IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 4, Ver. I (Jul-Aug. 2014), PP 74-79
www.iosrjournals.org

Predict Fault-Proneness Module Using Pattern Recognization (A Review)

[1]Puneet Chopra, [2]Er. Gurdeep Kaur
[1]*CSE dept, Desh Bhagat University, India*
[2]*(Asst. Professor) CSE dept, Desh Bhagat University, India*

***Abstract**: Background: The accurate prediction of where faults are likely to occur in code can help direct test effort, reduce costs and improve the quality of software. Objective of this paper is We investigate how the context of models, the independent variables used and the modeling techniques applied, influence the performance of fault prediction models. Method on We used a systematic literature review to identify 208 fault prediction studies published from January 2000 to December 2012. We synthesise the quantitative and qualitative results of 36 studies which report sufficient contextual and methodological information according to the criteria we develop and apply. Combinations of independent variables have been used by models that perform well. Feature selection has been applied to these combinations when models are performing particularly well.*
Conclusion: The methodology used to build models seems to be influential to predictive performance. Although there are a set of fault
***Keywords:** Systematic Literature Review, Software Fault Prediction*

I. Introduction

The prediction of fault-proneness has been studied extensively. In a stable development environment, metrics can be used to predict modules that are likely to Critical faults. Some researchers en dorse to use of product metrics, such as Halstead complexity McCabe's cyclomatic complexity, and various code size measures to predict fault-prone modules while others are believe of such simplistic approaches V&V textbooks, for example, recommend using static code metrics to decide whether modules are worthy of manual inspections. NASA software Independent Verification and Validation facility and with several large government software contractors is that they won't review software modules unless tools like McCabe predict that they are fault prone. The use of such measures is controversial. Fenton offers an example where the same program functionality is achieved using different programming language constructs resulting in different static measurements for that module.

Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between Fault and non-Fault modules after learning; it can then be used to classify new email messages into Fault and non-Fault modules.

The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. Generalization is the property that the system will perform well on unseen data instances; the conditions under which this can be guaranteed are a key object of study in the subfield of computational learning theory.

Fenton uses this example to argue the uselessness of static code attributes. Fenton & Pfleeger note that the main McCabe's attribute (cyclomatic complexity, or v(g)) is highly correlated with lines of code. Also, Shepperd&Ince remark that for a large class of software cyclomatic complexity is no more than a proxy for, and in many cases out performed by, lines of code. Therefore, they argue against the use of single features to predict for defects. Further, they reject other commonly used indicators since they are all highly correlated and, so they argue, just as uninformative.

When individual features fail, combinations can succeed. This paper argues that combinations of static features ex-tracted from requirements and code can be exceptionally good predictors for modules that actually harbor defects. We do not intend to propose yet another classification algorithm. Our overall goal is to explore whether prediction of fault prone modules can be achieved using the information available in the early phases of software development.

As software systems are increasingly deployed in mission critical applications, it has become imperative that they operate reliably and in accordance with the requirements.

Experts to concentrate their attention and resources at problem areas of the system under development. Thus, applying software quality models early in the software life cycle con- tributes to efficient defect removal and results in delivering more reliable software products. The basic hypothesis of software quality prediction is

that a module currently under development is likely to be fault prone, if a module with the similar product or process metrics in an earlier project (or release) was fault prone. Therefore, the information from the previous project can be used in making a prediction for the current project, if the development environment is stable. This methodology is very useful for large-scale projects or projects with multiple releases. Many modeling techniques have been developed and applied to software quality prediction.

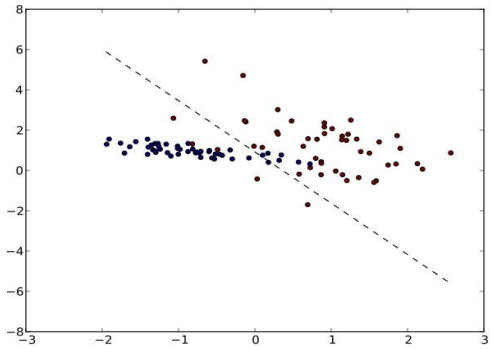

Fig 1: Machine Learning Approach

II. Related Work

Parvinder S. Sandhu, et.al [1] have proposed the Genetic Algorithm based software fault prediction models with Object-Oriented metrics. It has used Metric values of JEdit open source software for generation of the rules for the classification of source software modules in the categories validation is performed. Result shows that Genetic algorithm approach can be used for finding the fault proneness in object oriented software components. Fault-proneness of software modules is the probability that the module contains faults.

Yan Ma, et.al[2] have proposed the a methodology for predicting fault prone modules using a modified random forests algorithm. Random forests improve classification accuracy by growing an ensemble of classification trees and letting them vote on the classification decision. We applied the methodology to five NASA public domain defect data sets. These data sets vary in size, but all typically contain a small number of defect samples in the learning set. If overall accuracy maximization is the goal, then learning from such data usually results in a biased classifier, i.e. the majority of samples would be classified into non-defect class. To obtain better prediction of fault-proneness, two strategies are investigated: proper sampling technique in constructing the tree classifiers, and threshold adjustment in determining the winning class. Both are found to be effective in accurate prediction of fault prone modules. In addition, the paper presents a thorough and statistically sound comparison of these methods against ten other classifiers frequently used in the literature. Accurate prediction of fault prone modules in software development process enables effective discovery and identification of the defects.

Lan Guo, et.al [3] have proposed a novel methodology for predicting fault prone modules, based on random forests. Random forests are an extension of decision tree learning. Instead of generating one decision tree, this methodology generates hundreds or even thousands of trees using subsets of the training data. Classification decision is obtained by voting. We applied random forests in five case studies based on NASA data sets. The prediction accuracy of the proposed methodology is generally higher than that achieved by logistic regression, discriminant analysis and the algorithms in two machine learning software packages, WEKA and See5.

Rubinderjit Kaur, et.al[4] have proposed that evaluation of the fault proneness of modules in open source software
system using k-NN clustering algorithm based on Object-Oriented metrics. It has Metric values of JEdit open source software for generation of the rules for the classification of software modules and thereafter empirically validation is performed. Result show that the proposed approach can be used satisfactorily for finding the fault proneness in object oriented software components.

Mattew Evett, et.al[5] have proposed the genetic programming (GP) based system for targeting software modules for reliability enhancement. The GP system ,and provides a case study using software quality data from two actual industrial projects. The system is shown to be robust enough for use in industrial domains.

Marshima M. Rosli, et.al[6] have proposed the software development industry is to deliver an application with

100% defects free. However, this challenge is difficult to achieve by the software industries because it involve humans and is not an automated process done by applications, which having faults is a common things. Fault prediction is identified as one major area to predict the probability that the software contains faults. The objective of the fault prediction is to classify the software modules in the categories of faulty and non-faulty modules as early as possible in software development life cycle. Fault prediction model using object oriented metrics values from web application as input values to the genetic algorithm to predict the fault probability. The aim of the proposed design model is to develop an automated tool for software development group to discover the most likely software modules in web applications to be high problematic in the future.

Manpreet Kaur, et.al[7] have proposed that fault-proneness of a software is the probability that the module contains faults. To predict fault proneness of modules different techniques have been proposed which includes statistical methods, machine learning techniques, neural network techniques and clustering techniques. This approach has been tested with real time defect C programming language datasets of NASA software projects. Result show that the fusion of requirement and code metric is the best prediction model for detecting the faults as compared with commonly used code based model.

Robert Hochman, et.al[8] have proposed the genetic algorithm is applied to developing optimal or near optimal back propagation neural networks for fault-prone/not fault-prone classification of software modules. The algorithm considers each network in population of neural networks as a potential solution to the optimal classification problem. Variables governing the learning and other parameters and network architecture are represented as substrings (genes) in a machine level bit string (chromosome). When the population undergoes simulated evolution using genetic operations selection based on a fitness function, crossover and mutation the average performance increases in successive generations. We found that. On the same data, compared with the best manually developed networks, evolved networks produced improved classifications in considerably less time, with no human effort, and with greater confidence in their optimality or near optimality. Strategies for devising a fitness function specific to the problem are explored and discussed.

A. Discriminant Analysis

Discriminant analysis is a very useful tool to determine which variables discriminate between two or more naturally occurring groups. It can also be used to classify cases into two or more groups. In our study, the DISCRIM procedure (linear discriminant function) in SAS [4], a commercial statistics software, was employed as a classifier on the five NASA data sets.

B. Logistic Regression

Logistic regression is useful to predict a dependent vari-able on the basis of independents (predictors). For comparison, the LOGISTIC procedure in SAS was used as a classifier to predict fault prone modules for the five NASA data sets.

C. Random Forests

A random forest is a classifier consisting of a collection of tree-structured classifiers. The random forest classifies a new object from an input vector by examining the input vector on each tree in the forest. Each tree casts a unit vote at the input vector by giving a classification. The forest selects the classification having the most votes over all the trees in the forest.

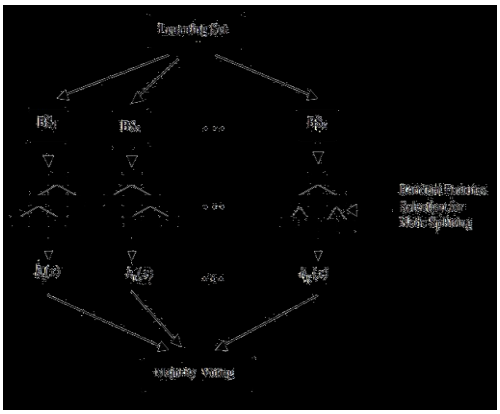

Fig. 2 Construction of a Random Forest

D. See5/C5

See5/C5 is a commercial machine learning software [5]. Its earlier version is called C4.5. There are three classifiers in See5: Decision Tree, Rule Set, and Boosting. When See5 is invoked with the default values of all options, it constructs a decision tree for classification. Decision trees can some-times be quite difficult to understand. An important feature of See5 is its ability to generate classifiers called Rule Sets that consist of unordered collections of (relatively) simple if-then rules, derived from the constructed decision trees. Another innovation incorporated in See5 is adaptive boosting. The idea is to generate several classifiers (either decision trees or rule sets) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class. The three classifiers of See5 were used to predict fault prone modules for the five NASA data sets.

E. Genetic Algorithm

A genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics and a particular class of Evolutionary Algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. With help of Genetic algorithm classification of the software components into faulty/fault-free systems is performed. The flowchart of the Genetic Algorithm based approach is shown in the following figure:

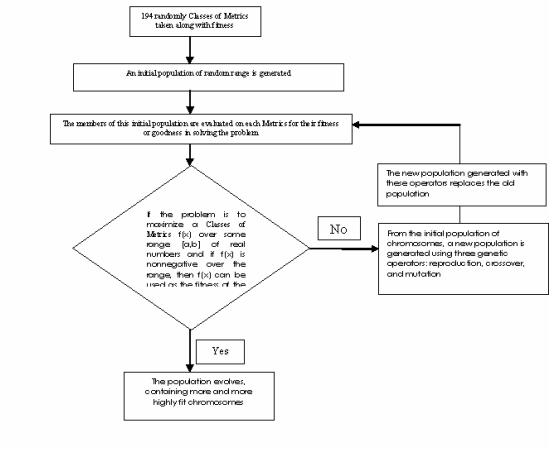

Fig. 3 Flowchart of use of GA

III. Experimental Database

Data set describing project JM1 contains some missing values. These were removed prior to the statistical analysis. Each data set contains twenty-one software product metrics which describe product's size, complexity and some structural characteristics.

Table 1: Metric Descriptions of Five Data Sets

Metric Type	Metric	Definition
McCabe	v(G)	Cyclomatic Complexity
	ev(G)	Essential Complexity
	iv(G)	Design Complexity
	LOC	Lines of Code
Derived Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Count
	E	Effort
	B	Effort Estimate
	T	Programming Time

Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAndComment	Lines of Code and Comment
Basic Halstead	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

Defect Predicted?

Module with Defect?	No	Yes
No	True Negative (TN)	False Positive (FP)
Yes	False Negative (FN)	True Positive (TP)

Fig 4: Confusion Matrix of Defect Prediction

$$ACC = \frac{TP + TN}{TN + FP + FN + TP}$$

The True Negative Rate (TNR) is the proportion of correctly identified defect-free modules. It is calculated as:

$$TNR = \frac{TN}{TN + FP} = 1 - PF$$

The precision is proportion of correctly fault-prone modules, calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

The number of fault-prone modules is much smaller than the number of non-defective modules. Therefore, the accuracy is a good measure of performance. The margin plot (please review Section 2 for the definition of "margin") for projects PC1 and KC2 using the random forest algorithm (majority voting). Since maximizing overall accuracy is the goal when the traditional random forest algorithm is applied, a good overall accuracy measures ware obtained. However, a large proportion of fault-prone modules (cases belonging to minority class) were misclassified. The performance measures that take the "imbalance of data" into account when assessing classification success include geometric mean (G-mean) and F-measure, defined as follows:

$$G - mean\ 1 = \sqrt{PD * Precision}$$

$$G - mean\ 2 = \sqrt{PD * TN}$$

$$F - measure = \frac{(\beta 2 + 1) * Precision * PD}{\beta 2 * Precision + PD}$$

References

[1]. T.M. Khoshgoftaar and N. Seliya, The Necessity of Assuring Quality in Soft-ware Measurement Data, Proc. 10th Int'l Symp. Software Metrics (MET-RICS 04), IEEE CS Press, 2004, pp. 119_130.

[2]. L. Guo et al., "Robust Prediction of Fault Proneness by Random Forests," Proc. 15th Int'l Symp. Software Reliability Eng. (ISSRE 04), IEEE CS Press, 2004, pp. 417_428.

[3]. A.G. Koru and J. Tian, "An Empirical Comparison and Characterization of High Defect and High Complexity Modules," J. Systems and Software, vol.67, no. 3, 2003, pp. 153_163.

[4]. J.S. Shirabad and T.J. Menzies, "The PROMISE Repository of Software Engineering Databases," School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[5]. Lan Guo, Yan Ma, Bojan Cukic, Harshinder Singh, "Robust Prediction of Fault-proneness of Random Forests".

[6]. An H. Witten and Eibe Frank, "Data Mining- Practical Machine learning Tools and Techniques," Second Edition, © 2005 by Elsevier Inc.

[7]. A. Idri, T.M. Khoshgoftaar, A. Abran, Can Neural Networks be easily Interpreted in Software Cost Estimation, IEEE International Conference of Fuzy Systems, pp. 1162-1167, (2012).

[8]. A. Mittal, K. Parkash, H. Mittal, Software Cost Estimation Using Fuzzy Logic, ACM Software Engineering, Vol. 35, No. 1, USA, (2011).

[9]. B. Stewart, Predicting Project Delivery Rates Using the Naïve-Bayes Classifier, Journal of Software Maintance and Evolution, vol. 14, pp. 161-179, (2011).

[10]. Barry W. Boehm, Ricardo Valerdi, Achievements and Challenges in Cocomo- Based Software Resource Estimation, IEEE Software Published by the IEEE Computer Society 0740- 7459 / 08 /2 008 IEEE,Jan. 2007.

[11]. C. Olaru, L. Wehenkel, A Compelete fuzzy Decision Tree techniques, Fuzzy Sets and Systems, vol. 138, pp. 221-254, (2003).

[12]. Eibe Frank and Ian H. Witten, "Data Mining: Practical Machine Learning Tools and Techniques" Second Edition © 2005 by Elsevier Inc. Pages 70-75.

[13]. F.V. Jensen, "An Introduction to Baysian Networks", UCL Press, London, (2010).

[14]. I.Attarzadeh, S. Hockow, Improving the Accuracy of Software Cost Estimation Model Based on a New Fuzzy Logic Model, World Applied Sciences Journal 8(2):177-184,(2010).

[15]. Ian H. Witten and Eibe Frank, "Data Mining: Practical Machine Learning Tools and Techniques" Second Edition ISBN: 0-12-088407-0 © 2005 by Elsevier Inc. Pages 36-44, 398- 400.

[16]. L. Zadeh, "Fuzzy Sets" , Journal of Information and control, vol. 8, pp. 338-353, (2011).

[17]. Luigi Buglione and Christof Ebert, "Estimation Tools and Techniques," IEEE Software published by IEEE Computer Society. (ISSN 0740-7459) www.computer.org/software.

[18]. M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 33-53.

[19]. M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," J. Systems and Software, vol. 70, nos. 1–2, 2004, pp. 37–60.

[20]. Mike Cohn,"Agile Estimating and Planning", Copyright 2005 Addison-Wesley.

[21]. Nikolaos Mittas, and Lefteris Angelis, "Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING © 2012 IEEE.