A K-Means Based Clustering Approach for Finding Faulty Modules in Open Source Software Systems

Parvinder S. Sandhu, Jagdeep Singh, Vikas Gupta, Mandeep Kaur, Sonia Manhas, Ramandeep Sidhu

Abstract—Prediction of fault-prone modules provides one way to support software quality engineering. Clustering is used to determine the intrinsic grouping in a set of unlabeled data. Among various clustering techniques available in literature K-Means clustering approach is most widely being used. This paper introduces K-Means based clustering approach for software finding the fault proneness of the Object-Oriented systems. The contribution of this paper is that it has used Metric values of JEdit open source software for generation of the rules for the categorization of software modules in the categories of Faulty and non faulty modules and thereafter empirically validation is performed. The results are measured in terms of accuracy of prediction, probability of Detection and Probability of False Alarms.

Keywords—K-Means, Software Fault, Classification, Object Oriented Metrics.

I. INTRODUCTION

Faults in software systems continue to be a major problem. 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. 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). 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. 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.

Clustering is used to determine the intrinsic grouping in a set of unlabeled data. It is the process of organizing objects into groups whose members are similar in some way. Among various clustering techniques available in literature K-Means clustering approach is most widely being used. K-Means is an unsupervised clustering technique used to classify data in to K clusters. It is partitional clustering approach, each cluster is associated with a centroid (center point), each point is assigned to the cluster with the closest centroid, Number of clusters, K, must be specified. Hence, in this study, a K-Means Based Clustering Approach is used for finding faulty Modules in Open Source Software Systems. In order to perform the analysis we validate the performance of the K-Means based clustering method for dataset derived from open source software JEdit [1]. We investigate the accuracy of the fault proneness predictions using object oriented design using metrics suite given by Chidamber and Kemerer [2] and used in [3] for fault prediction. In the literature [3]-[17] various types of Fault-Proneness Estimation Models are discussed.

The paper is organized as follows: section II explains about the methodology followed and section III the result of the study. Finally conclusions of the research are presented in section IV.

II. METHODOLOGY FOLLOWED

The following are the steps used for the prediction of fault prone modules:

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 [18]. 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 [19]. 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 [20]. The following are the metrics used in the classification process:

- Coupling between Objects (CBO)
- Lack of Cohesion (LCOM)
- Number of Children (NOC)
- Depth of inheritance (DOI)
- Weighted Methods per Class (WMC)
- Response for a class (RFC)
- Number of Public Methods (NPM)
- Lines of Code (LOC)

The data collection is performed and subsequently the it is tried to use Correlation-based Feature Subset Selection and Chi-squared Ranking Filter is applied to find the attributes that are important for the prediction. Correlation-based Feature Subset Selection Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them [21]. Subsets of features that are highly correlated with the class while having low intercorrelation are preferred. Chi-squared Ranking Filter evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class.

Thereafter, the reduced number of attributes are given as input to the K-means clustering algorithm. As Clustering is a technique that divides data in to two or more clusters depending upon some criteria. As, in this study data is divided in to two clusters depending upon that whether they are fault free or fault prone. In the K-means technique Euclidean distance as well as Manhattan distance measures are experimented. If the Manhattan distance is used, then centroids are computed as the component-wise median rather than mean.

To predict the results, we have 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.

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.
  \[
  PD = \frac{TP}{TP + FN} \quad (1)
  \]

- **Probability of False Alarms (PF)** is defined as the ratio of false positives to all non-defect modules.
  \[
  PF = \frac{FP}{FP + TN} \quad (2)
  \]

Basically, PD should be maximum and PF should be minimum.

### Table 1

A CONFUSION MATRIX OF PREDICTION OUTCOMES

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

### III. RESULT AND DISCUSSION

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 II

STATICS OF THE WMC METRIC VALUES IN JEDIT DATA

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>607</td>
</tr>
<tr>
<td>Mean</td>
<td>11,728</td>
</tr>
<tr>
<td>StdDev</td>
<td>31,202</td>
</tr>
</tbody>
</table>

### Table III

STATICS OF THE DIT METRIC VALUES IN JEDIT DATA

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>7</td>
</tr>
<tr>
<td>Mean</td>
<td>2.496</td>
</tr>
<tr>
<td>StdDev</td>
<td>1.937</td>
</tr>
</tbody>
</table>

### Table IV

STATICS OF THE NOC METRIC VALUES IN JEDIT DATA

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>56</td>
</tr>
<tr>
<td>Mean</td>
<td>0.715</td>
</tr>
<tr>
<td>StdDev</td>
<td>5.1</td>
</tr>
</tbody>
</table>

### Table V

STATICS OF THE CBO METRIC VALUES IN JEDIT DATA

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>195</td>
</tr>
<tr>
<td>Mean</td>
<td>12,942</td>
</tr>
<tr>
<td>StdDev</td>
<td>24,121</td>
</tr>
</tbody>
</table>

### Table VI

STATICS OF THE RFC METRIC VALUES IN JEDIT DATA

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>848</td>
</tr>
<tr>
<td>Mean</td>
<td>174.978</td>
</tr>
<tr>
<td>StdDev</td>
<td>269.591</td>
</tr>
</tbody>
</table>
First, the dataset is evaluated using Correlation-based Feature Subset Selection using BestFirst Search. The parameters are:

- **locallyPredictive** -- Identify locally predictive attributes. Iteratively adds attributes with the highest correlation with the class as long as there is not already an attribute in the subset that has a higher correlation with the attribute in question. The default value is True in the experiment.
- **missingSeparate** -- Treat missing as a separate value. Otherwise, counts for missing values are distributed across other values in proportion to their frequency. The default value False is used in the experiment.

BestFirst searches the space of attribute subsets by greedy hillclimbing augmented with a backtracking facility. Setting the number of consecutive non-improving nodes allowed controls the level of backtracking done. Best first may start with the empty set of attributes and search forward, or start with the full set of attributes and search backward, or start at any point and search in both directions (by considering all possible single attribute additions and deletions at a given point).

The parameters used are:

- **direction** -- Set the direction of the search. The default value ‘forward’ is used.
- **lookupCacheSize** -- Set the maximum size of the lookup cache of evaluated subsets. This is expressed as a multiplier of the number of attributes in the data set. It is set to 1.
- **searchTermination** -- Set the amount of backtracking. It is set to 5.

The figure 2 shows the results after applying Correlation-based Feature Subset Selection using BestFirst Search. It has proposed the use of DIT, CBO, RFC, NPM and LOC metric as significant metrics for the prediction.

In case of Chi-squared Ranking Filter selection Ranks attributes by their individual evaluations. Use in conjunction with attribute evaluators (ReliefF, GainRatio, Entropy etc). The following parameters are used:

**Search Method**

- **Best-first**
- **Start set: no attributes**
- **Search direction: forward**
- **Stop after 5 node expansions**
- **Total number of subsets evaluated: 43**
- **Best subset found: 0.226**

**Selected attributes: 2,4,5,7,8 : 5**

- **DIT**
- **CBO**
- **RFC**
- **NPM**
- **LOC**

- **Fig. 1 Graphical Representation of the Statistics of the Metric Data of the Attributes of Dataset**

- **Fig. 3 Snapshot of the Output of Correlation-based Feature Subset Selection using BestFirst Search**
• binarizeNumericAttributes -- Just binarize numeric attributes instead of properly discretizing them. The false value of this parameter is used.
• missingMerge -- Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value. The false value of this parameter is used.
• generateRanking -- A constant option. Ranker is only capable of generating attribute rankings. The true value of this parameter is used.
• numToSelect -- Specify the number of attributes to retain. The default value (-1) indicates that all attributes are to be retained. Use either this option or a threshold to reduce the attribute set.
• startSet -- Specify a set of attributes to ignore. When generating the ranking, Ranker will not evaluate the attributes in this list. This is specified as a comma separated list of attribute indexes starting at 1. It can include ranges. E.g. 1,2,5-9,17.
• threshold -- Set threshold by which attributes can be discarded. Default value results in no attributes being discarded. Use either this option or numToSelect to reduce the attribute set. The Default value -1.7976931348623157E308 is used in the experimentation.

Snapshot of the Output of Chi-squared Ranking Filter is shown in figure 3. It shows that rank of the five attributes recommended by the CFS is better than other attributes. So, we selected the 5 attributes as selected by CFS algorithm

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, class (nominal): 9 Bug count): Chi-squared Ranking Filter

Ranked attributes:
65, 872 8 LOC
48, 86 5 RFC
47, 516 2 E2T
45, 46 4 CBO
35, 243 7 NPM
33, 811 1 WMC
17, 4 6 LCOM
0 3 NOC

Selected attributes: 8,5,2,47,1,6,3 : 8

Fig. 3 Snapshot of the Output of Chi-squared Ranking Filter

Thereafter, K-Means algorithm is applied on the reduced dataset. First of all the Euclidean distance is used and results are shown in figure 4 and values of the cluster centroids are tabulated in table XI. Second, ManhattanDistance is used and cluster centroids are tabulated in table XII. Manhattan distance (or Taxicab geometry) is the distance between two points is the sum of the (absolute) differences of their coordinates [22]. In both the cases the following parameters are used:

• distanceFunction -- The distance function to use for instances comparison. First set to EuclideanDistance and thereafter set to ManhattanDistance.
• dontrReplaceMissingValues -- Replace missing values globally with mean/mode. Default value False is used.
• maxIterations -- Set maximum number of iterations. It is set to 500.
• numClusters -- Set number of clusters. It is set to 2.
• preserveInstancesOrder -- Preserve order of instances. It is set to False.
• seed -- The random number seed to be used. It is set to 10.

Fig. 4 Snapshot of the Output Kmeans algorithm using Euclidean Distance

<table>
<thead>
<tr>
<th>Cluster centroids:</th>
<th>Cluster#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Full Data</td>
</tr>
<tr>
<td>DIC</td>
<td>2.4964</td>
</tr>
<tr>
<td>CBO</td>
<td>12.6423</td>
</tr>
<tr>
<td>RFC</td>
<td>174.9731</td>
</tr>
<tr>
<td>NPM</td>
<td>7.7847</td>
</tr>
<tr>
<td>LOC</td>
<td>205.2117</td>
</tr>
</tbody>
</table>

TABLE XI
K-MENAS CLUSTER CENTROIDS USING EUCLIDEAN DISTANCE
Table II

K-Means Cluster Centroids Using Manhattan Distance

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Full Data</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>CBO</td>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>RFC</td>
<td>33</td>
<td>553</td>
<td>24</td>
</tr>
<tr>
<td>NFM</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>LOC</td>
<td>76.5</td>
<td>134</td>
<td>59</td>
</tr>
</tbody>
</table>

The 61 (means 22%) Clustered Instances belongs to cluster 0 and 213 (means 78%) instances belongs to cluster 1. In both the cases the same confusion matrix is recorded as shown in Table XIII.

Table III

Recorded Confusion Matrix of Prediction Outcomes

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Real Data</th>
<th>Fault</th>
<th>No Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault</td>
<td>46</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>No Fault</td>
<td>15</td>
<td>125</td>
<td></td>
</tr>
</tbody>
</table>

The Accuracy of classification, Probability of detection (PD) and Probability of False Alarms (PF) values are 62.4% 0.754 and 0.413 respectively.

IV. CONCLUSION

This paper empirically evaluates performance of Kmeans based clustering technique in predicting fault-prone classes using open source software. The proposed KMeans based classification technique shows 62.4 percent accuracy. It also shows high value of Probability of detection (PD) i.e. 0.754 and low value of Probability of False Alarms (PF) i.e. 0.413.

This study confirms that construction of Kmeans based model is feasible, adaptive to Object Oriented systems and useful in predicting faulty prone classes. It is therefore concluded that model is implemented using Kmeans 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 K-Means clustering method to predict the faulty classes with better accuracy.

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. We plan to replicate our study to predict model based on hybrid genetic algorithms or soft computing techniques.

REFERENCES