Contemporary Engineering Sciences, Vol. 7, 2014, no. 31, 1691 - 1697
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2014.411227

Classification Model for Maintainability Prediction

Mikyeong Park and Euyseok Hong*

School of Information Technology
Sungshin Women's University, Korea
* Corresponding author

Abstract

Software maintainability prediction is important because it enables organizations to effectively manage maintenance resources and improve design and coding. Most studies have concentrated on estimating the number of changes or changed lines of code during maintenance period. On the other hand, we propose classification models that determine maintainability levels of software units. Classification model is simple to use and makes it easy for developers to analyze results. Using widely used classification algorithms, we build and evaluate two types of prediction models. The experimental results show that decision tree with a CfsSubsetEval attribute selection method has the best performance in binary classifications and Naïve Bayesian outperforms others in ternary classifications.

Keywords: software maintainability, maintainability prediction, classification

1 Introduction

As software maintainability becomes more and more significant, predicting maintainability is also getting important. If a maintainability prediction model makes an accurate result, the model enables organizations to effectively manage maintenance resources. Furthermore, it can help developers improve design and coding. By improving maintainability, cost and time spent on software development are reducing [1].

Figure 1 presents the structures of an estimation model and a classification model. Software units mean software modules or classes. Inputs of an estimation model are these units and outputs are the corresponding values that enable to evaluate their maintainability. In general, output values are the number of changes or changed lines of code during a maintenance phase. However, it is difficult to

determine levels of maintainability using the numbers. In addition, estimating exactly the correct number is not easy. In contrast, we propose classification models that classify software units into several groups representing different maintainability levels. It is simple to use the models and they make it easy for developers to analyze classification results at a time so that it can be more useful than an estimation model.

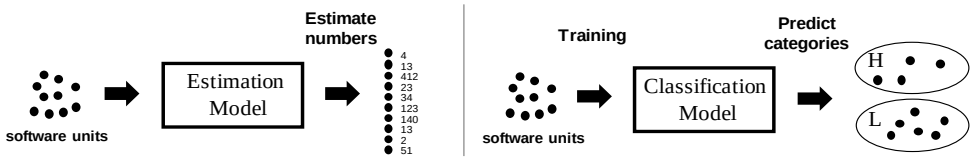


Figure 1. Comparison of an estimation model and a classification model

2 Related Works

There are many studies about models to predict software maintainability [1]. Methods used in these models vary from statistical techniques to machine learning approaches. Most of the studies are about estimation models and we describe some important ones. Li and Henry used multiple linear regression for prediction [2]. Neural network modeling techniques were employed to predict the number of software defects and the maintainability [3]. Koten and Gray proposed Bayesian network models at first time [4] and Zhou and Leung used a multivariate adaptive regression splines model [5]. However, using these methods, determining and analyzing levels of maintainability makes developers do laborious works. On the contrary, there are few studies about classification models. Dallal constructed a model using logistic regression techniques [6]. Although he used a classification algorithm, however, did not focus on the maintainability levels.

3 Model Construction

The aim of our study is building and evaluating classification models for maintainability prediction. Maintainability levels are composed of *High* and *Low* in binary-classification or *High*, *Medium* and *Low* in ternary-classification. This being so, ternary-classification can classify software units in more detail than binary-classification. High/low maintainability means the number of changes is few/large. It is to say that *Low* is more serious; it is important to predict software units labeled *Low* more accurately than those labeled *High*.

Figure 2 shows the process of constructing and using a classification model to predict maintainability. The process is divided into three phases: preprocessing phase in which data preprocessing and output labeling are conducted, training phase in which labeled data are trained by training algorithms, and prediction phase in which a trained model predicts the maintainability levels of the new input units.

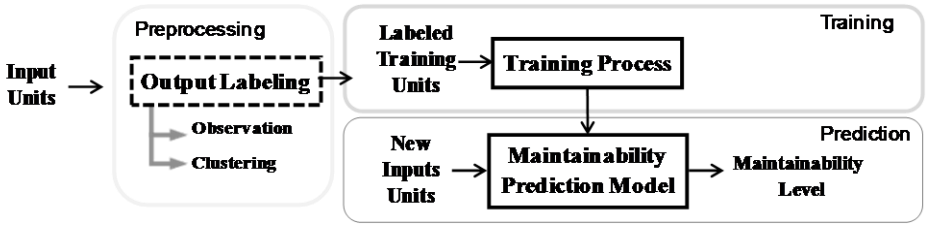


Figure 2. Process of constructing and using a maintainability

Output labeling in the preprocessing phase is to determine the labels for classification outputs of training data. Mostly, the outputs before labeling are the number of changes or changed lines of code. It is needed to make new outputs labeled with maintainability levels according to original output values. The new output values are represented in the form of categories such as *High*, *Low* and/or *Medium*. As shown in dashed box, output labeling can be performed by two ways: by observing the distribution of the original output values or by the result of clustering algorithms using the original values. After labeling, classification algorithms are conducted in the training phase. The four typical classification algorithms are selected in the paper: J48 decision tree, Naïve Bayesian, Support Vector Machine (SVM), and MultiLayer Perceptron (MLP). In the prediction phase, the trained model accepts new units and it predicts their maintainability levels.

4 Empirical Study

4.1 Experimental Setting

We use one of Li & Henry datasets [2], UIMS, which many researchers have used in their analysis. The other dataset, QUES, cannot be labeled properly using output labeling methods. That is why we use only UIMS. UIMS is a dataset with 11 metrics collected from 39 classes of a user interface management system implemented in Ada. The input metric vector is (DIT, NOC, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE2, SIZE1) and the output metric is CHANGE. CHANGE is the number of lines changed per class during a maintenance period.

Figure 3 shows the CHANGE values of UIMS. 39 classes can be divided into two groups according to the CHANGE values intuitively. Figure 4 and 5 present the output labeling results after arranging the classes in order of the CHANGE value. Output labeling is conducted by (a) intuitive observation by human and (b) the result of k-means clustering. Figure 4 shows the result of labeling into binary groups. In binary groups, the result of observation is the same as that of clustering. The labeling results are 34 *Low* classes and 5 *High* classes. Figure 5 shows the result of labeling into ternary groups. Unlike binary groups, the result of (a) is different from that of (b). Building and evaluating models are conducted with all three results of output labeling.

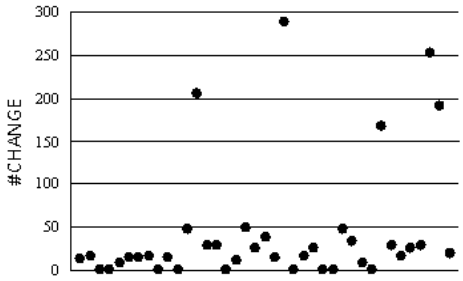


Figure 3. Values of CHANGE

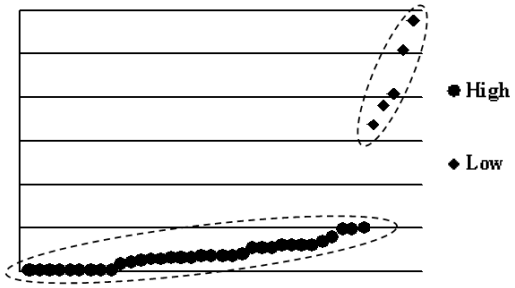
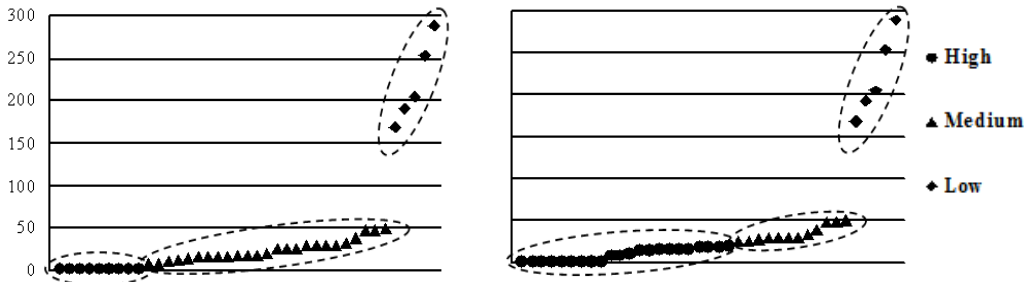


Figure 4. Binary-group of CHANGE



(a) (b)

Figure 5. Ternary-group of CHANGE by (a) observation and (b)

For better performance of prediction, we use a CfsSubsetEval attribute selection method. Table 1 shows all attributes and the results of attribute selection.

Table 1. Results of Attribute Selection

Method		Attributes
all attributes		dit, noc, mpc, rfc, lcom, dac, wmc, nom, size2, size1
CfsSubsetEval	Binary-classification	rfc, dac, size2
	Ternary-classification	dit, noc, mpc, lcom, dac, wmc, size2, size1

4.2 Performance Measure

There are many measures that can be used to evaluate classification performance. We use accuracy, Type I/II error rates and area under ROC curve(AUC). Accuracy is a measure of an overall effectiveness of a classifier and AUC is that of an ability of classifier to avoid *Low* [7]. A Type I error rate is the number of *High* mislabeled as *Low* while a Type II error rate is the number of *Low* mislabeled as *High*. A Type II error is usually more critical than a Type I error because it requires heavy costs at the later phases of system development. The evaluation of multi-classification is more complicated than binary-classification.

In order to evaluate ternary-classification models, accuracy and two types of errors are used. AUC cannot be used because it is difficult to define in multi-classification [7]. Type I/II error rates in multi-classification are similar in form to those in binary-classification but have different meanings. Therefore, we define Type I'/II' error rates newly. A Type I' error rate means the number of *High* mislabeled as *Low* or *Medium* while a Type II' error rate means the number of *Low* mislabeled as *Medium* or *High*. There is also another error type meaning the number of *Medium* mislabeled as *Low* or *High*, but it is negligible.

4.3 Experimental Result

All experiments are run 10 times to obtain accurate results. A model with NotReduction is a model with all attributes. Table 2 shows the evaluation results in binary-classification and ternary-classification.

Table 2. Results of 4 classifiers for binary-classification and ternary-classification

Model		binary-classification				ternary-classification		
		Accuracy	Type I	Type II	AUC	Accuracy	Type I'	Type II'
J48	NotReduction	87.18	2/34	3/5	0.47	79.79	1/9	3/5
	CfsSubsetEval	92.31	1/34	2/5	0.84	79.49	1/9	3/5
NB	NotReduction	89.74	3/34	1/5	0.79	82.05	1/9	1/5
	CfsSubsetEval	87.18	4/34	1/5	0.80	84.62	1/9	1/5
MLP	NotReduction	84.62	2/34	4/5	0.68	76.92	0/9	4/5
	CfsSubsetEval	92.31	1/34	2/5	0.79	74.36	0/9	4/5
SVM	NotReduction	87.18	0/34	5/5	0.50	79.49	2/9	5/5
	CfsSubsetEval	87.18	0/34	5/5	0.50	79.49	2/9	5/5

In binary-classification, the accuracies of the models with CfsSubsetEval are higher than or equal to those with NotReduction except Naïve Bayesian. J48 with CfsSubsetEval achieves 92.31 percent accuracy, with a Type I error rate of 1/34 and a Type II error rate of 2/5. SVM predicts all of *Low* but does not predict *High* at all, thereby a Type I error rate is as low as 0 and a Type II error rate is as high as 1. Both J48 with CfsSubsetEval and MLP with CfsSubsetEval are much better than others in the analysis of accuracy. However AUC which is another measure to evaluate the classification performance is different between them. According to [8], 0.84 of AUC values means an excellent discrimination. Therefore, J48 with CfsSubsetEval gives the best performance in all parts.

The performances of ternary-classification models using k-means clustering for output labeling are worse than those using intuitive observation. Table 2 shows the result of ternary-classification classified only by observation. Although overall the accuracies of ternary-classifications are lower than those of binary - cllassifica-

tions, it is not to say ternary-classifications are bad as predictors. Similar results with binary-classification are shown for SVM that still cannot predict *High* at all. As the result, Naïve Bayesian outperforms others in ternary-classification.

5 Conclusions

Most of the previous researches on software maintainability prediction have focused on estimation models. On the other hand, we built two types of classification models, of which software units are classified into two or three groups representing different maintainability levels. A classification model is useful in that it is simple to use and make it easy for developers to analyze prediction results. According to the experimental results, J48 decision tree with CfsSubsetEval has the best performance in binary classifications and Naïve Bayesian outperforms others in ternary classifications.

Acknowledgements. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2010-0021902).

References

[1] M. Riaz, E. Mendes and E. Tempero, A systematic review of software maintainability prediction and metrics, *Proc. of ESEM 2009*, (2009) 367 - 377. http://dx.doi.org/10.1109/esem.2009.5314233

[2] W. Li and S. Henry, Object-oriented metrics that predict maintainability, *Journal of Systems and Software*, **23 (2)** (1993), 111 - 122. http://dx.doi.org/10.1016/0164-1212(93)90077-b

[3] M. M. Thwin and T. S. Quah, Application of neural networks for software quality prediction using object-oriented metrics, *Journal of Systems and Software*, **76 (2)** (2005), 147 - 156. http://dx.doi.org/10.1016/j.jss.2004.05.001

[4] C. V. Koten and A. R. Gray, An application of Bayesian network for predicting object-oriented software maintainability, *Information and Software Technology*, **48 (1)** (2006), 59 - 67. http://dx.doi.org/10.1016/j.infsof.2005.03.002

[5] Y. Zhou and H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, *Journal of Systems and Software*, **80 (8)** (2007), 1349 - 1361. http://dx.doi.org/10.1016/j.jss.2006.10.049

[6] J. A. Dallal, Object-oriented class maintainability prediction using internal quality attributes, *Information and Software Technology*, **55 (11)** (2013), 2028-2048. http://dx.doi.org/10.1016/j.infsof.2013.07.005

[7] M. Sokolova and G. Lapalme, A systematic analysis of performance measures for classification tasks, *Information Processing and Management*, **45 (4)** (2009), 427 - 437. http://dx.doi.org/10.1016/j.ipm.2009.03.002

[8] R. Shatnawi and W. Li, The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process, *Journal of Systems and Software*, **81 (11)** (2008), 1868 - 1882. http://dx.doi.org/10.1016/j.jss.2007.12.794

Received: October 1, 2014; Published: December 2, 2014