IJCSC International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 645-648

INVESTIGATE THE PERFORMANCE OF SOFTWARE DEFECT PREDICTION FOR DECISION TREE ALGORITHM USING QUANTATIVE AND QUALATATIVE FACTORS

R. P. S. Bedi*, Priyanka Anand[1], Sunil Khullar[2] and Parvinder. S. Sandhu[3]

[1]Student, Department of Computer Science, Rayat Institute of Engineering & Information Technology
[2]Senior Lecturer, Department of Computer Science, Rayat Institute of Engineering & Information Technology
E-mail: sunilkhullar222@yahoo.co.in
[3]Professor, Department of Computer Science & Engg., Rayat & Bahra Institute of Engineering & Biotechnology, Mohali, India.

ABSTRACT

Identifying and locating defects in software projects is a difficult work. Especially, when project size grow, this task become expensive with sophisticated testing and evaluation with sophisticated testing and evaluation mechanism. On the other hand, measuring software is a continuous and disciplined manner brings many advantage such as accurate estimation of project cost and schedules and improving product and process qualities. Detailed analysis of software metric data also gives significant clues about the location of possible defect in a programming code. The aim of this paper is to establish a method for identifying software defect using decision tree algorithm. We investigate whether qualitative and quantitative factors can be used to identify level of number of faulty software modules. We compare the performance of Decision Stump based decision Tree, M5 Model and fast decision Tree learner known as REPTree, for the fault dataset.

Keywords: Decision Stump, Decision Tree, Software Fault, M5 Model, REPTree.

1. INTRODUCTION

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. Early lifecycle data includes metrics describing unstructured textual requirement and static code metrics. Various researches show that use of static code metrics (such as Halstead complexity, Cyclomatic complexity, McCabe's complexity etc.) to measure quality is inefficient complexity, The use of single features of software to predict faults is uninformative. 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. In the past, several metrics for measuring software complexity and testing thoroughness have been proposed. Static metrics, e.g., the McCabe's cyclomatic number or the Halstead's Software Science, statically computed on the source code and tried to quantify software complexity. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. Decision tree algorithms begin with a set of cases, or examples and create a tree data structure that can be used to classify new cases. Each case described by a set of attributes, which can have numeric or symbolic values. Associated with each training case is a label representing by the name of a class. Each internal node of a decision tree contains a test, the result of which is used to decide what branch to follow from that node. The leaf nodes contain class labels instead of tests. In order to perform the analysis we validate the Performance of Decision Stump based decision Tree for the Software Defect Prediction and Study the Performance of M5 Model for the Software Defect Prediction. In the literature [3]-[17] various types of Fault-Proneness Estimation Models are discussed The paper is organized as follows: section 2 explains about the methodology followed and section 3 the result of the study. Finally conclusions of the research are presented in section 4.

2. METHODOLOGY FOLLOWED

The following are the steps used for the prediction of fault prone modules: First of all, find the Qualitative and Quantitative attributes of software systems i.e. software metrics. The real-time defect data sets are taken from data repository. Next step is to Select the suitable metric values as representation of statement and The suitable metrics

* Joint Registrar PTU Jallandhar.

like product requirement metrics and product module metrics out of these data sets are considered. The term product is used referring to module level data. The term metrics data applies to any finite numeric values, which describe measured qualities and characteristics of a product. The term product refers to anything to which defect data and metrics data can be associated. And the next step is to Analyze, refine metrics and normalize the metric values and Explore different Decision Tree based Techniques It is very important to find the suitable algorithm for modeling of software components into different levels of fault severity in software systems. The following three decision tree algorithms are experimented:

- Decision Stump based decision Tree
- M5 Model
- fast decision Tree learner, known as REPTree.

The next step is to comparisons are made on the basis of the more accuracy and least value of *MAE and RMSE* error values. *Accuracy value* of the prediction model is the major criteria used for comparison. The mean absolute error is chosen as the standard error. The technique having lower value of mean absolute error is chosen as the best fault prediction technique.

2.1 Mean Absolute Error

Mean absolute error, MAE is the average of the difference between predicted and actual value in all test cases; it is the average prediction error [13]. The formula for calculating MAE is given in equation 1.

$$\frac{|a_1 - c_1| + |a_2 - c_2| + \ldots + |a_n - c_n|}{n} \quad (1)$$

Assuming that the actual output is *a*, expected output is *c*.

2.2 Root Mean-squared Error

RMSE is frequently used measure of differences between values predicted by a model or estimator and the values actually observed from the thing being modeled or estimated [13]. It is just the square root of the mean square error as shown in equation 2.

$$\sqrt{\frac{(a_1 - c_1)^2 + (a_2 - c_2)^2 + \ldots + (a_n - c_n)^2}{n}} \quad (2)$$

The mean-squared error is one of the most commonly used measures of success for numeric prediction. This value is computed by taking the average of the squared differences between each computed value and its corresponding correct value. The root mean-squared error is simply the square root of the mean-squared-error. The root mean-squared error gives the error value the same dimensionality as the actual and predicted values.

The mean absolute error and root mean squared error is calculated for each machine learning algorithm i.e. various algorithms for Neural Networks.

3. RESULT AND DISCUSSION

The first step is to find the structural code and requirement attributes of software systems i.e. software metrics. The real-time defect data sets are taken from http://promisedata.org/repository. The Qualitative and quantitative dataset is about 31 projects completed in a consumer electronics company (one row per project). There is a mixture of qualitative attributes (these are measured on a 5 point ranked scale VL, L, M, H, VH) and quantitative attributes whose scale is stated [12].

3.1 Qualitative Factors

The Quantitative factors are grouped under five topics [12]:

- Specification and Documentation process
- New Functionality
- Design and Development process
- Testing and Rework
- Project Management

3.2 Quantitative Factors

The following are the Quantitative factors are [12]:

- *Software size:* the size, in KLOC of the developed code and the development language,
- *Effort:* development effort measured in person hours for the software development, from specification review to unit test.

The proposed Decision Tree based methodology is implemented in WEKA environment is one such facility which lends a high performance language for technical computing.

In case of Decision Stump algorithm after 10 fold cross validation the value of MAE and RMSE is calculated as is:

- Mean absolute error 303.42
- Root mean squared error 422.6276

In case of M5 pruned model tree using smoothed linear models the total number of faults can be calculated using following equation:

TD = 200.6171 * F1=VH
+ 258.9974 * D1=M,VL
+ 10.0736 * K
– 52.3791

Implements base routines for generating M5 Model trees and rules. The original algorithm M5 was invented

by *R*. Quinlan and Yong Wang made improvements. The following parameters are used for building the model:

- *buildRegressionTree:* Whether to generate a regression tree/rule instead of a model tree/rule.
- *debug:* If set to true, classifier may output additional info to the console.
- *minNumInstances:* The minimum number of instances to allow at a leaf node. The value is set to 4.
- *saveInstances:* Whether to save instance data at each node in the tree for visualization purposes.
- *unpruned:* Whether unpruned tree/rules are to be generated.
- *use Unsmoothed:* Whether to use unsmoothed predictions.

Time taken to build model was 0.14 seconds. In case of 10 fold cross validation the following MAE and RMSE values are obtained:

- Mean absolute error 226.023
- Root mean squared error 301.9067

Fast decision tree learner, builds a decision/ regression tree using information gain/variance and prunes it using reduced-error pruning (with back fitting). Only sorts values for numeric attributes once. Missing values are dealt with by splitting the corresponding instances into pieces (i.e. as in C4.5). In case of 10 fold cross validation the following MAE and RMSE values are obtained:

- Mean absolute error 381.126
- Root mean squared error 530.7604

The following parameter options are used for building the REPTree Model:

- *debug:* If set to true, classifier may output additional info to the console.
- *maxDepth:* The maximum tree depth (–1 for no restriction).
- *minNum:* The minimum total weight of the instances in a leaf.
- *minVarianceProp:* The minimum proportion of the variance on all the data that needs to be present at a node in order for splitting to be performed in regression trees.
- *noPruning:* Whether pruning is performed.
- *numFolds:* Determines the amount of data used for pruning. One fold is used for pruning, the rest for growing the rules.
- *seed:* The seed used for randomizing the data.

4. CONCLUSION

In this paper, we investigate whether qualitative and quantitative factors can be used to identify level of number of faulty software modules. We compare the performance of Decision Stump based decision Tree, M5 Model and fast decision Tree learner known as REPTree, for the fault dataset. M5 based Decision Tree shows best results than among other decision tree algorithms experimented in the study with lower values of MAE and RMSE calculated as 226.023 and 301.9067 values respectively. It is therefore, concluded the M5 Based Decision Tree model is implemented and the best algorithm for classification of the fault prone modules from the faultless modules of the software systems.

The future work can be extended in following directions:

- Most important attribute can be found for fault prediction and this work can be extended to further programming languages. More algorithms can be evaluated and then we can find the best algorithm.
- Further investigation can be done and the impact of attributes on the fault prediction can be found.
- Other dimensions of quality of software can be considered for mapping the relation of attributes and fault tolerance.

REFERENCES

[1] Seliya N., Khoshgoftaar T.M. and Zhong S. (2005), "Analyzing Software Quality with Limited Fault-proneness Defect Data", In *Proceedings of the Ninth IEEE International Symposium on High Assurance System Engineering*, Germany, pp. 89-98.

[2] Jiang Y., Cukic B. and Menzies T. (2007), "Fault Prediction Using Early Lifecycle Data", *ISSRE 2007, the 18th IEEE Symposium on Software Reliability Engineering*, IEEE Computer Society, Sweden, pp. 237-246.

[3] Bezdek J.C., Ehrlich R., and Full W. (1984), "FCM: Fuzzy c-means Algorithm", *Computers and Geoscience*, **10**, pp. 191-203.

[4] Khoshgoftaar T.M. and Munson J.C. (1990), "Predicting Software Development Errors using Complexity Metrics", *IEEE Journal on Selected Areas in Communications*, **8**, Issue: 2, pp. 253 -261.

[5] Pigoski M. and Nelson E. (1994), "Software Maintenance Metrics: A Case Study", *Proceedings of IEEE Conference on Software Maintenance*, Canada, pp. 392-401.

[6] Khoshgoftaar, T.M., Allen E.B., Ross F.D., Munikoti R., Goel N. and Nandi A. (1997), "Predicting Fault-prone Modules with Case-based Reasoning", *ISSRE 1997*, the Eighth International Symposium on Software Engineering, Mexico, pp. 27-35.

[7] Menzies T., Ammar K., Nikora A., and Stefano S. (2003), "How Simple is Software Defect Prediction?", *Journal of Empirical Software Engineering*, **32**, Issue: 2, pp.1156-1161.

[8] Eman K., Benlarbi S., Goel N., and Rai S. (2001), "Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components", *Journal of Systems Software*, **55**, Issue: 3, pp. 301-310.

[9] Munson J.C. and Khoshgoftaar T.M. (1992), "The Detection of Fault-prone Programs", *IEEE Transactions on Software Engineering*, **18**, Issue: 5, pp. 423-433.

[10] Yaun X., Khoshgoftaar M. and Allen B. (2000), "An Application of Fuzzy Clustering to Software Quality Prediction", *Information Sciences: An International Journal*, **179**, Issue: 8, pp. 1040-1058.

[11] Seliya N. and Khoshgoftaar M. (2007), "Software Quality Analysis of Unlabeled Program Modules with Semi Supervised Clustering", *Software Quality Journal*, **37**, Issue: 2, pp. 201-211.

[12] Challagula, Bastani B. and Yen (2006), "A Unified Framework for Defect Data Analysis using the MBR Technique", *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, Washington, pp. 39-46.

[13] Dav'e N. and Krishnapuram R. (1997), "Robust Clustering Methods: A Unified View", *IEEE Transactions on Fuzzy Systems*, **5**, Issue: 2, pp. 270–293.

[14] Saux B. and Boujemaa N. (2002), "Unsupervised Robust Clustering for Image Database Categorization", In *Proceedings of the IEEE-IAPR International Conference on Pattern Recognition (ICPR'2002)*, Turkey, pp. 259-262.

[15] Stark E. (1996), "Measurements for Managing Software Maintenance", *International Conference on Software Maintenance*, USA, pp. 152-161.

[16] Bellini P. (2005), "Comparing Fault-Proneness Estimation Models", *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, China, pp. 205-214.

[17] Lanubile F., Lonigro A., and Visaggio G. (1995), "Comparing Models for Identifying Fault-Prone Software Components", *Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering*, USA, pp. 12-19.

[18] Fenton N.E. and Neil M. (1999), "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, **25**, Issue: 5, pp. 675-689.

[19] Runeson, Wohlin C. and Ohlsson M.C. (2001), "A Proposal for Comparison of Models for Identification of Fault-Proneness", *Journal of System and Software*, **56**, Issue: 3, pp. 301–320.

[20] Deodhar M. (2002), "Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems", *Journal of System and Software*, **56**, Issue: 3, pp. 157-162.

[21] Ma Y. and Guo L. (2006), "A Statistical Framework for the Prediction of Fault-Proneness", *Product Focused Process Improvement*, Edition: First, Publisher: Springer Berlin/Heidelberg, pp. 204-214.

[22] Challagulla V.U.B., Bastani F.B., Yen I. L. and Paul (2005), "Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques", *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, USA, pp. 263-270.