Comparison of Machine Learning Techniques for Software Quality Prediction

Somya Goyal, Manipal University Jaipur, Jaipur, India

ABSTRACT
Software quality prediction is one the most challenging tasks in the development and maintenance of software. Machine learning (ML) is widely being incorporated for the prediction of the quality of a final product in the early development stages of the software development life cycle (SDLC). An ML prediction model uses software metrics and faulty data from previous projects to detect high-risk modules for future projects, so that the testing efforts can be targeted to those specific ‘risky’ modules. Hence, ML-based predictors contribute to the detection of development anomalies early and inexpensively and ensure the timely delivery of a successful, failure-free and supreme quality software product within budget. This article has a comparison of 30 software quality prediction models (5 technique * 6 dataset) built on five ML techniques: artificial neural network (ANN); support vector machine (SVMs); Decision Tree (DTs); k-Nearest Neighbor (KNN); and Naïve Bayes Classifiers (NBC), using six datasets: CM1, KC1, KC2, PC1, JM1, and a combined one. These models exploit the predictive power of static code metrics, McCabe complexity metrics, for quality prediction. All thirty predictors are compared using a receiver operator curve (ROC), area under the curve (AUC), and accuracy as performance evaluation criteria. The results show that the ANN technique for software quality prediction is promising for accurate quality prediction irrespective of the dataset used.

KEYWORDS
Area Under the Curve (AUC), Artificial Neural Network (ANN), Classification Tree, Fault Prediction, KNN, Machine Learning (ML), Naïve-Bayes, Receiver Operator Curve (ROC), Software Quality, Support Vector Machine (SVM)

1. INTRODUCTION
Software with high quality which meets the user’s needs, requirements and performs as per the expectation, are tempted in the entire world since always. The Software must ensure the failure-free performance; that is the Reliability of product. So many quality attributes and metrics with numerous Quality Assurance techniques are developed, but still the question: how to ensure that the resulting product will possess good quality is an open problem. The early detection of failure-prone modules directly correlates with the quality of end-product. The fault prediction involves the early detection of those “risky” modules of the software which prone to errors, impair the quality and will surely incur heavy development (testing) and maintenance cost. The early detection of faulty (buggy) modules improves the effectiveness of quality enhancement activities and ultimately improves the overall

DOI: 10.4018/IJKSS.2020040102
quality. The software fault prediction is used as an indicator for software quality for two reasons: (1) Quality is inversely proportional to the failures, which in turn caused by faults (development anomalies). To ensure high quality, failures must be minimized to zero. It can be achieved only by 100% detection and removal of defects from the modules. Hence, the accuracy in defect prediction is the most crucial factor to judge the quality of software product. (2) The early fault detection provides decisive power to the entire development team to strategically allocate the testing resources. Quality improvement activities can intensively be applied to the detected risky modules (Kan, 2002). Scheduling can be done more effectively to avoid delays. Prioritization of failure-prone modules for testing can be done. Ultimately, the quality can be improved and ensured by prediction of faults in early phases of development cycle. In case, faulty modules are not detected in early development phases, then the cost of getting the defect fixed increases manifold. Along with the increased cost, the chances of getting the defect detected by the customer in the live environment also increase. In situ, defect is found in the live environment, may stop the operational procedures which can eventually result in fatal consequences too. Software industry already witnessed such failures like NASA Mass Climate Orbiter (MCO) spacecraft worth $125 million lost in the space due to small data conversion bug (NASA, 2015).

Till today, so many researchers have contributed in this problem domain, but the gap is that their results are not univocal. Their work is not generalized but biased to the specific datasets or the techniques used to solve the problem.

Hence, the more accurate fault prediction is done, the more precise quality prediction is achieved. The present work is focused on the following research goals:

**R1:** To transform the software quality prediction problem as a learning problem (classification problem).

**R2:** To create ML prediction models using static code metrics as predictor.

**R3:** To evaluate the accuracy of prediction models empirically.

**R4:** To find which ML technique outperforms other ML techniques.

The major contribution of this work, is to develop and validate a cross-platform generalized model, to accurately predict the faulty modules in the software during development, so that the defects cannot propagate to the final phases and ultimately, the quality of software can be improved. As the quality of the software is inversely proportional to the number of defects in the end-product (Kan, 2002).

In this Paper; (1) The Software quality prediction problem is formulated as a two-class classification problem. Following this approach, a few features (attributes) from the previous project dataset are used as predictors and the quality of the module is generated as response by the classifier in terms of the predicted class label. (2) In total, 30 models are built for software quality prediction using 5 ML techniques (ANN, SVM, Naïve-Bayes classifier, DT, and k-nearest neighbor) over 6 datasets. Each model is validated using 10-fold cross-validation. (3) An empirical comparison among the developed prediction models is made using ROC, AUC, and accuracy as performance evaluation criteria.

The rest of the paper is organized as follows. Section 2 provides the glimpses of the related work from the literature. Section 3 gives the detailed description for the experimental set-up. The details to datasets, metrics, methods and performance evaluation criteria as research methodology are included in Section 4. Section 5 covers the results and the discussions. The work is concluded in Section 6, with remarks on the future work.

2. RELATED WORK

In this section, a review of literature is presented on the use of machine learning techniques for the prediction of software quality as well as the work already carried out about the static code metrics to detect faulty modules. The study is tabulated as Table 1, and shows that the static code metrics
Related Content

Automatic Personalization in E-Learning Based on Recommendation Systems: An Overview
www.igi-global.com/chapter/automatic-personalization-learning-based-recommendation/56070?camid=4v1a

An Adaptive ICT-Enabled Model for Knowledge Identification and Management for Enterprise Development
www.igi-global.com/article/an-adaptive-ict-enabled-model-for-knowledge-identification-and-management-for-enterprise-development/109066?camid=4v1a
A Semantic Map for Indoor Robot Navigation Based on Predicate Logic
www.igi-global.com/article/a-semantic-map-for-indoor-robot-navigation-based-on-predicate-logic/247975?camid=4v1a

Stochastic Utility Evaluation
www.igi-global.com/chapter/stochastic-utility-evaluation/74437?camid=4v1a