Volume 2, Issue 6, June 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Applying Machine Learning for Fault Prediction Using Software Metrics

Shanthini. A
Research scholar,
Department of Computer Science and Engineering,
Annamalai University,
Annamalai nagar, Tamil Nadu, India.

Chandrasekaran.RM
Professor,
Department of Computer Science and Engineering,
Annamalai University,
Annamalai nagar, Tamil Nadu, India.

***Abstract*— Defective modules pose considerable risk by decreasing customer satisfaction and by increasing the development and maintenance costs. Therefore, in software development life cycle, it is essential to predict defective modules as early as possible so as to improve software developers' ability to identify the defect-prone modules and focus quality assurance activities such as testing and inspections on those modules. Software metrics play an important role in measuring the quality of software. It is desirable to predict the quality of software as early as possible, and hence metrics have to be collected early as well. This study is focused on the high-performance fault predictors based on machine learning algorithms. The defect prediction is performed using method-level metrics and class-level metrics for KC1 dataset. We used public NASA datasets from the PROMISE repository. Various classifiers were examined for kc1 dataset using different metrics level data SVM provides the best prediction performance for both metrics level in terms of precision, recall and accuracy.**

***Keywords*— Defect prediction, Software metrics, Machine learning, Class level metrics, Method level metrics .**

I. INTRODUCTION

Faults and failures in software are costly factors. They account for a significant amount of any project budget. This cost cannot be removed completely as methods are needed to ensure the quality of the software, but the costs for fault handling should be possible to decrease considerably by introducing and improving methods for early fault identification. Moreover, this information can be used as feedback to the development and the enhancement of the development processes. Thus, it can be used for process improvement and hence cost reduction. Based on the above reasoning, it is clear that methods are needed to predict, control and improve fault handling in general. The type of methods can be divided into two major classes: methods for prediction of the number of faults in a specific module and methods for identification of fault-prone modules. The first type of methods has been investigated and evaluated, but it has been difficult to develop a valid model, in particular a model which is transferable between projects or organisations. Thus, methods for identification of fault- and failure prone modules and models for fault prediction are a potential way to improve software quality and to reduce cost.

Effective defect prediction models can help developers focus quality assurance activities on defect-prone modules and thus improve software quality by using resources more efficiently. These models often use static measures obtained from source code—mainly size, coupling, cohesion, inheritance, and complexity measures—which have been associated with risk factors, such as defects and changes.

This study investigates the significance of different software metrics with regard to defect prediction. In this paper, we used two different machine learning algorithms to find the relationship between software metrics and fault proneness. The proposed model is empirically evaluated using public domain KC1 NASA data set. Finally an analysis is conducted in order to determine the power of the metrics in classifying classes as defective or defect free. Then the results of machine learning models are compared with regard to defect prediction.

The rest of this paper is organized as follows. Subsequent sections describe related work. The 3rd section is for the data source. The 4th and 5th sections present details about metrics and machine learning approaches used. Performance evaluation is discussed in the 6th section. Conclusions are given in the 7th section.

II. REVIEW OF RELATED LITERATURE

Amjan Shaik et al. [2] have done statistical analysis for object oriented software metrics on CK metric suite by validating the data collected from the projects of some students. Metrics data provided quick feedback for software designers and managers. They found out that if

appropriately used; it could lead to a significant reduction in cost of the overall implementation and improvement in quality of the final product. Dr. B.R. Sastry et al. [1] tried to implement software metrics with aid of GUI & also analyzed relationships of metrics to determine quality and quantity of software attributes measured with regard of object oriented software development life cycle.

C. Neelamegan et al. [3] did survey on four object oriented quality metrics. In this study, the metrics for object oriented design focused on measurements that were applied to the class and design characteristics. Dr Kadhim M. Breesam [4] validated a set of metrics empirically that could be used to measure the quality of an object oriented design in terms of using class inheritance.

Considerable research has been performed on software metrics and defect prediction models. However, only few of the studies considered identifying small number of metrics that would be effective with regard to defect prediction. Knab, Pinzger and Bernstein [6] conducted an experiment, where different defect prediction models were constructed using decision trees over seven versions of the Mozilla project. Product metrics as well as process metrics were used. Different models used different sets of metrics. And therefore, it was possible to compare the efficiency only the metrics, which described the historical defects (NDPV related) and it gave good predictions in 62% of the investigated cases.

Catal et al. [5] examined Chidamber-Kemerer metrics suite and some method-level metrics (the McCabe's and Halstead's ones) for a defecdel which is based on Artificial Immune Recognition System (AIRS) algorithm. The authors investigated together 84 metrics from the method-level metrics transformation and 10 metrics from the class-level metrics. The dataset under study was a storage management project for receiving and processing ground data in the NASA. The project had been implemented in C++. According to obtained results the authors concluded that the best fault prediction is achieved when CK metrics are used with the lines of code (LOC) metric.

Olague et al. [7] investigated three metrics suites (CK, MOOD and QMOOD). The metrics suites were validated for their ability in predicting fault-proneness. The authors used defect data from six versions of Mozilla Rhino, an open-source implementation of JavaScript written entirely in Java. The authors obtained the best results for CK metrics. However, they concluded that both CK and QMOOD suites contain metrics, which are effective with regard to detecting error-prone classes.

III. DATA SOURCE

The data set used in this research is obtained from the NASA IV & V Facility Metrics Data Program (MDP) data repository.1 The primary objective of the MDP is to collect, validate, organize, store and deliver software metrics data. The repository contains software metrics and associated error data for several projects. The data is made available to general users and has been sanitized by officials representing the projects from which the data originated. For each project in the database, unique numeric identifiers are used to describe product entries. A product refers to anything with which defect data and metrics can be associated. In most cases, it refers to code-related project modules such as functions. For each module, metric values were extracted and mapped to a defect log. Because the recorded metric values for a module correspond to those obtained before eliminating faults in the module, there is no risk of the metric values changing as a result of structural changes in the module that may occur during fault elimination. We use the data associated with the KC1 project. This is a real-time project written in C++ consisting of approximately 315,000 LOC. There are 10,878 modules and 145 classes.

IV. METRICS

The binary dependent variable in our study is fault proneness. The goal of our study is to explore empirically the relationship between software metrics and fault proneness. Fault proneness is defined as the probability of fault detection in a class. We used machine learning methods to predict the probability of fault proneness. Our dependent variable is predicted based on the faults found during software development life cycle. The metrics are used as independent variable. Both method level metrics and class level metrics are used in defect prediction. Public NASA datasets include 21 method-level metrics proposed by Halstead and McCabe. Additional to these it also has class-level metrics, too. Method-level metrics are suitable for both procedural and object-oriented programming paradigms, whereas, class-level metrics can be only calculated for programs developed with object-oriented programming languages.

The 21 method level metrics are shown below:

- LOC_BLANK
- BRANCH_COUNT
- LOC_CODE_AND_COMMENT
- LOC_COMMENTS
- CYCLOMATIC_COMPLEXITY
- DESIGN_COMPLEXITY
- ESSENTIAL_COMPLEXITY
- LOC_EXECUTABLE
- HALSTEAD_CONTENT
- HALSTEAD_DIFFICULTY
- HALSTEAD_EFFORT
- HALSTEAD_ERROR_EST
- HALSTEAD_LENGTH
- HALSTEAD_LEVEL
- HALSTEAD_PROG_TIME
- HALSTEAD_VOLUME

- NUM_OPERANDS
- NUM_OPERATORS
- NUM_UNIQUE_OPERANDS
- NUM_UNIQUE_OPERATORS
- LOC_TOTAL

In addition to the method level metrics, 10 class-level oriented metrics are used. The object oriented metrics are given below which all 6 Chidamber- kemerer metrics:

- WEIGHTED METHODS PER CLASS
- DEPTH OF INHERITANCE TREE
- RESPONSE FOR A CLASS
- NUMBER OF CHILDREN
- COUPLING BETWEEN OBJECT CLASSES
- LACK OF COHESION IN METHODS
- PERCENT_PUB_DATA
- ACCESS_TO_PUB_DATA
- DEP_ON_CHILD
- FAN_IN

V. EXPERIMENT DESCRIPTION

We used machine-learning algorithms on the datasets .Normally, the choice of machine-learning algorithm and model parameter adjustments affect prediction performance. Classification is a procedure that assigns a class label to a sample from a given set of samples that have labels. The classification is done using Naive Bayes, K-star, Random forest and SVM. Naive Bayesian Classification is the most known and used classification method. It is not only easy to implement on various kinds of datasets, but also it is quite efficient. Naive Bayes is a probabilistic classifier based on Bayes' theorem and it has strong independence assumptions. Naive Bayes assumes that the existence of a class's feature does not depend on the existence of the other features. When ompared to the other algorithms, Naive Bayes requires a smaller data quantity to estimate the parameters.

Support vector machine (SVM) is also well known tool for performing data classification, and have been successfully used in many applications. SVM constructs an N-dimensional hyperplane that optimally separates the data set into two categories. The purpose of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that cases with one category of the dependent variable on one side of the plane and the cases with the other category on the other side of the plane . The support vectors are the vectors near the hyperplane. The SVM modeling finds the hyperpalne that is oriented so that the margin between the support vectors is maximized. When the points are separated by a nonlinear region, SVM handles this by using a kernel function inorder to map the data into a different space when a hyperplane can be used to do the separation.

Random Forests represents a major advance in data mining and knowledge discovery, offering high levels of predictive accuracy. Random Forests is best suited for the analysis of complex data structures embedded in small to moderate data sets. Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and the tree "votes" for that class. The forest chooses the classification having the most votes. K-star is an instance-based classifier, which is the class of a test instance is based upon the class of those training instances similar to it, as determined by similarity function. K-star retrieves the nearest stored example using an entropic measure instead of Euclidean distance to represent instance-based learning instead of the k-Nearest Neighbor because it produced better results for the selected data sets. Each defect prediction model is validated using 10 fold cross validation.

VI. PERFORMANCE

In this paper, we have evaluated two different levels of metrics i.e., 21 method level metrics and 10 class level metrics using four different classifiers such as Naive Bayes, K-star, Random forest and SVM. We have done an analysis to find out the nature of relationship of these metrics level with defects. For method level analysis 21 metrics are used. The KC1 dataset used has 2109 methods. 15% of which are defective and 85% of which are non defective. For class level analysis 10 metrics are used. The KC1 dataset used has 145 classes. Based on the conducted experiments, the Precision, recall F-Measure and accuracy results for NB, K-star, SVM, and random forest using method level and class level metrics as independent variable are presented in Table 1 and Table 2 respectively.

From the performance measures it is observed that Random Forest outperforms the other algorithms in KC1 data set for method level metrics. K-Star is the second best algorithm for method level metrics. In case of Class level metrics, SVM on the average gives better performance than the other classifiers.

TABLE 1
PERFORMANCE OF CLASSIFIERS USING METHOD LEVEL METRICS

Classifier	Precision	Recall	F-Measure	Accuracy
Naïve Bayes	81.6	82.4	82	82.4
SVM	81.6	84.8	78.6	82.8
K-star	82.6	84	83.2	83.97
Random Forest	**82.9**	**85.3**	**83.4**	**85.3**

TABLE 2
PERFORMANCE OF CLASSIFIERS USING CLASS LEVEL METRICS

Classifier	Precision	Recall	F-Measure	Accuracy
Random forest	71.9	70.3	70.6	70.34
K-star	73	72.4	72.6	72.41
Naïve Bayes	68.8	68.3	66	68.27
SVM	**80.2**	**80**	**80.1**	**80**

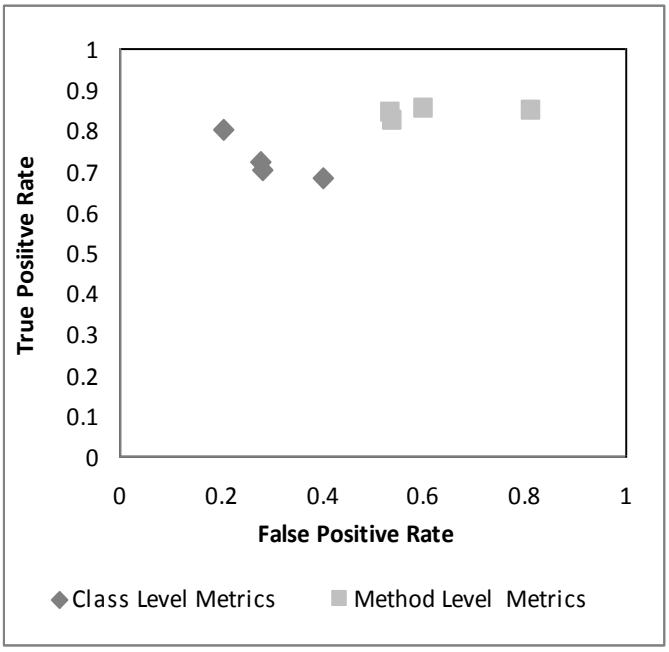

Fig. 1 Performance of classifiers using class level and method level metrics

Besides the commonly used evaluation metrics namely precision, recall, f-measure and Accuracy, Receiver operative Characteristics (ROC) curve, is used as an additional alternative evaluation metric. ROC curve plots the classes correctly classified as defective (True positive) against the classes incorrectly classified as defective (False positive). Fig 1 shows the ROC curve evaluating the performance curve of various classifiers on the KC1 data set. The diagonal represents the expected performance of a random classifier. From the fig 1, it is evident that, for class level metrics most of the classifiers provide conservative performance. For method level metrics, the classifier occupies the liberal performance region. So on the average, classifiers in general perform better for class level rather than method level metrics.

VII. CONCLUSIONS

The goal of our research is to analyze the performance of various classifiers on defect prediction Based on public domain NASA data set KC1; we analyzed the performance of the classifiers using traditional measures such as precision, recall and F-measure. Roc is also used as an alternative metric. The SVM method out performs other classifiers for class level metrics .Random forest shows better performance for method level metrics. This study confirms that construction of the SVM models is feasible, adaptable to OO systems, and useful in predicting fault prone classes for higher level of metrics (class). SVM is not much feasible for low level metrics (methods). While research continues, practitioners and researchers may apply machine learning methods for constructing the model to predict faulty classes. We plan to replicate our study to predict the models based on other machine learning algorithms such as genetic algorithms.

REFERENCES

[1] Dr B.R. Sastry, M.V. Vijaya Saradhi, "Impact of software metrics on Object Oriented Software Development life cycle", *International Journal of Engineering Science and Technology*, Vol 2 (2), pp 67-76, 2010.

[2] Amjan Shaik, Dr C.R.K. Reddy, Dr A Damodaran, "Statistical Analysis for Object Oriented Design Software security metrics", *International journal of engineering and technology*, vol 2, pp 1136-1142,2010.

[3] C. Neelamegan, Dr. M. Punithavalli, "A Survey- Object Oriented quality metrics", *Global journal of Computer Sc.And Technology*, Vol 9, no 4, 2009.

[4] Dr Kadhim M. Breesam, "Metrics for Object Oriented design focusing on class Inheritance metrics", *2nd International conference on dependability of computer system IEEE*, 2007.

[5] C.Catal, B.Diri and B.Ozumut, "An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software," in *2nd International Conference on Dependability of Computer Systems* DepCoS-RELCOMEX, 2007.

[6] P.Knab, M.Pinzger and A.Bernstein, "Predicting defect densities in source code files with decision tree learners," in *the 2006 International Workshop on Mining Software Repositories*, 2006.

[7] H.Olague, L.Etzkorn, S.Gholston and S.Quattlebaum, "Empirical Validation of Three Software Metrics Suites to Predict Fault- Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402-419.

[8] T.Ostrand, E.Weyuker and R.Bell, "Where the bugs are," in the 2004 ACM SIGSOFT *International Symposium on Software Testing and Analysis*, ISSTA'04, pp. 86–96, 2004.

