An Approach to Reduce Test Effort by Using Machine Learning Techniques

Mohammed Inayathulla, Silpa C
Department of Information Technology
Sree Vidyanikethan Engineering College
Tirupathi, AP, India

Abstract
It is obvious that testing consumes more than fifty percent of development effort. Hence it may be advantageous for any organization if the testing effort is reduced. Various fault prediction models have been proposed but their effectiveness in reducing testing effort or improving quality is not addressed. An approach(TERA) is proposed which uses the machine learning techniques in order to reduce the testing effort. Initially by using the prediction models the number of defects are predicted and based on these defects appropriate testing effort is allocated to each module. The test effort can be reduced only if the suitable test strategy is used with appropriate fault-prediction accuracy.

Keywords
Fault Prediction, Machine Learning, Software Testing, Test Effort

1. Introduction

As recent software systems have grown in size and complexity, quality assurance activities such as testing and inspections have become increasingly important, for software developers. Since resources are limited and scheduling is tight in most cases, quality assurance must be performed as efficiently as possible. Advance knowledge of which files of a software system are most likely to contain faults can be a valuable asset. To prioritize quality assurance efforts, various models for predicting fault-prone modules have been proposed in order to select software modules based on their probability of having a fault, the number of expected faults or the fault density. The basic hypothesis of software quality prediction is that a module currently under development is likely to be fault prone, if a module with the similar product or process metrics in an earlier project was fault prone.

Based on the prediction results, practitioners can allocate limited testing (or inspection) efforts to fault-prone modules so as to find more faults with smaller effort [1]. Based on the prediction results, practitioners can allocate limited testing efforts to fault-prone modules so as to find more faults with smaller effort. Here fault prediction models refer to machine learning techniques. To adopt fault prediction techniques in industry, one needs to be able to assess the cost effectiveness of the prediction because not

only poor predictions but also a poor resource allocation strategy could even increase the test effort. Fault prediction allows software practitioners to direct their resources into the areas with the highest impact on the bottom line. However, while the prediction performances of have been evaluated in terms of recall/ precision/F1-measure[1] Alberg diagrams and/or ROC curves, the final goal of reducing the test effort or increasing software quality has been rarely explored. To adopt fault prediction techniques in industry, one needs to be able to assess the cost effectiveness of the prediction because not only poor predictions but also a poor resource allocation strategy could even increase the test effort.

From Fig. i, the collected data i.e. the historical data is categorized into two parts: a training dataset for building learning models with the available learning schemes or techniques, and a test dataset for evaluating the performances of the learner models or conducting prediction. It is very important that the test dataset is not used in any way to build the learning model initially. This is a necessary condition to evaluate the generalization ability of a model that is built according to a learning technique and to further determine whether or not to apply the learning technique or select one best scheme from the given schemes.

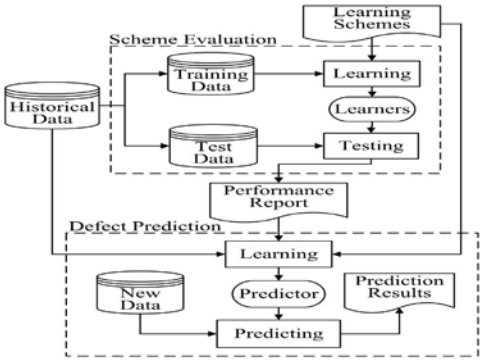

Fig. 1 Software Fault Prediction Methodology

Manuscript received September 5, 2015
Manuscript revised September 20, 2015

At the defect prediction stage, according to the performance report generated from the first stage, a learning technique is selected and used to build a prediction model which can be used predict software fault. From Fig. 1.1, it is observed that all of the historical data are used to build the prediction model here. This is very different from the first stage; it can be used for improving the generalization ability of the prediction model. After the prediction model is built, the same model can be used to predict the defective nature of new software components or similar software components.
The primary goal of TERA is to estimate the reduction of test effort that fault prediction can achieve. Apart from this, at the end after implementation following questions are required to be answered:

(Q1) What is the appropriate strategy to allocate test effort to each module after prediction?
This question is difficult to answer although several strategies can be easily developed. The simplest one is to allow the test effort be proportional to the predicted number of faults in a module. The concern with this strategy is that it does not consider the size of a module, which may affect the ease of discovering a fault. Another strategy is to allocate test effort based on the fault density, but this may also not be the best choice because it concentrates on modules with high fault density no matter what their size, even though larger modules will contain more faults than smaller ones. Moreover, the best strategy may depend on the available test effort. To answer this question, we need to be able to find the expected number of discoverable faults that can be discovered with respect to the given test resources, the effort allocation strategy (with fault prediction result), and the set of modules to be tested. Then, we could estimate the test effort needed to discover a desired number of faults. In this approach a fault discovery model that can represent the relationship among the prediction results, test efforts, module sizes, and the probability of fault discovery.

(Q2) How prediction accuracy is calculated for a prediction model?
If the prediction accuracy is very low, we cannot rely on any test effort allocation strategy that uses the prediction result. Among various evaluation measures such as recall, precision, F-value, Alberg diagram and ROC curve we decided to use the normalized Popt because it can evaluate the prediction performance in terms of testing effort while ROC curve assumes equal test effort among all modules.

(Q3) How much is test effort reduced by the prediction?
This is the primary question everyone want answered. To answer this question, an assumption is made that there are still some faults remaining after testing because most software systems contain faults after release. Therefore, a parameter called the remaining fault rate is defined required test effort is computed that potentially discovers as many faults as actual testing.

2. Related Work

In 2005, Thomas J. Ostrand, Elaine J. Weyuker and Robert M. Bell proposed a model to predict which files are likely to have the largest concentrations of faults in the next release of a system. They used negative binomial regression theorem in their prediction model. Their predictions allowed testers to target their efforts on those files with high fault rates, enabling them to identify faults more quickly. But their model required a lot of statistical experience to conduct the prediction[2]. How testing effort was allocated to fault prone modules was also not addressed.
In 2008, Stefan Lessmann and Christophe Mues used ROC(receiver operating characteristic area)curve to evaluate the prediction[3]. The ROC graph is a 2D illustration of TPR(true positive rate) on the Y-axis versus FPR(false positive rate) on the X-axis. An ROC curve is obtained by varying the classification threshold over all possible values. Thereby, each ROC curve passes through the points (0, 0), representing a classifier that always predicts nfp(non fault-prone) and(1, 1),the opposite case. In[4] the prediction performance was evaluated in terms of F1 which was a combination of precision and Recall values. In[5][6] also different prediction models were used but how test effort can be reduced was not addressed.

3. Tera Approach

A. Approach Implementation: Initially prediction models are implemented and the predicted number of faults are used to calculate the test effort with different test effort allocation strategies. With this test effort the total number of discoverable faults are calculated and reported. Then the 6 strategies are compared considering faults discovered by each strategy.

i) *Train/Test Dataset:* The size of dataset (for building a prediction model) has a great impact on the quality of prediction, training and testing datasets must be prepared with at most care.
ii) *Objective Variable:* There are three candidates for the objective variable. They are the probability of having a fault, the number of faults and the fault density. In this approach the number of faults and fault density are predicted.
iii) *Predictor Variables:* Metrics for each module can be computed from the code and design documents. LOC, Cyclomatic Complexity, Design Complexity are some of the metrics used.

iv) *Prediction Models:* Until now, various types of fault-prone module predictors have been used. Since we need to predict a "number" (the number of faults) rather than predicting a probability or conducting a classification, we decided to use random forest, which can predict a number and is one of the promising approaches in fault-prone module prediction. Linear Regression and CART also are implemented.

Random Forest: Random Forest is a machine learning algorithm for classification and regression that operates by constructing multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The term random forest came from random decision forests that was first proposed by "Tin Kam Ho" of Bell Labs in 1995. The method combines Breiman's bagging idea and the random selection of features.

Linear Regression: Linear regression has the longest history in statistics, well understood and is the most popular machine learning model. It is based on the assumption of a linear relationship exist between the input x1, x2 ... and output variable y (numeric). If only one predictor variable is used it is called as simple linear regression. If more than one predictor variable is used then it is called as multiple linear regression. In TERA multiple linear regression is used.

CART(Classification and Regression trees): Classification tree analysis is used when the predicted result is the class to which the data belongs. Regression tree analysis is used when the predicted result can be considered a numeric value. In TERA regression tree analysis is implemented as we have to predict a numeric outcome.

B. R-Statistical Computing and Graphics Tool Kit

R provides an Environment for both statistical computing and graphics. It is Free and Open Source Software. It provides Data Storage, Analysis and Graphing. To build prediction models, the statistical computing and graphics toolkit R and its MASS, rpart and randomForest libraries are used. To implement Random Forest prediction model in R following syntax is used:

randomForest(obj_var, data=train, mtry=sqrt(p), ntree=200)

p: no of predictor variables

ntree: no of trees to grow

To implement linear regression in R following syntax is used:

lm(response_var ~ var1+var2+var3- - -+varp)

To implement CART following syntax is used:

rpart(obj_var ~ var1+var2+var3- - -+varp, ethod="anova", data=train)

C. Effort Allocation Strategies: *There are several possible* strategies to assign test effort to each module after prediction. In TERA Following strategies are used to allocate test effort:

i) Equal test effort to all modules: This is the most naïve strategy, which we consider as a baseline strategy.

ii) Test effort α module size: Given a module set(m1;...;mn,) the allocated test effort ti for the ith module mi is defined as

$$ti = ttotal.Si/Stotal \quad (1)$$

iii) Test effort α # of predicted faults: Allocated test effort ti is defined as:

$$ti = ttotal.Fi/Ftotal \quad (2)$$

where Fi is the number of predicted faults in the ith module and Ftotal is the total number of predicted faults in all modules.

iv) Test effort α predicted fault density:- Allocated test effort ti is defined as

$$ti = ttotal(Fi/Si) / \sum_{i=1}^{n} (Fi/Si) \quad (3)$$

v) Test effort α # of predicted faults × module size: Allocated test effort is defined as

$$ti = ttotal.Fi.Si / \sum_{i=1}^{n} (Fi.Si) \quad (4)$$

vi) Test effort α # of predicted faults × log(module size): Allocated test effort is defined as

$$ti = ttotal.Fi.\log(Si) / \sum_{i=1}^{n} (Fi.\log(Si)) \quad (5)$$

D. Fault Discovery Model: A fault discovery model is used in order to compute the number of discoverable faults with respect to the given test resources, the effort allocation strategy, and the set of software modules to be tested. The relationship between testing time (effort)and the cumulative number of detected faults as shown in below equation:

$$H_i(t_i)=a[1-\exp(b_i t_i)] \ , \ b_i=b0/S_i \quad (6)$$

H(t): Expected value of the cumulative number of faults detected by a given testing effort.

t: Testing time (effort).

b: Probability of detecting each fault per unit time.

a: The number of initial faults before testing

The value of a is estimated as per the equation below

$$a_i=H_i+(R.\ S_i\ /1000) \quad (7)$$

H: Actual faults

R: 0.3,0.5,1.0 (Remaining fault rate for new code)

$b0=-S_{total}/t_{total}.\log(1-H_{total}/a_{total})$

E. Evaluation Criteria of Fault Prediction: Among various evaluation measures such as recall, precision, F-value[4], Alberg diagram and ROC curve the normalized Popt is used because it can evaluate the prediction

performance in terms of testing effort. Popt is defined as 10-Δopt, where Δopt is the area between the LOC-based cumulative lift charts of the optimal model and the prediction model. In the Fig.ii a sample graph is shown where x-axis is considered as SLOC y-axis represents defects.

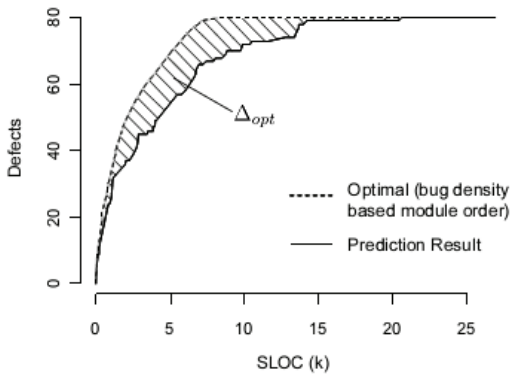

Fig 2 Example of LOC based Popt chart

4. Results

Fault prediction is meaningless when plenty of resources are available. Prediction is used only when resources are less. R-3.0.2 is used to build prediction models. The dataset used is one of the NASA Metrics Data Program defect data sets. It is a Promise Software Engineering Repository data set made publicly available in order to encourage predictive models of Software Engineering. The dataset had 145 modules information out of which 140 modules are used for training dataset and remaining five are used for testing dataset. The dataset contained only static code measures. For Random Forest all variables were used as predictor variables where as for linear regression and CART only few variables were selected. Among the available static measures the measures which had impact on prediction were selected in case of CART and Linear Regression(LOC, Cyclomatic Complexity). The selection of predictor variables also has a great impact on prediction.

(Q1) What is the appropriate strategy to allocate test effort to each module after prediction?
After conducting prediction and allocating test effort with three different remaining fault rate values strategy *A5* showed best results.

(Q2) How prediction accuracy is calculated for a prediction model?
The following table below shows the P_{opt} value of three prediction models. The three machine learning models are evaluated and rated on a scale of ten.

Table I

Prediction Model	**P_{opt} Value**
Random Forest	8.8
Linear Regression	8
CART	6.6

Random Forest has shown best performance in terms of P_{opt} value. The next step is to allocate test effort based upon the predicted number of faults. Effort is allocated based upon the faults predicted by Random Forest Model as it has shown best performance among the three models.

(Q3) How much is test effort reduced by the prediction?
This is the most important requirement that has to be answered. The test effort required for discovering faults is calculated based upon the prediction results of best prediction model. Table 2 shows the test required to discover 100 percent of faults by each strategy based on the prediction results of each model. CART and Linear Regression have shown similar results with three remaining fault rates whereas Random Forest has shown a bit different.

Table II. Test Effort Required to Discover 100 Percent Faults

(a) Remaining Fault Rate=1

Prediction Model	P_{opt}	Testing Effort(%)					
		A1	A2	A3	A4	A5	A6
Random Forest	8.8	100	87	95	104	83	95
CART	6.6	100	86	101	101	93	103
Linear Regression	8	100	85	98	95	98	101

(b) Remaining Fault Rate=0.5

Prediction Model	P_{opt}	Testing Effort(%)					
		A1	A2	A3	A4	A5	A6
Random Forest	8.8	100	85	91	98	81	96
CART	6.6	100	86	102	102	93	103
Linear Regression	8	100	85	98	95	98	101

(c) Remaining Fault Rate=0.3

Prediction Model	P_{opt}	Testing Effort(%)					
		A1	A2	A3	A4	A5	A6
Random Forest	8.8	100	85	91	91	81	96
CART	6.6	100	86	102	102	93	106
Linear Regression	8	100	85	98	95	98	101

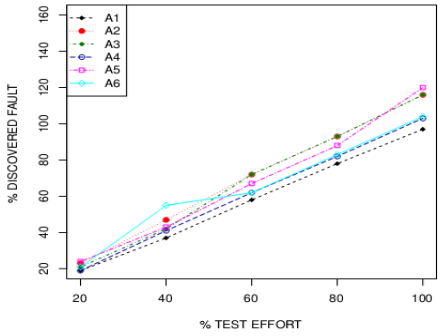

Fig. 3 Comparison with R=1

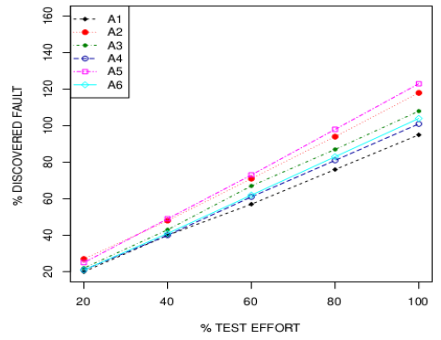

Fig. 4 Comparison with R=0.5

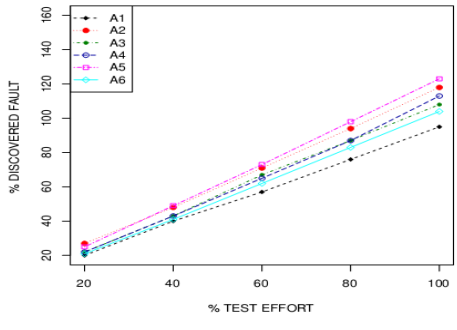

Fig. 5: Comparison with R=0.3

5. Conclusion

This approach compares test allocation strategies. The results suggest that strategy A5detects more number of defects with less test effort among all strategies. When prediction is evaluated Random Forest showed the best performance. The graphs in Fig iii,iv,v shows the comparison of all strategies. The results also suggest that reduction of the test effort is achieved only if the suitable test strategy is employed with appropriate prediction accuracy. However, where sufficient data are available to fit a prediction model and develop good fault prediction accuracy, the best test strategy can significantly reduce the amount of test effort while still maintaining the same level of fault detection rate or provide a better level of fault detection with the same amount of test effort. The strategy A5 is able to detect 100% of faults with only 80-85% of effort. Thus by using fault prediction 15-20% of effort can be reduced. Fault Prediction allows the testers to focus more on fault prone modules. Considerable future work is required to generalize these results. TERA is evaluated only on one dataset(KC1). As part of future work TERA can be evaluated on other datasets.

REFERENCES

[1] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker "Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing"IEEE Transactions on Software Engineering vol.39, No.10, October 2013.

[2] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, "Predicting the Location and Number of faults in Large Software Systems,"IEEE Trans. Software Eng.,vol. 31, no. 4, pp. 340-355, Apr. 2005 .

[3] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," IEEE Trans. Software Eng., vol. 34, no. 4, pp. 485-496, July/Aug. 2008.

[4] Y. Kamei, A. Monden, and K. Matsumoto, "Empirical Evaluation of SVM-Based Software Reliability Model,"Proc. Fifth ACM/IEEE Int'l Symp. Empirical Software Eng.,vol. 2, pp. 39-41, 2006.

[5] P. Knab, M. Pinzger, and A. Bernstein, "Predicitng Defect Densities in Source Code F iles with Decision Tree Learners,"Proc. Third Working Conf. Mining Software Repositories,pp. 119-125, 2006.

[6] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," IEEE Trans. Software Eng., vol. 34, no. 4, pp. 485-496, July/Aug. 2008.

[7] Y. Kamei, S. Matsumoto, A. Monden and A.E. Hassan, "Revisiting Common Bug Prediction Findings Using Effort Aware Models,"Proc. IEEE Int'l Conf. Software Maintenance,pp. 1-10, 2010.