Journal of Computer Applications ISSN: 0974 – 1925,
Volume-5, Issue EICA 2012-2, February 10, 2012

A Comparative Analysis of General Defect Proneness in the Competing Software Systems

Aishwarya Devi R
M.E - Software Engineering
Jayaram College of Engineering and Technology
Anna University, Trichy
Pagalavadi
E-mail: aishwaryadevicse@gmail.com

Dr. Sahaaya Arul Mary S A
Prof, Head of the Department,
Computer Science and Engineering,
Jayaram College of Engineering and Technology
Anna University, Trichy
Pagalavadi

Abstract - Predicting defect-prone software components are an economically important activity and so has received a good deal of attention. The main objective of this software defect-proneness is to propose and evaluate a general framework for defect prediction in software that supports 1) unbiased and 2) comprehensive comparisons between competing prediction systems. Generally, before building defect prediction model and using them for prediction purposes, first it is necessary to decide which learning schemes should be used to construct the model. Thus the predictive performances of the learning scheme(s) should be determined, especially for future data. However, this step is often neglected and so the resultant prediction model may not be trustworthy. Consequently a new software defect prediction is proposed that provides guidance to address these potential shortcomings. The framework method comprises of 1) Scheme Evaluation and 2) Defect Prediction components. The Scheme Evaluation analyses the prediction performance of competing learning schemes for given historical data sets. And the Defect Predictor builds models according to the evaluated learning scheme, predicts software defects with new data according to the constructed model. The proposed framework is more effective and less prone to bias than previous approaches. This proposed software prediction framework is applicable for all the situations i.e., unbiased.

Keywords – *software defect prediction software defect-proneness, machine learning, scheme evaluation***.**

I. INTRODUCTION

Software defect prediction has been an important research topic in the software engineering field for more than 30 years. Current defect prediction focuses on 1) Estimating the number of defects remaining in software systems. 2) Discovering the defect associations and 3) classifying the defect-proneness of software components into two classes, defect- prone and not defect-prone. This project is concerned with the third approach.

The first type of work employs Statistical approach, Capture-Recapture (CR) models and Detection Profile Method (DPM) to estimate the number of defects remaining in software systems by inspecting the data. The prediction result can be used as an important measure for software developer and hence this can be used to control the software process to decide whether to schedule further inspections or pass the software artifacts to the next development step.

The second type of work borrows association rule mining algorithms from the data mining community to reveal software defect associations and this can be used for three purposes.

They are:

1. Finding as many related defects as possible to the detected defect(s) and consequently make more corrections to the software.
2. This type helps to evaluate reviewer's results during inspection. And finally,
3. Assisting the managers in improving software process through analysis of the reasons about the frequently occurring defects to take corrective action.

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. Classification remains a largely unsolved problem. In order to address this, researchers have been using increasingly sophisticated techniques drawn from machine learning. This sophistication has led to the challenges in how such techniques are configured and how they should be validated.

Incomplete or inappropriate validation can result in unintentionally misleading results and occurs in failure. For this reason a new and more general framework is proposed within which to conduct such validations.

Much of the research activity has followed the path of using software metrics extracted from the code as candidate actors to reveal whether a software component is defect-prone or not. To accomplish this, a variety of machine learning algorithms have been used to inductively find patterns or rules within the data to classify software components as either defect- prone or not.

In order to motivate the need for more systematic and unbiased methods for comparing the performance of machine-learning-based defect prediction, a recent paper published by Menzies is focused here. This is referred as MGF paper.

There are three reasons to choose MGF paper. Because,
1. The MGF paper has been widely cited and is therefore influential.
2. The approach might be regarded as state of the art for this kind of research.
3. Because the MGF analysis is based on datasets in the public domain; the work can be replicated.

In the MGF study, publicly available data sets from different organizations are used. This allows exploring the impact of data from different sources on different processes for finding appropriate classification models apart from evaluating these processes in a fair and reasonable way.

Additionally, the MGF study used 12 learning schemes resulting from two data pre-processors, two feature selectors, and three classification algorithms are designed to access the effects of different elements of a learning scheme on defect prediction. Although *balance* is a uncommon measure in classification, the results of MGF were reported with it.

The contribution of this paper includes:

- A new and more general software defect-proneness prediction framework within which appropriate validations conducted is proposed
- The impacts of different elements of a learning scheme on the evaluation and prediction are explored and it should be noted that the learning schemes should be evaluated as holistically and no learning scheme dominates.
- The potential bias and the comprehensive comparisons between the MGF and the current framework are evaluated.

II. RELATED WORK

As already mentioned in the introduction that this research activity has followed the research of Menzies, Greenwald and Frank (MGF). Also there is a comparison between the current research and the MGF in the following. MGF published a study in the journal in 2007 in which they compared the performance of two machine learning techniques (Rule induction and Naive Bayes) to predict the software components which contains defects. For this research activity, they used the NASA MDP repository, which, at the time of research it contained 10 separate data sets.

Traditionally, many researchers have explored issues already like the relative metrics of McCabe's cyclomatic complexity, Halstead's software science measures, and lines of code (LOC) counts for building defect predictors. However, MGF claim that these issues are irrelevant since "*how* the attributes are used to build predictors is much more important than *which* attributes are used". And "the choice of using learning method is far more important than which subset of the available data is used for learning". This analysis found that a Naive Bayes classifier, after *log filtering* and attribute selection on *InfoGain*, had a mean probability of error detection of 71 percent and the mean false rates of 25 percent. This significance is based on the rule induction methods of J48 and OneR.

The choice of which attribute subset is used for learning is not only circumscribed by the attribute subset itself and the available data, but also by the attribute selectors, learning algorithms and data preprocessors. But there is always an intrinsic relationship between a learning method and attribute selection method. Therefore, by the comparisons of all similar research it is concluded that before building prediction models, it should be needed to choose the combination of all three of learning algorithm, data preprocessing, and attribute selection method for better results.

Lessmann et al. have also conducted a follow-up research similar to MGF on defect predictions. And this research also provides an additional results as well as suggestions for a methodological framework. But Lessmann did not perform with the attribute selection when building prediction models.

There is an argument that the MGF's attribute selection approach is problematic. And hence that yields a bias in the evaluation results, despite the use of a M x N-*way cross- validation* method. The reason is that ranked attributes on the entire data set, including both the training and test data is used. The potential result is that this validation method over estimates the performance of their learning model and thereby reporting a potentially misleading result.
Moreover, after ranking the attributes each individual attribute is separated and evaluated and chose those *n* features with the highest scores. Unfortunately, this strategy cannot

consider features with complementary information. The major difference between MGF framework, Lessmann and some other prediction framework are:

a. MGF compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict the defect containing components.MGF claim that "*how* the attributes are used to build predictors is much more important than *which* attributes are used". And "the choice of using learning method is far more important than which subset of the available data is used for learning"
b. Lessmann have also conducted the same research as like MGF on defect predictions, providing an additional results as well as suggestions for a methodological framework. But Lessmann did not perform with the attribute selection when building prediction models.
c. Other research method as such Hall and Holmes concluded that the forward selection(FS) search was well suited to Naïve Bayes but the backward elimination(BE) search is more suitable for Rule Induction. Cardie found using a decision tree to select attributes helped the neighbor algorithm to reduce its prediction error.

III. PROPOSED FRAMEWORK

A Overview of the Framework

The proposed framework consists of two parts: Scheme Evaluation and Defect prediction. Scheme evaluation focuses on evaluating the performance of learning schemes. Generally, before building defect prediction model and using them for prediction purposes, first it is necessary to decide which learning schemes should be used to construct the model. Thus the predictive performances of the learning scheme(s) should be determined, especially for future data. However, this step is often neglected and so the resultant prediction model may not be trustworthy. Consequently a new software defect prediction is proposed that provides guidance to address these potential shortcomings.

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

At the defect prediction stage, according to the performance report of the first stage, a leaning scheme is selected and used to build a prediction model and predict software defect. This stage is very useful improving the generalization ability of the predictor.

A.1. Scheme Evaluation

The scheme evaluation is the 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-validation is 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 the current framework, an MxN-*waycross-validation* for estimating the performance of each perspective model, that is, each data set is first divided into N bins, and after that a predictor is learned on (N-1) bins, and then tested on the remaining bin. This is repeated for the N folds so that each bin is used for training and testing while minimizing the sample bias. To overcome any 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, MxN models are built in all during the period of evaluation; thus MxN 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 built to build a learner.

A learning scheme is comprised of:

1. Data preprocessor,
2. An attribute selector,
3. A learning algorithm

A.2. Observations

1. MGF proposed 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 inappropriate data.
2. This is because they used both the training and test data to rank attributes, which the labels of the new data are unavailable when choosing attributes in practice.

A.3. Pseudo code

This is the detailed scheme evaluation process is described with the pseudocode which consists of function *Learning* and function *AttrSelect.* The function *Learning* is used to build a learner with a given learning scheme, and the function *AttrSelect* performs attribute selection with a learning algorithm.

```
Function 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.  m=10; */ number of folds for attribute
        Selection                                  */

    3.  d= Pre-processing(data, scheme.preprocessor);
```

```
4.  bestAttrs=AttrSelect(d,scheme.algorithm, scheme.attrSelector,m,n);
5.  d'= select bestAttrs from d;
6.  learner=Build Classifier(d'scheme.algorithm);
    /*build a classifier on d' with the learning algorithm of scheme */
```

IV. EXPERIMENTS

A.4 .Data Sets

We used the data taken from the public NASA MDP repository, which was also used by MGF and many others, the AR data from the PROMISE repository were also used. Thus, there are 17 data sets in total.13 from NASA and the remaining 4 from the P

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 pre-processing, modules that contain one or more defects were labeled as defective. Besides LOC counts, the data sets include Halstead attributes, as well as McCabe complexity measures.

A4 Performance Measures

The receiver operating characteristics (ROC) curve is often used to evaluate the performance of binary predictors.A typical ROC curve is shown in the following fig. 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 in (1) and (2), respectively. Obviously, higher *pds and* lower *pfs*are desired. The point (*pf=0,pd=1*) is the ideal position where all the defective modules are recognized.

$$Pd = tpr = \frac{TP}{TP+FN}, \quad (1)$$

$$Pf = fpr = \frac{FP}{FP+TN}, \quad (2)$$

$$balance = 1- \frac{\sqrt{(1-pd)}2+\sqrt{(0-pf)}2}{\sqrt{2}} \quad (3)$$

MGF introduced a performance measure called *balance,* which is used to choose the optimal(*pd*, *pf*) pairs. The definition is shown in (3) from which the observation shows that it is equivalent to the normalized Euclidean distance from the desired point (0,1) to (*pf, pd*) in a ROC curve. In order to compare the current prediction with the MGF, *balance* is used as a performance measure.

Zhang and Reformat argue that using (*pd*, *pf*) performance measures in the classification of imbalanced data is not practical due to low precisions. By contrast, MGF argue that precision has an unstable nature and can be misleading to determine the better predictor. We also think that the predictors with high *pd* have practical usage even when their *pf*is also high. Nevertheless, such a predictor could still be helpful for software testing, especially in mission critical and safety critical systems, where the cost of many false positives is far less than that of false negatives. However, it should be noted that *balance* should be used carefully to determine the best among a set of predictors. Since it is a distance measure, predictors with different (*pf, pd*) values can have same *balance* value.

A5 Experiment Design

Two experiments are designed in the experiment one is to compare our framework with that of MGF, the second is intended to demonstrate our framework in practice and explore whether we should choose a particular learning scheme or not.

Framework Comparison

1. We divided each data set into two parts: one is used as *historical* data and the other is viewed as new data. To make most use of the data, we performed 10 *pass* simulations.
2. We replicated MGF's work with the *historical* data. First, all of the historical data were preprocessed in the same way by a log-*filtering* preprocessor. Then an iterative attribute subset selection as used in MGF's study was performed. In the subset selection method, the *i*=1,2,..,*N*th top ranked attribute(s) were evaluated step by step. Each subset was evaluated by a 10 x 10- *way cross- validation* with the Naïve Bayes algorithm.
3. We also simulated the whole defect prediction process presented in our framework. In order to be comparable with MGF, we restricted our learning scheme to the same pre processor method, attribute selection and the same learning algorithm.

V. CONCLUSION

Software defect proneness is an important activity to detect the defects present in the software. In this paper, we have presented a novel bench mark 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 historical data and predictor is finally used to predict defect on the new data.

There is a comparison of the proposed framework with MGF's study and pointed out the potential bias in their baseline experiment. Here also performed a baseline experiment to stimulate the whole process of defect prediction in both MGF's study and the current framework. From these experimental results, there is a observation that there is a bigger difference between evaluation performance and the actual prediction performance in MGF's study than with the current framework. This means that the results they report are over optimistic. While this might seem like some small technicality, the impact is profound. When performing the statistical significance testing dramatically different findings that are highly statistically significant is found in opposite direction. The real point is not which learning scheme does "better" but how should one set about answering this question. From the current experiment we also observe that the predictions of Menzies et al. On the AR data are much more biased than that on the NASA data and the performance of the MGF framework varies greatly with data from different resources. Thus we contend our framework is less biased and more capable of yielding results closer to the "true" answer. And hence our framework is more stable.

From the results, a data pre-processor /attribute selector can play different roles with different learning algorithms for different data sets can be seen and that no learning scheme dominates, i.e., always out performs the others for all data sets. This means that different learning schemas for different data sets should be chosen and consequently, the evaluation and decision process is important.

REFERENCES

1. A.Porter and R.Selby, "Empirically Guided Software Development Using Metric-Based Classification Trees," IEEE software, vol. 7, no.2, pp.46-54, Mar 1992.
2. B.T.Compton and C.Withrow,"Prediction and Control of ADA Software Defects," J.Systems and Software,vol.12,no.3,pp 199-207,1990.
3. C.Wohlin and P. Runeson, "Defect Content Estimations From Review Data", Proc. 20th Int'l conf. Software Eng., pp 400-409, 1998.
4. F.Padberg, T.Ragg and R.Schoknecht, "Using Machine Learning for Estimating the Defect Content After an Inspection," IEEE Trans.SoftwareEng ., vol.30,no.1,pp 17-28,Jan.2004.
5. G.Q. kenney, "Estimating Defects in commercial software during operational use,"IEEE Trans. Relaiabilty, vol. 42, no.1, pp.107-115, Mar.1993
6. J.Munson and T.M.Khosgotaar,"Regression Modelling of Software Quality:Empirical Investigation," J.Electronic Materials,vol.19,no.6,pp.106-114,1990.
7. J.C.Munson and T.M.khoshgoftaar,"The Detection of fault-Prone Programs,"IEEETrans.Software Engg.,vool.18,no.5,pp.423-433, may 1992.
8. J.Tian and M.Zelkowitz,"Complexity Measure Evaluation and Selection,"IEEETrans.Software Eng.,vol.21,no.8,pp.641-649,Aug.1995.
9. K.Ganesan, T.M. Khoshgoftaar, and E. Allen,"Case-Based Software Quality Prodiction," Int'l J.Software Eng. and knowledge Eng., vol.10,no.2,pp.139-152,2000.
10. K.Srinivasan and D.Fisher,"Machine Learning Approaches to Estimating Development Effort,"IEEETrans.Software Eng.,vol.21,no.2,pp.126-137,Feb.1995.
11. K. El Emam, S.Benlarebi, N. Goel, and S.N..Rai, "Comparing Case- Based Reasoning Classifiers for Predicting High Risk Software Components," J. Systems and software, vol. 55, no. 3, pp. 301-320, 2001.
12. L.Zhan and M.Reformat, "A Practical Method for the Software Fault Prediction," Proc.IEEE Int'l Conf.Information Reuse and Integration,pp.659-666,2007.
13. N.E. Fenton and M.Neil,"A Critique of software Defect Prediction Models,"IEEETrans,Software Eng.,vol.25,no.5.