Software Defect Prediction from Historical Data

K.B.S Sastry, Lecturer in Computer Science, Andhra Loyola College, Vijayawada, India
Dr.B.V.Subba Rao, Professor in IT, PVP Siddhartha Inst. of Technology, Vijayawada, Andhra Pradesh, India
Dr K.V.Sambasiva Rao, Principal, MVR College of Engg. and Technology, Paritala, Vijayawada, India

Abstract—Software Testing Consumes major percentage of project cost, so researchers focuses of the “How to minimizes cost of testing in order to minimize the cost of the project”. The Software defect prediction is a method which predicts defects from historical database. Data mining Techniques are used to predict Software defects from historical databases. This paper describes frame work to produce software defect from the historical database and also present one pass data mining algorithm used find rules to predict software defects. The experimental results shows that, one pass algorithm generate rules for software defect prediction with consider amount of time and with better performance.

Keywords—Testing, defect, Data mining, cost, database

I. INTRODUCTION

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. A set of activities conducted with the intent of finding errors in software. Testing is a process used to help identify the correctness, completeness and quality of developed computer software. A Software Defect / Bug is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectations (which may not be specified but are reasonable). In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results. Current defect prediction work focuses on (i) estimating the number of defects remaining in software systems, (ii) discovering defect associations, and (iii) classifying the defect-proneness of software components, typically into two classes defect-prone and not defect-prone. The second type of work borrows association rule mining algorithms from the data mining community to reveal software defect associations, which can be used for three purposes. The third type of work classifies software components as defect-prone and non-defect-prone by means of metric-based classification. Being able to predict which components are more likely to be defect-prone supports better targeted testing resources and therefore improved efficiency.

Data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those item sets whose occurrences exceed a predefined threshold in the database; those item sets are called frequent or large item sets. The design and study of one-pass algorithms has a long tradition in many areas of computer science. For example, they are used in the area of data stream processing, where streams of huge amounts of data have to be monitored on-the-fly without prior storing the entire data. But also, e.g., a deterministic finite automaton on words can be viewed as a (very simple) example of a one-pass algorithm whose memory size and processing time per data item is constant, i.e., does not depend on the input size. For most computational problems, however, the amount of memory necessary for solving the problem grows with increasing input size. The remainder of the paper is organized as follows. Section 2 provides related work. Section 3 describes problem description. Section 4 is devoted proposed framework and one pass algorithm. In Section 5 results are documented. Conclusions and consideration of the significance of this work are given in the final section.

II. RELATED WORK

MGF [1] published a study in this journal in 2007 in which they compared the performance of two machine learning techniques (Rule Induction and Naïve Bayes) to predict software components containing defects. Hall and Holmes[2] concluded that the forward selection search was well suited to Naïve Bayes but the backward elimination search is more suitable for C4.5. Cardie [3] found using a decision tree to select attributes helped the nearest neighbor algorithm to reduce its prediction error. Kebat et al. [4] used a decision tree . That is, which attribute subset is more useful for defect prediction not only depends on the attribute subset itself but also on the specific data set. This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Filtering attributes for use with a Naïve Bayesian classifier and obtained a similar result. However, Kibler and Aha [5] reported more mixed results on two medical Classification tasks. Therefore, before building prediction models, we should choose the combination of all three of learning algorithm, data pre-processing and attribute
selection method, not merely one or two of them. Lessmann et al. [6] have also conducted a follow-up to MGF on defect predictions, providing additional results as well as suggestions for a methodological framework. However, they did not perform attribute selection when building prediction models. Thus this work has wider application.

III. PROBLEM DESCRIPTION

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those item sets whose occurrences exceed a predefined threshold in the database; those item sets are called frequent or large item sets. The second problem is to generate association rules from those large item sets with the constraints of minimal confidence. Suppose one of the large item sets is L_k, L_k = \{I_1, I_2, \ldots , I_k\}, association rules with this item sets are generated in the following way: the first rule is \{I_1, I_2, \ldots , I_k -1\} \Rightarrow \{I_k\}, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent, further the confidences of the new rules are checked to determine the interestingness of them. Those processes iterated until the antecedent becomes empty. Since the second sub problem is quite straightforward, most of the researches focus on the first sub problem.

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Features</th>
<th>Instances</th>
<th>Recorded Values</th>
<th>% Defective Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>C</td>
<td>40</td>
<td>505</td>
<td>20200</td>
<td>19</td>
</tr>
<tr>
<td>JM1</td>
<td>C</td>
<td>21</td>
<td>10878</td>
<td>228438</td>
<td>19</td>
</tr>
<tr>
<td>KC1</td>
<td>C++</td>
<td>21</td>
<td>2107</td>
<td>44247</td>
<td>15</td>
</tr>
<tr>
<td>KC3</td>
<td>Java</td>
<td>40</td>
<td>458</td>
<td>18320</td>
<td>9</td>
</tr>
<tr>
<td>KC4</td>
<td>Perl</td>
<td>40</td>
<td>125</td>
<td>5000</td>
<td>49</td>
</tr>
<tr>
<td>MC1</td>
<td>C &amp; C++</td>
<td>39</td>
<td>9466</td>
<td>369174</td>
<td>0.7</td>
</tr>
<tr>
<td>MC2</td>
<td>C</td>
<td>40</td>
<td>161</td>
<td>6440</td>
<td>32</td>
</tr>
<tr>
<td>MW1</td>
<td>C</td>
<td>40</td>
<td>403</td>
<td>16120</td>
<td>8</td>
</tr>
<tr>
<td>PC1</td>
<td>C</td>
<td>40</td>
<td>1107</td>
<td>44280</td>
<td>7</td>
</tr>
<tr>
<td>PC2</td>
<td>C</td>
<td>40</td>
<td>5589</td>
<td>223560</td>
<td>0.4</td>
</tr>
<tr>
<td>PC3</td>
<td>C</td>
<td>40</td>
<td>1563</td>
<td>62520</td>
<td>10</td>
</tr>
<tr>
<td>PC4</td>
<td>C++</td>
<td>39</td>
<td>1458</td>
<td>58320</td>
<td>12</td>
</tr>
</tbody>
</table>

Fig 1. Statics of defect in software

Let I=I_1, I_2, \ldots, I_m be a set of m distinct attributes, T be transaction that contains a set of items such that T \subseteq I, D be a database with different transaction records T_s. An association rule is an implication in the form of X \Rightarrow Y, where X, Y \subseteq I are sets of items called item sets, and X \cap Y =\emptyset. X is called antecedent while Y is called consequent, the rule means X implies Y. There are two important basic measures for association rules, support(s) and confidence(c). Since the database is large and users concern about only those frequently purchased items, usually thresholds of support and confidence are predefined by users to drop those rules that are not so interesting or useful. The two thresholds are called minimal support and minimal confidence respectively. Support(s) of an association rule is defined as the percentage/fraction of records that contain X \cup Y to the total number of records in the database. Suppose the support of an item is 0.1\%, it means only 0.1 percent of the transaction contain purchasing of this item. Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain X \cup Y to the total number of records that contain X. Confidence is a measure of strength of the association rules, suppose the confidence of the association rule X \Rightarrow Y is 80\%, it means that 80\% of the transactions that contain X also contain Y together.

IV. PROPOSED SOLUTION

1) Framework
The framework consists of two components: (i) scheme evaluation and (ii) defect prediction. Historical data are divided into two parts: training set for building learners with the given learning schemes, and a test set for evaluating the performances of the learners. It is very important that the test data are not used in any way to build the learners. This is a necessary condition to assess the generalization ability of a learner that is built according to a learning scheme, and further to determine whether or not to apply the learning scheme, or select one best scheme from the given schemes. Defect prediction stage, according to the performance report of the first stage, a learning scheme is selected and used to build a prediction model and predict software defect.
2) **One pass Algorithm**

**ALG Evaluation** (historical Data, scheme)

**Input:** historical Data - the historical data;  
Scheme - the learning scheme.  
**Output:** AvgResult - the mean performance over the M×N-way cross-validation.

```plaintext
1 M = 10; /*number of repetitions */  
2 N = 10; /*number of folds */  
3 repeat  
4 D = Randomize (historical Data); /*randomize the order of instances */  
5 Generate N bins from D;  
6 for i = 1 to N do  
7    test = bin[i];  
8    train = D − test;  
9    [learner, bestAttrs ] = Learning (train, scheme);  
10   test' = select bestAttrs from test ;  
11   Result = Test Classifier (test', learner); /*Compute the performance measures of the learner on data test' */  
12 end  
13 until M times ;  
AvgResult = 1/M×N  
14 Results;
```

**Algorithm Learning** (data, scheme)

**Input:** data - the data on which the learner is built;  
Scheme - the learning scheme.  
**Output:** learner - the final learner built on data with scheme;  
bestAttrs - the best attribute subset selected by the attribute selector of scheme

```plaintext
1 m = 10; /*number of repetitions for attribute selection */  
2 n = 10; /*number of folds for attribute selection */  
3 d = Preprocessing (data, scheme, preprocessor);  
4 bestAttr = AttrSelect (d, scheme. Algorithm, scheme.attrSelector, m, n);  
5 d' = select bestAttr from d ;  
6 learner = BuildClassifier (d', scheme. Algorithm); /*build a classifier on d' with the learning algorithm of scheme */  
**Algorithm Prediction**(historical Data, new Data, scheme)

**Input:** historical Data - the historical data; new Data - the new data;  
Scheme - the learning scheme.  
**Output:** Result - the predicted result for the new Data

```plaintext
1 [predictor, bestAttrs ] = Learning (historical Data, scheme);  
2 d = select bestAttrs from new Data;
```
3 Result = Predict (d, predictor);
/*predict the class label of d with predictor

V. CONCLUSION

This paper, presented a novel benchmark framework for software defect prediction. The framework involves evaluation and prediction. In the evaluation stage, different learning schemes are evaluated and the best one is selected. Then in the prediction stage, the best learning scheme is used to build a predictor with all this article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Historical data and the predictor is finally used to predict defect on the new data. This paper described framework to produce software defects from the historical database and also presented one pass data mining algorithm used to find rules to predict software defects.

REFERENCES