforts to modules that are likely to have a high number of faults.

From previous experiments it has been observed that: -

- Existence of a correlation between a reasonable set of static metrics and software fault proneness [1].
- Use of various methods for predicting fault-prone modules in software development projects [5].
- Size and complexity metrics are not sufficient for accurate prediction.
- There is a need to find the best prediction techniques for a given prediction problem.

A wide variety of techniques have been proposed [2]. The modeling techniques cover the main classification paradigms, including principal component analysis, discriminate analysis, logistic regression, logical classification models, layered neural networks etc. Principal component analysis has been often used in the software engineering field to improve the accuracy of Discriminant models or regression models.

- Discriminant analysis, which has been previously applied to detect fault-prone programs.

- Logistic regression, which has been included in empirical comparisons between models identifying high-risk components.
- Logical classification models, which have been extensively used in software engineering issues, such as the identification of high-risk modules or the detection of reusable software components.
- Layered neural networks, which have been already applied in software engineering applications, to build reliability growth models predict the gross change or reusability metrics.
- Holographic networks, a non-connectionist kind of neural network, which has been proposed for evaluating software quality. Empirical investigations have been already performed in other fields, including financing and manufacturing.

Researchers have also investigated methods to combine machine learning and statistical methods. Neumann used principal component analysis (PCA) to enhance the performance of neural networks. However, we find that feature subset selection (FSS) methods, such as correlation-based feature selection technique (CFS), consistency based subset evaluation technique (CBS), information gain attribute ranking technique, and relief techniques perform better than principal component analysis. Hence, the cumbersome procedure of calculating and deducing from principal components can be possibly eliminated. While the above prediction models have their advantages and disadvantages, Fenton and Neil [2] argued that these models can lead to inappropriate risk management decisions, since they do not effectively take dependencies between the “attributes” into consideration. So they proposed Bayesian belief networks (BBN) as an alternative to current approaches. However, constructing BBN topologies is not an easy task. Mitchell provides a general comprehensive discussion of the machine learning algorithms used for identifying faults in software components.

III. RELATED WORK

Lanubile presented an empirical investigation of the modeling techniques for identifying fault-prone software components early in the software life cycle. Using software complexity measures, the techniques build models, which classify components as likely to contain faults or not. The modeling techniques applied in their study cover the main classification paradigms, including principal component analysis, discriminate analysis, logistic regression, logical classification models and layered neural networks.

P. Bellini compared Fault-Proneness Estimation Models and concluded that over the last years, software quality has become one of the most important requirements in the development of systems. Fault-proneness estimation could play a key role in quality control of software products. In this area, much effort has been spent in defining metrics and identifying models for system assessment. Using these metrics

to assess which parts of the system is more fault-proneness is of primary importance. The objective has been to find a compromise between the fault-proneness estimation rate and the size of the estimation model in terms of number of metrics used in the model itself. The methodologies were the logistic regression and the discriminate analyses.

Atchara Mahaweerawat analyzed that to remain competitive in the dynamic world of software development; organizations must optimize the usage of their limited resources to deliver quality products on rears in the system under development. His paper presents time and within budget. This requires prevention of fault introduction and quick discovery and repair of residual faults. In particular, faults due to the use of inheritance and polymorphism are considered, as they account for significant portion of faults in object-oriented systems. The proposed fault prediction model is based on supervised learning using Multilayer Perceptron Neural Network and the results are analyzed in terms of classification correctness and based on the results of classification, faulty classes are further analyzed and classified according to the particular type of fault.

Yan Ma suggested that accurate prediction of fault prone modules in software development process enables effective discovery and identification of the defects. Such prediction models are especially valuable for the large-scale systems, where verification experts need to focus their attention and resources to a problem methodology for predicting fault-prone modules using a modified random forests algorithm which improve classification accuracy by growing an ensemble of trees and letting them vote on the classification decision.

Runeson, Claes Wohlin and Magnus C. Ohlsson proposed that in striving for high-quality software, the management of faults plays an important role. The faults that reside in software products are not evenly distributed over the software modules; some modules are more fault-prone than others. Various models are presented in the literature, which identify fault-prone modules, which can be used in quality management processes. They proposed the schemes, which would help researchers to compare the capabilities of different models and to choose a suitable model.

Eric Rotenberg speculates that technology trends pose new challenges for fault tolerance in microprocessors. He proposed a new time redundancy fault-tolerance approach in which a program is duplicated and the two redundant programs simultaneously run on the processor.

IV. PUBLIC DOMAIN DEFECT DATA SET

The real-time faulty data sets used in this paper can be accessed from NASA's MDP (Metric Data Program) data repository, available online at <http://mdp.jvv.nasa.gov>. The CM1 data is obtained from a spacecraft instrument, written in C, containing approximately 506 modules. The JM1data is obtained from a predictive ground system project, written in C++, containing 10879 modules. The KC1 data is obtained from a science data processing project coded in C++,

containing 2108 modules. The PC1 data is collected from a flight software system coded in C, containing 1108 modules. All these data sets varied in the percentage of defect modules, with the CM1 dataset containing the least number of defect modules and the KC1 dataset containing the largest.

Different types of predictor software metrics (independent variables) are used in our analysis. These complexity and size metrics include well known metrics, such as Halstead, McCabe, line count, operator/operand count, and branch count metrics.

Halstead metrics are sensitive to program size and help in calculating the programming effort in months. The different Halstead metrics include length, volume, difficulty, intelligent count, effort, error, and error estimate. McCabe metrics measure code (control flow) complexity and help in identifying vulnerable code.

The different McCabe metrics include cyclometric complexity, essential complexity, design complexity and lines of code. The target metrics (dependent variables) are the "Error Count" and "Defects". "Error Count" refers to the number of module errors associated with the corresponding set of predictor software metrics, while "Defect" metric refers to whether the module is fault prone or fault-free. Regarding the software; we used the freely available WEKA machine learning tool kit and Mat lab to conduct these experiments. The WEKA software computes the mean absolute error, root mean squared error, relative absolute error, and root relative squared error. However, the most commonly reported error is the mean absolute error / root mean squared error. The root mean squared error is more sensitive to outliers in the data than the mean absolute error.

V. PRESENT WORK

A variety of many machine learning algorithms and neural network techniques are analyzed. As a broad subfield of artificial intelligence, machine learning is concerned with the design and development of algorithms and techniques that allow computers to "learn".

The major focus of machine learning research is to extract information from data automatically, by computational and statistical methods. Machine Learning is part of Machine Intelligence but addresses a more specialized purpose and scope. Machine learning algorithms and applications adapt themselves to the behavior of a system usually through the discovery of time-varying patterns in the data. These algorithms typically fuse linear and nonlinear regression, adaptive control theory, neural networks, statistical learning theory, rule induction, and decision tree generation.

Neural networks are non-linear sophisticated modeling techniques that are able to model complex functions. These are used when exact nature of input and outputs is not known. A key feature is that they learn the relationship between input and output through training.

Various neural network techniques like Gradient Descent back propagation(BGD), Batch Gradient Descent with

momentum(BGDWM), Variable Learning Rate(VLR), Variable Learning Rate with momentum(VLRM), Resilient back propagation(RB), Fletcher-Reeves conjugate gradient Algorithm(FRCG), Polak-Ribiere Update conjugate gradient Algorithm(PRUCG), Powell-Beale Restarts conjugate gradient Algorithm(PBRCG), Scaled Conjugate Gradient Algorithm(SCG), Quasi-Newton BFGS Algorithm(QNBFGS), Quasi-Newton One Step Secant Algorithm(QNOSS), Levenberg-Marquardt Algorithm(LM), Generalized Regression Networks(GRNN) and Self organizing Map/Network(SON) are explored.

Various machine learning algorithms like Naïve Bayes Algorithm, Multinomial Naïve Bayes Algorithm, Updateable Naïve Bayes Algorithm, Logistic Regression Modeling, Multilayer Perceptron Algorithm, RBF Network Algorithm, Linear Logistic Algorithm, Sequential Minimal Optimization Algorithm (SMO), Nearest-neighbour Classifier, k- Nearest-neighbour Classifier, k* instance-based Classifier, Locally Weighted Learning Algorithm, Additive Regression Algorithm, Attribute Selected Classifier, Bagging Algorithm, Classification Via Regression, Cross-validation Parameter Selection, Decorate Algorithm, Grading Algorithm, Additive Logistic Regression, MetaCost Algorithm, MultiBoosting Algorithm, Metaclassifier For Multi-Class Datasets, Ordinal Class Classifier, Raced Incremental Logit Boost, Random Committee Algorithm, Stacking Algorithm, StackingC Algorithm, Vote Algorithm, HyperPipe Algorithm, Voting Feature Intervals Algorithm, Decision Stump Algorithm, Pruned C 4.5 Decision Tree Algorithm, Logistic Model Tree Algorithm, Random Forest Algorithm, Random Tree Algorithm and REP Tree Algorithm are also analyzed. The model is implemented and then best algorithm is found.

CONCLUSION

Being able to estimate software faultiness before and during testing and analysis activities would greatly help

software testing and analysis. Prediction of fault-prone modules supports software quality engineering through improved scheduling and project control. It is a key step towards steering the software testing and improving the effectiveness of the whole process. This paper has presented an analysis of classification models predicting quality classes on the basis of software complexity measures and tried to find the best model on the basis of software complexity measures. Out of these entire machines learning algorithm, generalized regression networks has the minimum value of mean absolute error and root mean square error. So, it gives the best performance among all these machine learning and neural network techniques.

REFERENCES

- [1] Fenton, N. E. and Neil, M. 1999, "A Critique of Software Defect Prediction Models", *IEEE Trans. Software Engineering*, vol no. 25, 5 Sep 1999, 675-689, Bellini, I. Bruno, P. Nesi, D. Rogai, University of Florence.
- [2] F. Lanubile, A. Lonigro and G. Visaggio, "Comparing Models for Identifying Fault-Prone Software Components", 1995.
- [3] Giovanni Denaro and Mauro, "An Empirical Evaluation of Fault Proneness Models".
- [4] Mahaweerawat, A. "Fault-Prediction in object oriented software's using neural network techniques".
- [5] Ohlsson, R. "A Proposal for Comparison of Models for Identification of Fault-Proneness", Dept. of Communication Systems, Lund University.
- [6] Rotenberg, E. "A Microarchitectural Approach to Fault Tolerance in Microprocessors".
- [7] Yan Ma, Lan Guo and Bojan Cukic: "A Statistical Framework for the Prediction of Fault-Proneness", Department of Statistics, West Virginia University, Morgantown.
- [8] www.cs.waikato.ac.nz.