Information and Software Technology 000 (2017) 1–13

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Which type of metrics are useful to deal with class imbalance in software defect prediction?

Muhammed Maruf Öztürk

Department of Computer Engineering, Engineering Faculty, Suleyman Demirel University, Isparta, Turkey

ARTICLE INFO

Article history:
Received 25 May 2016
Revised 29 June 2017
Accepted 4 July 2017
Available online xxx

Keywords:
Static code metrics
Process metrics
Class imbalance
Defect prediction

ABSTRACT

Context: There are various ways to cope with class imbalance problem which is one of the main issues of software defect prediction. Sampling algorithms are implemented on both industrial and open-source software defect prediction data sets by practitioners to wipe out imbalanced data points. Sampling algorithms, up-to-date, have been employed either static or process code metrics.
Objective: In this study, sampling algorithms including Virtual, SMOTE, and HSDD (hybrid sampling for defect data sets) are explored using static code and quality metrics together. Our goal is not only to lead practitioners to decide the type of the metrics in defect prediction but also provide useful information for developers to design less defective software projects.
Method: We ran sampling experiments with three sampling algorithms on ten data sets (from GitHub). Feature selection is applied on large features of the data sets. Using five classifiers, the performance of the data sets after sampling is compared with initial data sets. Regression analyzes are implemented on quality metrics to find the most influential metrics for detecting defect proneness.
Results: Regardless of the type of the sampling, prediction performances are similar. Quality metrics surpassed static code metrics with respect to training times and prediction accuracies.
Conclusion: Using quality metrics yields better prediction results rather than static code metrics in imbalanced data sets. As the count of project cloning increases, the number of defects decreases. Thus, approaches, related to the class imbalance, should be evaluated not only in terms of static code metrics but also for quality metrics.

1. Introduction

Verification and validation of a software is critical to plan allocated budget to development [1]. However, software systems require many efforts to be validated. In order to validate software systems, software defect prediction (SDP) is a way to check whether they are validated. Prediction of defective parts of a software facilitates planning of the software development budget in SDP. SDP is twofold. First is the process which is conducted using historical data to predict future defects. The second is the prediction of the remaining defects. SDP is conducted considering some properties of defect data sets of software systems.

Some researchers strongly advise the use of static code metrics [2–5] in SDP. Prediction models employing these metrics include statistical methods and machine learning. Here, the main objective is to find the metrics of which is far more correlated with defect-proneness than the others. Fuzzy rules help practitioners to extract new biases from static code metrics [6]. In recent years, it has shown that process metrics giving information about software processes yield better prediction results than static code metrics [7–12]. Therefore, static code metrics should be compared with process metrics in every SDP issue.

One of the issues of SDP is class imbalance and it has also been the focus of SDP in recent years. It is the main factor of causing poor performance on SDP data sets. This occurs when defective or not defective samples outnumber the others. Class imbalance learning aims to deal with this problem. It presents either data or algorithm focused solutions. Feature selection, ensemble methods, and sampling techniques are applied on data sets to handle with class imbalance. In order to overcome this problem, some methods have been developed so far [13–15], but there is not any comparison work in literature in terms of static code and process metrics. For instance, Wang et al. investigated class imbalance learning methods on the data sets having static code metrics [13]. Likewise, there are some works performed with publicly available data sets having static code metrics [16–18]. On the other hand, there are few works comparing static code and process metrics with regard to the predictive performance. One of them is Rahman and

E-mail address: muhammedozturk@sdu.edu.tr

http://dx.doi.org/10.1016/j.infsof.2017.07.004
0950-5849/

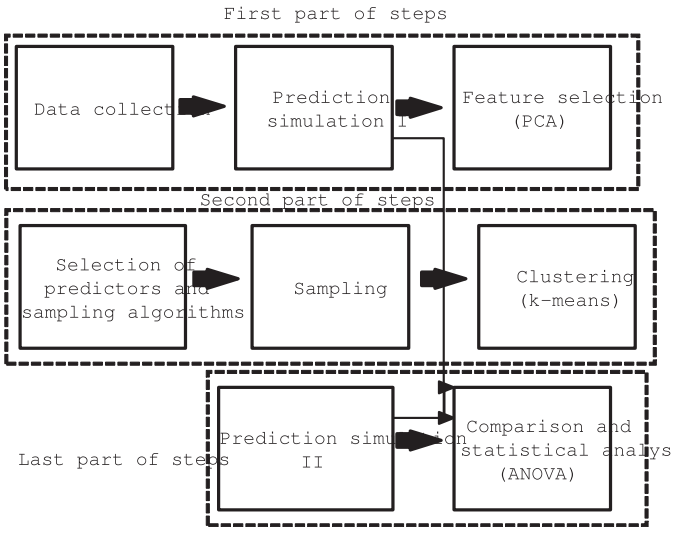

Fig. 1. Main steps in our work.

Devanbu's work [12] in which process and static code metrics were compared with 12 real-world data sets. They conducted the experiment to reveal which type of metrics is likely to evolve in the changing distribution of defects. However, they did not present any findings about the strength of the type of metrics in handling with class imbalance.

This work investigates which type of metrics is useful in class imbalance learning methods. We focus on these research questions: Which type of process metrics are linearly correlated with defect count? Are static code metrics preferable in predicting defects for every experimental condition including class imbalance? Are sampling methods quite different in terms of prediction results and which sampling method is preferable in coping with class imbalance? Do clustering give tips for researchers during defect prediction simulation?

In order to answer the questions, we discuss the results of sampling methods used in class imbalance problem with regard to static code and quality metrics. The performance of the classifiers including LIBSVM, Bayes, naive bayes, random forest, and J48 were recorded before the pre-processing in the experiment. We then completed feature selection, clustering and sampling using HSDD [19], SMOTE [20], and Virtual [21]. Finally, the performance of the classifiers were re-measured and recorded to compare with the pre-processing manner. Experimental data sets have static code and process (quality) metrics. Main steps of our study are seen in Fig. 1. The study consists of three phases. Each of the phases is detailed in the following subsections. The contribution and novelty of the paper can be summarized as follows:

1. Correlation between quality metrics and the number of defects is investigated.
2. Static code metrics and process metrics are compared in terms of area under the curve (AUC) on both industrial and open-source projects using four predictors.
3. It is investigated which type of metric yields remarkable success if class imbalance is addressed with sampling algorithms.
4. Common sampling algorithms are compared with both static code and quality metrics to determine the churn of prediction success.
5. Clustering structures of defect prediction data sets are explored to detect whether clusters of defect prediction data sets give tips for performance parameters such as AUC and g-mean.

The rest of the paper is organized as follows: The works correspond to ours are presented in Section 2 and the difference of our work from these works is also stressed in this section. The notions and the definitions are presented in Section 3. Section 4 details the method. Experimental design is in Section 5. Last, we conclude in Section 6 and discuss future works.

2. Literature review

The main purpose of this section is to give information about the works related to the process metrics. In addition, we intend to show the need of a class imbalance comparison work including static code and process metrics together.

Rahman and Devanbu [12] compared static code metrics with process metrics by investigating 12 project data sets and they depicted that static code metrics are stagnant in prediction performance. Performance parameters AUC and *F*-measure were used in the comparison. The experimental data retrieved from GitHub include 14 different process metrics. Moreover, they gathered 37 static code metrics for each project. The experiment, performed using four classifiers including logistic regression, LIBSVM, J48, and naive bayes, showed that prediction performance does not evolve using static code metrics. However, this study did not evaluated class imbalance in terms of static code and process metrics together.

Kaur et al. compared static code metrics with process metrics in mobile applications. The prediction performed using process metrics yielded better results than static code metrics [7]. Madeyski ve Jureczko [8] investigated process metrics in industrial and open-source projects. The main objective of this work is to find the correlation between process metrics and defect-proneness. Prediction models having 12 different projects including "number of distinct committers" and "number of modified lines" metrics produced good results. Additionally, the method is best in metrics extracted from recently changed codes.

Foucault et al. investigated ownership metrics and software quality in open-source projects. One of the results of this work is that simple metrics performs better. In this work, it is ambiguous that how the methods employed for class imbalance problem behave in data sets including ownership metrics [10].

McIntosh et al. [22] stressed that code review metrics are very effective in software quality. Their study, that includes 14 process metrics, did not explore the impact of process metrics in class imbalance.

In another work [11], which is related to how process metrics should be collected and analyzed, a new software evaluation model is developed. Evaluating historical processes, the model adds new process metrics to the evaluation structure. Thus, the aim of this operation is that it helps practitioners to interpret process tables and develop software system. Bird et al. [23] investigated the correlation between defectiveness and ownership metrics. In the experiment, which was performed using two versions of Windows operating system, it was concluded that as the level of ownership of a software system increases, the number of total defects decreases. However, it was remained unclear how the ownership metrics affect the success of the prediction. Another work [24] having similar results examined comment and ownership metrics on four data sets. It could not decide whether recall or precision outperforms the other.

It is also possible to make foresight about software quality using linear models on process metrics [25]. But, applied models should be changed depending on the type of the project and developer teams. Weyuker et al. [26] investigated the correlation between developer metrics and the success of the SDP. One of the findings of this work is that it is not suitable to apply one model to various software systems. Because it is desired to use historical data of project teams while predicting defects. The number of developer is not the main factor affecting the ratio of defects

Please cite this article as: M.M. Öztürk, Which type of metrics are useful to deal with class imbalance in software defect prediction? Information and Software Technology (2017), http://dx.doi.org/10.1016/j.infsof.2017.07.004