ART OF SOFTWARE DEFECT ASSOCIATION & CORRECTION USING ASSOCIATION RULE MINING

Nilamadhab Mishra, MCA, M.PHIL (C S), M.Tech (CSE)
Asst.prof, Krupajal Group of Institution, Orissa
E-mail : nilamadhab.mishra@rediffmail.com

ABSTRACT:

Much current software defect prediction work focuses on the number of defects remaining in a software system. In this paper, I focus about association rule mining based methods to predict defect associations and defect correction effort. This is to help developers detect software defects and assist project managers in allocating testing resources more effectively. Software process improvement method is based on the analysis of defect data. I applied the proposed methods to the SEL defect data consisting of more than 200 projects over more than 15 years. The method is classification of software defects using attributes which relate defects to specific process activities. The results show that, for defect association prediction, the accuracy is very high and the false-negative rate is very low. The comparison of defect correction effort prediction method with other types of methods—PART, C4.5, and Naïve Bays show that accuracy can be improved. I also evaluate the impact of support and confidence levels on prediction accuracy, false negative rate, false-positive rate, and the number of rules. Thus higher support and confidence levels may not result in higher prediction accuracy and a sufficient number of rules must be a precondition for high prediction accuracy.

Key words: Defect association, defect correction effort, defect isolation effort, defect association prediction, Software defect prediction

1. INTRODUCTION

The success of a software system depends not only on cost and schedule, but also on quality. Among many software quality characteristics, residual defects have become the de facto industry standard [1, 2]. Therefore, the prediction of software defects, i.e.,
deviations from specifications or expectations which might lead to failures in operation, has been an important research topic in the field of software engineering for more than 30 years. Clearly, they are a proxy for reliability, but, unfortunately, reliability is extremely difficult to assess prior to full deployment. Current defect prediction work focuses on estimating the number of defects remaining in software systems with code metrics, inspection data, and process-quality data by statistical approaches, capture-recapture (CR) models, and detection profile methods (DPM)\[4\].

The prediction result, which is the number of defects remaining in a software system, can be used as an important measure for the software developer, and can be used to control the software process and gauge the likely delivered quality of a software system. In contrast, propose that the defects found during production are a manifestation of process deficiencies, so they present a case study of the use of a defect based method for software in-process improvement. In particular, they use an attribute-focusing (AF) method to discover associations among defect attributes such as defect type, source, and phase introduced, phase found, component, impact, etc [9]. By finding out the event that could have led to the associations, they identify a process problem and implement a corrective action. This can lead a project team to improve its process during development. I restrict my work to the predictions of defect (type) associations and corresponding defect correction effort.

1. For the given defect(s), what other defect(s) may co-occur?
2. In order to correct the defect(s), how much effort will be consumed?

I use defect type data to predict software defect associations that are the relations among different defect types such as: If defects a and b occur, then defect c also will occur. This is formally written as \( a \land b \Rightarrow c \). The defect Associations can be used for three purposes [3, 4].

First, find as many related defects as possible to the detected defect(s) and, consequently, make more-effective corrections to the software. For example, consider the situation where we have classes of defect a, b, and c and suppose the rule \( a \land b \Rightarrow c \) has been obtained from a historical data set, and the defects of class a and b have been detected occurring together, but no defect of class c has yet been discovered. The rule indicates that a defect of class c is likely to have occurred as well and indicates that we
should check the corresponding software artifact to see whether or not such a defect really exists. If the result is positive, the search can be continued, if rule $a \land b \land c \Rightarrow d$ holds as well, we can do the same thing for defect $d$.

Second, help evaluate reviewers’ results during an inspection. For example, if rule $a \land b \Rightarrow c$ holds but a reviewer has only found defects $a$ and $b$, it is possible that he missed defect $c$. Thus, a recommendation might be that his/her work should be reinserted for completeness.

Third, to assist managers in improving the software process through analysis of the reasons some defects frequently occur together. If the analysis leads to the identification of a process problem, managers have to come up with a corrective action.

At the same time, for each of the associated defects, we also predict the likely effort required to isolate and correct it. This can be used to help project managers improve control of project schedules. Both of our defect association prediction and defect correction effort prediction methods are based on the association rule mining method which was first explored by Agrawal. Association rule mining aims to discover the patterns of co-occurrences of the attributes in a database [5, 6]. However, it must be stressed that associations do not imply causality. An association rule is an expression $A \rightarrow C$, where $A$ (Antecedent) and $C$ (Consequent) are sets of items. The meaning of such rules is quite intuitive: Given a database $D$ of transactions, where each transaction $T \in D$ is a set of items, $A \rightarrow C$ expresses that whenever a transaction $T$ contains $A$, then $T$ also contains $C$ with a specified confidence. The rule confidence is defined as the percentage of transactions containing $C$ in addition to $A$ with regard to the overall number of transactions containing $A$. The idea of mining association rules originates from the analysis of market-basket data where rules like “customers who buy products $p1$ and $p2$ will also buy product $p3$ with probability $c$ percent” are extracted. Their direct applicability to business problems together with their inherent understandability make association rule mining a popular data mining method.
2. PAPER CONTENTS

General method:

The objective of the study is to discover software defect associations from historical software engineering data sets, and help determine whether or not a defect(s) is accompanied by other defect(s). If so, we attempt to determine what these defects are and how much effort might be expected to be used when we correct them. Finally, we aim to help detect software defects and effectively improve software control. For this purpose, first, we preprocessed the NASA SEL data set (see the following section for details), and obtained three data sets: the defect data set, the defect isolation effort data set, and the defect correction effort data set. Then, for each of these data sets, we randomly extracted five pairs of training and test data sets as the basis of the research. After that, we used association rule mining based methods to discover defect associations and the patterns between a defect and the corresponding defect isolation/correction effort. Finally, we predicted the defect(s) attached to the given defect(s) and the effort used to correct each of these defects. We also compared the results with alternative methods where applicable.

Data Source and Data Extraction:

The data we used is SEL Data which is a subset of the online database created by the NASA/GSFC Software Engineering Laboratory (SEL) for storage and retrieval of software engineering data for NASA Goddard Space Flight Center. The subset includes defect data of more than 200 projects completed over more than 15 years. The SEL Data is a database consisting of 15 tables that provide data on the projects’ software characteristics (overall and at a component level), changes and errors during all phases of development, and effort and computer resources used. In the SEL Data, the defects are divided into six types, Table 1 contains the details. In addition, the effort used to correct defects falls into four categories: One Hour or Less, One Hour to One Day, One Day to Three Days, and More Than Three Days.

Analysis approach:

I use the five-fold cross-validation method as the overall analysis approach. That is, for each D of the defect data set, the defect isolation effort data set, and the defect correction effort data set, the inducer is trained and tested a total of five times. I use the
association rule mining method to learn rules from the training data sets. For defect association prediction, the rule learning is straightforward, while for defect correction effort prediction, it is more complicated because the consequent of a rule has to be defect correction effort [7, 8]. Considering the target of association rule mining is not predetermined and classification rule mining has only one predetermined target, the class, I integrate these two techniques to learn defect correction effort prediction rules by focusing on a special subset of association rules whose consequents are restricted to the special attribute, the effort. Once we obtain the rules, we rank them, and use them to predict the defect associations and defect correction effort with the corresponding test data sets. The predictions of defect associations and defect correction effort are both based on the length-first (in terms of the number of items in a rule) strategy.

**Association Rule Discovery:**

Both positive and negative association rules are to be discovered by the project. First, the frequent item sets of potential interest and infrequent item sets of potential interest are generated in a database. Then Extract positive rules of the form \( A \rightarrow B \), and negative rules of the forms \( A \rightarrow \neg B \), \( \neg A \rightarrow B \), and \( \neg A \rightarrow \neg B \).

**Rule ranking Strategy:**

Before prediction, we rank the discovered rules according to the length-first strategy. The length-first strategy was used for two reasons. First, for the defect association prediction, the length-first strategy enables us to find out as many defects as possible that coincide with known defect(s), thus preventing errors due to incomplete discovery of defect associations. Second, for the defect correction effort prediction, the length-first strategy enables us to obtain more-accurate rules, thus improving the effort prediction accuracy. Specifically, the length-first rule-ranking strategy is as follows:

1. Rank rules according to their length. The longer the rules, the higher the priority.
2. If two rules have the same length, rank them according to their confidence values. The greater the confidence values, the higher the priority. The more-confident rules have more predictive power in terms of accuracy; thus, they should have higher priority.
3. If two rules have the same confidence values, rank them according to their support values. The higher the support values, the higher the priority. The rules with higher support value are more statistically Significant, so they should have higher priority.

4. If two rules have the same support value, rank them in alphabetical order.

2.1 GRAPHS, TABLES, AND PHOTOGRAPHS

Defect Types:

Here we are concentrated on some specified defects that are the most frequent defects that may occur in the Sel data.

Defect Types:

<table>
<thead>
<tr>
<th>Defect type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational defect</td>
<td>Computational defects are those that cause a computation to erroneously evaluate a variable’s value. These defects could be equations that are incorrect not because of the incorrect use of a data structure within the statement but by miscalculation.</td>
</tr>
<tr>
<td>Data value defect</td>
<td>Data value defects are those that are a result of the incorrect use of a data structure. Examples of this type of defects errors are the use of incorrect subscripts for an array, the use of the wrong variable in an equation, the use of the wrong unit of measurement, or the inclusion of an incorrect declaration of a variable local to the module.</td>
</tr>
<tr>
<td>Internal interface defect</td>
<td>Internal interface defects are those that were associated with internal structures of a module.</td>
</tr>
<tr>
<td>External interface defect</td>
<td>External interface defects are those that were associated with structures existing outside the module’s local environment but which the module used.</td>
</tr>
<tr>
<td>Initialization defect</td>
<td>Initialization defects are those that result from an incorrectly initialized variable, failure to re-initialize a variable, or because a necessary initialization was missing; failure to initialize or re-initialize a data structure properly upon a module’s entry/exit is also considered an initialization defect.</td>
</tr>
<tr>
<td>Logic/control structure defect</td>
<td>Logic/control structure defects are those that cause an “incorrect path” in a module to be taken. Such a control defect might be a conditional statement causing control to be passed to an incorrect path.</td>
</tr>
</tbody>
</table>
Rule Ranking Algorithm:

| Input: Rules – the defect association rules generated by the association rule mining Apriori algorithm. |
| Output: Rules – the ranked defect association rules by applying the proposed rule ranking length-first strategy, the most priori first. |
| 1: for each rule \( r_i \in \text{Rules} \) do |
| 2: for each rule \( r_j \in \text{Rules} \) do |
| 3: if \( \text{Length}(r_i) < \text{Length}(r_j) \) then |
| 4: \( r_i \leftarrow r_j \); // make more priori first |
| 5: else if \( \text{Length}(r_i) = \text{Length}(r_j) \) then |
| 6: if \( \text{Conf}(r_i) < \text{Conf}(r_j) \) then |
| 7: \( r_i \leftarrow r_j \); // the priority depends on confidence values for rules with the same length |
| 8: else if \( \text{Conf}(r_i) = \text{Conf}(r_j) \) then |
| 9: if \( \text{Supp}(r_i) < \text{Supp}(r_j) \) then |
| 10: \( r_i \leftarrow r_j \); // the priority depends on support values for rules with the same length \& the same confidence value |
| 11: else if \( \text{Supp}(r_i) = \text{Supp}(r_j) \) then |
| 12: if \( \text{AlphabetOrder}(r_i) > \text{AlphabetOrder}(r_j) \) then |
| 13: \( r_i \leftarrow r_j \); |
| 14: end if |
| 15: end if |
| 16: end if |
| 17: end if |
| 18: \( r_i \leftarrow r_j \); // \( r_i \) has higher priority |
| 19: end for |
| 20: end for |

Defect Isolation and Correction Effort Prediction:

The constraint-based association rules mining method is used for the purpose of defect isolation and correction prediction [10]. The steps are:

1. Compute the frequent item sets that occur together in the training data set at least as frequently as a predetermined min: Supp. The item sets mined must also contain the effort labels.

2. Generate association rules from the frequent item sets, where the consequent of the rules is the effort. In addition to the min: Supp threshold, these rules must also satisfy a minimum confidence.
Effort Prediction Rules:

Input: Rules – the output of rule ranking RR;

\( dAttri \) – a defect and its attributes.

Output: Eff – the predicted effort for correcting defect \( dAttri \).

1. \( Sim \leftarrow 0; \)\( Eff' \leftarrow 0; \) // candidates
2. for each element \( e \in dAttri \) do
3. \( Def_{0} \leftarrow e; \)
4. for each rule \( r \in Rules \) do
5. \( \text{if} \exists Def_{0} \in \text{Antecedent}(r) \land Length(Def_{0}) < Length(dAttri) \) then
6. \( \text{Def}_{0} \leftarrow \text{Def}_{0} \cup \text{Antecedent}(r); \)
7. \( \text{Eff}_{0} \leftarrow \text{Consequent}(r); \)
8. end if
9. end for
10. \( \text{Eff}' \leftarrow \text{Eff}' \cup \text{Eff}_{0}; \)
11. \( \text{Sim} \leftarrow \text{Sim} \cup \frac{\| dAttri \cap \text{Def}_{0} \|}{\| dAttri \|}; \)
12. end for
13. \( \text{Eff} \leftarrow \{ \text{Eff}' : \text{Eff}' \text{ with the max } \{ \text{Sim} \} \}; \)

Defect association prediction procedure.

We use the association rule mining method to learn rules from the training data sets. For defect association prediction, the rule learning is straightforward, while for defect correction effort prediction, it is more complicated because the consequent of a
rule has to be defect correction effort. Considering the target of association rule mining is not predetermined and classification rule mining has only one predetermined target, the class, we integrate these two techniques to learn defect correction effort prediction rules by focusing on a special subset of association rules whose consequents are restricted to the special attribute, the effort. Once we obtain the rules, we rank them, and use them to predict the defect associations and defect correction effort with the corresponding test data sets. The predictions of defect associations and defect correction effort are both based on the length-first strategy.

There is no work on software defect association prediction and there is no other method that can be used for this purpose. Therefore, I am unable to directly compare our defect association prediction method with other studies. As the defect correction effort is represented in the form of categorical values and there are some attributes to characterize it, it can be viewed as a classification problem. This allows us to compare our defect correction prediction method with three different types of methods. These methods are the Bayesian rule of conditional probability-based method, Naïve Bayes [, the well-known trees-based method, C4.5, and the simple and effective rules-based method, PART.

3. CONCLUSION

In this paper, I have given an application of association rule mining to predict software defect associations and defect correction effort with SEL defect data. This is important in order to help developers detect software defects and project managers improve software control and allocate their testing resources effectively. The ideas have been tested using the NASA SEL defect data set. From this, I extracted defect data and the corresponding defect isolation and correction effort data.

I have also compared the defect correction effort prediction method with three other types of machine learning methods, namely, PART, C4.5, and Naïve Bayes. The results show that for defect isolation effort, our accuracy is higher than for the other three methods b. Likewise, for defect correction effort prediction, the accuracy is higher than the other three methods.
I also have explored the impact of support and confidence levels on prediction accuracy, false negative rate, false positive rate, and the number of rules as well. I found that higher support and confidence levels may not result in higher prediction accuracy, and a sufficient number of rules is a precondition for high prediction accuracy. While we do not wish to draw strong conclusions from a single data set study, I believe that my results suggest that association rule mining may be an attractive technique to the software engineering community due to its relative simplicity, transparency, and seeming effectiveness in constructing prediction systems.

REFERENCES:

