APPENDIX
Software Defect Prediction using Classifier Mining

K.B.S. Sastry¹, Dr. R. Satya Prasad²

¹Lecturer, Dept. of Computer Science, Andhra Loyola College, Vijayawada, sastrykbs@gmail.com
²Associate Professor, Dept. of Computer Science, Acharya Nagarjuna University, Guntur, profrsp@gmail.com

There has been rapid growth of software development. Due to various causes, the software comes with many defects. In Software development process, testing of software is the main phase which reduces the defects of the software. If a developer or a tester can predict the software defects properly then, it reduces the cost, time and effort. In this paper, we show a comparative analysis of software defect prediction based on classification rule mining. We propose a scheme for this process and we choose different classification algorithms. Showing the comparison of predictions in software defects analysis. This evaluation analyzes the prediction performance of competing learning schemes for given historical data sets (NASA MDP Data Set). The result of this scheme evaluation shows that we have to choose different classifier rule for different data set.

Keywords: Software defect prediction, classification Algorithm, Confusion matrix

1.1 Mining for software Engineering

To improve the software productivity and quality, software engineers are applying data mining algorithms to various SE tasks. Many algorithms can help engineers figure out how to invoke API methods provided by a complex library or framework with insufficient documentation. In terms of maintenance, such type of data mining algorithms can assist in determining what code locations must be changed when another code location is changed. Software engineers can also use data mining algorithms to hunt for potential bugs that can cause future in-field failures as well as identify buggy lines of code (LOC) responsible for already-known failures. The second and third columns of Table 2.1 list several example data mining algorithms and the SE tasks to which engineers apply them [1].
Proposed Scheme

2.1 Overview of the Framework

In General, before building defect prediction model and using them for prediction purposes, we first need to decide which learning scheme or learning algorithm should be used to construct the model. Thus, the predictive performance of the learning scheme should be determined, especially for future data. However, this step is often neglected and so the resultant prediction model may not be Reliable. As a consequence, we use a software defect prediction framework that provides guidance to address these potential shortcomings.

The framework consists of two components:
1) scheme evaluation and
2) defect prediction.

Figure 2.1 contains the details. At the scheme evaluation stage, the performances of the different learning schemes are evaluated with historical data to determine whether a certain learning scheme performs sufficiently well for prediction purposes or to select the best from a set of competing schemes.

From Figure 2.1, we can see that the historical data are divided into two parts: a training set for building learners with the given learning schemes, and a test set.
Figure 2.1: Proposed framework 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 to further determine whether or not to apply the learning scheme or select one best scheme from the given schemes.

At the defect prediction stage, according to the performance report of the rst stage, a learning scheme is selected and used to build a prediction model and predict software defect. From Fig. 2.1, we observe that all of the historical data are used to build the predictor here. This is very different from the first stage; it is very useful for improving the generalization ability of the predictor. After the predictor is built, it can be used to predict the defect-proneness of new software components.

MGF proposed [5] a baseline experiment and reported the performance of the Naive Bayes data miner with log-filtering as well as attribute selection, which performed the scheme evaluation but with in appropriate data. This is because they used both the training (which can be viewed as historical data) and test (which can be viewed as new data) data to rank attributes, while the labels of the new data are unavailable when choosing attributes in practice.

2.2 Scheme Evaluation

The scheme evaluation is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated by building and evaluating learners with them. The first problem of scheme evaluation is how to divide historical data into training and test data. As mentioned above, the test data should be independent of the learner construction. This is a necessary precondition to evaluate the performance of a learner for new data. Cross-validations usually used to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a data set into complementary subsets, performing the analysis on one subset, and validating the analysis on the other subset. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

In our framework, an percentage split used for estimating the performance of each predictive model, that is, each data set is first divided into 2 parts, and after that a
predictor is learned on 60% instances, and then tested on the remaining 40%. To overcome any ordering effect and to achieve reliable statistics, each holdout experiment is also repeated M times and in each repetition the data sets are randomized. So overall, M*N(N=Data sets) models are built in all during the period of evaluation; thus M*N results are obtained on each data set about the performance of the each learning scheme.

After the training-test splitting is done each round, both the training data and learning scheme(s) are used to build a learner. A learning scheme consists of a data preprocessing method, an attribute selection method, and a learning algorithm.

Evaluation of the proposed framework is comprised of:

1. A data preprocessor

   - The training data are preprocessed, such as removing outliers, handling missing values, and discretizing or transforming numeric attributes.

   - Here Preprocessor used-NASA Preprocessing Tool

2. An attribute selector

   - Here we have considered all the attributes provided by the NASA MDP Data Set.

3. Learning Algorithms

   { NaiveBayseSimple from bayse classification

   { Logistic classification

   { From Rule based classification {

       DecisionTable

       { OneR

       { JRip {

       PART

   { From Tree based classification{ {

       J48
2.3 Scheme Evaluation Algorithm

Data: Historical Data Set
Result: The mean performance values

1. M=12 : No of Data Set
2. i=1;
3. while i<=M do
   4. Read Historical Data Set D[i];
   5. Split Data set Instances using % split;
   6. Train[i]=60% of D; % Training Data;
   7. Learning(Train[i],scheme);
   8. Test Data=D[i]-Train[i];% Test Data;
   9. Result=TestClassifier(Test[i],Learner);
10. end

Algorithm 1: Scheme Evaluation

2.4 Defect prediction

The defect prediction part of our framework is straightforward; it consists of predictor construction and defect prediction. During the period of the predictor construction:

1. A learning scheme is chosen according to the Performance Report.
2. A predictor is built with the selected learning scheme and the whole historical data. While evaluating a learning scheme, a learner is built with the training data and tested on the test data. Its final performance is the mean over all rounds. This reveals that the evaluation indeed covers all the data. Therefore, as we use all of the historical data to build the predictor, it is expected that the constructed predictor has stronger generalization ability.
3. After the predictor is built, new data are preprocessed in same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.
2.5 Difference between Our Framework and Others

So, to summarize, the main difference between our framework and that of others in the following:

1) We choose the entire learning scheme, not just one out of the learning algorithm, attribute selector, or data preprocessor;
2) We use the appropriate data to evaluate the performance of a scheme.
3) We choose percentage split for training data set(60%) and test dataset(40%).

2.6 Data Set

We used the data taken from the public NASA MDP repository, which was also used by MGF and many others, e.g., [10], [11], [12], [13]. Thus, there are 12 data sets in total from NASA MDP repository.

Table 3.1, and 3.2 provides some basic summary information. Each data set is comprised of a number of software modules (cases), each containing the corresponding number of defects and various software static code attributes. After preprocessing, modules that contain one or more defects were labeled as defective. A more detailed description of code attributes or the origin of the MDP data sets can be obtained from [5].

<table>
<thead>
<tr>
<th>Data Set</th>
<th>System</th>
<th>Language</th>
<th>Total Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1-5</td>
<td>Spacecraft Instrument</td>
<td>C</td>
<td>17K</td>
</tr>
<tr>
<td>KC3-4</td>
<td>Storage management for ground data</td>
<td>JAVA</td>
<td>8K and 25K</td>
</tr>
<tr>
<td>KC1-2</td>
<td>Storage management for ground data</td>
<td>C++</td>
<td>*</td>
</tr>
<tr>
<td>MW1</td>
<td>Database</td>
<td>C</td>
<td>8K</td>
</tr>
<tr>
<td>PC1,2,5</td>
<td>Flight Software for Earth orbiting Software</td>
<td>C</td>
<td>26K</td>
</tr>
<tr>
<td>PC3,4</td>
<td>Flight Software for Earth orbiting Software</td>
<td>C</td>
<td>30-36K</td>
</tr>
</tbody>
</table>
Table 2.2: Data Sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Attribute</th>
<th>Module</th>
<th>Defect</th>
<th>Defect(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>38</td>
<td>344</td>
<td>42</td>
<td>1.22</td>
</tr>
<tr>
<td>JM1</td>
<td>22</td>
<td>9593</td>
<td>1759</td>
<td>18.34</td>
</tr>
<tr>
<td>KC1</td>
<td>22</td>
<td>2096</td>
<td>325</td>
<td>15.5</td>
</tr>
<tr>
<td>KC3</td>
<td>40</td>
<td>200</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>MC1</td>
<td>39</td>
<td>9277</td>
<td>68</td>
<td>0.73</td>
</tr>
<tr>
<td>MC2</td>
<td>40</td>
<td>127</td>
<td>44</td>
<td>34.65</td>
</tr>
<tr>
<td>MW1</td>
<td>38</td>
<td>264</td>
<td>27</td>
<td>10.23</td>
</tr>
<tr>
<td>PC1</td>
<td>38</td>
<td>759</td>
<td>61</td>
<td>8.04</td>
</tr>
<tr>
<td>PC2</td>
<td>37</td>
<td>1585</td>
<td>16</td>
<td>1.0</td>
</tr>
<tr>
<td>PC3</td>
<td>38</td>
<td>1125</td>
<td>140</td>
<td>12.4</td>
</tr>
<tr>
<td>PC4</td>
<td>38</td>
<td>1399</td>
<td>178</td>
<td>12.72</td>
</tr>
<tr>
<td>PC5</td>
<td>39</td>
<td>17001</td>
<td>503</td>
<td>2.96</td>
</tr>
</tbody>
</table>

2.7 Performance Measurement

The Performance measured according to the Confusion matrix given in table 2.3, which is used by many researchers e.g. [14], [5]. Table 2.3 illustrates a confusion matrix for a two class problem having positive and negative class values.
Table 2.3: Confusion Matrix

<table>
<thead>
<tr>
<th>Predicted Class</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td>True Positive</td>
<td>False Negative</td>
</tr>
<tr>
<td>Negative</td>
<td>False Positive</td>
<td>True Negative</td>
</tr>
</tbody>
</table>

Software defect predictor performance of the proposed scheme based on Accuracy, Sensitivity, Specificity, Balance, ROC Area defined as

\[
\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}
\]

\[
= \frac{True Positive + True Negative}{True Positive + False Positive + True Negative + False Negative}
\]

\[
= \text{The percentage of prediction that is correct.}
\]

\[
pd = \text{True Positive Rate (tpr) = Sensitivity} = \frac{TP}{TP + FN}
\]

\[
= \text{The percentage of positive labeled instances that predicted as positive}
\]

\[
\text{Specificity} = \frac{TN}{FP + TN}
\]

\[
= \text{The percentage of positive labeled instances that predicted as negative.}
\]

- \( pf = \text{False Positive Rate (fpr) = 1 - specificity} \)

\[
= \text{The percentage of Negative labeled instances that predicted as negative}
\]

Formal definitions for pd and pf are given in the formula. Obviously, higher pds and lower pfs are desired. The point (pd=1, pf=0) is the ideal position where we recognize all defective modules and never make mistakes.

MGF introduced a performance measure called balance, which is used to choose the optimal (pd, pf) pairs. The definition is shown below from which we can see that it is equivalent to the normalized euclidean distance from the desired point (0, 1) to (pf, pd) in a ROC curve.
The receiver operating characteristic (ROC) [15] [28], curve is often used to evaluate the performance of binary predictors. A typical ROC curve is shown in Fig. 2.2. The y-axis shows probability of detection (pd) and the x-axis shows probability of false alarms (pf).

Formal definitions for pd and pf are given above. Obviously, higher pds and lower pfs are desired. The point (pf=0, pd=1) is the ideal position where we recognize all defective modules and never make mistakes.

\[ \text{Balance} = 1 - \frac{\sqrt{(1-pd)^2 + (0-pf)^2}}{\sqrt{2}} \]

The Area Under ROC Curve (AUC) is often calculated to compare different ROC curves. Higher AUC values indicate the classifier is, on average, more to the upper left region of the graph. AUC represents the most informative and commonly used, thus it is used as another performance measure in this paper.

Figure 2.2: Scheme evaluation of the proposed framework
3. Result Discussion

This section provides simulation results of some of the Classification algorithm techniques collected by simulation on Software tool named weka (version 3.6.9). In the thesis, however, proposed schemes are more comprehensively compared with competent schemes.

According to best accuracy value we choose 8 classification algorithm among many classification algorithms. All the evaluated values are collected and compare with different performance measurement parameter.

3.1 Accuracy

From the accuracy table 3.1 we can see different algorithm giving different accuracy on different dataset. But the average performance nearly same.

For Storage management software (KC1-3) LOG, J48G giving better Accuracy value. For database software written in c programming language (MW1) only PART giving better accuracy value.

The performance graph is given in the figure 3.3.

Table 3.1: Accuracy

<table>
<thead>
<tr>
<th>Methods</th>
<th>NB</th>
<th>LOG</th>
<th>DT</th>
<th>JRip</th>
<th>OneR</th>
<th>PART</th>
<th>J48</th>
<th>J48G</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>83.94</td>
<td>87.68</td>
<td>89.13</td>
<td>86.23</td>
<td>89.13</td>
<td>73.91</td>
<td>86.23</td>
<td>86.96</td>
</tr>
<tr>
<td>JM1</td>
<td>81.28</td>
<td>82.02</td>
<td>81.57</td>
<td>81.42</td>
<td>79.67</td>
<td>81.13</td>
<td>79.8</td>
<td>79.83</td>
</tr>
<tr>
<td>KC1</td>
<td>83.05</td>
<td>86.87</td>
<td>84.84</td>
<td>84.84</td>
<td>83.29</td>
<td>83.89</td>
<td>85.56</td>
<td>85.56</td>
</tr>
<tr>
<td>KC3</td>
<td>77.5</td>
<td>71.25</td>
<td>75</td>
<td>76.25</td>
<td>71.25</td>
<td>81.25</td>
<td>80</td>
<td>82.5</td>
</tr>
<tr>
<td>MC1</td>
<td>94.34</td>
<td>99.27</td>
<td>99.25</td>
<td>99.22</td>
<td>99.3</td>
<td>99.19</td>
<td>99.3</td>
<td>99.3</td>
</tr>
<tr>
<td>MC2</td>
<td>66</td>
<td>66.67</td>
<td>56.86</td>
<td>56.86</td>
<td>56.86</td>
<td>70.59</td>
<td>52.94</td>
<td>54.9</td>
</tr>
<tr>
<td>MW1</td>
<td>79.25</td>
<td>77.36</td>
<td>85.85</td>
<td>86.79</td>
<td>85.85</td>
<td>88.68</td>
<td>85.85</td>
<td>85.85</td>
</tr>
<tr>
<td>PC1</td>
<td>88.82</td>
<td>92.11</td>
<td>92.43</td>
<td>89.14</td>
<td>91.45</td>
<td>89.8</td>
<td>87.83</td>
<td>88.49</td>
</tr>
<tr>
<td>PC2</td>
<td>94.29</td>
<td>99.05</td>
<td>99.37</td>
<td>99.21</td>
<td>99.37</td>
<td>99.37</td>
<td>98.9</td>
<td>98.9</td>
</tr>
<tr>
<td>PC3</td>
<td>34.38</td>
<td>84.67</td>
<td>80.22</td>
<td>82.89</td>
<td>82.89</td>
<td>82.67</td>
<td>82.22</td>
<td>83.56</td>
</tr>
<tr>
<td>PC4</td>
<td>87.14</td>
<td>91.79</td>
<td>90.18</td>
<td>90.36</td>
<td>90.18</td>
<td>88.21</td>
<td>88.21</td>
<td>88.93</td>
</tr>
<tr>
<td>PC5</td>
<td>96.56</td>
<td>96.93</td>
<td>97.01</td>
<td>97.28</td>
<td>96.9</td>
<td>96.93</td>
<td>97.13</td>
<td>97.16</td>
</tr>
</tbody>
</table>

VOL 2 ISSUE 4 APRIL 2015 Paper 32
Sensitivity

From the accuracy table 3.2 we see that NB algorithm gives better performance in maximum data set.

In case of DecisionTable gives the sensitivity zero(sometimes), that means it considering all the class as a true negative. It can not be consider for defect prediction. LOG, OneR, PART, J48, J48G algorithms giving average performance.

<table>
<thead>
<tr>
<th>Methods</th>
<th>NB</th>
<th>LOG</th>
<th>DT</th>
<th>JRip</th>
<th>OneR</th>
<th>PART</th>
<th>J48</th>
<th>J48G</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>0.4</td>
<td>0.267</td>
<td>0</td>
<td>0.2</td>
<td>0.133</td>
<td>0.333</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>JM1</td>
<td>0.198</td>
<td>0.102</td>
<td>0.07</td>
<td>0.157</td>
<td>0.109</td>
<td>0.03</td>
<td>0.131</td>
<td>0.123</td>
</tr>
<tr>
<td>KC1</td>
<td>0.434</td>
<td>0.238</td>
<td>0.197</td>
<td>0.328</td>
<td>0.254</td>
<td>0.32</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>KC3</td>
<td>0.412</td>
<td>0.412</td>
<td>0.118</td>
<td>0.118</td>
<td>0.176</td>
<td>0.353</td>
<td>0.353</td>
<td>0.353</td>
</tr>
<tr>
<td>MC1</td>
<td>0.548</td>
<td>0.161</td>
<td>0.194</td>
<td>0.161</td>
<td>0.161</td>
<td>0.194</td>
<td>0.161</td>
<td>0.161</td>
</tr>
<tr>
<td>MC2</td>
<td>0.571</td>
<td>0.545</td>
<td>0</td>
<td>0</td>
<td>0.091</td>
<td>0.5</td>
<td>0.045</td>
<td>0.045</td>
</tr>
<tr>
<td>MW1</td>
<td>0.429</td>
<td>0.286</td>
<td>0.429</td>
<td>0.143</td>
<td>0.071</td>
<td>0.286</td>
<td>0.214</td>
<td>0.214</td>
</tr>
<tr>
<td>PC1</td>
<td>0.28</td>
<td>0.24</td>
<td>0.16</td>
<td>0.16</td>
<td>0.08</td>
<td>0.36</td>
<td>0.24</td>
<td>0.24</td>
</tr>
<tr>
<td>PC2</td>
<td>0.333</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PC3</td>
<td>0.986</td>
<td>0.178</td>
<td>0</td>
<td>0.233</td>
<td>0.014</td>
<td>0.137</td>
<td>0.288</td>
<td>0.288</td>
</tr>
<tr>
<td>PC4</td>
<td>0.431</td>
<td>0.538</td>
<td>0.231</td>
<td>0.508</td>
<td>0.323</td>
<td>0.677</td>
<td>0.692</td>
<td>0.677</td>
</tr>
<tr>
<td>PC5</td>
<td>0.427</td>
<td>0.308</td>
<td>0.332</td>
<td>0.521</td>
<td>0.303</td>
<td>0.474</td>
<td>0.498</td>
<td>0.479</td>
</tr>
</tbody>
</table>

3.3 Specificity

From the specificity table we can see some of the algorithm are giving 100 percent specificity, that can not be consider as there respective sensitivity zero. These algorithms can give wrong prediction.

So According to the sensitivity and specificity DecisionTable algorithm should not consider for software defect prediction as they giving high 100% specificity bt 0% sensitivity.
3.4 Balance

Looking to the Accuracy, Sensitivity, and Specificity performance table we consider the NB, LOG, JRip, OneR, PART, J48, J48G, as there performance are average. From the graph figure 3.1 we see that, in maximum of cases the OneR algorithm giving lowest balance value than others. So, no need to use for defect prediction.

Table 3.3: Specificity

<table>
<thead>
<tr>
<th>Methods</th>
<th>NB</th>
<th>LOG</th>
<th>DT</th>
<th>JRip</th>
<th>OneR</th>
<th>PART</th>
<th>J48</th>
<th>J48G</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>0.893</td>
<td>0.951</td>
<td>1</td>
<td>0.943</td>
<td>0.984</td>
<td>0.789</td>
<td>0.943</td>
<td>0.951</td>
</tr>
<tr>
<td>JM1</td>
<td>0.956</td>
<td>0.988</td>
<td>0.99</td>
<td>0.968</td>
<td>0.957</td>
<td>0.994</td>
<td>0.954</td>
<td>0.956</td>
</tr>
<tr>
<td>KC1</td>
<td>0.898</td>
<td>0.976</td>
<td>0.959</td>
<td>0.937</td>
<td>0.932</td>
<td>0.927</td>
<td>0.947</td>
<td>0.947</td>
</tr>
<tr>
<td>KC3</td>
<td>0.873</td>
<td>0.794</td>
<td>0.921</td>
<td>0.937</td>
<td>0.857</td>
<td>0.937</td>
<td>0.921</td>
<td>0.952</td>
</tr>
<tr>
<td>MC1</td>
<td>0.947</td>
<td>1</td>
<td>0.999</td>
<td>0.999</td>
<td>1</td>
<td>0.999</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MC2</td>
<td>0.724</td>
<td>0.759</td>
<td>1</td>
<td>0.931</td>
<td>0.862</td>
<td>0.897</td>
<td>0.931</td>
<td>0.931</td>
</tr>
<tr>
<td>MW1</td>
<td>0.848</td>
<td>0.848</td>
<td>0.924</td>
<td>0.978</td>
<td>0.978</td>
<td>0.978</td>
<td>0.957</td>
<td>0.957</td>
</tr>
<tr>
<td>PC1</td>
<td>0.943</td>
<td>0.982</td>
<td>0.993</td>
<td>0.957</td>
<td>0.989</td>
<td>0.946</td>
<td>0.935</td>
<td>0.943</td>
</tr>
<tr>
<td>PC2</td>
<td>0.946</td>
<td>0.997</td>
<td>1</td>
<td>0.998</td>
<td>1</td>
<td>0</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>PC3</td>
<td>0.219</td>
<td>0.976</td>
<td>0.958</td>
<td>0.944</td>
<td>0.987</td>
<td>0.96</td>
<td>0.926</td>
<td>0.942</td>
</tr>
<tr>
<td>PC4</td>
<td>0.929</td>
<td>0.968</td>
<td>0.99</td>
<td>0.956</td>
<td>0.978</td>
<td>0.909</td>
<td>0.907</td>
<td>0.917</td>
</tr>
<tr>
<td>PC5</td>
<td>0.983</td>
<td>0.99</td>
<td>0.991</td>
<td>0.987</td>
<td>0.99</td>
<td>0.985</td>
<td>0.986</td>
<td>0.987</td>
</tr>
</tbody>
</table>

Table 3.4: Balance

<table>
<thead>
<tr>
<th>Methods</th>
<th>NB</th>
<th>LOG</th>
<th>DT</th>
<th>JRip</th>
<th>OneR</th>
<th>PART</th>
<th>J48</th>
<th>J48G</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM1</td>
<td>0.569</td>
<td>0.481</td>
<td>0.293</td>
<td>0.433</td>
<td>0.387</td>
<td>0.505</td>
<td>0.433</td>
<td>0.433</td>
</tr>
<tr>
<td>JM1</td>
<td>0.432</td>
<td>0.365</td>
<td>0.342</td>
<td>0.403</td>
<td>0.369</td>
<td>0.314</td>
<td>0.385</td>
<td>0.379</td>
</tr>
<tr>
<td>KC1</td>
<td>0.593</td>
<td>0.461</td>
<td>0.431</td>
<td>0.523</td>
<td>0.47</td>
<td>0.516</td>
<td>0.518</td>
<td>0.518</td>
</tr>
<tr>
<td>KC3</td>
<td>0.575</td>
<td>0.559</td>
<td>0.374</td>
<td>0.375</td>
<td>0.409</td>
<td>0.54</td>
<td>0.539</td>
<td>0.541</td>
</tr>
<tr>
<td>MC1</td>
<td>0.678</td>
<td>0.407</td>
<td>0.43</td>
<td>0.407</td>
<td>0.407</td>
<td>0.43</td>
<td>0.407</td>
<td>0.407</td>
</tr>
<tr>
<td>MC2</td>
<td>0.639</td>
<td>0.636</td>
<td>0.293</td>
<td>0.293</td>
<td>0.355</td>
<td>0.633</td>
<td>0.321</td>
<td>0.323</td>
</tr>
<tr>
<td>MW1</td>
<td>0.582</td>
<td>0.484</td>
<td>0.593</td>
<td>0.394</td>
<td>0.343</td>
<td>0.495</td>
<td>0.443</td>
<td>0.443</td>
</tr>
<tr>
<td>PC1</td>
<td>0.489</td>
<td>0.462</td>
<td>0.406</td>
<td>0.405</td>
<td>0.349</td>
<td>0.546</td>
<td>0.461</td>
<td>0.461</td>
</tr>
<tr>
<td>PC2</td>
<td>0.527</td>
<td>0.293</td>
<td>0.293</td>
<td>0.293</td>
<td>0.293</td>
<td>0.293</td>
<td>0.293</td>
<td>0.293</td>
</tr>
<tr>
<td>PC3</td>
<td>0.448</td>
<td>0.419</td>
<td>0.292</td>
<td>0.456</td>
<td>0.303</td>
<td>0.389</td>
<td>0.494</td>
<td>0.495</td>
</tr>
<tr>
<td>PC4</td>
<td>0.595</td>
<td>0.673</td>
<td>0.456</td>
<td>0.651</td>
<td>0.521</td>
<td>0.763</td>
<td>0.772</td>
<td>0.764</td>
</tr>
</tbody>
</table>
Depending on Accuracy, Sensitivity, Specificity, Balance performance we choose 6 Algorithms from 8 algorithms are:

- NaiveBayesSimple
- Logistic
- JRip
- PART
- J48 and J48Graft

![Figure 3.1: Balance](image)

### 3.5 ROC Area

And the Software defect prediction performance based on ROC Area simulated by our scheme given in the table:3.5..

According to ROC Area Logistic and Navey based algorithm gives the better performance for software defect prediction.
Table 3.5: Comparative Performance (ROC Area) of Software defect prediction.

<table>
<thead>
<tr>
<th>Methods</th>
<th>CM1</th>
<th>JM1</th>
<th>KC1</th>
<th>KC3</th>
<th>MC1</th>
<th>MC2</th>
<th>MW1</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
<th>PC5</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>0.685</td>
<td>0.681</td>
<td>0.801</td>
<td>0.745</td>
<td>0.861</td>
<td>0.745</td>
<td>0.666</td>
<td>0.736</td>
<td>0.846</td>
<td>0.793</td>
<td>0.84</td>
<td>0.804</td>
</tr>
<tr>
<td>Log</td>
<td>0.668</td>
<td>0.709</td>
<td>0.808</td>
<td>0.604</td>
<td>0.893</td>
<td>0.686</td>
<td>0.592</td>
<td>0.821</td>
<td>0.7</td>
<td>0.802</td>
<td>0.911</td>
<td>0.958</td>
</tr>
<tr>
<td>JRip</td>
<td>0.572</td>
<td>0.562</td>
<td>0.633</td>
<td>0.527</td>
<td>0.58</td>
<td>0.5</td>
<td>0.561</td>
<td>0.499</td>
<td>0.589</td>
<td>0.735</td>
<td>0.755</td>
<td></td>
</tr>
<tr>
<td>PART</td>
<td>0.492</td>
<td>0.713</td>
<td>0.709</td>
<td>0.612</td>
<td>0.773</td>
<td>0.639</td>
<td>0.611</td>
<td>0.566</td>
<td>0.481</td>
<td>0.728</td>
<td>0.821</td>
<td>0.942</td>
</tr>
<tr>
<td>J48</td>
<td>0.537</td>
<td>0.67</td>
<td>0.698</td>
<td>0.572</td>
<td>0.819</td>
<td>0.259</td>
<td>0.5</td>
<td>0.646</td>
<td>0.39</td>
<td>0.727</td>
<td>0.784</td>
<td>0.775</td>
</tr>
<tr>
<td>j48G</td>
<td>0.543</td>
<td>0.666</td>
<td>0.698</td>
<td>0.587</td>
<td>0.819</td>
<td>0.274</td>
<td>0.5</td>
<td>0.651</td>
<td>0.39</td>
<td>0.738</td>
<td>0.778</td>
<td>0.775</td>
</tr>
</tbody>
</table>

Figure 3.2: ROC Area

3.6 Comparison with other's results

- In 2011 Song, Jia, Ying, and Liu proposed a general framework. In that framework they used One R algorithms for defect prediction, But that should not be considered for defect prediction as it gives 0 sensitivity sometimes, and balance values are very low than others.

- In 2007 MGF used considers only 10 data set, whereas in our research we used 12 data set with more modules in every data set. And in our result the balance values are also greater than there results.

- In others works different machine learning algorithms are used. In our research
the results of comparative measurement values are increases. Mainly in accuracy increases as we used percentage split.

Figure 3.3: Accuracy

Figure 3.4: Sensitivity
Figure 3.5: Specificity

Figure 3.6: Balance
4. Conclusion

4.1 Concluding Remarks

In our research work we have attempted to solve the Software defect prediction problem through different Data mining (Classification) algorithms.

In our research NB and Logistic algorithm gives the overall better performance for defect prediction. PART and J48 gives better performance than OneR and JRip.

From these results, we see that a data preprocessor/attribute selector can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important.

In order to improve the efficiency and quality of software development, we can make use of the advantage of data mining to analysis and predict large number of defect data collected in the software development. This paper reviewed the current state of software defect management, software defect prediction models and data mining technology briefly. Then proposed an ideal software defect management and prediction system, researched and analyzed several software defect prediction methods based on data mining techniques and specific models (NB, Logistic, PART, J48G).

4.2 Scope for Further Research

- Clustering based classification can be used.
- Future studies could focus on comparing more classification methods and improving association rule based classification methods
- Furthermore, the pruning of rules for association rule based classification methods can be considered.
Bibliography


Predictive Classification Framework for Software Defect Prediction

K.B.S Sastry¹ and Dr. R. Satya Prasad²

¹Lecturer in Computer Science, Andhra Loyola College, Vijayawada.
²Associate Professor, Dept. of Computer science, Acharya Nagarjuna University, Nagarjuna nagar, Guntur Dist.

Abstract

Consider the fast development of the product improvement application transform in present days. Programming advancement application holds a few deserts in presenting/executing programming items. They are expense and powerful movement programming advancement in testing the result of the product. Customarily a portion of the information digging systems were produced for distinguish programming deformity forecast from different information set applications from recorded information. One pass calculation is one of the systems for getting to administrations and different gimmicks of the preparing units continuously programming application improvement including the peculiarities of programming application like item cost and testing item. Programming quality and testing proficiency are the fundamental peculiarities in programming deformity forecast. So in this paper we propose to create prescient arrangement calculation to reduction expense of the product testing advancement and expense estimation for programming application process. This method propose to create programming quality and testing proficiency in by developing prescient modules from code properties exhibit in discharged thing sets. In this strategy, use information affiliation guideline digging occasions for identifying backing and trust for every information thing present progressively programming application improvement with characteristic representation. This methodology is help to designers locate programming deserts and support venture administration in distributing testing strategies with assets adequately.

Index Terms: Software defect production, association rule mining, classification, Defect testing, cost and database.
I. INTRODUCTION
Software engineering is the process of develop, design and maintenance of the software product with updated features of the earliest application development. These features termed as software development and information technology from various updated and outdated application events from other equality applications. Software development is one of the computer programming, documenting and testing involved in real time application development. Software development may include various features in recent research development processes, reuse, reengineering maintenance, prototyping and other activities in software development application process. Programming improvement technique is edge work used to create structure, arrange, and control the procedure of creating data frameworks with practical skeleton engendering with applicable information. Included matter with proselyte the peculiarities in practical in qualities and shortcomings. Figure 1 shows technique for advancement programming application including all the utilitarian, programming and outline usage with testing feasibilities. Consider the above examination we watch that product testing is one of the key element for exploring the gimmicks of machine program that creates surprising results.
Figure 1: Software development process.
The achievement of the product prepare relies on upon the expense and timetable, as well as relies on upon quality, quality may rely on upon programming deformity forecast. Programming imperfection is a slip, defect, mix-up, and blame in machine program that delivers unforeseen results with relative programming improvement application process. Programming deformity expectation is the procedure of model that causes with viable modules in programming advancement process administration operations. [2] [6] To lessening programming absconds progressively application process administration with their delegate accomplishments of item discharge quality is a primary sacred occasion in programming forecast. Programming imperfection is condition in programming discharged item does not rely upon programming prerequisites and end client desires. [6] For viable tweak of the product deformity forecast continuously application advancement, information mining methods were acquainted with produce proficient and fantastic information administration operations in discharge programming item handle. Information mining is the methodology of examining information from alternate point of view information dissection and after that outlining the information with helpful data. Distinctive information digging methods were presented for illuminating programming deformity handle continuously programming application advancement with relative information administration operations. [3] According to the affiliation tenet mining in information mining system occasions in programming application improvement characteristics they incorporate all the information measures that attains relations in semantic information representation of authentic information utilizing affiliation digging for concentrating the peculiarities of the examined information movement. [2] In affiliation mining distinctive guidelines are sorted out to foresee programming deformity expectations with partner relations in diverse methods for transforming guideline set eras in semantic information representation of information dissection. It uncovers example and patterns that are characterized in verifiable database process. Affiliation guideline mining is not just limit the administrations with reliance dissection in the connection application improvement furthermore find co-happening examples of the traits in information bases [13] [14]. It takes after affiliation arrangement for running across fulfilled example grouping and different gimmicks of the obliged information base outline process. One pass calculation is one of the affiliated calculations for depicting these occasions focused around arrangement and acquainted order for higher grouping exactness with progressive information occasion associations. One pass calculation portrays administration of recorded information from different information transactions; calculation takes after productive process on assessment and imperfection forecast for catching programming deformity forms in authentic information when we apply pattern assessment and other gimmick exercises in pertinent information movement. This methodology takes after affiliation tenet digging for computing least backing and least trust in every information thing present in chronicled information with successive property check in information examination of the delivered information movement [10][14]. Using this methodology we diminish the product item cost and booking occasions of chronicled information in illustrative occasions to other progressing
occasions with business movement era. Programming quality is one of the key viewpoint in programming improvement process, because of this application structure of programming items we propose to create prescient grouping calculation for expanding proficient gimmick eras on every information set representation of most punctual information things with continuous property representation in authentic information.

![Defect Prevention Strategy for Software Development Process.](image)

Figure 2: Defect Prevention Strategy for Software Development Process.

We utilize deformity information sort as prescient grouping for partitioning information sets into number of information things with incorporating quality representation continuously programming handling units in information consistence with backing and certainty of the continuous information sets for expanding programming item confirmation.

The basic idea of the association rule mining that originates from retrieving data items from historical data, for example customers buy three products p1, p2, and p3 and organized with probability of all the product information in finding efficient data extraction processes using support and confidence with other format results. In our technique the following procedure will occurs and calculating the patterns of attributes in data bases [18][19]. Association rule mining explores high confidence associations among multiple variables it may overcome some constraints introduced by other techniques. The success of the software prediction assessment with association rule mining in various fields motivates in us to apply software defect data set.

Remaining part of this paper proposes as follows: Section II explains all the literature review of the previous used techniques with feasibility access on advanced techniques. Section III explains Back ground work propagation of software product assurance with different values. Section IV explains Research approach of predictive classification procedures. Section V explains experimental evaluation of each sub ordinate software product development.
II. RELATED WORK
To enhance the product quality, gainfulness programming designers with information mining calculations to different programming applications with application programming client interface systems by a complex library or schema with lacking documentation [16]. Programming deformity expectation is a device and after that it creates among testing exercises and programming improvement process applications. [6] Defect indicators used to make as assets of the product measurements and other gimmick exercises in business occasion movement. Martin Shepperd presents K-implies and neural information mining strategies on distinctive information sets and afterward illustrative those terms in reasonable module representation with investigating information into diverse groups with deficiency tolerance. In their study they have displayed relative results performed on same information sets. [14][16] Partha Sarathi Bishnu presents an application of affiliation guideline mining to foresee the administrations programming partners and imperfection amendment with SEL right information. Grouping is the procedure of displaying occasions that depicts programming models. In this models a preparation information set and other test information sets with proficient information extraction on chronicled information. The proposed expectation characterization calculation helps distinctive affiliation tenets with quantitative and qualitative information thing accomplishment in sensible information representation of the handling of properties presentation of the consecutive preparing of every information thing designated to other information characteristics [1]. The proposed work concurs with retraining information from different information things introduce in the business occasions. To begin with it recovers information from different information sets display in the concentrated information sets with handling occasion administration. It assesses all the information things from concentrated information sets with nature of programming application improvement. Affiliation forecast calculation does not accomplishes representation of all the preparing units.

III. BACKGROUND WORK
Association rule mining is to find the association rules that satisfy data set report generation in minimum support and minimum confidence with data results in comparative data process [2] [5]. Historical datasets are reported with proceeding events in real time software application development. In this technique historical data can be divided into two ways with further report processing of all the commercial event software applications. It divides into two ways evaluation and defect prediction using the services of software product evaluation with relative and successive coordination with semantic and other features are activated historical data representation. In this technical aspect representation of the data computation there is a relative conceptual presentation of each and verified results in commercial progression.
Data sets are loaded into memory process for computing additional requirements of association rule mining with conceptual and physical processes with equality sharing between each item set present historical data [3][9]. Procedure of the one pass algorithm can be satisfied following consequences, upload data sets and then training some datasets into necessary action in to test data for accessing services from testing data then apply one pass algorithm on historical data sets. Compare data sets into one pass procedure in real time processing operations. Association rules are applied in following terminology events with reserved data items in calculating minimum support and confidence for accessing services with relevant features of each data item.

IV. RESEARCH APPROACH

According to the utilization data sets from historical datasets, then each data item calculated association of memorials and processing features of the extracted data sets. [4] These procedures are evaluated into number of item sets with including each item set representation in proceeding on each data item. One pass algorithm achieves software product cost and scheduling of services in recommended data events. Consider the representation of software quality assurance we propose to develop Predictive Classification algorithm for increasing the services of all the features presented in data modulation. We propose to develop predictive classification algorithm to decrease cost of the software testing development. This algorithm consists 3 phases
1. Extraction phase
2. Learning Phase
3. Prediction Result Analysis Phase
In Extraction phase the process will include following procedure
Initialize Datasets, Transactions, Training set (TS), Testing set (TS1)
Extract data set from D=d0,d1,d3,………………..dn
Perform transactions T=t0,t1,t2,…………tn on retrieving data sets
Calculate each transaction r with processing operations on training and testing extraction datasets.

Learning Phase performs following procedures.
Training data from each Data item D=d0,d1,………..dn
Test training data from Extracted data sets D=d0,d1,………..dn.
learner = Build Classifier (d’, scheme. Algorithm)
Calculate Support(S) and Confidence(C) for each attribute in data item including transactions

Prediction results
[predictor, bestAttrs ] = Learning (historical Data)
d = select bestAttrs from new Data
Classify data with when transaction occurs Support as P(d0Ud1) for each attribute in data item.
Classify data with when transaction occurs Confidence as P(d0/d1) for each attribute in data item.
Result = Predict (d, predictor).

Algorithm 1: Predictive Classification algorithm for software defect predictions.

Calculation 1 takes after productive methodology for ascertaining least and most extreme information utility focused around affiliation principle mining operations which incorporates clear clarification with respect to information investigation and information visualization which incorporates preparing of information things with quantitative and qualitative information things for expanding limit religious in information representation in present procedure [16] [20].

Information Source and Extraction: Data we utilize SEL (Software Engineering Laboratory) information sets which is sub situated of online information sets like NASA and other stockpiling and recovery of programming designing information things. [1][3] The SEL is information builds that furnish ventures with respect to programming necessities with determined characters changes and blanders being developed of all the product applications. With the end goal of programming imperfection expectation which incorporates the methodology of concentrated information from distinctive tables of SEL information research facilities. The deformity is exceptionally basic in concentrating information sets from distinctive gimmick forms.

Examination Approach: For this methodology we utilize acceptance technique on intersection information occasions with results in all the introduced information which prepare effective work spread [15] [18]. We utilize affiliation principle mining system to recover preparing information sets from information angles introduce in the business work engendering occasion administration. For
deformity affiliation procedure of concentrated information sets standard learning is straight advance while imperfection forecast the ensuing of a tenet must be abscond amendment exertion.

V. EXPERIMENTAL RESULTS
In this section we present experimental results for defect transactional data operations when performing the operations of the association which includes data processing unit as product releasing cost and product formative cost and scheduling the product with consequence event management operations. [4] Extracted defect association correction effort data set with different minimum support and minimum confidence threshold for association rule mining based defect correction and effort predictions. For example we describe to take five different data sets as training and as test data sets are used to defect software effort prediction.

| Input : Data extraction with rule ranking RR, data sets with attributes, defects. |
| Output: Predicted effort for correcting defect attributes. |
| Initialize the Simulation------> 0, Effect<--- --- Φ |
| For each data item e€ attribute |
| Do |
| Defect<---- e; |
| Apply each rule on data item r € rules |
| Defect <------ Consequence(r); |
| Each data item attribute calculate the support and confidence |
| Effect<------Effect U Defect; |
| Simulation <------- Sim U D.attrib)/attri. |
| Defect<----Effect.attrib). |
| End if |
| End for. |
| Release defect factor. |

Figure 4: Effort prediction procedure

Perform operation with each data item then calculate the performance of accuracy and quality of software product in commercial event management system application progress. Our proposed work follows the following procedure events in query evaluation.

**Defect Association Prediction:** When mining defect association rules minimum support 10,20, and 30 four minimum confidence values.
Defectf{DataValue}\}

DefectfNullg@(32.5%; 79.9%).

Defectf{Comput:g ^ DefectfIni:g}

DefectfEx:Interfaceg@(34.3%; 75.1%).

The first defect rule that defect the DataValue occurred in the defect process. The second rule defines example interface the systematic data representation in calculating support and confidence.

**Table 1: Defect Association Prediction Accuracy.**

<table>
<thead>
<tr>
<th>Constraining</th>
<th>Accuracy</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.Supp=20%</td>
<td>Mean=96.59%</td>
<td>FP=13.17%</td>
</tr>
<tr>
<td>Min.Conf=30%</td>
<td>Min = 95.83%</td>
<td>FN=2.84%</td>
</tr>
</tbody>
</table>

The quality of the defect association depends upon the association rules which include perfect process of explored with impact of min.supp and min.conf on each association rule progressed with commercial data aggregation on each data item presented in attribute representation.

**Defect Effort Prediction:** We use same configurations for extracting and mining defect isolation which includes defect rules for accessing services. [1] The distribution effort formation of all the taken five data sets give minimum support and confidence effort with equal formalism with sufficient data processing units.

**Table 2: Defect Isolation Effort Prediction Accuracy.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Medium</td>
</tr>
<tr>
<td>Apriori</td>
<td>93.80</td>
<td>93.52</td>
</tr>
<tr>
<td>PART</td>
<td>69.52</td>
<td>68.54</td>
</tr>
<tr>
<td>C4.5</td>
<td>68.59</td>
<td>95.26</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>67.95</td>
<td>68.18</td>
</tr>
</tbody>
</table>

We observe that each data set the average number of rules decreases then min.supp can be increased from 10 to 40 percent with equal sharing of data between each data item present in requested protocol hierarchy [20] [22]. As shown in the above table 2 achieves efficient comparison results like apriori association rule mining gives higher performance in comparison working procedure with equality sharing with other PART, C4.5, and other Naïve Bayes methodologies.
The process of executing informative representation of each data item which include process applications in real time processing units.

The reason association rule mining based prediction performs so much better than other methods is that it explores high confidence associations among multiple variables and discovers interesting rules, i.e., rules that are useful, strong, and significant. [16] [21] Concluded experimental result show efficient data set representation in increasing the quality of software in processing utilization event management operations.

VI. CONCLUSION
Software quality and maintenance is main achieving term in present research approaches. Conventional used technology is one pass algorithm for finding association rules with minimum support and minimum confidence. In this paper we
propose to develop predictive classification algorithm to decrease cost of the software testing development and cost estimation for software application process. Due to this software model application format we increase software assurance with all the properties in presented data set representation in association rule mining. It further supports the conclusion that a sufficient number of rules is a precondition for the high prediction accuracy we obtained in the context of defect isolation effort prediction.

VII. ACKNOWLEDGEMENT
The authors who access services from proposed methodology, they can derive processes from NASA/GSFC Software Engineering Laboratory (SEL) for providing the defect data for this analysis. The authors specially thank for anonymous reviewers for their helpful comments in a different way of execution of data sets. The authors also thanks to all the viewers for success of publishing this document.

VIII. REFERENCES


[2] “Software Defect Prediction from Historical Data”, K.B.S Sastry, Dr.B.V.Subba Rao, and Dr K.V.Sambasiva Rao, Volume 3, Issue 8, August 2013 ISSN: 2277 128X.

[3]. Software Defect Prediction Based on Classification Rule Mining by Dulal Chandra Sahana.

[4]. Data Mining Techniques for Software Defect Prediction by Ms. Puneet Jai Kaur, Ms. Pallavi


[6]. Significance of Different Software Metrics in Defect Prediction by Marian Jureczko.

[7]. Using Classification to Evaluate the Output of Confidence-Based Association Rule Mining by Stefan Mutter, Mark Hall, and Eibe Frank.

[8]. Association Rule Mining: A Survey by Qiankun Zhao and Sourav S. Bhowmick.

[9]. Association Rule Based Classification by Senthil K. Palanisamy.

[10]. Software Defect Prediction Based on Association Rule Classification by Baojun Ma Karel Dejaeger Jan Vanthienen Bart Baesens.

Mahendra Tiwari, Randhir Singh, Shivendra Kumar Singh.


Software Defect Prediction from Historical Data

K.B.S Sastry,  
Lecturer in Computer Science,  
Andhra Loyola College,  
Vijayawada.  
E-mail: sastrykbs@gmail.com

Dr.B.V.Subba Rao,  
Professor in IT, PVP Siddhartha Inst. of Technology, Vijayawada,  
Andhra Pradesh, INDIA.  
E-mail: bvsrau@gmail.com

Dr K.V.Sambasiva Rao,  
Principal, MVR College of Engg. and Technology, Paritala, Vijayawada.  
E-mail: principal@mvrcoe.ac.in

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 Naive 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. Kubat 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 Naive 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 Lk, Lk = {I1, I2, … , Ik}, association rules with this item sets are generated in the following way: the first rule is {I1, I2, … , Ik-1} \( \Rightarrow \) {Ik}, 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 straight forward, most of the researches focus on the first sub problem.

\[\begin{array}{|c|c|c|c|c|}
\hline
\text{Name} & \text{Language} & \text{Features} & \text{Instances} & \text{Recorded Values} & \% \text{Defective Instances} \\
\hline
\text{CM1} & \text{C} & 40 & 505 & 20200 & 10 \\
\hline
\text{JM1} & \text{C} & 21 & 10878 & 228438 & 15 \\
\hline
\text{KC1} & \text{C++} & 21 & 2107 & 44247 & 15 \\
\text{KC3} & \text{Java} & 40 & 458 & 18320 & 9 \\
\text{KC4} & \text{Perl} & 40 & 125 & 2000 & 45 \\
\hline
\text{MC1} & \text{C} \& \text{C++} & 39 & 9466 & 369174 & 0.7 \\
\text{MC2} & \text{C} & 40 & 161 & 6440 & 32 \\
\text{MW1} & \text{C} & 40 & 403 & 16120 & 8 \\
\hline
\text{PC1} & \text{C} & 40 & 1197 & 44280 & 7 \\
\text{PC2} & \text{C} & 40 & 5589 & 222560 & 0.4 \\
\text{PC3} & \text{C} & 40 & 1553 & 62520 & 10 \\
\text{PC4} & \text{C} & 40 & 1458 & 58320 & 12 \\
\text{PC5} & \text{C++} & 39 & 17186 & 670254 & 3 \\
\hline
\end{array}\]

Fig 1. Statistics of defect in software

Let \( I = \{I1, I2, \ldots, Im\} \) 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 \( Ts \). 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.

![Diagram of Proposed Solution](image)

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.

```
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 ;
```
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

1. \( m = 10; \) /*number of repetitions for attribute selection */
2. \( n = 10; \) /*number of folds for attribute selection */
3. \( d = \text{Preprocessing} \text{(data, scheme. preprocessor)}; \)
4. \( \text{bestAttrs} = \text{AttrSelect} \text{(d, scheme. Algorithm, scheme.attrSelector, m, n)}; \)
5. \( d' = \text{select bestAttrs from d }; \)
6. learner = \text{BuildClassifier} \text{(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

1. \([\text{predictor, bestAttrs }] = \text{Learning} \text{(historical Data, scheme)};\)
2. \(d = \text{select bestAttrs from new Data};\)
3. Result = \text{Predict} \text{(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

Comparison of Classification Mining models for Software Defect Prediction

K.B.S. Sastry¹, Dr. R. Satya Prasad²

¹Lecturer, Dept. of Computer Science, Andhra Loyola College, Vijayawada, sastrykbs@gmail.com
²Associate Professor, Dept. of Computer Science, Acharya Nagarjuna University, Guntur, profrsp@gmail.com

Software defects are expensive in quality and cost. The accurate prediction of defect-prone software modules can help direct test effort, reduce costs, and improve the quality of software. Machine learning classification algorithms is a popular approach for predicting software defect. Various types of classification algorithms have been applied for software defect prediction. However, no clear consensus on which algorithm perform best when individual studies are looked at separately. In this research, a comparison framework is proposed, which aims to benchmark the performance of a wide range of classification models within the field of software defect prediction. For the purpose of this study, 10 classifiers are selected and applied to build classification models and test their performance in 9 NASA MDP datasets. Area under curve (AUC) is employed as an accuracy indicator in our framework to evaluate the performance of classifiers. Friedman and Nemenyi post hoc tests are used to test for significance of AUC differences between classifiers. The results show that the logistic regression perform best in most NASA MDP datasets. Naïve bayes, neural network, support vector machine and k* classifiers also perform well. Decision tree based classifiers tend to underperform, as well as linear discriminant analysis and k-nearest neighbor.

Keywords: Software Defect Prediction, Machine Learning, Classification Model, Comparison Framework

1. INTRODUCTION

A software defect is an error, failure, or fault in a software [1], that produces an incorrect or unexpected result, or causes it to behave in unintended ways. It is a deficiency in a software product that causes it to perform unexpectedly [2]. Software defects or software faults are expensive in quality and cost. Moreover, the cost of capturing and correcting defects is one of the most expensive software development activities [3]. Recent studies show that the probability of detection through defect prediction models may be higher than the probability of detection through software reviews [4]. The accurate prediction of defect-prone software modules can certainly assist testing effort, reduce costs and improve the quality of software [5].
Classification algorithm is a popular machine learning approach for software defect prediction. It categorizes the software code attributes into defective or not defective, which is collected from previous development projects. Classification algorithm is also able to predict which components are more likely to be defect-prone, supports better targeted testing resources and therefore, improved efficiency. For prediction modeling, software metrics are used as independent variables and fault data is used as the dependent variable [6]. A wide range of classification techniques have already been proposed in the predicting software defect. Since 1990, more than 20 classification algorithms have been applied and proposed as the best method for predicting the software defect, including logistic regression (LR) [7], decision trees (DT) [8], neural network (NN) [9], naive bayes (NB) [10], and etc.

While many studies individually report the comparative performance of the modelling techniques they have used, no clear consensus on which perform best emerges when individual studies are looked at separately. Bibi et al. [11] report that regression via classification (RvC) works well. Hall et al. [5] suggests that studies using support vector machine (SVM) techniques perform less well. These may be underperforming as they require parameter optimization for best performance. Models based on C4.5 seem to underperform if they use imbalanced data [12] [13], as the algorithm seems to be sensitive to this. NB and LR, in particular, seem to be the techniques used in models that are performing relatively well overall [10] [14]. NB is a well understood algorithm that is in common use. Studies using random forests (RF) have not performed as well as might be expected [5], although many studies using NASA dataset use RF and report good performances [15]. However, models seem to have performed best where the right technique has been selected for the right set of data. No particular classifiers that performs the best for all the datasets [14].

However, we need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models [15] [14] [4]. In this research, we propose a comparison framework, which aims to benchmark the performance of a wide range of classification models within the field of software defect prediction.
This paper is organized as follows. In section 2, the proposed comparison framework is explained. The experimental results of classification models comparison are presented in section 3. Finally, our work of this paper is summarized in the last section.

2. PROPOSED COMPARISON FRAMEWORK

The proposed framework is shown in Figure 1. The framework is comprised of 1) a dataset 2) a classification algorithms, 3) a model validation, 4) a model evaluation and 5) a model comparison.

2.1 Dataset

One of the most important problems for software fault prediction studies is the usage of nonpublic (private) datasets. Several companies developed fault prediction models using proprietary data and presented these models in conferences. However, it is not possible to compare results of such studies with results of our own models because their datasets cannot be reached. The use of public datasets makes our research repeatable, refutable, and verifiable [16]. Recently, state-of-the-art public datasets used for software defect prediction research is available in NASA Metrics Data (MDP) repository [17].

The data used in the proposed framework are collected from the NASA MDP repository. NASA MDP repository is a database that stores problem, product, and metrics data [17]. Each NASA dataset is comprised of several software modules, together with their number of faults and characteristic code attributes. After preprocessing, modules that contain one or more errors were labeled as fault-prone, whereas error-free modules were categorized as not-fault-prone. Besides line of codes (LOC) counts, the NASA MDP datasets include several Halstead attributes [18] as well as McCabe complexity measures [19]. The former estimates reading complexity by counting operators and operands in a module, whereas the latter is derived from a module’s flow graph. Some researchers endorse the static code attributes defined by McCabe and Halstead as defect predictors in the software defect prediction. McCabe and Halstead are module-based metrics, where a module is the smallest unit of functionality. Static code attributes are used as defect predictors, since they are useful, generalizable, easy to use, and widely used.
Fig. 1. Proposed Comparison Framework of Classification Models for Software Defect Prediction
In this research, we use nine software defect datasets from NASA MDP. Individual attributes per dataset, together with some general statistics and descriptions, are given in Table 1. These datasets have various scales of LOC, various software modules coded by several different programming languages including C, C++ and Java, and various types of code metrics including code size, Halstead’s complexity and McCabe’s cyclomatic complexity.

Table 1. Characteristics of NASA MDP Datasets
2.2 Classification Algorithms

The proposed classification framework aims to compare the performance of a wide range of classification models within the field of software defect prediction. For the purpose of this study, 10 classifiers have been selected, which may be grouped into the categories of traditional statistical classifiers (LR, LDA, and NB), nearest neighbors (k-NN and K*), NN, SVM, and decision tree (C4.5, CART, and RF). The selection aims at achieving a balance between established classification algorithms used in software defect prediction.

2.3 Model Validation

We use a stratified 10-fold cross-validation for learning and testing data. This means that we divide the training data into 10 equal parts and then perform the learning process 10 times. As shown in Table 2, each time, we chose another part of dataset for testing and used the remaining nine parts for learning. After, we calculated the average values and the deviation values from the ten different testing results. We employ the stratified 10-fold cross validation, because this method has become the standard and state-of-the-art validation method in practical terms. Some tests have also shown that the use of stratification improves results slightly [20].

Table 2. Stratified 10 Fold Cross Validation

<table>
<thead>
<tr>
<th>n-validation</th>
<th>Dataset’s Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

2.4 Model Evaluation

We apply area under curve (AUC) as an accuracy indicator in our experiments to evaluate the performance of classifiers. AUC is area under ROC curve. Lessmann et al. [15] advocated the use of the AUC to improve cross-study comparability. The AUC has the potential to significantly improve convergence across empirical experiments in software defect prediction, because it
separates predictive performance from operating conditions, and represents a general measure of predictiveness. Furthermore, the AUC has a clear statistical interpretation. It measures the probability that a classifier ranks a randomly chosen fault-prone module higher than a randomly chosen non-fault-prone module. Consequently, any classifier achieving AUC well above 0.6 is demonstrably effective for identifying fault-prone modules and gives valuable advice as to which modules should receive particular attention in software testing.

A rough guide for classifying the accuracy of a diagnostic test using AUC is the traditional system, presented by Gorunescu [21]. In the proposed framework, we added the symbols for easier interpretation and understanding of AUC (Table 3).

<table>
<thead>
<tr>
<th>AUC</th>
<th>Meaning</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.90 - 1.00</td>
<td>excellent classification</td>
<td>▲</td>
</tr>
<tr>
<td>0.80 - 0.90</td>
<td>good classification</td>
<td>△</td>
</tr>
<tr>
<td>0.70 - 0.80</td>
<td>fair classification</td>
<td>◊</td>
</tr>
<tr>
<td>0.60 - 0.70</td>
<td>poor classification</td>
<td>☆</td>
</tr>
<tr>
<td>&lt; 0.60</td>
<td>failure</td>
<td>○</td>
</tr>
</tbody>
</table>

2.5 Model Comparison

There are three families of statistical tests that can be used for comparing two or more classifiers over multiple datasets: parametric tests (the paired t-test and ANOVA), non-parametric tests (the Wilcoxon and the Friedman test) and the non-parametric test that assumes no commensurability of the results (sign test). Demsar recommends the Friedman test for classifier comparisons, which relies on less restrictive assumptions [22]. Based on this recommendation, in our framework Friedman test is employed to compare the AUCs of the different classifiers. The Friedman test is based on the average ranked (R) performances of the classification algorithms on each dataset.
Let $r^j_i$ rank of the $j$-th of algorithms on the $i$-th of $D$ datasets The Friedman test compares the average rank of algorithm as

$$R_j = \frac{1}{D} \sum_{i=1}^{D} r^j_i.$$ 

Under the null-hypothesis, which states that all the algorithms are equivalent and so their ranks $R_j$ should be equal. The Friedman statistic is calculated as follows, and distributed according to $X^2$ with $c-1$ degrees of freedom, which $D$ and $C$ are big enough.

$$X^2 = \frac{12D}{C(C+1)} \left[ \sum_{j=1}^{D} R_j^2 - \frac{C(C+1)^2}{4} \right]$$

If the null-Hypothesis is rejected, we can proceed with a post-hoc test. The Nemenyi test is used when all classifiers are compared to each other. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference, given by

$$CD = q_\alpha \sqrt{\frac{C(C+1)}{D}}$$

With critical values $q_\alpha$

3. EXPERIMENTAL RESULTS

The experiments were conducted using a computing platform based on Intel Core i7 2.2 GHz CPU, 16 GB RAM, and Microsoft Windows 7 Professional 64-bit with SP1 operating system. The development environment is Netbeans 7 IDE, Java programming language, and RapidMiner 5.2 library. We used the default parameter settings provide my RapidMiner 5.2 library.

We conducted experiments on 9 NASA MDP datasets by using 10 classification algorithms. Table 4 reports the AUCs of all classification algorithms. The last column of Table 4 reports the mean rank of each classifier over all datasets, which constitutes the basis of the Friedman test.