2012 12th International Conference on Quality Software

Compressed C4.5 Models for Software Defect Prediction

Jun Wang, Beijun Shen and Yuting Chen
School of Software
Shanghai Jiao Tong University
Shanghai 200240, China
deathspeeder@sjtu.edu.cn, {shen-bj,chenyt}@cs.sjtu.edu.cn

***Abstract*—Defects in every software must be handled properly, and the number of defects directly reflects the quality of a software. In recent years, researchers have applied data mining and machine learning methods to predicting software defects. However, in their studies, the method in which the machine learning models are directly adopted may not be precise enough. Optimizing the machine learning models used in defects prediction will improve the prediction accuracy.**

In this paper, aiming at the characteristics of the metrics mined from the open source software, we proposed three new defect prediction models based on C4.5 model. The new models introduce the Spearman's rank correlation coefficient to the basis of choosing root node of the decision tree which makes the models better on defects prediction. In order to verify the effectiveness of the improved models, an experimental scheme is designed. In the experiment, we compared the prediction accuracies of the existing models and the improved models and the result showed that the improved models reduced the size of the decision tree by 49.91% on average and increased the prediction accuracy by 4.58% and 4.87% on two modules used in the experiment.

I. Introduction

Software defect prediction is a technique used to predict both the number and type of defects in a software system on the basis of some software metrics, such as source code changes, previous defects, etc. It has drawn great attentions from both researchers and software practitioners. One main reason for this is that during the software development process, a defect prediction technique allows a manager to invest resources proactively (rather than reactively) and therefore saves the time and cost for finding bugs and then fixing them. However, some software defect prediction techniques assume that the number of software defects depends on the size of the software, which is not true for most modern software systems. For this reason, software repository mining techniques can provide us the opportunities to mine these metrics and apply them in the prediction of software defects (i.e., [1]–[4]).

Usually, to predict software defects, the prediction model needs to be built on the basis of which an algorithm takes the objective software as input, and produces some predictions on defects as output. Some machine learning algorithms (i.e. naive Bayes, Bayesian networks, decision trees, and neural networks, etc.) have been adopted to software defect prediction (i.e., [4]–[7]), but they may not be precise enough. For example, Knab et al. have applied decision tree learners to predict defects on the basis of source code metrics, modification report metrics and defect report metrics and achieved 62 percentages on prediction accuracy [4] but wrongly classified nearly 40 percent of instances. An optimization of the decision tree algorithm is expected to help us achieve an accurate prediction.

In this paper we investigate some software prediction models and then aiming at the characteristics of the metrics mined from some open source software repositories, we propose three new defect prediction models based on C4.5 algorithm. C4.5 algorithm is an algorithm developed by Quinlan which is to build a decision tree from a set of training data, using the concept of information entropy [8]. The Spearman's rank correlation coefficient is introduced into the new models to make the choosing of attribute as the root node of the decision tree much more in line with the actual situation. In order to verify the effectiveness of the new models, we have designed an experiment to compare the prediction accuracies of the existing model and the improved models. Experiment on two Eclipse modules has shown that the improved models can significantly reduce the size of the decision trees and increase the prediction accuracy of these two modules by 4.58% and 4.87% respectively.

The remainder of this paper is organized as follows: Section II presents some background knowledge. In section III, we present the improved models. In section IV, we explain the experiment designed for comparison of defects prediction models. The result of the experiment and the discussion on it are presented in section V. Section VI draws the conclusions and indicates the future work.

II. Preliminaries

We introduce some concepts from C4.5 algorithm [8] and then introduce the Spearman's rank correlation coefficient.

A. The C4.5 Algorithm

Let S be a set with n data samples. Divide the sample set S into c different classes $C_i (i = 1, 2, \cdots, c)$, and every class C_i has n_i samples. Then the *entropy* of dividing S into c classes is defined,

$$E(S) \equiv -\sum_{i=1}^{c} p_i \log_2(p_i) \tag{1}$$

1550-6002/12 $26.00 © 2012 IEEE
DOI 10.1109/QSIC.2012.19

where $p_i = \frac{n_i}{n}$ is the probability of a sample in S that belongs to class C_i. Entropy characterizes the purity of a sample set.

Let the set of all the different values of attribute A be X_A, and S_v be the sub-set of samples with value v on attribute A, that is $S_v = \{s \in S | A(s) = v\}$. After an attribute A is chosen to be the root of a sub-tree, the entropy of classifying S_v is defined,

$$E(S, A) \equiv \sum_{v \in X_A} \frac{|S_v|}{|S|} E(S_v) \tag{2}$$

where $E(S_v)$ is the entropy of dividing samples in set S_v into c classes. The *information gain* of attribute A to the sample set S is,

$$Gain(S, A) \equiv E(S) - E(S, A) \tag{3}$$

C4.5 uses gain ratio as the basis of choosing attributes as the root of a sub-tree when the decision tree is constructed. The *gain ratio* is,

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInfo(S, A)} \tag{4}$$

where, *split information* is,

$$SplitInfo(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \tag{5}$$

where S_i is c sample sub-sets by dividing S using c values of attribute A. Split information is the entropy of S on all values of attribute A.

B. Spearman's Rank Correlation Coefficient

The Spearman's rank correlation coefficient is used to study the relationship between two variables and to quantify the degree of correlation of two columns of data which is defined as the Pearson correlation coefficient between the ranked variables. It is calculated during the construction of the decision tree by choosing an attribute with values from all instances as X_i letting defects number of every instance be Y_i and then converting the n raw scores X_i, Y_i to ranks x_i, y_i. The Spearman correlation coefficient ρ is computed:

$$\rho \equiv \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}} \tag{6}$$

Tied values are assigned a rank equal to the average of their positions in the ascending order of the values. For example, the third and forth values are equal, and the corresponding rank would be $\frac{3+4}{2} = 3.5$.

III. Compressed C4.5 Models

A. Compressed C4.5 Model I

Our first model multiplies the Spearman's correlation coefficient and the gain ratio, and then use the product to replace the original gain ratio for selecting test attributes. The Spearman's rank correlation coefficient can be positive or negative.

The new gain ratio in compressed C4.5 model I is defined,

$$GRMod1(S, A) \equiv GainRatio(S, A) \times \rho \tag{7}$$

We introduce the Spearman's rank correlation coefficient by multiplying it with the gain ratio. The reason is that both the gain ratio and the Spearman's rank correlation coefficient can represent the relationship between the metrics and the defects, while the weights of them are not clear. So we would adopt multiplication instead of using addition.

B. Compressed C4.5 Model II

We propose our second compressed C4.5 model, in which the Spearman's rank correlation coefficients are sorted in ascending order. Let the ranks of the coefficients be $Rank(\rho)$. Compressed C4.5 model II uses $Rank(\rho)$ instead of ρ as multiplier. This method ignores the value of the Spearman's rank correlation coefficient, and takes the importance of every attribute into account.

The gain ratio for compressed C4.5 model II is defined,

$$GRMod2(S, A) \equiv GainRatio(S, A) \times Rank(\rho) \tag{8}$$

C. Compressed C4.5 Model III

Compressed C4.5 model III introduces the Spearman's rank correlation coefficient into the process of calculating gain ratio in order to balance the fluctuation of gain ratio in different metrics. The main idea of C4.5 algorithm is to choose the attribute with the biggest information gain (in C4.5, the basis of choosing root node is gain ratio which is calculated by dividing information gain by split information, however, split information is introduced only to solve the multi-valued bias problem in ID3 [9], the ancestor of C4.5.) as the root node of a sub-tree in which the information gain is the compression of the entropy expectation caused by assigning the value of attribute A. Along with the information gain, we adopt Spearman's rank correlation coefficient as the basis too. The first step of the compressed C4.5 model III is to reduce the importance of information gain. Therefor, we re-define $E(S, A)$, and the entropy of classifying S_v by attribute A is,

$$E'(S, A) \equiv \sum_{v \in X_A} (\frac{|S_v|}{|S|} + \rho) E(S_v) \tag{9}$$

Since the split information is used to reduce the influence of the multi-valued bias problem, we keep it in step two of the compressed C4.5 model III, but make it more significant in calculating gain ratio. Split information in formula (5) is re-defined,

$$SplitInfo'(S,A) \equiv -\sum_{i=1}^{c}(\frac{|S_i|}{|S|}+\rho)\log_2\frac{|S_i|}{|S|} \quad (10)$$

The modified gain ratio is calculated,

$$GRMod3(S,A) \equiv \frac{Gain'(S,A)}{SplitInfo'(S,A)} = \frac{E(S)-E'(S,A)}{SplitInfo'(S,A)} \quad (11)$$

IV. Experiment Preparation

Our experiment is designed to evaluate the new models for prediction of software defects.The predict process is divided into three steps: mining metrics from software repository including source code metrics, change metrics and defect history metrics; comparing existing models by inputting formatted metrics into several existing machine learning models and outputting prediction accuracy; verifying the effectiveness of the improved models by comparing with the existing models. Fig. 1 shows the work-flow of the experiment, where *Understand* tool[1] is used to extract source code metrics from source codes; *FilterMetrics* tool, *LinkBugs* tool and *GenerateWekaData* tool are utilities developed by the authors to filter metrics from *Understand* tool and to mine change metrics and defect metrics and format metrics according to *ARFF* format[2] respectively.

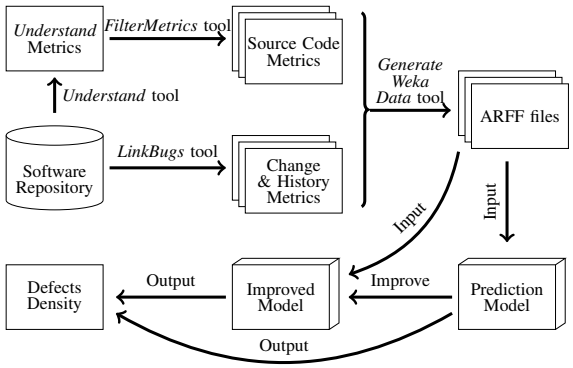

Fig. 1. Defect Prediction Process

Along with our three new models, three existing machine learning models are selected to predict defects: naive Bayes, C4.5 and CART. For every model, the size of the model and the classification accuracy of training data set and testing data set are recorded. The inputs of models include 11 source code metrics, NR metric and defect history metric. The output of models is the classification of the source code files into ten levels by their defects number. In this paper, a tool named *WEKA*[3] is used to predict defects.

[1]http://www.scitools.com/index.php

[2]An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. http://www.cs.waikato.ac.nz/~ml/weka/arff.html

[3]http://www.cs.waikato.ac.nz/ml/weka/

Two modules from Eclipse[4] are selected as the subjects of our experiment: *org.eclipse.jdt.core* and *org.eclipse.pde.ui*. There are nine versions are studied in our experiment.

TABLE I
Versions of Eclipse

Version	Release Date	Classes in jdt.core	Classes in pde.ui
2.0	June 27, 2002	811	538
2.1	March 27, 2003	842	720
3.0	June 24, 2004	940	793
3.1	June 27, 2005	1045	921
3.2	June 6, 2006	1112	1200
3.3	June 21, 2007	1132	1525
3.4	June 13, 2008	1181	1671
3.5	May 27, 2009	1189	1499
3.6	June 3, 2010	1181	1573

The CVS repository of Eclipse[5] is downloaded at March 29, 2011. And the bug reports[6] of *org.eclipse.jdt.core* are exported at April 8, 2011 and the bug reports of *org.eclipse.pde.ui* are exported at April 21, 2011.

V. Result and Analysis

In our experiment, three existing models are compared with each other. We can compare these three existing models by analyzing their confusion matrixs (when classifying instances, a confusion matrix will be generated by *WEKA* for every model.) and the classification accuracy in table II. Note that CART model does not build a valid decision tree on *org.eclise.pde.ui* (CART model seems to simplify a decision tree as much as possible so that it can classify all instances into Level 1, Level 2 or Level 10. But in practice, this is not the case.). By comparison, we conclude that C4.5 is the best while naive Bayes ranks second and CART follows.

TABLE II
Comparison of Existing Models

		Naive Bayes	CART	C4.5
jdt.core	Training Set (%)	48.5973	55.5853	71.8057
	Testing Set (%)	75.3181	83.1213	80.3223
pde.ui	Training Set (%)	61.7496	-	76.354
	Testing Set (%)	75.8003	-	83.2266

We compare our three compressed C4.5 models with C4.5. The prediction accuracies of compressed C4.5 model I and II are close to each other both on training set and testing set which indicates that ρ and $Rank(\rho)$ have the same contribution to prediction accuracy. Compared with C4.5 model, our compressed C4.5 model III improves the prediction accuracy on testing data sets by 4.5802 percentages and 4.8656 percentages on modules *org.eclipse.jdt.core* and *org.eclipse.pde.ui* respectively.

[4]http://www.eclipse.org

[5]http://archive.eclipse.org/arch/

[6]https://bugs.eclipse.org/bugs/

TABLE III
COMPARISON OF COMPRESSED C4.5 MODELS AND C4.5 MODEL

		C4.5	CCM I	CCM II	CCM III
jdt.core	Training set (%)	71.8057	65.5828	67.8526	57.1538
	Testing set (%)	80.3223	80.5768	80.4071	84.9025
pde.ui	Training set (%)	76.354	74.5633	72.3323	67.4886
	Testing set (%)	83.2266	83.2907	85.5314	88.0922

The size of a generated decision tree represented by leaf nodes number and non-leaf nodes number is the key factor that determines the running time of classifying defects and therefore reflects the efficiency of the models. In our experiment, CART model, C4.5 model and three Compressed C4.5 Models generate decision trees. The sizes of these trees are shown in table IV.

TABLE IV
SIZE OF DECISION TREES

Model	*jdt.core*		*pde.ui*	
	non-leaves	leaves	non-leaves	leaves
CART	33	17	0	1
C4.5	1729	865	831	416
CCM I	1093	520	637	319
CCM II	1401	701	507	254
CCM III	193	97	33	17

The sizes of compressed C4.5 models are smaller than C4.5, however, the prediction accuracies are higher than C4.5 model's. From table III we know that these three compressed C4.5 models are more accurate than C4.5 on classifying testing set data. This means the effectiveness of the compressed C4.5 models is verified.

VI. CONCLUSION AND FUTURE WORK

The goal of this paper is to improve the software defect prediction accuracy. To achieve this, we designed a research scheme, and finally proposed Compressed C4.5 Models. By experiment, we conclude that:

- When predicting defects on two modules from Eclipse, the accuracy of C4.5 model is better than that of naive Bayes model both on training set and testing set;
- When applying CART model to software with few defects may easily generate a simple decision tree which classifies all instances into one or two levels and gives meaningless prediction;
- The prediction accuracies of Compressed C4.5 Model I and II are close to each other, which means the introduction of Spearman's rank correlation coefficient ρ and the introduction of $Rank(\rho)$ have generally the same effects;
- The Compressed C4.5 Model III decreases the size of decision tree and improves the prediction accuracy on testing data set by 4.58% and 4.87% on modules *org.eclipse.jdt.core* and *org.eclipse.pde.ui* compared with C4.5 model.

The disadvantage of our experiment is the choosing of modules. Both of the two modules studied in this paper have few defects and the distribution of defects tends to one or two defects in one file. This generates a little interference to the comparison of compressed C4.5 models and existing models.

The future work can be applying Compressed C4.5 Models to more open source software to overcome the problem of few defects in a file. We can also focus on mining metrics based on the characteristics of open source software to achieve higher defect prediction accuracy.

VII. ACKNOWLEDGMENTS

We would like to thank Hongyu Zhang (Tsinghua University) for reviewing earlier drafts of this paper and giving several valuable comments. In addition Yuting Chen is also supported by the NSFC Grant (No. 61100051).

REFERENCES

[1] M. Takahashi and Y. Kamayachi, "An empirical study of a model for program error prediction," *Software Engineering, IEEE Transactions on*, vol. 15, no. 1, pp. 82 –86, Jan. 1989.

[2] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," *Software Maintenance, IEEE International Conference on*, vol. 0, p. 23, 2003.

[3] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 563–572. [Online]. Available: http://portal.acm.org/citation.cfm?id=998675.999460

[4] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 119–125. [Online]. Available: http://doi.acm.org/10.1145/1137983.1138012

[5] F. Xing, P. Guo, and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," *Software Reliability Engineering, International Symposium on*, vol. 0, pp. 213–222, 2005.

[6] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, and R. Mishra, "Predicting software defects in varying development lifecycles using bayesian nets," *Information and Software Technology*, vol. 49, no. 1, pp. 32–43, 2007, most Cited Journal Articles in Software Engineering - 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0B-4M4CN0C-3/2/1fde90c721bf40bca6a3c91128794148

[7] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. D. Backer, and R. Haesen, "Mining software repositories for comprehensible software fault prediction models," *Journal of Systems and Software*, vol. 81, no. 5, pp. 823–839, 2008, software Process and Product Measurement. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0N-4PHSC3R-2/2/129ce670cc51b090011f548066bb5711

[8] J. R. Quinlan, *C4.5: Programs for machine learning*. The Morgan Kaufmann Series in Machine Learning, San Mateo, CA: Morgan Kaufmann, 1993.

[9] ——, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986, 10.1007/BF00116251. [Online]. Available: http://dx.doi.org/10.1007/BF00116251