A Bayes Network Classification Approach For Finding Faulty Modules In Open Source Software Systems

Aarti Mahajan, Vikas Gupta, Parvinder S. Sandhu

**Abstract**—Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control. There are many metrics and techniques available to investigate the accuracy of fault prone classes which may help software organizations for planning and performing testing activities. Bayes algorithms are being successfully applied for solving both classification and regression problems. It is therefore important to investigate the capabilities of Bayes Network Classification algorithm in predicting software quality. In order to perform the analysis we validate the performance of the Bayes Network based Algorithm using open source software JEdit. In this paper, we investigate the capability of a Bayes Network Algorithm in predicting faulty classes. We investigate the accuracy of the fault proneness predictions using object oriented design metrics suite. By using Bayes Network Algorithm technique on fault prone classes may enable the software organizations, for planning and performing testing by focusing on accuracy of fault prone classes. This may result in significant improvement in software quality.

**Keywords**— A bayes network classification approach, Software fault, Object Oriented Metrics.

I. INTRODUCTION

**FAULTS** in software systems continue to be a major problem. A software bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from behaving as intended (e.g., producing an incorrect result). A software fault is a defect that causes software failure in an executable product. In software engineering, the non-conformance of software to its requirements is commonly called a bug. Most bugs arise from mistakes and errors made by people in either a program's source code or its design, and a few are caused by compilers producing incorrect code. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and work. The possibility of early estimating the potential faultiness of software could help on planning, controlling and executing software development activities.

Software is said to contain a fault if for some input data the output is incorrect. For each execution of the software program where the output is incorrect, we observe a failure. Software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects but on the other hand fault is a hidden programming error that may or may not actually manifest as a failure. Many systems are delivered to users with excessive faults. This is despite a huge amount of development effort going into fault reduction in terms of quality control and testing. 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. A limited amount of valuable work in this area has been carried out previously. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control. Quality of software is increasingly important and testing related issues are becoming crucial for software. Although there is diversity in the definition of software quality, it is widely accepted that a project with many defects lacks quality. 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. Predictive modeling is the process by which a model is created or chosen to try to best predict the probability of an outcome. The objective of a fault-proneness model is to identify faulty classes and focus testing effort on them.

II. METHODOLOGY USED

First of all, find the structural code and design attributes of software systems. Thereafter, select the suitable metric values as representation of statement. Next step is to analyze, refine metrics and normalize the metric values. We used JEdit open source software in this study [2]. JEdit is a programmer's text editor developed using Java language. JEdit combines the functionality of Window, Unix, and MacOS text editors. It was
released as free software and the source code is available on [3]. JEdit includes 274 classes. The number of developers involved in this project was 144. The project was started in 1999. The number of bugs was computed using SVC repositories. The release point for the project was identified in 2002. The log data from that point to 2007 was collected. The header files in C++ were excluded in data collection. The word bug or fixed was counted. Details on bug collection process can be found in [4]. The following is the details of the metrics used in the classification process:


The 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.

**A. 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. The formula for calculating MAE is given in equation 7.

\[
\frac{|a_1-c_1|+|a_2-c_2|+\ldots+|a_n-c_n|}{n}
\]  

(1)

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

**B. 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. It is just the square root of the mean square error as shown in equation 2.

\[
\sqrt{\left(a_1-c_1\right)^2+\left(a_2-c_2\right)^2+\ldots+\left(a_n-c_n\right)^2}/n
\]  

(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 Bayes Network Classification algorithms.

To predict the results, we have also used confusion matrix. The confusion matrix has four categories: True positives (TP) are the modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) are the fault-free modules correctly labeled as such. False negatives (FN) refer to faulty modules incorrectly classified as fault-free modules.

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Real Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault</td>
<td>TP</td>
</tr>
<tr>
<td>No Fault</td>
<td>FN</td>
</tr>
</tbody>
</table>

The following set of evaluation measures are being used to find the results:

- Probability of Detection (PD), also called recall or specificity, is defined as the probability of correct classification of a module that contains a fault.

\[
PD = \frac{TP}{TP + FN}
\]  

(3)

- Probability of False Alarms (PF) is defined as the ratio of false positives to all non defect modules.

\[
PF = \frac{FP}{FP + TN}
\]  

(4)

### III. RESULTS AND DISCUSSIONS

The data is collected from [1] and the statistics of the metric data of the WMC, DIT, NOC, CBO, RFC, LCOM, NPM, LOC metrics is tabulated in Table II, III, IV, V, VI, VII, VIII and IX metrics respectively. The details of the number of Faulty and Non-Faulty Modules present in the dataset is shown in Table X.

<table>
<thead>
<tr>
<th>Table II</th>
<th>Statics of the WMC Metric Values in JEdit Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>Value</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>907</td>
</tr>
<tr>
<td>Mean</td>
<td>11.726</td>
</tr>
<tr>
<td>StdDev</td>
<td>31.292</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table III</th>
<th>Statics of the DIT Metric Values in JEdit Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>Value</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>907</td>
</tr>
<tr>
<td>Mean</td>
<td>11.726</td>
</tr>
<tr>
<td>StdDev</td>
<td>31.292</td>
</tr>
</tbody>
</table>
Bayes Network learning using various search algorithms and quality measures. The following parameters are used for running the program (as shown in fig. 2):

- **BIFFile** -- Set the name of a file in BIF XML format. A Bayes network learned from data can be compared with the Bayes network represented by the BIF file.
- **debug** -- If set to true, classifier may output additional info to the console.
- **estimator** -- Select Estimator algorithm for finding the conditional probability tables of the Bayes Network.
- **useADTree** -- When ADTree (the data structure for increasing speed on counts, not to be confused with the classifier under the same name) is used learning time goes
down typically. However, because ADTrees are memory intensive, memory problems may occur. Switching this option off makes the structure learning algorithms slower, and run with less memory. By default, ADTrees are used.

- searchAlgorithm -- Select method used for searching network structures. This Bayes Network learning algorithm uses a hill climbing algorithm restricted by an order on the variables.

![Fig. 2 Snapshot of the Bayes Network GUI Parameter setting](image)

IV. CONCLUSION AND FUTURE SCOPE

This study empirically evaluates performance of Bayes Network Classification technique in predicting fault-prone classes using open source software. The proposed Bayes Network technique based classification technique shows 70.8 percent accuracy. It also shows high value of Probability of detection (PD) for non faulty modules as compared to faulty modules as 0.75 and 0.664 values respectively. A lower value of Probability of False Alarms (PF) is indicated for the faulty modules as compared to non-faulty modules with 0.25 and 0.336 values respectively.

This study confirms that construction of Bayes Network classification model is feasible, adaptable to Object Oriented systems and useful in predicting faulty prone classes. It is therefore concluded that model is implemented using Bayes Network based technique for classification of the software components into faulty/fault-free systems is found satisfactory. The contributions of the study can be summarized as follows: First open source software systems analyzed. These systems are developed with different development methods than proprietary software. In previous studies mostly proprietary software were analyzed. Second, we examine Bayes Network method to predict the faulty classes with better accuracy.

The future work can be extended in following directions:

- More algorithms can be evaluated and then we can find the best algorithm. We plan to replicate our study to predict model based on hybrid genetic algorithms or soft computing techniques.

REFERENCES