Towards a Better Understanding of Static Code Attributes for Defect Prediction

Muhammed Maruf Öztürk and Ahmet Zengin

Department of
Computer Engineering
Faculty of Computer
and Information Sciences
Sakarya, Turkey 54187
Email: muhammedozturk@sakarya.edu.tr, azengin@sakarya.edu.tr

***Abstract*—Defect prediction requires intensive effort and includes operations which are focused on reducing the cost of software development. These operations involving the use of machine learning algorithms could produce wrong results originated from skewed or missing data. In order to increase the success rate of predictors, defect data sets are either pruned or duplicated. To address this problem, we observe the effects of the derivation of low level metrics using statistical methods in prediction performance. The performance of predictions are evaluated using 10-fold cross-validation on each data set. Experimental results obtained by using 15 data sets show that naive Bayes classifier improved values of Area Under the Curve (AUC) with the rate of 0,1 in average.**

Keywords–Defect prediction; Low level metrics; Metric derivation

I. INTRODUCTION

Properties of software codes vary depending on development processes, functional goals, and development constraints [1][2]. In order to comprehend this variety in depth, we should examine software behaviours and tendencies, in which versions of software changes, along with specific software metric models [3][4][5]. Developers need metric tables to advance their understanding of how software changes across it's newer versions [6]. The standards, which were developed by McCabe and Halstead, are widely used ones while generating software metric tables [7][8]. These standards do not require an in-depth analysis in the structure of codes; however, the model presented by McCabe is more suitable than the others in the design level [9].

Metric tables of software components have a property that indicates the defect-proneness of software. Thanks to this property, a defect prediction can be conducted on the basis of binary classification. However, each data set has potential problems caused by noise or repeated data points that this issue reduces the success of prediction [10]. One of the mostly known problems in defect prediction is class-imbalanced data sets. In such cases, defects are generally intensified on specific parts of software so that the reliability of the prediction is not as desired. In this respect, it is rather difficult to determine a general bias about the software modules [11]. We have two ways to cope with class-imbalance: undersampling, and oversampling. Although undersampling is an efficient method, it causes the hiding of useful data. Likewise, oversampling may cause an unrealistic increase in the success of learning [12], [13][14].

In this study, we investigate metric derivation methods and its effects on defect prediction. Defect data sets consist of 15 data sets including NASA metrics data program (NASA MDP) and Softlab. The common feature of these data sets is that they were generated using McCabe & Halstead metrics. After adding some metrics to the data sets such as character count (cCount) and class size (cS), the variation recorded on the performance parameters such as accuracy and AUC was observed. Moreover, the relationship between low level and other metrics was strived for the exploration. The results obtained from the experiment show that the proposed method increased the success of prediction on 15 data sets in general.

The rest of the paper is organized as follows. Section 2 provides a background describing the relevant terms and approaches. Related works are mentioned in Section 3 and this section also discusses the distinctive aspect of our work when it is compared to similar works. The proposed approach is in Section 4. The results, we have obtained so far, are explained in Section 5. The novelty and the contribution of the paper are presented in Section 6.

II. BACKGROUND

Two types of learning are used in defect prediction: supervised and unsupervised learning. Supervised learning is the most commonly used technique [15][16]. It includes SMV, ANN, decision trees etc.. Although unsupervised learning does not requie a labelling on training data, supervised learning analyzes the data only labeled. Researchers generally want to see which supervised learning techniques are suitable for defect data sets to be predicted. Learning techniques also called predictors are to predict defect-proneness of modules for the next version of software.

Properties of code are prepared using a particular measuring standard namely metrics [17]. Even though researches published in last five years are focused on process metrics that yielded promising results [18][19], code metrics have some gaps that are worthy to explore [20][21]. One of them is

 ISBN: 978-1-61208-438-1

the reliability of defect data sets. As the defect data sets are generally prepared by combining all related developer's comments, they may have missing or noisy data points. In order to cope with this problem, the data are re-sampled or reduced by using particular preprocessing techniques. SMOTE is one of the widely used sampling strategy for defect prediction [22]. However it is sensible to combine a sample reduction method with an over-sampling technique [23].

III. Related Works

One of the leading fields to explore static code properties is machine learning. Menzies et al.'s work, published in 2007, is a much cited work in this field [24]. This work stressed that the type of the metric set is more important than the selected predictor in the success of precision. The promising result of this work is that Bayes classifier showed better performance than J48 with the rate of 71%. Likewise, we have taken naive Bayes among performance measurement algorithms.

The framework developed by Song et al. showed that every data set may not be suitable for every prediction model [25]. This especially changes depending on the type of the data set. Using this result we can say that every learning method is not suitable for every defect data set. A two-phase prediction model was developed in Kim and Kim's work [26], the reports considered as eligible were eliminated in the first phase and the prediction accuracy was obtained as 70%. This work also proved the importance of preprocessing in defect data sets.

One of the works which used NASA MDP data sets is Gray et al.'s work [27]. This work, especially focused on data cleansing, removed some properties of the metrics obtained from 13 data sets to be suitable for binary classification. Missing values were assigned to zero. The first of these results is that used data sets should be extended. Thus, we can determine whether the repeated data points are in general. Second, low level metrics should be used to detect repeated data. Third is the presence of the issues caused by the repeated data.

The studies above all use static code metrics to build a proper prediction model. However, the most relevant work to ours is Gray et al's work which is explained in the preceding paragraph. This work and our work have similarities: they use the same experimental data sets and have claimed the importance of the use of low level metrics.

IV. Proposed Approach

NASA MDP and SOFTLAB data sets consisting of metric values that range from 21 to 40. Tests including ANOVA, t-test, and chi-square unveiled the relationship between character count and LOC (number of lines of code) as below:

$$cCount \sim= lCode * 30. \quad (1)$$

Lorenz and Kidd presented object-oriented metric tables [28]. The main reason why object-oriented metrics are widely used is that such metrics are the best indicator of system reliability at design level [29]. cS is also a low level metric but it is not available in the data sets of NASA MDP and SOFTLAB (CS=total number of operations+ number the attributes) [28]. In order to explore the relationships of defects, Linear and Multiple Regression analyses were used. If the binary-dependencies of the metrics are desired to be extracted, Linear Regression is a convenient method. This method assumes that relations between variables can explained through a linear model [30][31]. Also our approach is to unveil the linear relationships between defect data set values. Given a dependent variable as y-f(x), the assumption having independent variable(x) emerges as y=ax+b. This is called as Curve Fitting [32]. The aim of this process is to find the most suitable a and b variables for f(x). As the value of R^2 closes to the one, a rather suitable curve is obtained. If e_i is regarded as error term, the formula is $e_i = y_{i,measured} - y_{i,model}$. We aim at minimizing S_r in the formula of $S_r = \sum_i^n (e_i)^2$. Linear and nonlinear distribution samples are seen in Figure 1 and Figure 2. The more function curve fits the real data, including large samples up to the count of 17186, the more accurate model is obtained.

If f(x) linear function is to be expressed with more than one independent variable, Multiple Linear Regression is used. For two variables, we have:

$$f(x) = b + a_0 x_1 + a_1 x_2 \quad (2)$$

Our approach can be summarized as follows: 1. The extraction

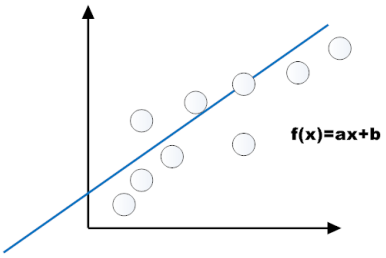

Figure 1. Curve Fitting (Linear).

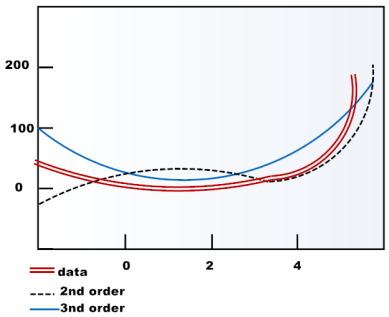

Figure 2. Nonlinear distribution.

of characteristic properties of software defect data sets and exploring required models. 2. The derivation of new low level metrics regarding defect data sets and adding to the data sets. 3. The comparison of data sets including low level metrics with preceding situation.

V. Results

cCount and cS are obtained by using the relationships of data. To test the use of low level metric, we have used 15 data sets including NASA MDP and SOFTLAB. These data sets belong to software projects developed using C, C++, and Java programming languages. The data sets have some metrics range from 21 to 40 including large samples up to the count

 ISBN: 978-1-61208-438-1

of 17186. Data sets, having skewed samples at a certain ratio, comprise 25 missing values. The experimental study has been tested by using the framework we have been developing. This framework is able to generate over the given codes and drives defect prediction with defect prediction algorithms.

The regression analysis results between class size and the other three metrics are illustrated in Figure 3. According to these results, a formula $y = 0,5244x - 14,679, R^2 = 0,9453$ has been found using cS-comment_loc. R^2 is close to one that verifies the consistency of the equation. When it comes to the relation of CS-Executable_loc, an equation is obtained as $y = 9,5518ln(x) - 34,278, R^2 = 0,523$. On the other hand, the effects of Code_and_comment_loc and unique_operand are close to the zero.

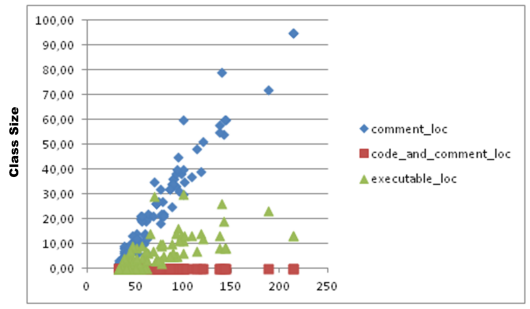

Figure 3. Relations between Class Size and other metrics.

Before the prediction, definitions including defect-prone or not-defect-prone property of software modules should be prepared. If a module does not include any defect and rightly biased then it is labeled as TN. In such cases if the module is wrongly biased then it is labeled as FP. If any module including defects is wrongly biased, labeled as FN. Last, if the bias and the prediction is the same for a defect-prone module, it is labeled as TP. Using these parameters, a table confusion matrix is organized as in Table 1. The success of the proposed method is compared to the others by benefiting the formulas defined in Listing 3.

TABLE I. CONFUSION MATRIX

		PREDICTED	
		nfp	fp
REAL	nfp	TN	FP
	fp	FN	TP

$$Precision = TP/(TP + FP), Recall = TP/(TP + FN) \quad (3)$$

$$TPR = (TP/TP+FN)*100\%, FPR = (FP/FP+TN)*100\% \quad (4)$$

$$Accuracy = TP + TN/(TP + FP + FN + TN) \quad (5)$$

Four classifiers including naive Bayes, Bayes, Random Forest and J48 have been used for the experiment. 10 fold cross-validation has been used along with 10 iteration. One of the evaluation parameters is AUC that is the indicator of the probability of false alarm versus the probability of detection.

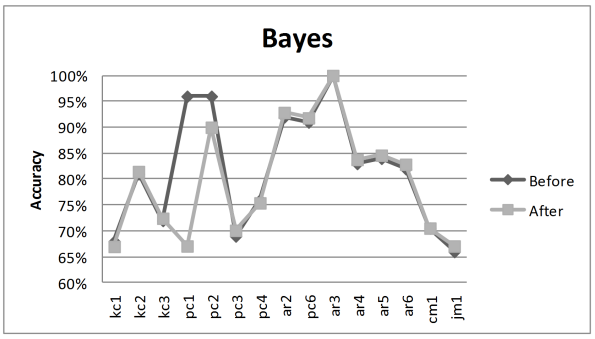

Figure 4. Accuracy values of Bayes.

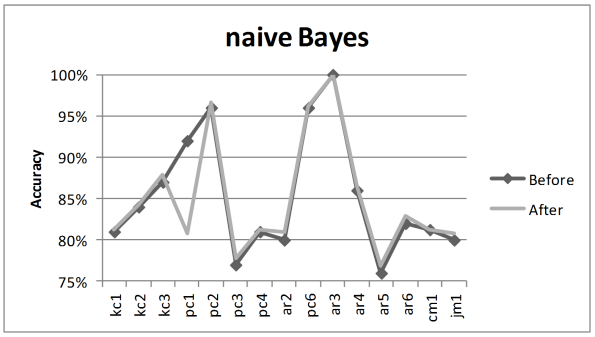

Figure 5. Accuracy values of naiveBayes.

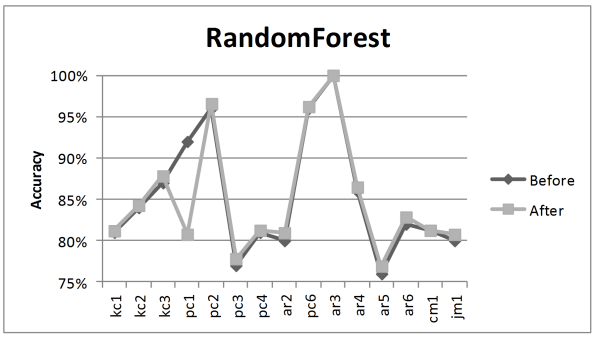

Figure 6. Accuracy values of RandomForest.

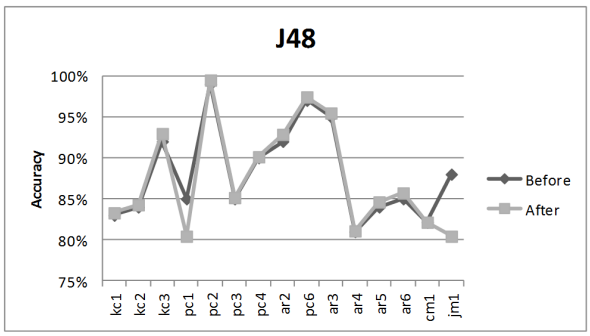

Figure 7. Accuracy values of J48.

 ISBN: 978-1-61208-438-1

On 15 data sets naive Bayes increased the AUC values in general with the rate of 0.1. Figure 4-Figure 7 show some results that explain the successes of the predictors both before the use of low level metrics and after. First, naive Bayes and RandomForest have increased the success of the prediction in all data sets except for the pc1. Second, Bayes has produced worse results than the other algorithms. Last, while the success of J48 on jm1 data set has been reduced, successes of the other algorithms have been increased. Figure 8 and 9 show the AUC values that measures testing reliability. Having low level metrics, remarkable improvement has been achieved on testing set as seen in Figure 9.

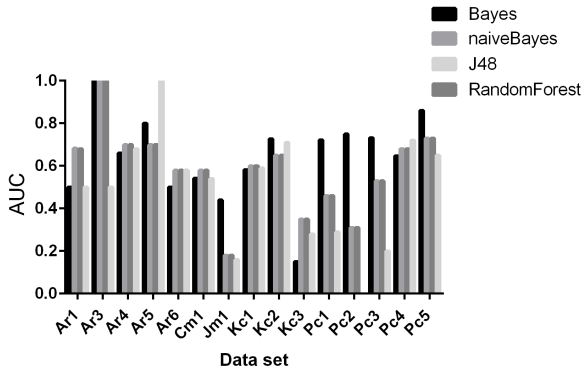

Figure 8. AUC values before preprocessing.

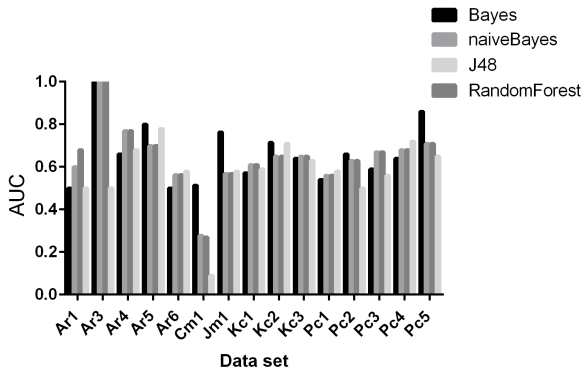

Figure 9. AUC values after preprocessing.

VI. Conclusion

Here, we want to discuss the use of low level metrics in defect prediction and present our approach based on least-square using metric relationships. Thus, extracting mathematical models of the metrics has raised some bias. The first results showed that the use of low level metrics has achieved an unprecedented success in NASA MDP and SOFTLAB data sets.

Low level metrics help us to better understand the details of software systems. However, the success of learning algorithms may not be improved with increasing count of the metrics at steady state. Furthermore, skewness of data sets should be fixed by exposing all data to a preprocessing. To gain better insight, we should develop a preprocessing algorithm which uses some tests such as ANOVA, t-test, and chi-square. In addition, the software, in which data sets are extracted, are coded by using various languages including C, C++, and Java. Therefore, the types of coding should be considered during the extension of metric tables.

The contributions of this paper can be summarized as follows: (i) proposed method for deriving low level metrics could shed new light to researchers in terms of valuable data sets that are not publicly available. (ii) metric relations change depending on the type of coding as in the range of ar3-pc1 coded with C programming language. (iii) using few samples does not produce consistent results such as ar3 data set having 64 samples.

Our current approach has been merely tried on NASA MDP and SOFTLAB data sets. Therefore, one of the purposes which will extend this study is the testing of the approach on other publicly available data sets. An important issue that could arise during the experiment is the ambiguous effects of repeated data points. In this respect, our future work aims to investigate the contribution of the low level metric in the detection of repeated data.

Acknowledgment

The authors would like to thank Tim Menzies who is one of the co-founders of tera-Promise.

References

[1] I. Herraiz, D. Rodriguez, and R. Harrison, “On the statistical distribution of object-oriented system properties,” in Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on. IEEE, 2012, pp. 56–62.

[2] J. Highsmith, Adaptive software development: a collaborative approach to managing complex systems. Addison-Wesley, 2013.

[3] N. Fenton and J. Bieman, Software metrics: a rigorous and practical approach. CRC Press, 2014.

[4] R. J. Leach, Software Reuse: Methods, Models, Costs. AfterMath, 2012.

[5] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, “Choosing software metrics for defect prediction: an investigation on feature selection techniques,” Software: Practice and Experience, vol. 41, no. 5, 2011, pp. 579–606.

[6] L. Putnam and W. Myers, Five core metrics: the intelligence behind successful software management. Addison-Wesley, 2013.

[7] T. J. McCabe, “A complexity measure,” Software Engineering, IEEE Transactions on, no. 4, 1976, pp. 308–320.

[8] M. Halstead, “Potential impacts of software science on software life cycle management,” Purdue University Library, 1977.

[9] T. J. McCabe and C. W. Butler, “Design complexity measurement and testing,” Communications of the ACM, vol. 32, no. 12, 1989, pp. 1415–1425.

[10] L. Pelayo and S. Dick, “Applying novel resampling strategies to software defect prediction,” in Fuzzy Information Processing Society, 2007. NAFIPS’07. Annual Meeting of the North American. IEEE, 2007, pp. 69–72.

[11] G. M. Weiss, “Mining with rarity: a unifying framework,” ACM SIGKDD Explorations Newsletter, vol. 6, no. 1, 2004, pp. 7–19.

[12] X.-Y. Liu, J. Wu, and Z.-H. Zhou, “Exploratory undersampling for class-imbalance learning,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 39, no. 2, 2009, pp. 539–550.

[13] T. M. Khoshgoftaar and K. Gao, “Feature selection with imbalanced data for software defect prediction,” in Machine Learning and Applications, 2009. ICMLA’09. International Conference on. IEEE, 2009, pp. 235–240.

 ISBN: 978-1-61208-438-1

[14] T. M. Khoshgoftaar, K. Gao, and N. Seliya, “Attribute selection and imbalanced data: Problems in software defect prediction.” in ICTAI (1), 2010, pp. 137–144.

[15] H. Lu, B. Cukic, and M. Culp, “A semi-supervised approach to software defect prediction,” in Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual. IEEE, 2014, pp. 416–425.

[16] H. Lu, E. Kocaguneli, and B. Cukic, “Defect prediction between software versions with active learning and dimensionality reduction,” in Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on. IEEE, 2014, pp. 312–322.

[17] C. Kaner et al., “Software engineering metrics: What do they measure and how do we know?” in In METRICS 2004. IEEE CS. Citeseer, 2004.

[18] F. Rahman and P. Devanbu, “How, and why, process metrics are better,” in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE ’13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 432–441. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486846

[19] I. S. Wiese, F. R. Côgo, R. Ré, I. Steinmacher, and M. A. Gerosa, “Social metrics included in prediction models on software engineering: A mapping study,” in Proceedings of the 10th International Conference on Predictive Models in Software Engineering, ser. PROMISE ’14. New York, NY, USA: ACM, 2014, pp. 72–81. [Online]. Available: http://doi.acm.org/10.1145/2639490.2639505

[20] P. Oliveira, M. T. Valente, and F. Paim Lima, “Extracting relative thresholds for source code metrics,” in Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014, pp. 254–263.

[21] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, “Towards building a universal defect prediction model,” in Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014, pp. 182–191.

[22] R. Pears, J. Finlay, and A. M. Connor, “Synthetic minority oversampling technique (smote) for predicting software build outcomes,” arXiv preprint arXiv:1407.2330, 2014.

[23] L. Chen, B. Fang, Z. Shang, and Y. Tang, “Negative samples reduction in cross-company software defects prediction,” Information and Software Technology, vol. 62, 2015, pp. 67–77.

[24] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” Software Engineering, IEEE Transactions on, vol. 33, no. 1, 2007, pp. 2–13.

[25] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, “A general software defect-proneness prediction framework,” Software Engineering, IEEE Transactions on, vol. 37, no. 3, 2011, pp. 356–370.

[26] D. Kim, Y. Tao, S. Kim, and A. Zeller, “Where should we fix this bug? a two-phase recommendation model,” Software Engineering, IEEE Transactions on, vol. 39, no. 11, 2013, pp. 1597–1610.

[27] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, “Reflections on the nasa mdp data sets,” Software, IET, vol. 6, no. 6, 2012, pp. 549–558.

[28] M. Lorenz and J. Kidd, Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., 1994.

[29] Y. Suresh, J. Pati, and S. K. Rath, “Effectiveness of software metrics for object-oriented system,” Procedia Technology, vol. 6, 2012, pp. 420–427.

[30] E. Kocaguneli, T. Menzies, and J. W. Keung, “On the value of ensemble effort estimation,” Software Engineering, IEEE Transactions on, vol. 38, no. 6, 2012, pp. 1403–1416.

[31] B. Kitchenham and E. Mendes, “Why comparative effort prediction studies may be invalid,” in Proceedings of the 5th international Conference on Predictor Models in Software Engineering. ACM, 2009, p. 4.

[32] R. A. Johnson, I. Miller, and J. E. Freund, Probability and statistics for engineers. Prentice-Hall, 2011.

 ISBN: 978-1-61208-438-1