Software Defect Prediction Using One Pass Data Mining Algorithm

[1]Marri Syam Sudhakar, [2]BR. Sarat Kumar

[1,2]Dept. of CSE Lenora College of Engineering, Rampachodavaram, AP, India

Abstract

The software defect prediction is one of the most important areas that attracts researcher's to prevent the defect that going to damage the system that will construct. The current software defect prediction focuses on First identifying numbers of defects remains in already constructed system. Second discovering software defects rules that are useful to future discussions about software defects. Finally classifying the defect proneness of software components. This paper focus only on second type i.e. describes the framework for finds defect Association in software defect database. Using association rules mining. It constructs all request patterns in the historical database. Then discover strong association rules in defect database. The experimental results shows that algorithm which presented in this paper at perform previous work.

Keywords

Data Mining, Association Rule Mining, Software Defect

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 re veal 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.

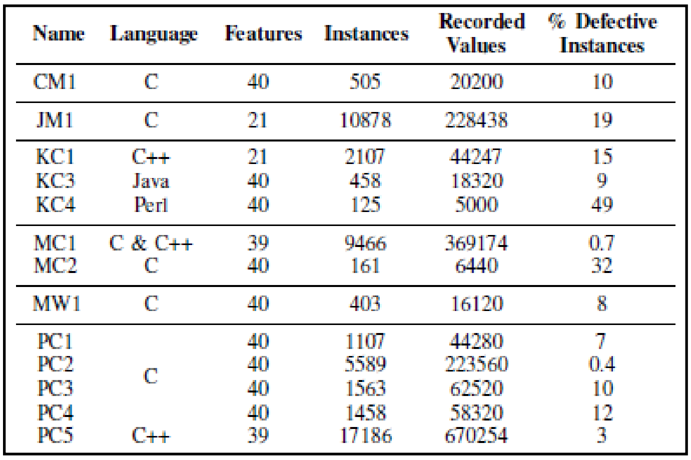
The remainder of the paper is organized as follows. Section 2 provides related work. Section 3 describes problem description. Section 4 is devoted proposed framework and one pass algorithm. In Section 5 results are documented. Conclusions and consideration of the significance of this work are given in 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 Na¨ıve Bayes) to predict software components containing defects. Hall and Holmes [2], concluded that the forward selection search was well suited to Na¨ıve 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 Na¨ıve 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. [5] 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}⇒ {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.



Name	Language	Features	Instances	Recorded Values	% Defective Instances
CM1	C	40	505	20200	10
JM1	C	21	10878	228438	19
KC1	C++	21	2107	44247	15
KC3	Java	40	458	18320	9
KC4	Perl	40	125	5000	49
MC1	C & C++	39	9466	369174	0.7
MC2	C	40	161	6440	32
MW1	C	40	403	16120	8
PC1	C	40	1107	44280	7
PC2		40	5589	223560	0.4
PC3		40	1563	62520	10
PC4		40	1458	58320	12
PC5	C++	39	17186	670254	3

Fig. 1: Stastics of Defect in Software's

Let I=I1, I2, … , I'm be a set of m distinct attributes, T be transaction that contains a set of items such that T ⊆ I, D be a database with different transaction records Ts. An association rule is an implication in the form of X⇒Y, where X, Y ⊂ I are sets of items called item sets, and X ∩ Y =∅. 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 ∪ 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 ∪ 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⇒Y is 80%, it means that 80% of the transactions that contain X also contain Y together.

IV. Proposed Solution

A. Frame Work

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.

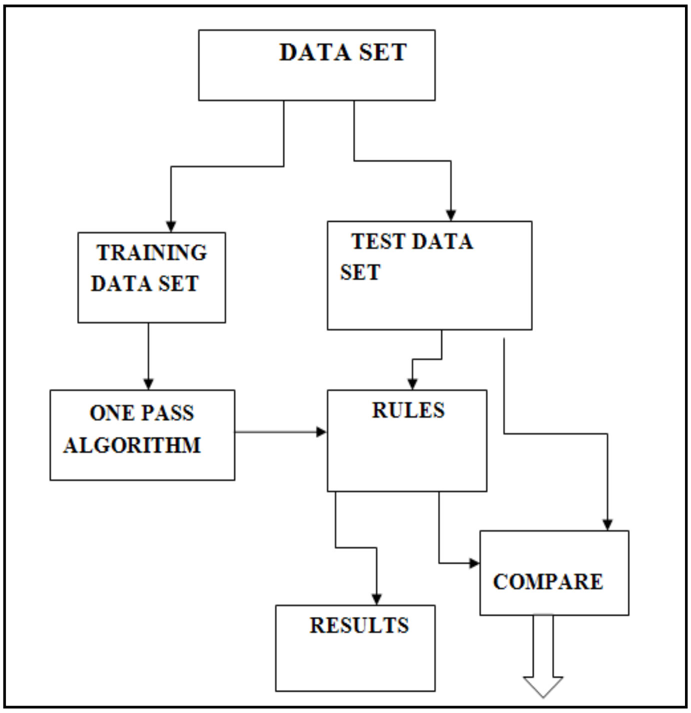


Fig. 2: Frame Work

B. 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.

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

Algorithm Learning (data, scheme)
Input: data - the data on which the learner is built;
Scheme - the learning scheme.
Output: learner - the final learner built on data with scheme;
BestAttrs - the best attribute subset selected by the attribute selector of scheme

```
m = 10; /*number of repetitions for attribute selection */
n = 10; /*number of folds for attribute selection */
d = Preprocessing (data, scheme. preprocessor);
bestAttrs=AttrSelect(d, scheme.Algorithm, scheme.attrSelector,
m, n);
5 d' = select bestAttrs from d;
6 learner = BuildClassifier (d', scheme. Algorithm);
/*build a classifier on d' with the learning algorithm of scheme
*/
```

```
Algorithm Prediction (historical Data, new Data, scheme)
Input: historical Data - the historical data; new Data - the new
data;
Scheme - the learning scheme.
Output: Result - the predicted result for the new Data

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

V. Results

The following are the results screens for finding association in the historical software defect database.

VI. 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 frame work 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

[1] B. T. Compton, C. Withrow,"Prediction and control of ada software defects", J. Syst. Softw., Vol. 12, No. 3, pp. 199–207, 1990.

[2] J. Munson, T. M. Khoshgoftaar,"Regression modeling of software quality: empirical investigation", J. Electron. Mater., Vol. 19, No. 6, pp. 106–114, 1990.

[3] N. B. Ebrahimi,"On the statistical analysis of the number of errors remaining in a software design document after inspection", IEEE Trans. Softw. Eng., Vol. 23, No. 8, pp. 529–532, 1997.

[4] S. Vander Wiel, L. Votta,"Assessing software designs using capture-recapture methods", IEEE Trans. Softw. Eng., Vol. 19, No. 11, pp. 1045–1054, 1993.

[5] P. Runeson, C. Wohlin,"An experimental evaluation of an experience-based capture-recapturemethod in software code inspections", Empirical Softw. Eng., Vol. 3, No. 4, pp. 381–406, 1998.