International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 481-483

COMPARING THE EFFECTIVENESS OF MACHINE LEARNING ALGORITHMS FOR DEFECT PREDICTION

Pradeep Singh

Software repositories with defect logs are main resource for defect prediction. In recent years, researchers have used the vast amount of data that is contained by software repositories to predict the location of defect in the code that caused problem. In this paper machine learning approach is used for predicting the modules with defect for embedded data set. Public datasets from the promise repository have been explored for identifying software defects using machine learning methods. The repository contains software metric data and error data at the function/method level. The aim of the paper is to classify embedded data set using J48, OneR and Naïve Bayes machine learning algorithms to construct a model that predicts potentially defected modules within a given set of software modules with respect to their metric data and study the performance of these machine learning algorithms. The result is compared on the basis of confusion matrices. The study showed that J48 and OneR performed better than Naive Bayes.

***Keywords**:* Machine Learning, Software Metrics, Defect Prediction.

1. Introduction

Software test engineers are always interested to identify the piece of software which are the most likely to fail. Various fault predictive models have been proposed for defect analysis so that fault may be detected at an early stage. Timely prediction of faults in software modules saves lot of testing budget. Repositories containing software metric data and error data at the function/method level can predict the number of faults in software module. In this paper comparative evaluation of machine learning algorithm on small embedded software implemented in *C* consists of 121 modules and 29 static code attributes and other consists of 101 modules 29 static code attributes is presented. In this paper section two briefly describes the software metrics used in this paper. In the section three brief description of machine learning algorithms. Section four contains the experimental evaluation of machine learning algorithms on the dataset. In the last section, on the basis of evaluation various conclusions are drawn and future scope for the present work is discussed.

2. Software Metrics

Software metric is a simple quantitative measure driveable from any attribute of software life cycle. Software metrics make it possible for software engineers to measure and predict software process, necessary resources for projects and work product relevant for a software development effort. Software metric is a measure of some property of a piece of software or its specification. Several data mining methods have been proposed for defect analysis in the past [8], [9], [10], [11]. Many researchers use static attributes to guide software quality predictions ([1], [2], [3], [4], [5], [6]). There is no single set of metrics that could act as a universally best defect predictor. So all 29 static code attributes (McCabe, Halstead and LOC measures) are used for defect prediction. The paper presents an application of machine learning algorithms in software engineering for predicting modules with defects. In order to do this 5 different lines of code measure, 3 McCabe metrics, 12 Halstead measures, branch count and few other metrics are used [1] [2].

3. Machine Learning Algorithms

Machine learning methods have been successfully applied for solving classification problems in many applications. The algorithms selected for classification on embedded dataset to compare the predictive effectiveness of J48 (Decision tree learner), OneR and Naïve Bayes (probabilistic learner). J48 is a JAVA implementation of Quinlan's C4.5 (version 8) algorithm [12]. The J48 Decision tree classifier algorithm recursively splits a data set according to tests on attribute values in order to separate the possible predictions. The algorithm uses the greedy top-down construction technique to induce decision trees for classification. A decision-tree model is built by analyzing training data and the model is used to classify unseen data. J48 generates decision trees, the nodes of which evaluate the existence or significance of individual features. The decision trees are constructed in a top-down fashion by choosing the most appropriate attribute each time. An information theory measure is used to evaluate features, which provides an indication of the "classification power" of each feature. Once a feature is chosen, the training data are divided into subsets, corresponding to different values of the selected feature, and the process is repeated for each subset, until a large proportion of the instances in each subset belong to a single

* Department of Computer Sc. & Engineering, National Institute of Technology, Raipur, INDIA. *E-mail: psingh.cs@nitrr.ac.in.*

class. This algorithm is chosen to compare the accuracy rate with other algorithms.

The next scheme, OneR, produces very simple rules based on a single attribute. OneR is also useful in generating a baseline for classification performance. OneR algorithm builds prediction rules using one or more values from a single attribute [13]. The OneR algorithm creates one rule for each attribute in the training data, and then selects the rule with the smallest error rate as its one rule. To create a rule for an attribute, the most frequent class for each attribute value must be determined. The most frequent class is simply the class that appears most often for that attribute value. A rule is simply a set of attribute values bound to their majority class. The error rate of a rule is the number of training data instances in which the class of an attribute value does not agree with the binding for that attribute value in the rule. OneR selects the rule with the lowest error rate. In the event that two or more rules have the same error rate, the rule is chosen at random. This algorithm is taken for comparing the strength of prediction with other algorithms, due to its simplicity and single attribute requirement.

The naive Bayes algorithms are based on theorem of Bayes posterior probability. Naive Bayes makes the assumption of class conditional independence i.e. there are no dependence relationship among the attributes. Learning a naive Bayes classifier is straightforward and involves estimating the probability of attribute values within each class from the training instances. Probabilities are estimated by counting the frequency of each discrete attribute values. For numeric attributes it is common practise to use the normal distribution [19]. *C*4.5 is an algorithm that summarises the training data in the form of a decision tree. Learning a decision tree and rule is different process than learning a Naive Bayes model.

4. EXPERIMENTAL RESULTS

The data sets used here is publicly available. The classification was done on freely distributed tool available online WEKA machine learning toolkit [15]. In the experiment machine learning algorithms was used for performance analysis of two dataset having 121 modules and 29 metrics having 9 defective modules and other having 101 modules 29 static codes attributes and 15 defective modules. The selected features were assessed by running them through a 10-way cross validation over the J48, OneR and Naïve Bayes. The 10-fold cross-validation process splits the data into 10-equal disjoint parts and uses 9 of these parts for training the framework and 1 for testing. This is done 10 times, each time using a different part of data for testing. The training data are used initially to discretize the faults and then to train a classification algorithm. The learned model is then applied to the test data. The outputs of various machine learning algorithms are compared on the basis of confusion matrix.

Confusion matrices are very useful for evaluating classifiers. The columns represent the predictions, and the rows represent the actual class. Table 1 shows the output of all three learners on the basis of confusion matrices. The output of J48 is shown in two columns. It shows that 107 instances were correctly predicted as not defective. These cases are also known as “True Positives”. The table also shows that 2 instances were correctly predicted as defective. These cases are also known as “True Negatives”. Correct predictions always lie on the diagonal of the table. On the other hand, it shows that 7 instances were predicted as not defective when they were having defects. These cases are also known as “False Positives”. Lastly, it shows 5 instances that were incorrectly predicted as defective. These cases are also known as “False Negatives”. The result is compared on the basis of confusion matrices. The can be seen that J48 and OneR performed better than Naive Bayes. The model is tested on same dataset and also by providing other test dataset of similar type of software. The performance of J48, OneR and Naïve Bayes for correctly classified instances are 90.086%, 89.2562% and 85.124% respectively for software having 121 modules.

Table 1
Results in Form of Confusion Matrix

		J48		*OneR*		*Naïve Bayes*	
		No Defect	*Defect*	*No Defect*	*Defect*	*No Defect*	*Defect*
Software 121 Modules	Non Defective	107	5	107	5	99	13
	Defective	7	2	8	1	5	4
Software 101 Modules	Non Defective	83	3	85	1	78	8
	Defective	14	1	13	2	10	5

5. CONCLUSIONS AND FUTURE WORK

The goal of this paper is to compare the performance of prediction model by using static attribute of embedded software. Three machine learning algorithms have been used for prediction purpose of two dataset. The study shows that J48 and OneR have outperformed in 10 fold cross validation with 90.086% and 89.2562 % as accuracy. In our evaluation it is found that J48 and OneR are better than Naïve Bayes learner. This paper establishes the fact that static measure can be useful indicator for defect prediction. The model could be further built in such a way so that it can predict defect for any software system .Further investigation can be done by different machine learning algorithm for improvement in the prediction accuracy.

REFERENCES

[1] M. Halstead, *Elements of Software Science.* Elsevier, (1977).

[2] T. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*,**2**(4) (1976) 308-320.

[3] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, *"Assessing Predictors of Software Defects,"* Proc. Workshop Predictive Software Models, (2004).

[4] N. Nagappan and T. Ball, "*Static Analysis Tools* as Early Indicators of Pre-Release Defect Density," Proc. Int'l. Conf. Software Eng., (2005).

[5] G. Hall and J. Munson, "Software Evolution: Code Delta and CodeChurn," *J. Systems and Software*, (2000) 111-118.

[6] Nikora and J. Munson, "Developing Fault Predictors for Evolving Software Systems," Proc. Ninth Int'l Software Metrics Symp. (METRICS '03) (2003).

[7] T. Khoshgoftaar, "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction," Proc. 12th Int'l Symp. Software Reliability Eng., (2001) 66-73.

[8] W. Tang and T. M. Khoshgoftaar, "Noise Identification with the KMeans Algorithm," Proc. Int'l Conf. Tools with Artificial Intelligence (ICTAI), (2004) 373-378.

[9] Zhong, S., Khoshgoftaar, T. M., and Seliya, N., "Analyzing Software Measurement Data with Clustering Techniques", IEEE Intelligent Systems, Special Issue on Data and Information Cleaning and Pre-processing, **2** (2004) 20-27.

[10] Fenton, N., Neil, M., "A Critique of Software Defect Prediction Models", *IEEE Transactions on Software Engineering*, **25**(5) (1999) 675-689.

[11] Khoshgoftaar, T. M., Seliya, M., "Tree-Based Software Quality Estimation Models For Fault Prediction", In Proceedings of the 8th IEEE International Conference on Software Metrics, (2002) 203-215.

[12] Quinlan, R. J., *"C4.5: Programs for Machine Learning"*, Morgan Kaufman, (1993).

[13] R. Holte, "Very Simple Classification Rules Perform Well on Most Commonly Used Data Sets," *Machine Learning*, **11** 63 (1993).

[14] Witten, I. H., and Frank E., J., "*Data Mining*: Practical Machine Learning Tools and Techniques Java Implementations", Morgan Kaufmann, (2005).

[15] *www.cs.waikato.ac.nz/~ml/weka/.*

[16] *http://promise.site.uottowa.ca/SERepository.*

[17] John, G. H. & Kohavi, R. "Wrappers for Feature Subset Selection." Artificial Intelligence, **97**(1-2) (1997) 273-324.

[18] Challagulla, V. U. B., Bastani, F. B., I-Ling Yen, Paul, *"Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques"*, 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2005 (2005) 263-270.

[19] Jiawei Han and Micheline Kamber. "*Data Mining*: Concepts and Techniques", Morgan Kaufmann, (2005).