An Application of a Rule-Based Model in Software Quality Classification

Lofton A. Bullard  
Taghi M. Khoshgoftaar  
Florida Atlantic University  
Boca Raton, Florida USA  
{taghi, lofton}@cse.fau.edu

Kehan Gao  
Eastern Connecticut State University  
Willimantic, Connecticut USA  
gaok@easternct.edu

Abstract

A new rule-based classification model (RBCM) and rule-based model selection technique are presented. The RBCM utilizes rough set theory to significantly reduce the number of attributes, discretion to partition the domain of attribute values, and Boolean predicates to generate the decision rules that comprise the model. When the domain values of an attribute are continuous and relatively large, rough set theory requires that they be discretized. The subsequent discretized domain must have the same characteristics as the original domain values. However, this can lead to a large number of partitions of the attribute’s domain space, which in turn leads to large rule sets. These rule sets tend to form models that over-fit. To address this issue, the proposed rule-based model adopts a new model selection strategy that minimizes over-fitting for the RBCM. Empirical validation of the RBCM is accomplished through a case study on a large legacy telecommunications system. The results demonstrate that the proposed RBCM and the model selection strategy are effective in identifying the classification model that minimizes over-fitting and high cost classification errors.

Keywords: rule-based classification model, rough set, reducts, discretization, software quality classification

1. Introduction

The quality of high-assurance and mission-critical systems has gained much attention in the modern software-infused technology infrastructure. Software quality can be measured in terms of software quality measurements. For example, number of system failures, number of fault-prone modules and so forth. An effective software quality improvement approach is to identify and target software modules that are most likely to be defective or of poor quality. A software metric-based quality classification model can be used by software analysts to identify faulty software modules and increase quality. Calibrated using measurements collected early in the development phase, these models have been shown by research to be effective quality predictors. For example, software complexity metrics and fault data collected from a previous system release or similar project can be used to model the relationship between the complexity metrics and the observed software quality, i.e., fault-prone (fp) or not fault-prone (nfp) [1]. Some software metric-based quality classification models that have been investigated and validated include logistic regression [6], classification trees [7], discriminant analysis [12], and computational intelligence-based methods [3, 4, 8, 10, 18, 19].

In comparison to existing software quality classification models, rule-based models are particularly attractive to the software quality team because of their simplicity, comprehensibility, and ease in interpretation of the obtained results. The key to the practical usefulness of such models is generating the appropriate rules that accurately depict the underlying relationship between the independent variables, i.e., software metrics, and the dependent variable, i.e., software quality. Various algorithms and techniques have been proposed for generating decision rules. Some are based on inductive learning, while others are based on rough set theory. A method to generate decision rules using dynamic reducts was proposed by Bazan [2]. More specifically, frequently occurring stable reducts of a given decision table that appear in random samples are taken from the decision table. Other related works include: Skowron [16] introduced a technique that, when applied over consistent decision tables, makes it possible to obtain decision rules with minimal sets of predictors; an incremental learning algorithm was proposed in [15] for computing a set of all minimal decision rules based on decision matrices.

Rough set theory [9] is an active research area for application to the data mining and knowledge discovery field. In
the context of software metrics collected for a given software system, not all metrics are equally significant indicators of software quality. Thus, attribute selection is an important aspect of software quality prediction. Rough set theory can be used to identify subsets of attributes, called reducts, that have the same discrimination power as a complete set of attributes. Once the reducts are identified, a set of decision rules are obtained to identify fault-prone software modules. When the domain values of an attribute are continuous and relatively large, rough set theory requires that they be discretized. The subsequent discretized domain must have the same characteristics as the original domain values. However, this can lead to a large number of partitions of the attributes’ domain space, which in turn leads to large rule sets. These rule sets tend to form models that over-fit. Over-fitting is observed when the performance of a classification model on the training dataset is over optimistic. To address this issue, in a previous study [11], we discussed the methodology, rough set theory, the equal frequency binning algorithm, as well as the model selection technique. Section 3 presents an analytical case study using a real-world data for a large telecommunications system. Finally, we conclude the paper in Section 4.

2. Methodology

2.1. Rough Sets

Based on classical set theory, rough sets were introduced in 1982 by Pawlak [9]. Using the concept of equivalence relations, partitions of a set of objects can be formed, subsets of significant attributes identified, and decision rules extracted, all based on the attribute values of the objects. Rough set theory can be used to analyze the dependency relationship between the independent and the dependent variables. This relationship is used to determine whether the dependent attribute can be characterized by the values of the independent attributes.

A main advantage of rough sets is that redundant attributes can be eliminated, and a subset of attributes with the same discrimination power as the original complete set of attributes emerges, called reducts. Once found, reducts can be used to generate decision rules to classify unseen objects. See [14] for a more detailed discussion on rough set theory.

2.2. Equal Frequency Binning Algorithm

Equal frequency binning is a simple method that can be used to discretize attributes with continuous domains. The algorithm is summarized below:

1. Sort the domain of values of each attribute in ascending order.

2. For each attribute divide the domain of values into \( N \) equal bins (intervals) using one of the following rules:
   - If the domain size is evenly divisible (no remainder) by \( N \), create \( N \) bins of equal size.
   - If the domain size is divisible by \( N \) but not evenly (remainder), let the first \( N-1 \) bins have the same number of elements and put the remaining elements in the \( N^{th} \) bin.
   - If the domain size is less than \( N \), find a value for \( N \) such that the bins are mutually exclusive.

3. Let the \( i^{th} \) interval’s lower bound of the \( j^{th} \) attribute be \( L_{ij} \), and the \( i^{th} \) interval’s upper bound of the \( j^{th} \) attribute be \( U_{ij} \), where \( i = 1 \ldots N \), and \( j = 1 \ldots m \), where \( m \) is the number of metrics.

2.3. Rule-Based Classification Modeling

In the context of a software quality classification program, the metric \( X_j \) for a given module \( i \), is either \((x_{ij} \geq L_{1j}) \land (x_{ij} < U_{1j})\) or \((x_{ij} \geq L_{2j}) \land (x_{ij} < U_{2j})\) or \(\ldots (x_{ij} \geq L_{Nj}) \land (x_{ij} < U_{Nj})\), where \(L_{kj}\) and \(U_{kj}\) are the lower and upper limits of the \( j^{th} \) attribute for the \( k^{th} \) interval. If \( m \) software complexity metrics are selected as the most significant attributes for the model, then at most \( I^m \) rules can be formed with each rule having \( 2m \) terms, where \( I \) is the number of intervals. For example, if \( m = 2 \) and \( I = 2 \) then at most four possible rules can be generated. They are listed in Table 1, where a ‘\&’ indicates a Boolean AND.

Each instance in the fit dataset can be classified by only one of the \( I^m \) rules. When a module satisfies a given rule, it implies that all \( 2m \) individual Boolean terms of the rule
**Table 1. Possible Rules with 2 Significant Attributes**

<table>
<thead>
<tr>
<th>Index</th>
<th>Boolean Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((x_1 \geq L_{11}) \land (x_1 &lt; U_{11}) \land (x_2 \geq L_{12}) \land (x_2 &lt; U_{12}))</td>
</tr>
<tr>
<td>2</td>
<td>((x_1 \geq L_{11}) \land (x_1 &lt; U_{11}) \land (x_2 \geq L_{22}) \land (x_2 &lt; U_{22}))</td>
</tr>
<tr>
<td>3</td>
<td>((x_1 \geq L_{21}) \land (x_1 &lt; U_{21}) \land (x_2 \geq L_{22}) \land (x_2 &lt; U_{22}))</td>
</tr>
<tr>
<td>4</td>
<td>((x_1 \geq L_{21}) \land (x_1 &lt; U_{21}) \land (x_2 \geq L_{12}) \land (x_2 &lt; U_{12}))</td>
</tr>
</tbody>
</table>

are true. This is an example of a complete matching classification scheme. In the case of software quality classification, a module is defined as fault-prone (fp) if its quality factor (such as number of faults) exceeds the selected threshold value, and not fault-prone (nfp) otherwise. The chosen threshold value is usually dependent on the project management’s quality improvement objectives. If we denote \(n_{fp}\) and \(n_{nfp}\) as the number of fp and nfp modules that satisfy a given rule, we can determine the probability of a rule as being fp or nfp. We use the following equations to compute these probabilities.

\[
p_{fp} = \frac{n_{fp}}{n_{fp} + n_{nfp}} \quad (1)
\]

\[
p_{nfp} = \frac{n_{nfp}}{n_{fp} + n_{nfp}} \quad (2)
\]

During the model training process if a rule does not classify any modules, it is deleted from the rule set.

Once we determine the probabilities of each rule, \(p_{fp}\) and \(p_{nfp}\), we can rank the rules in ascending \(p_{fp}\). An asset of ordering the rules, is that we can locate an optimum threshold value of \(p_{fp}\) to separate the fp from the nfp rules. We denote this particular value of \(p_{fp}\), as \(\theta\). All the rules with \(p_{fp} \geq \theta\) will be classified as fp rules, and those with \(p_{fp} < \theta\) will be classified as nfp rules. The implementation of the complete matching classification scheme we used for identifying a module, \(x_i\), if there is a rule with the characteristics of the module, as either \(fp\) or \(nfp\) is shown below.

\[
Class(x_i) = \begin{cases} 
fp & \text{if } p_{fp}(rule^r(x_i)) \geq \theta \\
nfp & \text{if } p_{fp}(rule^r(x_i)) < \theta 
\end{cases} \quad (3)
\]

where \(rule^r(x_i)\) implies that module \(i\) satisfies rule \(r\); \(p_{fp}(rule^r(x_i))\) represents the probability of rule \(r\), classifying the \(i^{th}\) module, being \(fp\). The rule index, \(r\), varies from 1 to \(I^n\). An analyst can vary \(\theta\) as per projects needs. For example, if \(\theta\) is relatively low, only a few rules will be nfp rules, implying that many modules will be classified as fp. On the other hand, if the selected \(\theta\) is relatively high, many modules will be classified as nfp.

Since the characteristics of the rules are inherited from the training dataset, the test datasets may contain characteristics that are not represented in the rules. In this case some modules will not be classified by the rules generated from the training dataset. In such situations, a partial matching classification scheme may be used to determine the class membership of an instance. The implementation of the partial matching classification scheme we used for identifying a module, \(x_i\), if there is a rule with the characteristics of the module, as either \(fp\) or \(nfp\) is shown below.

\[
Class(x_i) = \begin{cases} 
nfp & \text{if } \frac{R_{fp}}{R_{nfp}} \geq \frac{F_{fp}}{F_{nfp}} > F_{fp} \\
fp & \text{otherwise} 
\end{cases} \quad (4)
\]

where \(R_{fp}\) and \(R_{nfp}\) are the total number of fp rules and nfp rules, respectively, in the rule set derived from the training dataset; \(F_{fp}\) and \(F_{nfp}\) are the number of fp and nfp rules that fired using the partial matching classification scheme for a particular training instance. A rule only fires when \(l\) of its Boolean predicates are satisfied by the instance, where \(1 \leq l < 2m\). The term \(\frac{R_{fp}}{R_{nfp}}\) is used to normalize \(F_{nfp}\), because its magnitude is much greater than \(F_{fp}\). If this term was not included in the rule, there would be an inherent bias towards classifying a module as nfp. It is worthwhile noting that the fp and nfp rules were determined by the complete matching classification scheme.

### 2.4. Rule-Based Model Selection

In the context of a two-group classification problem, two types of misclassifications can occur: Type I and Type II. Table ?? presents the meaning of the two types of errors. The cost of a Type I error may involve wasted effort, and reviews in inspecting high quality program modules. A Type II error may indicate a missed opportunity of correcting a faulty module prior to operations. Since a Type II misclassification is more severe than a Type I misclassification, the model, which has the minimum difference between Type I and Type II misclassification rates with Type II misclassification rates as low as possible, is preferred. For the rule-based classification model, it was observed that, when training the model, as the number of intervals, \(I\), increases, the balance between the Type I and Type II misclassification rates improves. From previous empirical studies [11], we observed this to be an indicator of over-fitting. The major contribution of our model selection technique is its ability to identify a preferred-model with the least amount of over-fitting from an ensemble of RBCMs that were built using the same data and the same attributes. Over-fitting presents itself in a model whose performance on the training data is significantly better than the model’s performance on the test data.

In this model selection technique, the misclassification rates, Type I and Type II, are computed using the complete matching classification scheme and the partial matching classification scheme for the fit dataset, varying the values of \(I\) and \(\theta\). The rationale is that RBCMs that over-fit
usually do so on a new dataset which contains a sufficient number of instances that are not similar to instances in the fit dataset. When a RBCM is presented with instances like these, its partial classification scheme is used to determine the class an instance belongs to. Thus, we use the training dataset, the complete matching classification scheme, the partial matching classification scheme, and the rule set of a RBCM to approximate the amount of over-fitting.

Once each instance in the training dataset has been classified using the complete matching classification scheme and the partial matching classification scheme, we calculate the penalty distance between the Type I and Type II misclassification rates of the two matching classification schemes using the following formula:

\[ E = (1 + \Gamma^n - \varepsilon) \sqrt{(T1_c - T1_p)^2 + (T2_c - T2_p)^2} \]  

where \( \varepsilon \) is the number of rules generated from the fit dataset for the model, so \( \varepsilon = R1_p + R2_p; T1_c \) and \( T2_c \) are the Type I and Type II misclassification rates for the model using the complete matching scheme, respectively, and \( T1_p \) and \( T2_p \) are the Type I and Type II misclassification rates for the model using the partial matching scheme, respectively; the first term in the formula, \( 1 + \Gamma^n - \varepsilon \), is the penalty factor. It was observed in a previous case study, that as the number of intervals increased, the chance that models do not produce a complete rule set increased, and the amount of model over-fitting increased. The penalty factor punishes models that do not produce a complete rule set. The second term in the formula, \( \sqrt{(T1_c - T1_p)^2 + (T2_c - T2_p)^2} \), is the distance factor. It is the Euclidean distance between the Type I and Type II misclassification rates of the complete and partial matching classification schemes for a RBCM. It represents an approximate measure of the amount of over-fitting to expect for a particular RBCM. As its value increases, so does that amount of over-fitting. When the rule set of a RBCM is complete, the penalty factor will equal one, and the amount of over-fitting is directly proportional to the distance factor. However, when the rule set of a RBCM is not complete, the model is punished by an amount proportional to the product of the penalty factor and the distance factor.

Effective RBCMs have the characteristics of minimizing over-fitting and balancing the Type I and Type II misclassification rates, with the Type II misclassification error rate being as low as possible. The penalty distance measurement encapsulates these important characteristics of effective RBCMs. Models that do not generate a complete rule set tend to over-fit or have lower classification accuracies than models that generate a complete rule set, i.e., models with incomplete rule sets must rely on their partial matching classification scheme to classify any unseen instances. The penalty distance is able to identify the model with the least amount of over-fitting.

3. Case Study

In this section, we discuss the tool we used to generate the reducts, the datasets used in the case study, and the classification performance results of the RBCMs. A detailed analysis of the results as well as comparisons to other classification models are provided.

3.1. The Rough Set Exploration System

The Rough Set Exploration System (RSES) is a set of software tools that are used for rough set computations in data mining [2]. RSES implements algorithms to manage and edit data structures that are used in user experiments and defined in the RSES library, reduce data (objects and attributes) [17], quantify data [5], generate templates and decomposition trees [13], and classify objects [2]. The tool provides a graphical user interface which allows experiments to be constructed and executed with ease.

In our experiments we are interested in the discretization and the reduction algorithms. A discussion of background information for the discretization algorithm used by RSES is given in [5]. RSES also implements several reduction algorithms for reducing the number of irrelevant attributes [17]. These algorithms included an exhaustive search algorithm and several genetic algorithms. When the number of attributes is large (greater than 20), an exhaustive search for reducts is impractical. RSES uses genetic algorithms to find approximate and heuristic solutions to the attribute selection problem.

3.2. System Description

We conducted a case study of a very large legacy telecommunications system maintained by professional programmers in a large organization using the procedural-development paradigm, and written in a high-level language (Protel) similar to Pascal. This embedded-computer application included numerous finite-state machines. The entire system had significantly more than ten million lines of code. We studied four consecutive releases labelled 1 through 4. The earliest release was the basis for the training dataset,
and the subsequent releases were the basis for the evaluation data.

A module consisted of a set of functionally related source-code files. An average module had about four files. Fault data was collected at the module-level by the problem reporting system. A module was considered fault-prone if any problems discovered by customers resulted in changes to source code in the module, and not fault-prone otherwise. The numbers of modules considered in releases 1, 2, 3, and 4 were 3649, 3981, 3541, and 3978, respectively. The software metrics used in the study included 24 product metrics and 4 execution metrics. Refer to [8] for more detailed information about the system.

### 3.3. Results and Analysis

Due to the large number of independent attributes (28 in the experiment) and the large domain of possible values the attributes could be assigned, RSES required that the training dataset be discretized before generating any reducts. The discretization algorithm was implemented in RSES. We used three different genetic filter algorithms, also implemented in RSES, to determine the most significant reducts to be used to build the candidate classification models. The reduct selection criterion was to select the reducts that were generated by each of the three algorithms. Each algorithm had to generate at least 10 reducts, for a total of 30, before there were any reducts that were common to all three algorithms. We let the event of an algorithm generating a particular reduct be a vote for that reduct. Using this method, there was only one reduct that had three votes, FILINCUQ¹ and VARSPNSM², which was selected as the most significant reduct. Once we identified the most significant reduct, we used the equal frequency binning algorithm to discretize the attribute values of the instances. RSES did not save any intermediate data from the discretization process. Therefore, we implemented the equal frequency binning algorithm and applied it to the original domain values of the attributes that were included in the reduct. The equal frequency binning algorithm partitioned the values of the attributes into mutually exclusive intervals.

Table 3 shows an example of the results of the RBCM on the fit dataset we generated when \( I = 4 \) and \( m = 2 \), where \( I \) represents the \# of the intervals, and \( m \) represents the \# of the metrics selected for the model. Since the selected reduct contains two attributes, FILINCUQ and VARSPNSM, \( m = 2 \). In the table, the column labelled “Rule #” specifies the indexes of the Boolean rules used in the model; the column labelled “\( \theta \)” presents the threshold value of \( \theta \); the columns labelled “Type I,” “Type II,” and “Overall” presents the misclassification rates for the model that corresponds to each value of \( \theta \). The preferred model, which had the minimum difference between Type I and Type II misclassification rates, with Type II misclassification rate being as low as possible, is shown in **bold** in the table.

Table 4 displays the results on the fit dataset of the RBCM experiments performed using the proposed model selection strategy. We varied \( I \) from 2 to 20 to produce 19 RBCM. In the table, the column labelled “\# of Rules” presents the number of rules generated for a specified value of \( I \); the columns labelled **Fit CS** and **Fit PS** presents the Type I and Type II misclassification rates for the preferred model that corresponds to each value of \( I \) using the complete matching classification scheme (CS) and the partial matching classification scheme (PS); the column labelled “Penalty Distance” presents the distance between the misclassification rates for each RBCM for the two matching schemes. The preferred model is the one which has the smallest penalty distance. It is **bold** in the table.

Table 5 presents the quality-of-fit and the prediction performances of the RBCMs on the three test datasets (releases 2, 3, and 4). The preferred model is **bold** in the table. We can observe that for the selected preferred classification model, the Type I misclassification rates on releases 2 and 3, are about 2% lower than the Type I misclassification rate of the model on the fit dataset (release 1). The Type I misclassification rate on release 4 was an insignificant 0.12% higher than the Type I misclassification rate of the model on the fit dataset. However, the Type II misclassification rates of the preferred model were significantly lower on all three test datasets than its performance on the fit dataset, 3.6%, 5.0%, and 1.3%, respectively.

When compared to the results of other RBCM, we can observe the effectiveness of the model selection strategy. For example, referring to Table 4 and Table 5, consider the

---

¹An attribute that represents the \# of distinct include files

²An attribute that represents the total span of variables
model with the second lowest value of $E$ (penalty distance); the model with a value of $I = 5$. This model’s Type I misclassification rates are lower on releases 2 and 3 than on the fit dataset, 1.1% and 1.0%, respectively. On release 4, the model’s Type I misclassification rate was 0.8% higher than its performance on the fit dataset. On releases 2 and 3 the model’s Type II misclassification rates were 1.7% and 2.6% lower, respectively, however, on release 4 the model’s Type II misclassification was 3.4% higher than its performance on the fit dataset. This bias becomes more severe as the value of $I$ increases.

When compared to all the other models, the performance of the RBCM with the smallest value of $E$ on the test data had the least amount of overfitting. The preferred classification model is fairly balanced across the different software system releases, which is an indicator of stability. The results of this case study verify that we can use the model selection technique presented in Section 2.4 to identify a RBCM that minimizes over-fitting and balances the Type I and Type II misclassification rates, keeping the Type II misclassification rate as low as possible.

In the experiments the proposed RBCM was compared to the rule-based model (RBM) and case-based reasoning (CBR). We investigated RBM in a previous study [11]. Table 6 summarizes the results of the preferred model for each classification technique. Release 1 was used as the fit dataset for each method. Releases 2, 3, and 4 were used to test the RBCM and CBR models. However, RBM required
a validation dataset to determine the preferred classification model. We used release 2 as the required validation dataset for RBM, and releases 3 and 4 as the test datasets. The results demonstrate that although the proposed RBCM does not produce the quality-of-fit as good as the other two methods, it has better predictive performance than the RBM and CBR on the test datasets.

4. Conclusion

Accurate software metric-based classification models could be used by software developers to guide their efforts to reduce software developmental costs and produce a more reliable system. In this study, a new rule-based classification model and the model selection strategy were introduced and their effectiveness was demonstrated by the performance of the selected preferred classification model on the test datasets for a large legacy telecommunications system.

This study was our first investigation related to the proposed model, RBCM. Future research will involve conducting more experiments, including comparisons to other rule-based models on data in various domains, and using more sophisticated discretization methods instead of the simple equal frequency binning algorithm.

References


