A Methodology for Evaluation and Prediction of Defect-Proneness in software

[1]Silpa C, [2]Dr.K.Ramani
Department of Information Technology,
Sree Vidyanikethan Engineering College, Tirupati, India
[1]Silpa.c8@gmail.com [2]ramanidileep@yahoo.com

Abstract— Predicting defect-prone software components is an economically important activity. Software defect prediction work focuses on three ways 1) Estimating the number of defects remaining in software systems, 2) Discovering defect associations, and 3) Classifying the defect proneness of software components. The software defect prediction that supports both unbiased and comprehensive comparison between competing prediction systems. This methodology is comprised of 1) scheme evaluation and 2) defect prediction components. The scheme evaluation analyzes the prediction performance of competing learning schemes for given historical data sets. The defect predictor builds models according to the evaluated learning scheme and predicts software defects with new data according to the constructed model. In the evaluation stage different learning schemes are evaluated and best one is selected. In the prediction stage the best learning scheme is used to build a predictor with all historical data and the predictor is finally used to predict defect on new data. This system classifies the defect-proneness of software components into two classes, defect-prone and non defect-Prone.
Keywords— Scheme evaluation, Software defect prediction, Software defect-proneness prediction, Historical data, Learning schemes.

I. INTRODUCTION

Defect prediction work focuses on
1) Estimating the number of defects remaining in software systems,
2) Discovering defect associations, and 3) Classifying the defect-proneness of software components, typically into two classes, defect-prone and not defect-prone[6].

A. Estimating the number of defects remaining in software systems

This work employs statistical approaches, capture-recapture (CR) models [1], and detection profile methods (DPM) to estimate the number of defects remaining in software systems with inspection data and process quality data. The prediction result can be used as an important measure for the software developer.

B. Discovering defect associations

This work borrows association rule mining algorithms from the data mining community to reveal software defect associations which can be used for three purposes.
First finding as many related defects as possible to the detected defect(s) and consequently make more effective corrections to the software. This may be useful as it permits more directed testing and more effective use of limited testing resources.
Second, helping evaluate reviewers' results during an inspection. Thus, a recommendation might be that his/her work should be re-inspected for completeness.
Third, assisting managers in improving the software process through analysis of the reasons why some defects frequently occur together.

C. Classifying the defect-proneness of software

This work classifies software components as defect-prone and non-defect-prone by means of metric based classification.

II. SOFTWARE DEFECT PREDICTION

A. *Software Defect prediction Methodology*

The methodology consists of two components: 1) scheme evaluation and 2) defect prediction. The scheme evaluation focuses on evaluating the performance of a learning scheme, while the defect prediction focuses on building a final predictor using historical data according to the learning scheme and after which the predictor is used to predict the defect-prone components of a new (or unseen) software system. A learning scheme is comprised of: 1. a data preprocessor, 2. an attribute selector, 3. a learning algorithm.
So, the main difference between the given methodology and that of MGF[5] lies in the following: 1) In the methodology it choose the entire learning scheme, not just one out of the learning algorithm, attribute selector, or data preprocessor; 2) It use the appropriate data to evaluate the performance of a scheme. That is, it build a predictive model according to a scheme with only "historical" data and validate the model on the independent "new" data.
At the scheme evaluation stage, the performances of the different learning schemes are evaluated with historical data to determine whether a certain learning scheme performs sufficiently well for prediction purposes or to select the best from a set of competing schemes.

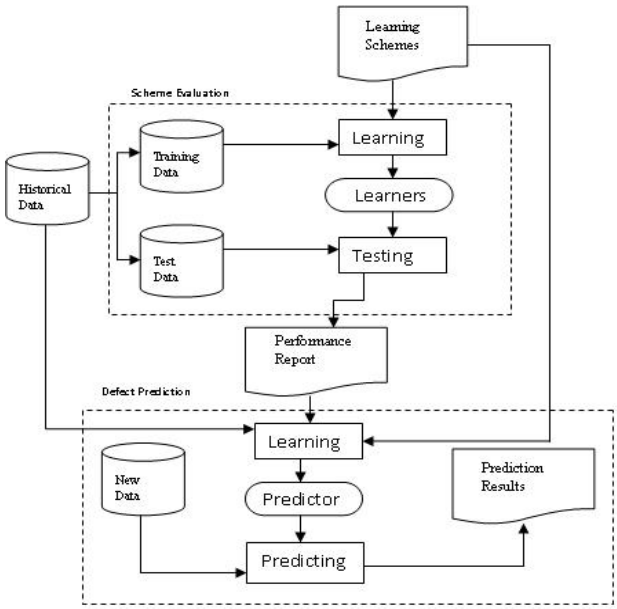

Fig. 1 Software defect prediction methodology

From Figure 1, the historical data are divided into two parts: a 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 to further determine whether or not to apply the learning scheme or select one best scheme from the given schemes.

At the 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. From Fig. 1, it is observed that all of the historical data are used to build the predictor here. This is very different from the first stage; it is very useful for improving the generalization ability of the predictor. After the predictor is built, it can be used to predict the defect-proneness of new software components.

B. Scheme Evaluation

The scheme evaluation is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated by building and evaluating learners with them. Figure 2 contains the details. 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.

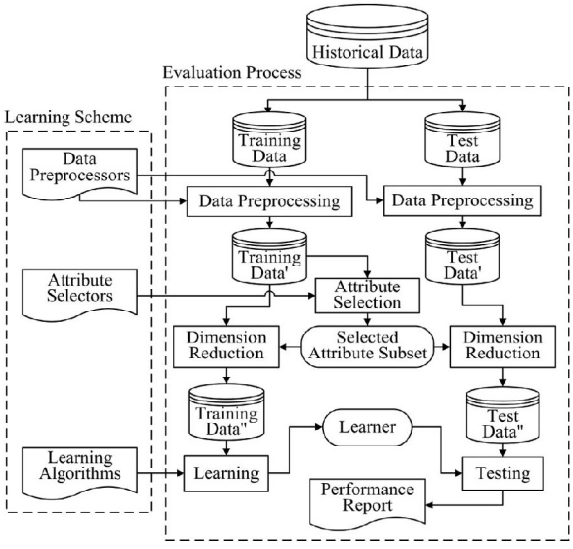

Fig. 2 Scheme Evaluation

In this Methodology, an M×N-way cross-validation is used for estimating the performance of each predictive 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 sampling bias.
The learner construction procedure is as follows:
i. Data preprocessing: This is an important part of building a practical learner. In this step, the training data are preprocessed, such as removing outliers, handling missing values, and discrediting or transforming numeric attributes. In our experiment, we use a log-filtering preprocessor which replaces all numerics n with their logarithms ln(n), such as used in MGF.
ii. Attribute selection: Attribute selection methods can be categorized as either filters or wrappers . It should be noted that both "filter" and "wrapper" methods only operate on the training data. A "filter" uses general characteristics of the data to evaluate attributes and operates independently of any learning algorithm.
iii. Learner construction: Once attribute selection is finished, the preprocessed training data are reduced to the best attribute subset. Then, the reduced training data and the learning algorithm are used to build the learner. Before the learner is tested, the original test data are preprocessed in the same way and the dimensionality is reduced to the same best subset of attributes. After comparing the predicted value and the actual value of the test data, the performance of one pass of validation is obtained. As mentioned previously, the final "evaluation" performance can be obtained as the mean and variance values across the M×N passes of such validation.
Defect Prediction:
The defect prediction methodology consists of predictor construction and defect prediction.
During the period of the predictor construction:
1. A learning scheme is chosen according to the Performance Report.
2. A predictor is built with the selected learning scheme and the whole historical data.
3. After the predictor is built, new data are preprocessed in same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.

III. DEFECT PREDICTION WITH DIFFERENT LEARNING SCHEMES

This Defect Prediction methodology illustrate that different elements of a learning scheme have different impacts on the predictions and to confirm that it should choose the combination of a data preprocessor, an attribute selector, and a learning algorithm, instead of any one of them separately. Then 12 different learning schemes were designed according to the following data preprocessors, attribute selectors, and learning algorithms.

A. Two data preprocessors

a. None: data unchanged;
b. Log: all of the numeric values are replaced by their logarithmic values.

B. Two attribute selectors

The standard wrapper method is employed to choose attributes. This means the performances of the learning algorithms are used to evaluate the selected attributes. Two different search strategies (based on greedy algorithms) are used as follows:

i. Forward selection: Starts from an empty set and evaluates each attribute individually to find the best single attribute. It then tries each of the remaining attributes in conjunction with the best to find the best pair of attributes. In the next iteration, each of the remaining attributes are tried in conjunction with the best pair to find the best group of three attributes. This process continues until no single attribute addition improves the evaluation of the subset.

ii. Backward elimination: Starts with the whole set of attributes and eliminates one attribute in each iteration until no single attribute elimination improves the evaluation of the subset.

C. Three learning algorithms

Naïve Bayes (NB), J48 (JAVA implementation of quinlan's C4.5 (version 8) algorithm), and OneR (one Rule).

i. Twelve learning schemes:

The combination of two data preprocessors, two attribute selectors, and three learning algorithms yields a total of 12 different learning schemes:

a. NB + Log + FS;
b. J48 + Log + FS;
c. OneR + Log + FS;
d. NB + Log + BE;
e. J48 + Log + BE;
f. OneR + Log + BE;
g. NB + None + FS;
h. J48 + None + FS;
i. OneR + None + FS;
j. NB + None + BE;
k. J48 + None + BE;
l. OneR + None + BE.

D. Data Sets

The data used in this methodology was taken from NASA repository [4]. Some of software defect prediction data sets are analyzed. Each data set is comprised of several software modules, together with their number of faults and characteristic code attributes. After preprocessing modules that contain one or more defects were labeled as fault prone (fp), where as error free modules are categorized as not defect prone modules (nfp). Static code measures are used for measuring data sets.

IV. ROC CURVE

The receiver operating characteristic (ROC) curve is used to evaluate the performance of binary predictors. A typical ROC curve is shown in Fig. 3. The y-axis shows probability of detection (pd) and the x-axis shows probability of false alarms (pf). The ROC curve must pass through the points pf = pd = 0 and pf = pd = 1.

Three interesting trajectories connect these points:

1. A straight line from (0,0) to (1,1) is of little interest since it offers no information: i.e. the probability of a predictor firing is the same as it being silent.

2. Another trajectory is the negative curve that bends away from the ideal point. It found that if predictors negate their tests, the negative curve will transpose into a preferred curve.

3. The point (pf = 0, pd = 1) is the ideal position (also known as "sweet spot") on a ROC curve. This is where it recognize all errors and never make mistakes. Preferred curves bend up towards this ideal point.

In the ideal point, a predictor has a high probability of detecting a genuine fault (pd) and a very low probability of false alarm (pf).

MGF[5] introduced a performance measure called balance, which is used to choose the optimal (pd, pf) pairs. The given equation (3) shows that it is equivalent to the normalized Euclidean distance from the desired point (0, 1) to (pf, pd) in a ROC curve.

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

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

Where as true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

Balance is based on the assumption that there is no difference between the cost of false positives (FP) and false negatives (FN).

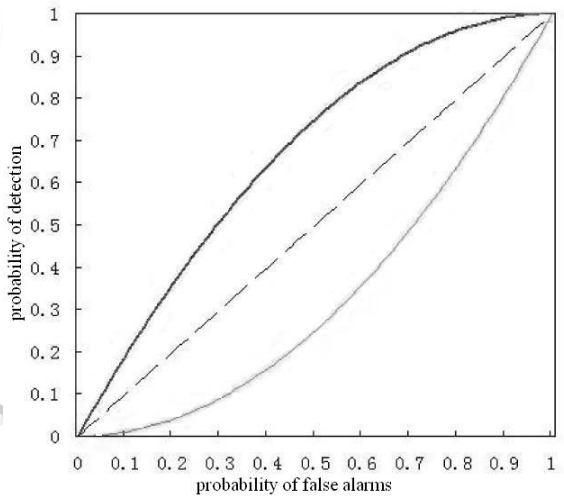

Fig. 3 ROC Curve

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

Each point in the ROC curve is mapped to the discrimination threshold. The point (0, 0) means the predictor treats all modules as non-defective when the corresponding threshold is 1. The point (1, 1) means the predictor treats all modules as defective when the corresponding threshold is 0.

The Area Under ROC Curve (AUC) is often calculated to compare different ROC curves. Higher AUC values indicate the classifier is, on average, more to the upper left region of the graph. AUC represents the most informative and commonly used, thus it is used as another performance measure in this paper.

Next, in order to more clearly compare the evaluation performance of the MGF and defect prediction methodology, here uses diff, which is defined as follows:

$$\text{diff} = \frac{EvalPerf - PredPerf}{PredPerf} \times 100\% \quad (4)$$

where EvaPerf represents the mean evaluation performance and PredPerf denotes the mean prediction performance. The performance measure can be either balance or AUC.

From the definition 1) a positive diff means that the mean evaluation performance is higher than the mean prediction performance, so the evaluation is optimistic; 2) a negative diff means a lower mean evaluation performance than the corresponding mean prediction performance, thus the evaluation is conservative. No matter whether the diff is positive or negative, it is clear that the smaller the absolute value of diff is, the more accurate the evaluation is. And this is our main goal, to have a framework that minimizes this gap.

V. CONCLUSION

The methodology 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 the predictor is finally used to predict defect on the new data.

1. In the Prediction Framework for data preprocessing log-filtering preprocessor was used. In the Defect prediction methodology some other preprocessor can be used. There are a number of different tools and methods used for preprocessing including: sampling, transformation, denoising, normalization, feature extraction.sampling which selects a representative subset from a large populationof data, transformation which manipulates raw data to produce a single input, denoising which removes noise from data, normalization, which organizes data for more efficient access, and feature extraction which pulls out specified data that is significant in some particular context.

2. In the Prediction Framework the attribute selection has been performed on the training data and by using wrappers attribute selection method. In the Defect prediction methodology selecting the best attributes by using filter method on the training data.

REFERENCES

[1]. S. Vander Wiel and L. Votta, "Assessing Software Designs Using Capture-Recapture Methods," IEEE Trans. Software Eng., vol. 19, no. 11, pp. 1045-1054, Nov. 1993.
[2]. N.E. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Trans. Software Eng., vol. 25, no. 5, pp. 675-689, Sept./Oct. 1999.
[3]. J.C. Munson and T.M. Khoshgoftaar, "The Detection of Fault- Prone Programs," IEEE Trans. Software Eng., vol. 18, no. 5, pp. 423-433, May 1992.
[4]. L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust Prediction of Fault-Proneness by Random Forests," Proc. 15th Int'l Symp. Software Reliability Eng., pp. 417-428, 2004.
[5]. T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Trans. Software Eng., vol. 33, no. 1, pp. 2-13, Jan. 2007.
[6]. Qinbao Song, Zihan Jia, Martin Shepperd, shi Ying and Jin Liu "A Gnereal Software Defect-Proneness Prediction Framework," IEEE Trans. Software Eng., vol. 37, no. 3, pp. 356- 370, May/June 2011.
[7]. K. Ganesan, T.M. Khoshgoftaar, and E. Allen, "Case-Based Software Quality Prediction," Int'l J. Software Eng. and Knowledge Eng., vol. 10, no. 2, pp. 139-152, 2000.
[8]. 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.
[9]. P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, "What Do We Know about Defect Detection Methods*?*" IEEE Software, vol. 23, no. 3, pp. 82-90, May/June 2006.
[10]. K.O. Elish and M.O. Elish, "Predicting Defect-Prone Software Modules Using Support Vector Machines," J. Systems and Software, vol. 81, no. 5,pp.649-660,2008