Software Defect Prediction from Code Quality Measurements via Machine Learning

Ross MacDonald(✉)

Saint Mary's University, Halifax, Canada
ross.e.macd@gmail.com

Abstract. Improvement in software development practices to predict and reduce software defects can lead to major cost savings. The goal of this study is to demonstrate the value of static analysis metrics in predicting software defects at a much larger scale than previous efforts. The study analyses data collected from more than 500 software applications, across 3 multi-year software development programs, and uses over 150 software static analysis measurements. A number of machine learning techniques such as neural network and random forest are used to determine whether seemingly innocuous rule violations can be used as significant predictors of software defect rates.

Keywords: Software defects · Defect prediction · Machine learning

1 Introduction

Lehman's Laws of Software Evolution have demonstrated that software development projects are becoming larger and more complex as the years go by [1]. With this ever-increasing complexity and size and decreasing quality, it is becoming increasingly difficult to accurately predict how a development program will unfold.

Research has demonstrated that the later a software defect is found during development, the more costly it becomes to fix [2]. One reason for this escalation has been attributed to the fact that the longer a development project runs, the higher the overhead cost and the higher the cost required to change the system [2].

Many static analysis tools exist today, each of which provides a particular focus on the quality of software. Research recommends that software development teams should utilize a "meta tool" that would allow them to combine the results from various static analysis tools together, thus achieving a better results [3]. SonarQube is such a "meta-tool" that provides a software development team with the ability to import, customize and automatically execute static analysis on code utilizing a variety of rule definitions.

E. Bagheri and J. C. K. Cheung (Eds.): Canadian AI 2018, LNAI 10832, pp. 331–334, 2018.
https://doi.org/10.1007/978-3-319-89656-4_35

Unfortunately, with the advent of meta-tools such as SonarQube, a new problem is created: Metrics Galore [4]. Metrics Galore is a situation where a team is paralysed by attempting to monitor too many metrics simultaneously. Many issues can occur such as de-motivation of the team, focusing on the wrong metrics, losing sight of the important goals of the program, etc. In order to address this issue, while still providing metrics for the software development team to track and improve upon, it is possible to employ machine learning algorithms to help. While unsupervised learning can provide a solution to the overwhelming metrics issue [5], regression and classification could serve to solve the prediction issues by providing a predictive model that could be used by the development team and by the software development managers [3].

The goal of the proposed research is to demonstrate the value of static analysis metrics in predicting software defects at a much larger scale than has been proposed previously as is indicated in Table 1:

Table 1. Program measurements

Measure	Program 1	Program 2	Program 3
Duration (Years)	2	1	7
Team size (People)	30	10	80
Applications	290	132	352
LoC (MSLOCs)	4.28	2.96	7.72

2 Methodology

Analyses will be performed on a data set consisting of software metrics data collected from three large-scale software development programs. These programs consisted of software with high-reliability, high-criticality and safety-critical performance requirements, and thus were extensively tested during and after software development activities.

The data are snapshots of a continuum of ever-changing values. In order to limit the scope of the research in this paper to a manageable size, it was determined that a single snapshot in time of software source code and a single snapshot in time of software defect metrics would be analysed.

In order to determine the optimum point in time to perform both of these snapshots, intimate knowledge of the program plans was required. The theory built around these data collections is based on two key points: The source code should be analysed after major software development was completed, but prior to acceptance testing was performed as this source repository will have a maximum number of undiscovered defects; and the software defect measurements should be analysed well after acceptance testing was performed in order to have high confidence that most major defects have been discovered.

Since there are several hundred static analysis metrics and rules that could be counted in the analysis, it will be necessary to narrow down the predictors to a more manageable number in order to aid in analysis as well as to avoid model over-fitting. This feature reduction will be performed in several steps as described below.

- Eliminate zero and near-zero variance predictors
- Eliminate single items of strongly intercorrelated predictor pairs
- Feature clustering using k-means in order to group similar features together
- Recursive feature elimination to remove insignificant features from the model

After feature selection is performed, analysis of each of the models will be compared where appropriate and statistical findings will be provided and analysed upon completion of the research. The research will focus on Regression and Classification prediction models. For regression, the following models will be used: Linear Regression, Neural Network Regression, and Support Vector Machine (SVM) Regression. For classification, the following models will be used: Decision Tree, Random Forest, Neural Network, and SVM.

Regression analysis will target the raw defect count as the outcome, while Binary classification will use a binary definition by answering the following question: "Does the application contain a defect?". Finally, Multi-class classification will be performed on the software defect count by stratifying it across several classes. These values will be defined as: No defects, Low defect rate, Moderate defect rate, High defect rate, and Extreme defect rate.

3 Results To-Date

Progress to date has followed the plan as laid out in the methodology. The data has been collected, cleaned and prepared for analysis. Feature elimination has been performed by variance elimination and inter-correlation elimination. The feature sets were further narrowed by clustering similar features together using a k-means analysis. Regression and Binary classification has been performed on the reduced feature set and the results can be observed in the following tables.

Table 2. Linear regression performance summary - top three

Name	RMSE (Root Mean Squared Error)
lm-2	11637.48
sl-2	11669.20
sl-1	11844.82

As shown in Table 2, the Regression performance is less than desirable for the dataset, as the RMSE is quite large. In fact, the best RMSE value is nearly three-times that of the mean of the outcome column in the original dataset.

Conversely, Table 3 shows favourable results for binary classification, with a top accuracy of over 70% using Random Forest Classification.

Table 3. Binary classification performance summary - top three

Name	Accuracy
rf-all	70.82%
rf-1	70.31%
rp-all	65.36%

4 Future Work

The thesis has made significant progress in generating the dataset and subsequent data preparation. This dataset is unique in terms of its size and scope - spanning 500 software applications over three years of software development and includes 150 software static analysis measurements. The initial results shown in Table 3 provides strong evidence for our original hypothesis that seemingly innocuous rule violations can be used as strong predictors of software defect rates. The remaining work includes experimentation with a number of machine learning techniques to create a credible model to predict the software defects. In addition to the successful use of Random Forest for predictions, the project will experiment with different neural network architectures. The results of predictions will be compared against multi-class classifications. Finally, the thesis will conclude with a comprehensive analysis of feature importance and make recommendations for best practices in software development process. The remaining work is intended to constitute the majority of the thesis research of the author.

References

1. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution - the nineties view. In: Proceedings of the 4th International Symposium on Software Metrics, METRICS 1997, pp. 20–32. IEEE Computer Society, Washington (1997)
2. Haskins, B., Stecklein, J., Dick, B., Moroney, G., Lovell, R., Dabney, J.: 8.4.2 error cost escalation through the project life cycle. In: INCOSE International Symposium, vol. 14, pp. 1723–1737 (2004)
3. Rutar, N., Almazan, C.B., Foster, J.S.: A comparison of bug finding tools for java. In: 15th International Symposium on Software Reliability Engineering, pp. 245–256, November 2004
4. Bouwers, E., Visser, J., van Deursen, A.: Getting what you measure. Commun. ACM **55**(7), 54–59 (2012)
5. Hinton, G., Sejnowski, T.: Unsupervised Learning: Foundations of Neural Computation. A Bradford Book, McGraw Hill Book Company (1999)