RESEARCH SCRIPT IJRCS - International Journal of Research in Computer Science ISSN: 2349-3828

METHOD LEVEL BUG PREDICTION USING INFORMATION GAIN

[1]Vaijayanthi Murugan | [2]Karthick M
[1,2](AP/CSE, SNS College of Technology, Coimbatore, India, vaijayanthimurugan25@gmail.com))

Abstract— *Software defects commonly known as bugs, present a serious challenge for the software developers to predict the bugs and to enhance the system reliability and dependability. The software defects are usually an incorrect output value, exceptions occurred in the source code, failure due to logical errors or due to any syntax errors. As the size of the programs grows and it may contain large number of methods, so, occurrence of bugs become more common and difficult to fix. It will take time to predict the bugs at the individual methods. Many techniques have been developed to mainly focus on method-level bug prediction. Several features are commonly used for method level bug prediction. To identify the best set of features it is proposed to use Filter Based Feature Selection (FBFS) using Information Gain. The Information Gain value is calculated for estimating the individual features. Based on the Information Gain values, the relevant features will be extracted for evaluation. In this work, the method-level bug prediction will be carried out using Support Vector Machine (SVM) classifier. Finally, the performance of the bug prediction models will be measured by using Precision, Recall and F-measure values. The volume of predicted bugs can be assessed by using the values of evaluation measures.*

Keywords— ***bug prediction, precision, recall, F-measure, method-level, information gain, accuracy, SVM classifier.***

1. INTRODUCTION

Bug prediction is considered to be an important field of research in software engineering. Usually software defects, such as, indefinite loop or an incorrect output values are commonly known as bugs. Typically bug prediction plays a major role to predict the bugs in the early stages of software development process. Recently many bug prediction techniques are based on method-level. These techniques include several Machine Learning Classifiers like Naïve Bayesian classifier, Support Vector Machine classifier.

Such techniques rely on some software quality metrics (say., Lines of Code, complexity). The software defects or bugs cannot be directly measured, so certain software quality metrics are collected from standard datasets. The dataset includes source code metrics like Chidamber and Kemerer (CK) metrics, Object Oriented (OO) metrics and Change metrics to predict bugs at the method-level.

To evaluate the performance of the bug prediction model, Machine Learning classifier called Naïve Bayesian classifier is used. The first step in classification process is that to construct a model using training set. Then the classifier will use the model to predict the bugs in future with unknown class values as testing set. The performance of the bug prediction model is estimated using some well-known performance evaluation criteria like Precision, Recall and Accuracy values.

The accuracy is the degree to which the algorithm correctly identifies future bugs. Precision is defined as the ratio of the number of modules correctly predicted as defective, to the total number of modules predicted in the set [1]. Recall is defined as the ratio of the number of modules predicted correctly as defective to the total number of defective modules in the set.

2. LITERATURE REVIEW

A. Machine Learning

There are two things that need to be achieved in machine learning process. First, the training needs to be done with known class labels. Second, the trained model needs an efficient algorithm to validate the unknown class labels by means of testing. Supervised learning is common in classification problems where the goal is to have the learner learn a predefined classification [1]. Table 1 shows the general structure of data used in supervised learning. Each instance of data is defined by a set of features and a class.

B. Ten-Fold Cross Validation

Cross-validation involves partitioning a sample of datasets into complementary subsets, performing the analysis on one subset (called the training set), and validating on the other subset (called the validation set or testing set) [2].

The validation process simplifies as:

TABLE I. GENERAL STRUCTURE OF DATA USED IN SUPERVISED LEARNING WITH KNOWN CLASSES

Data in Standard Format					
Case	Feature 1	Feature 2	….	Feature n	Class
1	XX	X	….	XXX	YES
2	XX	X	….	XXX	NO
3	XX	X	….	XXX	YES
N	XX	X	….	XXX	NO

1. Break data into 10 sets of size n/10.
2. Train on 9 datasets and test on 1 dataset.
3. Repeat 10 times and take a mean accuracy.

The initial stage to break all the given dataset into pieces. Then it takes the training dataset and to test for the remaining dataset and repeat the process till all the dataset completes.

3. CLASSIFICATION USING NAÏVE BAYESIAN APPROACH

A. Corpus Collection

To experiment with the machine learning classifier, the data are collected from the standard bug prediction dataset such as Lucene dataset. It consist of the source code

metrics, change metrics and bug metrics. The main focus of this paper is to construct a model to predict the method-level rather than at the file-level requires that all metrics are available at method-level.

B. Dataset

The standard bug prediction dataset includes source code metrics like Chidamber and Kemerer (CK) metrics and Object Oriented (OO) metrics. The change metrics includes, the file(s) being affected by the changes commonly known as Revision. The source code metrics include #methods, #fanin, #fanout, and #attributes. Bug prediction is an important challenge in Software Engineering research. The goal is to build reliable predictors that can indicate in advance about those components of a software system that are more likely to fail. Due to its relevance to software quality, various bug prediction techniques have already been proposed.

Essentially, such techniques rely on different predictors, including source code metrics, change metrics, etc. The main focus is to define the relationships between the defined metrics and the occurrences of bugs [3]. The metrics are already defined in the public dataset to evaluate the bug prediction techniques. This dataset provides the change log approaches and the single-version approaches and hence provide the necessary information for the defect prediction. The original dataset includes the Lucene dataset. The metrics that are available in the dataset are [4] Chidamber and Kemerer (CK) metrics and Object Oriented (OO) metrics. There are 6 CK metrics and 11 OO metrics listed in Table 2 which shows the metrics included in the original dataset.

TABLE II. CHIDAMBER AND KEMERER METRICS AND OBJECT ORIENTED METRICS

TYPES	METRICS	DESCRIPTION
CK	WMC	Weighted Methods per Class
CK	DIT	Depth of Inheritance Tree
CK	RFC	Response For Class
CK	NOC	Number of Children
CK	CBO	Coupling Between objects
CK	LCOM	Lack of Cohesion on Methods
OO	FANIN	Number of Classes that Reference the Class
OO	FANOUT	Number of Classes Referenced by the Class
OO	NOA	Number of Attributes
OO	NOPA	Number of Public Attributes
OO	NOPRA	Number of Private Attributes
OO	NOAI	Number of Attributes Inherited
OO	LOC	Number of Lines of Code
OO	NOM	Number of Methods
OO	NOPM	Number of Public Methods
OO	NOPRM	Number of Private Methods
OO	NOMI	Number of Methods Inherited

C. Code metrics

There are two traditional suits of code metrics exist:

- CK metrics suite.
- Set of metrics directly calculated at the method-level.

E. Naïve Bayes Classifier

A Naïve Bayes classifier is a probabilistic classifier based on applying Bayes' theorem with strong independence assumptions. When represented as a Bayesian network, a Naïve Bayes classifier has the structure depicted in [4] Figure 1. It shows the independence assumption among all features in a data instance.

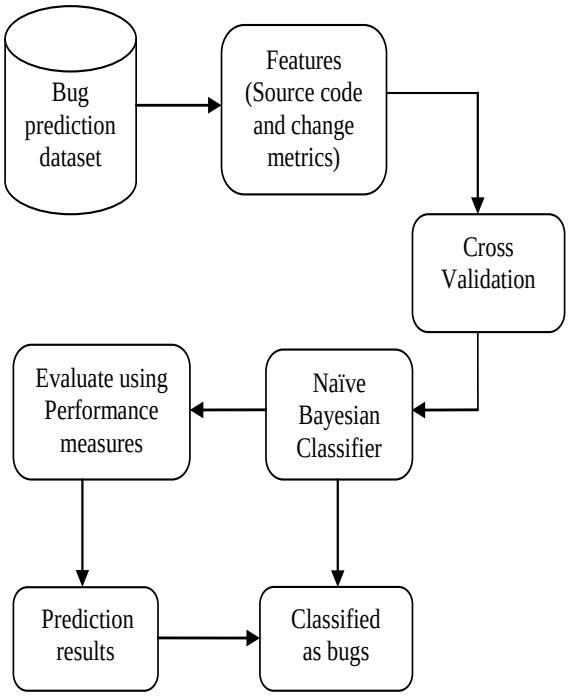

Fig. 1. Block Diagram of Naïve Bayesian Approach

F. Algorithm

Let **X** = {X1… Xn} be a finite set of observed random variables, called features, where each feature takes values from its domain Di. The set of all feature sets is denoted by Ω = D1 × … × Dn. Let C, such that C ∈ {0, u − 1}, be an unobserved random variable denoting the class of a set of features.

A hypothesis $h : \Omega \rightarrow \{0 \ldots u-1\}$, that assigns a class to any given set of variables is defined as a classifier. Each class c is assigned a discriminant function $f(c(\boldsymbol{x}))$, $c = 0 \ldots u-1$ [5]. The classifier selects the class with the maximum discriminant function on a given set of variables.

Thus the bayes theorem can be written as:

$$P(X) = \frac{P(X/H)\,P(H)}{P(X)} \qquad (1)$$

The equation 1 shows that, it predicts X belongs to C_i iff the probability $P(C_i/X)$ is the highest among all the $P(C_k/X)$ for all the k classes.

G. Performance Evaluation

The classifier is performed by evaluating which set of features yields the best overall classification accuracy and

recall, and also by examining the relative contributions of individual features. Table 3. explains the confusion matrix.

Precision

Precision is defined as the ratio of the number of modules correctly predicted as defective, to the total number of modules predicted in the set. Precision is also termed as True Positive which classified as truly predicted as bugs.

$$\frac{TP}{TP + FP} \quad (2)$$

Recall

This metric indicates the coverage of the accuracy. Recall is defined as the ratio of the number of modules predicted correctly as defective to the total number of defective modules in the set. Recall is also termed as True Negative which classified as not bugs.

$$\frac{TP}{TP + FN} \quad (3)$$

TABLE III. CONFUSION MATRIX TABLE

	Actual class (observation)	
Predicted class (expectation)	TP (True Positive) Correct result	FP (False Positive) Unexpected result
	FN (False Negative) Missing result	TN (True Negative) Correct absence of result

4. CLASSIFICATION USING SUPPORT VECTOR MACHINE CLASSIFIER

A. Feature Selection

In feature selection process, the source code metrics and the change metrics are selected from the whole bug prediction dataset. The features will be taken for classifier training once it is evaluated using K-Fold cross validation process [6]. The cross validation process will separate the training set and the testing set.

B. Information Gain Calculation

Once the metrics have been collected from the dataset, the small set of features alone will be selected for the evaluation of bug prediction process [7]. The subset of features will be selected from the whole dataset by calculating the information gain for all the features in the dataset. The Support Vector Machine classifier will take the testing set and will calculate the accuracy and then will take the testing samples to evaluate the accuracy for unknown labels

The features in the dataset include six Chidamber and Kemerer (CK) metrics and eleven Object Oriented (OO) metrics. It also contains nine change metrics, lines added to the source code, maximum lines added to the source code, average lines added, lines removed from the source code, maximum lines removed, average lines removed, maximum and average code churn added or deleted. The CK metrics include weighted methods per class, depth of inheritance tree, coupling between object classes, response for a class, lack of cohesion in methods. [13]The OO metrics include fanin, fanout, number of attributes, number of methods, number of methods inherited.

C. Performance evaluation of the features

Since the choice of features can affect the performance of classifiers, each feature's discriminative power for performing change classification is compared [8]. This is performed by evaluating which set of features yields the best overall classification accuracy and recall, and also by examining the relative contributions of individual features.

The accuracy of the classifier will be estimated by combining the terms Precision and Recall using the F-measure values.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4)$$

D. Evaluating the Approaches

The performance of bug prediction approaches is evaluated with several strategies, each according to a different usage scenario of bug prediction. We evaluate each technique in the context of classification (defective/non-defective) [9]. Prior to model building and classification we labeled each method in our dataset as either as bug-prone or not bug-prone as follows:

$$bugClass = \begin{cases} not\ bug - Prone: \#bugs = 0 \\ bug - Prone: \#bugs = 1 \end{cases} \quad (5)$$

(Eq. 3.1)

The equation 5 defines the classification of bugs in the dataset which classified as either bug-prone or not bug-prone [10]. If the class classified as bugs, it is defined as numeric values as 1, 2, 3, etc. otherwise 0.

5. EXPERIMENTAL RESULTS

The classifier is trained with the standard Lucene bug prediction dataset. The dataset contains the set of attributes, classes, set of data and instances. Based on the class values the testing set will predict the values similar to that of the training values. The original dataset will contain the information regarding the performance evaluation for the bug predictors. Each and every data will be considered as features in the dataset [11]. Once the data is selected as features, the cross validation will be performed for separating the features as training and testing samples. Then it will feed to the classifier for performance evaluation for calculating the accuracy.

The data are collected from the standard bug prediction dataset. The data will be in the form of excel format (say .xlsx). The original dataset will contain the information regarding the performance evaluation for the bug predictors. Each and every data will be considered as features in the dataset. Once the data is selected as features, the cross validation will be performed for separating the features as training and testing samples [12]. Then it will feed to the classifier for performance evaluation for calculating the accuracy. By analyzing the results for the Naïve Bayesian classifier, the accuracy is evaluated using cross validation method as 10 fold cross validation. By using this classifier, the accuracy obtained is 96.89%. But the accuracy for the support vector machine classifier is 91.02%. The proposed system proposes the feature selection method as Information Gain calculation. To increase the accuracy, the feature selection has to be carried out. The implementation is currently in progress.

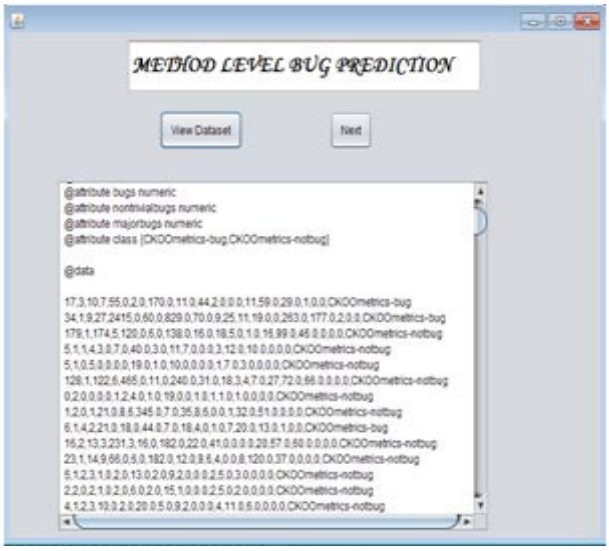

Fig. 2. Metrics in the Lucene Dataset

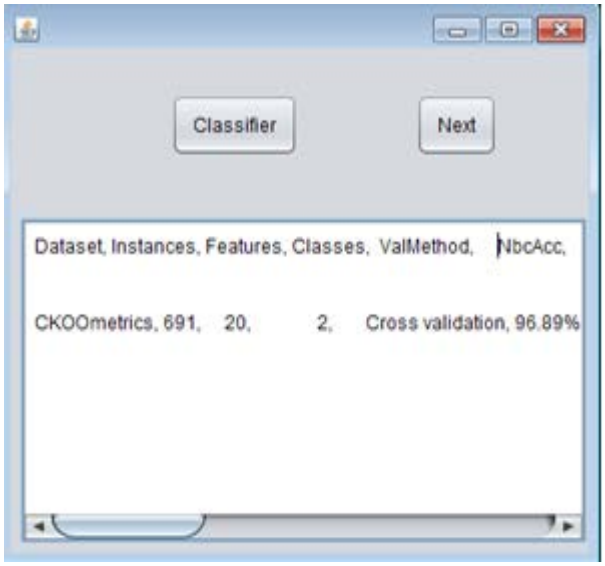

Fig. 3. Accuracy by Naïve Bayesian Classifier

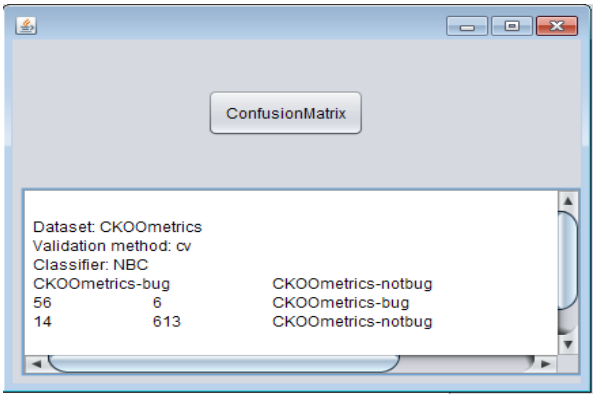

Fig. 4. Confusion Matrix

6. CONCLUSION AND FUTURE WORK

The accuracy of the classifier is evaluated using the performance evaluation measures. For the metrics dataset the Naïve Bayesian classifier performed well and the accuracy has been evaluated with the cross validation process. Finally the accuracy of the classifier is evaluated using the measures like Precision, Recall and F-measure values. The bug prediction process has been carried out with the standard bug prediction dataset.

Future work will include additional metrics related to standard bug prediction dataset and extend the analysis with the advanced feature selection process.

REFERENCES

[1] Giger, E. D'Ambros, M. Pinzger, M. Gall, H.C. (2012) 'Method-level bug prediction', ESEM '12 Proceedings of the ACM-IEEE international symposium on Empirical Software Engineering and Measurement, pp. 171-180.

[2] Bandana Garg. (2013) 'Design and Development of Naïve Bayes Classifier', Fargo, North Dakota.

[3] Langley, P. Iba, W. and Thompson, K. (1992) 'An analysis of Bayesian Classifiers', Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, pp. 223-228.

[4] Friedman, N. Geiger, D. and Goldszmidt, M. (1997) 'Bayesian Network Classifiers', Machine Learning, vol. 29, pp. 131-163.

[5] Kotsiantis, S.B. (2007) 'Supervised Machine Learning: A Review of Classification', Informatica pp. 249-268.

[6] Moser, R. Pedrycz, W. and Succi, G. (2008) 'A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction', Proceedings of ICSE , pp. 181-190.

[7] Couto, C. Silva, C. Valente, M.T. Bigonha, R. and Anquetil, N. (2012) 'Uncovering Causal Relationships between Software Metrics and Bugs', Proceedings of CSMR, pp. 223-232.

[8] Shanthini, A. Chandrasekaran, RM. (2012) 'Applying Machine Learning for Fault Prediction Using Software Metrics', Proceedings of IJARCSSE, pp. 274-278.

[9] Kim, S. Zhang, H. Wu, R. and Gong, L. (2011) 'Dealing with noise in defect prediction', Proceedings of ICSE, vol. 2, no. 6, pp. 481-490.

[10] Freund, Y. and Schapire, R. E. (1999) 'A Short Introduction to Boosting', Journal of Japanese Society for Artificial Intelligence, vol. 14, no. 5, pp. 771-780.

[11] Chidamber, S. R. and Kemerer, C. F. (1994) 'A metrics suite for object oriented design', IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493.

[12] Kim, S. Whitehead, E. J. Zhang, J. Y. (2008), 'Classifying Software Changes: Clean or Buggy?', vol. 34 no. 2, pp. 181-196.

[13] Subramanyam, R. and Krishnan, M.S. (2003), 'Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects', IEEE Transaction on Software Engineering, vol. 29 no. 4, pp. 297-310.