A HYBRID FEATURE SELECTION MODEL FOR SOFTWARE FAULT PREDICTION

C. Akalya devi¹, K. E. Kannammal² and B. Surendiran³

¹M.E (CSE), Sri Shakthi Institute of Engineering and Technology, Coimbatore, India
akalya.jk@gmail.com

²Associate Professor, Department of Computer Science and Engineering, Sri Shakthi
Institute of Engineering and Technology, Coimbatore, India
kekravi@gmail.com

³Assistant Professor (SrG), Department of Computer Science and Engineering,
Kumaraguru College of Technology, Coimbatore, India
surendir@gmail.com

ABSTRACT
Software fault prediction plays a vital role in software quality assurance. Identifying the faulty modules helps to better concentrate on those modules and helps improve the quality of the software. With increasing complexity of software nowadays feature selection is important to remove the redundant, irrelevant and erroneous data from the dataset. In general, Feature selection is done mainly based on filter and wrapper. In this paper a hybrid feature selection method is proposed which gives a better prediction than the traditional methods. NASA’s public dataset KC1 available at promise software engineering repository is used. To evaluate the performance of the software fault prediction models Accuracy, Mean absolute error (MAE), Root mean squared error (RMSE) values are used.

KEYWORDS
Fault prediction, feature selection, filter, software fault, software quality & wrapper.

1. INTRODUCTION
Quality of the software is an important aspect and software fault prediction helps to better concentrate on faulty modules. A fault in the software results in a change made to an executable file in response to an observed incorrect or undesirable behaviour of the software. And software fault prediction is to identify the modules most likely to have the faults in a future release. Software failures cost lot of amount and also human lives in critical systems. Software Reliability is hard to archive for complex software’s. Some of software quality assurance attributes are Fault proneness, Reliability, Reusability [1].

Fault prediction model has to consider two main things namely, accuracy and complexity. Better accuracy and least complexity can be the aim of developing a fault prediction model. To predict the faults techniques like statistical methods, machine learning methods, parametric models and mixed algorithms are used. Supervised machine learning technique is a method in which input and output in hand is used to train the machine to better predict in future. KC1 dataset available at promise repository is used for the experiments [2]. Feature selection is a pre-processing step in which irrelevant, redundant, missing and erroneous data are removed [3]. Feature selection plays a vital role in software fault prediction as complex software’s have
thousands and thousands of lines of code and has lot of attribute value pairs. Filters and wrappers are the two categories of feature selection. Wrapper based feature selection are computationally complex as they are based on complex classification algorithms. Filters are simple as they are based on the characteristics of data and hence preferred generally for large dataset. In this proposed model optimum feature selection is done by wrapper from the filtered features. This model is less complex than a simple wrapper and better predicts the fault than the simple filter. Naïve Bayes Classifier is used for testing the proposed model. Accuracy, Mean absolute error (MAE), Root mean squared error (RMSE) values are calculated to evaluate the proposed hybrid feature selection model.

The rest of the paper is organized as follows: Session 2 presents the literature study, Session 3 to 5 describes the hybrid feature selection model and the metrics used, results are presented in session 6 and the conclusion is presented in session 7.

2. LITERATURE REVIEW

Software fault prediction is interesting lot of people mainly due to its impact on software quality assurance. One of the main goals is to accurately predict the faulty modules. Today’s software’s for example aircraft software or defence systems are having millions of lines of code and are highly complex. Feature selection has to be considered as not all the attributes are important to consider to develop the fault prediction model and also to avoid redundant information. Also it is computationally expensive to consider all attributes as there where around some hundreds and more attributes.

Removing irrelevant and redundant features from the dataset, feature selection helps improve the performance of learning models by:

- Lessening the effect of the curse of dimensionality.
- Improving generality capability.
- Fast up learning process.
- Improving model interpretability.

Feature Selection Algorithms selects a set of ‘m’ features from set of ‘n’ features, Where m < n. Filters are simple hence preferred for large dataset but less accurate than computationally complex wrappers.

In [4] various classifiers like Naïve Bayes, Logistic Models Trees, RBF Network, Classification via Regression, RBF Network, and Simple Logistic are used for KC1 dataset with 2107 modules. However, no feature selection algorithm was used. Thus, using all features will increase classifier complexity. Max accuracy obtained is 70.99% only for CvR classification. Naïve Bayes gave a least accuracy value of 62.46% only.

In [5] proposed a feature selection method namely Decision Tree Induction Rule based (DTIRB) feature set. DTIRB uses three decision tree algorithms J48, CART, BFTree for feature selection. For KC1 dataset 15 features where selected by DTIRB. This model is evaluated with 18 classifiers and Naïve Bayes showed a better result of 0.25 MAE and 0.48 RMSE only.

3. METHODOLOGY

Removing the irrelevant and redundant features from the data helps to improve the performance of learning models. The two well-known feature selection methods are wrapper and filter. Filter
methods evaluate the goodness of the feature subset by using the intrinsic characteristic of the data. Following are the filter based feature selection algorithms that are evaluated:

- Correlation based Feature Selection
- Chi-Squared
- OneR
- Gain Ratio

3.1. Correlation based Feature Selection (CFS)
CFS identifies relevant features when moderate feature dependencies exist [6]. CFS selects a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. Here low correlated feature is preferred than a highly correlated one as they may be redundant instance.

3.2. Chi-Squared
Chi-squared Attribute evaluator evaluates the worth of an attribute by computing the value of the chi-squared statistic and is a widely used feature selection algorithm [7].

3.3. OneR
OneR, short for “One Rule”, is a simple classifier that generates one-level decision tree. OneR evaluates the worth of an attribute by using the OneR classifier. OneR classifier used cross-validation to estimate the accuracy of the learning scheme for a set of attributes [8]. It combines the C4.5 decision tree and Gaussian distribution.

3.4. Gain Ratio
Gain Ratio evaluates the worth of an attribute by measuring the gain ratio with respect to the class.

\[
\text{GainR}(\text{Class, Attribute}) = \frac{(H(\text{Class}) - H(\text{Class} \mid \text{Attribute}))}{H(\text{Attribute})}
\]

H specifies the entropy. Entropy is a measure of the uncertainty associated with a random variable. Information gain ratio biases the decision tree against considering attributes with a large number of distinct values. So it solves the drawback of information gain namely, information gain applied to attributes that can take on a large number of distinct values might learn the training set too well. For example, suppose that we are building a decision tree for some data describing a business's customers. Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high information gain, because it uniquely identifies each customer, but we do not want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize customers we haven't seen before.

On the other hand Wrapper methods searches in the space of subsets of set of features using cross validation. They use the induction algorithm to evaluate the feature subsets [9]. Wrappers are more complex and take more computation time but are more accurate than filter. Some of Wrapper based Feature Selection evaluated are:

- Naïve Bayes
3.5. Naïve Bayes

Naïve Bayes is the most used classifier and is based on Bayes theory. It assures that the presence of a particular feature is unrelated to another feature [10]. The advantage of Naïve Bayes is that it requires a small amount of training data to estimate the parameters necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire matrix.

3.6. RBF Network

Radial Basis Function Network implements a normalized Gaussian radial basis function network [11]. It uses the k-means clustering algorithm to provide the basis functions and learns either a logistic regression (discrete class problems) or linear regression (numeric class problems) on top of that. Symmetric multivariate Gaussians are fit to the data from each cluster. If the class is nominal it uses the given number of clusters per class. It standardizes all numeric attributes to zero mean and unit variance. RBF networks have the advantage of not being locked into local minima.

3.7. J48

J48 is a Weka implementation of Quinlan’s C4.5 decision tree algorithm [12]. A decision tree is a tree structure where non-terminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes. It is a popular and classic machine learning algorithm.

J48 is based on the ID3 algorithm developed by Ross Quinlan, with additional features to address problems that ID3 find difficult to deal. Given a set $S$ of cases, J48 first grows an initial tree using the divide-and-conquer algorithm as follows:

- If all the cases in $S$ belong to the same class or $S$ is small, the tree is leaf labelled with the most frequent class in $S$.

- Otherwise, choose a test based on a single attribute with two or more outcomes. Make this test as the root of the tree with one branch for each outcome of the test, partition $S$ into corresponding subsets $S_1, S_2, ...$ according to the outcome for each case, and apply the same procedure recursively to each subset.

J48 uses two heuristic criteria to rank possible tests: information gain and the default gain ratio. After the building process, each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequence (post condition). J4.8 is latest decision tree classifier better than CART, CHAID, ID3 classifiers.

In the Hybrid feature selection model, first step is to filter the feature set by using filter and use this selected feature set as the input to the second step of wrapper feature selection. By this method only a least set of features is produced with which the prediction can be done and this model also produces better result than a simple filter or a wrapper. The proposed Feature selection model is shown following fig.1. Here the prediction is done with least number of relevant necessary features alone.
4. DATASET

Data set used here for the evaluation of feature selection algorithms is KC1 NASA’s data set from promise software engineering repository [2]. KC1 is a “C++” system implementing storage management for receiving and processing ground data. It has a total of 2109 modules with 326 defective instances. The following table 1 depicts the source code metrics of KC1 dataset. Out of 22 attributes, defect is the string value of either true or false or 1/0 which says whether a module is defective or not. This DEFECT becomes the predicted value. And the rest of the attributes are numeric.

Table 1: LIST OF KC1 DATASET ATTRIBUTES

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loc</td>
</tr>
<tr>
<td>2</td>
<td>v(g)</td>
</tr>
<tr>
<td>3</td>
<td>ev(g)</td>
</tr>
<tr>
<td>4</td>
<td>iv(g)</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>L</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
</tr>
<tr>
<td>9</td>
<td>d</td>
</tr>
<tr>
<td>10</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>e</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>locCode</td>
</tr>
<tr>
<td>14</td>
<td>locComment</td>
</tr>
<tr>
<td>15</td>
<td>locBlank</td>
</tr>
<tr>
<td>16</td>
<td>locCodeAndComment</td>
</tr>
<tr>
<td>17</td>
<td>uniq_Op</td>
</tr>
<tr>
<td>18</td>
<td>uniq_Opnd</td>
</tr>
<tr>
<td>19</td>
<td>total_Op</td>
</tr>
<tr>
<td>20</td>
<td>total_Opnd</td>
</tr>
<tr>
<td>21</td>
<td>branchCount</td>
</tr>
<tr>
<td>22</td>
<td>defects {false,true}</td>
</tr>
</tbody>
</table>

1) loc: This metric describes the total number of lines for a given module. This is the sum of the executable lines and the commented lines of code and blank lines. Pure, simple count from open bracket to close bracket and includes every line in between, regardless of character content.

2) \( v(g) \): Cyclomatic Complexity \((v(g))\), measures the number of "linearly independent paths". A set of paths is said to be linearly independent if no path in the set is a linear combination of any other paths in the set through a program's "flow graph". A flow graph is a directed graph where each node corresponds to a program statement, and each arc indicates the flow of control from one statement to another. \( v(g) \) is calculated by,

\[
v(g) = e - n + 2
\]  

where "g" is a program's flow graph, "e" is the number of arcs in the flow graph, and "n" is the number of nodes in the flow graph. The standard McCabe rules ("v(g)">10), are used to identify fault-prone module.

3) \( ev(G) \): Essential Complexity \((ev(g))\) is the extent to which a flow graph can be reduced by decomposing all the sub-flow graphs of ‘g’ that are "D-structured primes". Such "D-structured primes" are also sometimes referred to as 'proper one-entry one-exit sub-flow graphs [1]'. \( ev(G) \) is calculated using,

\[
ev(G) = v(G) - m
\]  

where "m" is the number of sub-flow graphs of "g" that are D-structured primes.

4) \( iv(g) \): Design Complexity \((iv(g))\), is the Cyclomatic complexity of a module's reduced flow graph. The flow graph, "g", of a module is reduced to eliminate any complexity which does not influence the interrelationship between design modules. According to McCabe, this complexity measurement reflects the modules calling patterns to its immediate subordinate modules.

5) N: This metric describes the Halstead total operators + operands.

6) V: This metric describes the halstead \((V)\) metric of a module that contains the minimum number of bits required for coding the program.
7) L: This metric describes the halstead level (L) metric of a module i.e. level at which the program can be understood.

8) T: This metric describes the halstead programming time metric of a module. It is the estimated amount of time to implement the algorithm.

9) d: The difficulty level or error proneness (d) of the program is proportional to the number of unique operators in the program.

10) i: Intelligence(i) shows the complexity of a given algorithm independent of the language used to express the algorithm. The intelligence Content determines how much is said in a program.

11) e: This metric describes the halstead effort (e) metric of a module. Effort is the number of mental discriminations required to implement the program and also the effort required to read and understand the program.

12) B: This metric describes the halstead error estimate metric of a module. It is an estimate for the number of errors in the implementation.

13) I0Code: The number of lines of executable code for a module. This includes all lines of code that are not fully commented.

14) I0Comment: This metric describes the number of lines of comments in a module.

15) I0Blank: Halstead's count of blank lines.

16) locCodeAndComment: This metric describes the number of lines which contain both code & comment in a module.

17) uniq_Op: This metric describes the number of unique operators contained in a module i.e. the number of distinct operators in a module.

18) uniq_Opnd: This metric describes the number of unique operands contained in a module. It is a count of unique variables and constants in a module.

19) total_Op: This metric describes the total usage of all the operators.

20) total_Opnd: This metric describes the total usage of all the operands.

21) branchCount: It describes the number of decision points in a given module. Decisions are caused by conditionals statements.

22) defects: It describes weather the particular module is defective or not. This attribute is used for prediction.

5. CROSS-VALIDATION TEST

Most common 10-fold cross validation method is used for detecting faulty instances. One round of cross validation is partitioning a sample of data into complementary subsets, performing the analysis on one set known as training set and validating the analysis on other set known as testing set. In 10-fold one of them is treated as training set and rest 9 as testing set. This is repeated 10 times where each one of the subset is used as training set once.
6. EXPERIMENTAL RESULTS AND ANALYSIS

Waikato Environment for Knowledge Analysis (WEKA) [13] open source suite is used for analyzing the prediction accuracy. It was developed at the University of Waikato, New Zealand. It is written in Java and it is free software available under the GNU General Public License. Accuracy is calculated using the confusion matrix shown in table 2.

Table 2: Confusion Matrix

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Real data</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault</td>
<td>No Fault</td>
<td></td>
</tr>
<tr>
<td>Fault</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>No Fault</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Confusion matrix has 4 divisions which represent True Positives (a), False Positives (b), False Negatives(c), and True Negatives (d). True Positives are the correctly classified modules that has fault. False Positives are the fault free modules that are incorrectly classified as faulty. False Negatives are the faulty modules that are incorrectly classified as fault free modules. True Negatives are the non-faulty modules that are correctly classified as faulty. Accuracy is given by,

\[ \text{Accuracy} = \frac{a + d}{a + b + c + d} \]  

Mean absolute error (MAE) is the sum of the absolute error for all cases in the partition, divided by the number of cases in the partition, excluding cases with missing values.

\[ \text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |a_i - c_i| \]  

Root mean squared error (RMSE) is the square root of the mean of the squares of the probability score’s complement, divided by the number of cases in the partition.

\[ \text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - c_i)^2} \]  

MAE and RMSE should always be less for better prediction. Accuracy, Mean absolute error (MAE) and Root mean squared error (RMSE) values are measured to evaluate the performance. NASA’s KC1 dataset is used for this study. Here Naive Bayes Classifier is the classifier used for predicting the faulty instances. Table 3 shows the results of prediction on KC1 dataset.

Table 3: Comparison of Hybrid Feature Selection with Filter and Wrapper Feature Selection

<table>
<thead>
<tr>
<th>Feature selection Method</th>
<th>No. of Attributes Selected</th>
<th>Accuracy</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>All features</td>
<td>21</td>
<td>82.36 %</td>
<td>0.176</td>
<td>0.414</td>
</tr>
<tr>
<td>CFS</td>
<td>8</td>
<td>82.41 %</td>
<td>0.177</td>
<td>0.406</td>
</tr>
<tr>
<td>CFS-NB</td>
<td>1</td>
<td>85.59 %</td>
<td>0.201</td>
<td>0.352</td>
</tr>
<tr>
<td>CFS-RN</td>
<td>3</td>
<td>84.83 %</td>
<td>0.167</td>
<td>0.371</td>
</tr>
<tr>
<td>CFS-J48</td>
<td>2</td>
<td>84.16 %</td>
<td>0.183</td>
<td>0.371</td>
</tr>
<tr>
<td>CSq</td>
<td>top 11</td>
<td>82.98 %</td>
<td>0.171</td>
<td>0.403</td>
</tr>
<tr>
<td>CSq-NB</td>
<td>1</td>
<td>85.63 %</td>
<td>0.201</td>
<td>0.35</td>
</tr>
<tr>
<td>CSq-RN</td>
<td>2</td>
<td>83.74 %</td>
<td>0.179</td>
<td>0.377</td>
</tr>
</tbody>
</table>
From the above results when no feature selection is used only a least accuracy is obtained. CFS selected 8 attributes namely v, d, i, IOCode, IOComment, IOBlank, uniq_Opnd and branchCount. Naïve Bayes selected only one attribute ‘v’ alone and has a better prediction value of 85.59% which is better than other feature selection methods which also shows that reduced feature set helps in better feature selection than the complete set. For ChiSq, OneR and Gain Ratio which where ranker based top 11 features are considered as above threshold and are the selected attributes. total_Opnd, n, b, total_Op, d, l, v, t, e, uniq_Op, uniq_Opnd are the selected attributes for ChiSq, uniq_Op, l, IOCode, v(g), iv(g), branchCount, b, IOBlank, locCodeAndComment, IOComment, total_Opnd for OneR and uniq_OPnd, branchCount, d, v, v(g), i, t, e, IOComment, IOCode, n for Gain Ratio respectively. From the above results it is clear that the proposed hybrid feature selection model better predicts than the simple models.

Fig.2 shows the accuracy and fig.3 shows Mean absolute error (MAE) and fig.4 shows Root mean squared error (RMSE) values of various feature selection algorithms and the proposed model with that of all 21 features.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Attributes</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSq-J48</td>
<td>3</td>
<td>84.02%</td>
<td>0.191</td>
<td>0.364</td>
</tr>
<tr>
<td>OneR</td>
<td>top 11</td>
<td>82.50%</td>
<td>0.175</td>
<td>0.406</td>
</tr>
<tr>
<td>OneR-NB</td>
<td>1</td>
<td>85.63%</td>
<td>0.201</td>
<td>0.35</td>
</tr>
<tr>
<td>OneR-RN</td>
<td>2</td>
<td>84.12%</td>
<td>0.185</td>
<td>0.365</td>
</tr>
<tr>
<td>OneR-J48</td>
<td>3</td>
<td>84.54%</td>
<td>0.184</td>
<td>0.364</td>
</tr>
<tr>
<td>GainR</td>
<td>top 11</td>
<td>82.60%</td>
<td>0.173</td>
<td>0.406</td>
</tr>
<tr>
<td>GainR-NB</td>
<td>1</td>
<td>85.35%</td>
<td>0.203</td>
<td>0.351</td>
</tr>
<tr>
<td>GainR-RN</td>
<td>1</td>
<td>85.06%</td>
<td>0.207</td>
<td>0.345</td>
</tr>
<tr>
<td>GainR-J48</td>
<td>3</td>
<td>83.45%</td>
<td>0.175</td>
<td>0.384</td>
</tr>
</tbody>
</table>

Results show that the reduced data set by hybrid model better predicts. Short notations used in the table 2 and fig. 1 are CFS for Correlation based Feature Selection, NB for Naïve Bayes, CSq for Chi-Squared RN for RBF Network and GainR for Gain Ratio respectively.
7. CONCLUSION AND FUTURE WORK

Comparison of results shows that hybrid feature selection gives better performance. Reduced feature set plays an important role in software fault prediction. This reduced feature set improves the performance of learning algorithm and also reduces the computational cost. This further reduces the complexity of the classifier also. From the results the hybrid model with Naïve Bayes better predicts the faults. Hence accuracy is better and also the complexity of the classifier is reduced with reduced feature set that is selected by the hybrid feature selection approach. This prediction model will help to pay more attention to fault prone modules in future versions of same software or similar products and also will improve the productivity, easy maintenance and also the quality of software.

Only one dataset is considered for the work and in future more datasets of larger projects will be considered. Classifier must also be considered for better prediction in future work.

REFERENCES


Authors

C. Akalya devi is an M.E student in Sri Shakthi Institute of Engineering and Technology, Affiliated to Anna University of Coimbatore, Tamil Nadu, India. Her research interest includes software quality prediction and data mining.

K.E. Kannammal is working as Associate Professor in Department of Computer Science and Engineering at Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India. She is a research scholar. Her research interests include Sensor Networks and data mining. She has published papers in conferences and journals.

B. Surendiran is an Assistant Professor (SRG) at Kumaraguru College of Technology, Coimbatore, Tamil Nadu, India. He did his PhD at National Institute of Technology, Tiruchirappalli, India. His research interest includes digital mammogram analysis and image processing. He has received Best Paper Award for his paper at ARTcom2009 International Conference. He had published over 15 international journals and conference papers.