Software Quality Prediction Using Machine Learning

Bhoushika Desai and Roopesh Kevin Sungkur

Abstract In today's fast-changing environment, in order to create much more stable and complex software programs. With the emergence of Machine Learning, many companies are increasingly embracing this revolutionary approach, both in terms of growth and maintenance, to reduce software costs. As the size of applications increases in terms of functionality, when designing test cases, Software Quality Prediction is becoming more complex. Since the software measurement mechanism in a constant cycle has several benefits, namely reliable project cost estimation, process improvement and product quality compliance, it is vital to try further analysis of software metrics in order to implement the use of machine learning in software quality prediction. This research aimed at building two models which is Software Defect Prediction Model (SDPM) which will be used to predict defects in software and Software Maintainability Prediction Model (SMPM) which will be used for Software Maintainability. Different classifiers, namely Random Forest, Decision Tree, Naïve Bayes and Artificial Neural Networks have been considered and then evaluated using different metrics such as Accuracy, Precision, Recall and Area Under the Curve (AUC). The two models have successfully been evaluated and Decision Tree has been chosen as compared to other classifiers which tends to perform much better for both models. These models have been eventually been deployed as web services. Finally a framework based on a set of guidelines that can be used to improve software quality has been devised.

Keywords Software quality prediction · Software models · Machine learning · Classifiers · Score model

B. Desai · R. K. Sungkur (✉)
Department of Software and Information Systems, Faculty of Information and Communication Technologies, University of Mauritius, Moka, Mauritius
e-mail: r.sungkur@uom.ac.mu

M. Ben Ahmed et al. (eds.), *Innovations in Smart Cities Applications Volume 5*,
Lecture Notes in Networks and Systems 393, https://doi.org/10.1007/978-3-030-94191-8_32

1 Introduction

In software companies, system quality of the product is becoming a real concern. There are several variables that contribute to bad software products. Since the early 1970s, the software issue has arisen, with software engineers were unable deliver high-quality software at times and on budget [4]. Eventually, in their efforts to produce high-quality software and gain customer loyalty, software companies encounter numerous challenges. Inefficient management, administration, developer ego, strict schedules and strain, additional costs (e.g. buying new tools), contradictory views and values, lack of training on standards, inadequate resources to simplify the application development, lack of organizational quality management structure, poor knowledge of the process, Disapproval of senior management and the updated version of the system, poor communication, difficulty of coding and programming errors are ways of reducing software quality planning. There is a shortage of skilled workers in the technology sector that results in low performance. These could arise when programming and reviewing are done by the same individual or even when they're not comfortable with a computer language and are required to code. Such individuals are not skilled in this sector, leading to poor software quality. As this is a constantly evolving environment, the software business must provide its staff with frequent training to keep them aware with new technologies and resources. At the beginning of a project, it is advised to add developers not at the end or somewhere in between, that end up in a poor-quality product.

Another factor that causes applications to not achieve success is software testing. Software is not tested properly. In addition, each part or module should be tested during the manufacture of a product known as the system test. Testing is a procedure that guarantees that the system satisfies the client's criteria and specifications. Checking code is a way for software bugs to be detected and can also prevent program failure. Errors must be detected and corrected before the software is delivered to the customer, which contributes to a reliable product, maintains reliability, and decreases costs. Errors should usually be found early in the test phase. In addition, regression testing must be done to ensure the testing of the updated component of the program.

As reported by [6], it was observed that since the tester do not have enough software data to evaluate the system, and the most common quality issues emerged. He also suggested that monitoring was rigorous and that there was no written guideline for fair quality assurance of software. Modifications in requirements are most popular whilst designing complex systems. For instance, if the requirements are changing continuously in the early phases, and if the Software Development Life Cycle is defined, the new specification modification may still be managed. Additionally, company consumers start to understand what they can do to reduce

their everyday activities as the work proceeds and keep changing their minds by requesting more improved functionality. This problem affects the output of software products.

2 Literature Review

2.1 Software Defects and Software Maintainability

A software error is a software issue, mistake, defect, fault, failure, or flaw that provokes the system to generate an inaccurate or unforeseen result. Faults are critical aspects of a program. They are occurred from architecture or manufacture, or from outside environment. Faults in program are coding errors that affects various quality compared to expectation. Most of the errors are mostly from program code, several of them are from compiler-generated erroneous code [5]. Computer failures are a risk issue for computer developers and customers. Software errors not only reduce the consistency of the product, increase the costing but also slow the production timeline. Predicting software faults is presented to overcome this kind of problem. Software Defect Prediction can effectively advance software testing efficiency and guide the resource allocation. SDP detects the faulty element, which involves a wide variety of checks. Early detection of an error results in useful resource allocation, significantly decreases the amount of the time and cost of software and good quality software development. A SDP model consequently has a crucial role in understanding, assessing and enhancing the software quality of a software. [5]. Maintainability of software is defined as the correct in a software system to be changed adapt to environmental changes or meet specific needs prerequisites. This summary shows what software is maintained depending on different elements of program alteration (adaptation, rectification, enhancement, or prevention) [2].

2.2 Software Metrics

Software metric is a domain of computer science which is concerned with various software quantities and their developments [10]. Software metrics is among the main methods for successful analysis of the application [3]. Software metric is useful in improving software quality, cost analysis and budget planning [14] and with the aid of software measurement the software system can be interpret in an efficient way. Software quality metrics are the subgroup of computer metrics which, therefore, emphasizes the quality component of program, play a crucial role throughout the analysis and improvement of software quality.

There are three parameters of software metrics as shown in Fig. 1 below:

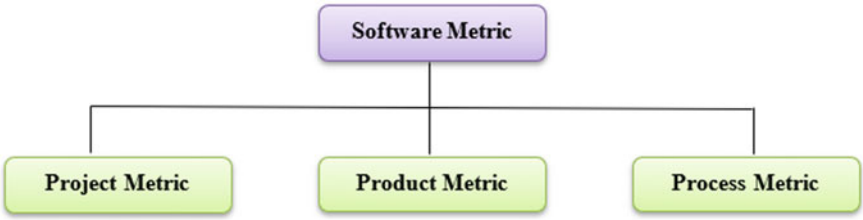

Fig. 1 Software metric parameters [5]

Project Metrics

Project metrics can be used for tracking the condition and progress of a project. By standardizing the work and helping to strengthen the software development strategy, project metrics avoid the issues or future risks [1]. Some scholars take the view that management is risk management [8]. This finding is attributed, in particular, to the belief that a large portion of project faults could have to do with poor risk management. Project metrics identify attributes of the project and its execution. For examples, the number of programmers, the personnel description over the software's life cycle, timetable, effectiveness and cost [1].

Product Metrics

Product metrics determines the software product's attributes at any point of its production. Product metrics can calculate system size, software design complexity, portability, maintenance capacity, project scale and efficiency. Product metrics are being used to extrapolate and reveal the product quality. Product metrics are used for regular or final service measurements [12]. Quality of the product is a crucial competitive concern when offering new products [9]. The quality of software products comprises of two stages: intrinsic product quality (reliability, density of defects) and customer satisfaction (user problems, customer satisfaction). The intrinsic quality of the software is determined by the number of functional faults in the program, or how often the program will operate before a malfunction occurs. Reliability is the likelihood that under defined circumstance a program will execute its specific task or a specific time period [13].

Process Metrics

Process metrics is oriented towards the software development method. It primarily addresses the length of the project, the form of technique used and the accumulated costs. Process metrics may be used to improve the production and maintenance of the software [7]. For examples of process metrics are the effectiveness of fault elimination throughout development, the analysis of test fault onset and the response time of the repair processes [12]. Experts have usually concentrated on two main categories of fault prediction metrics: code metrics that quantify application attributes such as size and complexity and process metrics are number of modifications and number of programmers [11].

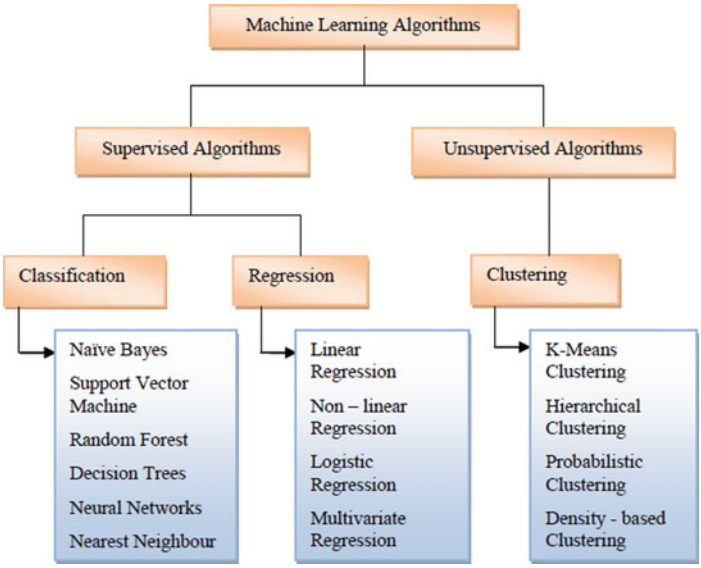

Fig. 2 Machine learning algorithms [5]

2.3 Machine Learning

Machine learning is a field in computer science that facilitates the learning of machines without complex programming. Machine learning is being used in a number of computer operations, where it is not easy to develop and relevant research algorithms with good results [15]. The problems can be fixed through machine learning easily by constructing a model which is a reasonable interpretation of a chosen dataset. Machine learning is becoming an innovative area from training the machines to emulate the nervous system, which has turned the domain of statistics into a large area that develops important mathematical concepts of learning processes. Machine learning consists of developing algorithms which enable the system to learn. Training is a method of finding out statistical regularities or other data patterns. The machine learning algorithms being developed to serve some role in the individual method of teaching (Talwar and Kumar, 2013). The below diagram illustrates the different machine learning algorithms (Fig. 2).

3 Methodology and Proposed Solution

The main objective of this research is to explore the issues and difficulties that lead to low software quality products and to demonstrate how machine learning can be used to predict the quality of application and provide the guidelines to address the

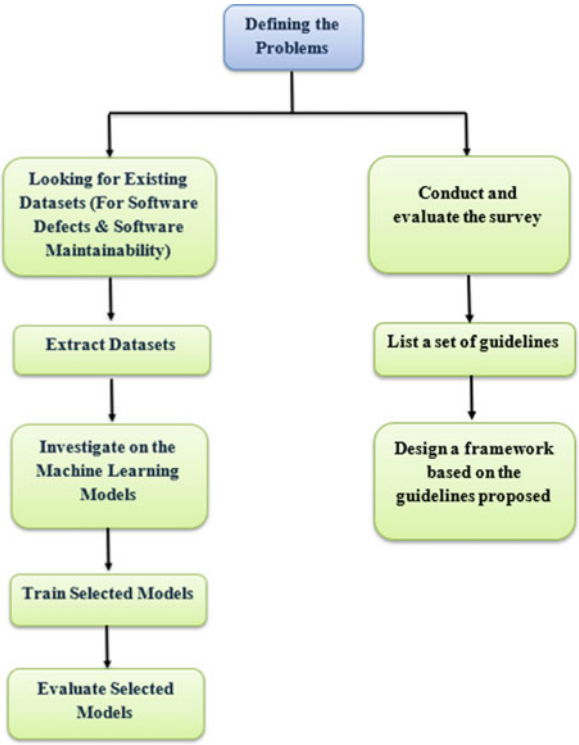

Fig. 3 Research design

problem encountered. The research objectives (ROs) of this research are depicted below.

RO1: To build a software defect prediction model.
RO2: To build a software maintainability prediction model.

The project will consist of two models: one for predicting defects in software which is software reliability and the other for predicting software maintainability. The model will be built and trained with several classifiers for both defect and maintainability. The best classifier will be chosen during evaluation of the model. Figure 3 below shows the research design whereas Fig. 4 shows the system design being used in this research. The diagram will be same for both model that is defect and maintainability prediction model. First, the data will be extracted from the dataset and then the data will be pre-processed by cleaning the data and removing missing values. The data will be split in sections and then the algorithms will be chosen. The model will be trained accordingly and finally the model will be evaluated.

Building the Model

The Azure Machine Learning Tool has been chosen to create the experiment because it is much easier as it is drag and drop. The aim of the model is to predict

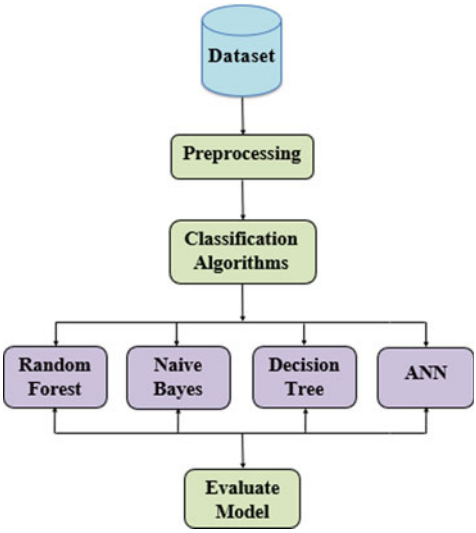

Fig. 4 System design

software defects and classification algorithms were used to train same. The SQL transformation has been used to remove outliers in the dataset. The data has been split into 2 sections: 80% for training the model which contain 8704 rows of records and 20% for testing the model which represents 2176 rows of records. The model is then trained with the below classifiers with the 80% of the training data and scored with the 20% of the testing data. After the scored model, it is then evaluated based on the accuracy, precision, recall and AUC. Several classifiers have been used as mentioned in the diagram, Two Class Neural Networks, Two Class Bayes Point Machine, Two Class Boosted Decision Tree and Two Class Decision Forest whose equivalent is Artificial Neural Network (ANN), Naive Bayes, Decision Tree and Random Forest.

About the Data

The software defect dataset that has been used in this research has been downloaded from Kaggle, and same is found on PROMISE repository which is accessible to the public. The defect dataset is called 'JM1'. The dataset contains 22 columns of attributes and has 10,885 instances. The dataset is in ARFF format and ready to use for creating the model. Before building the model, the dataset has been analysed and some graphs have been derived from the data. The below graph stated the number of defects in the dataset. The TRUE represents 2106 defects and FALSE represents 8779 which means non-defective (Fig. 5).

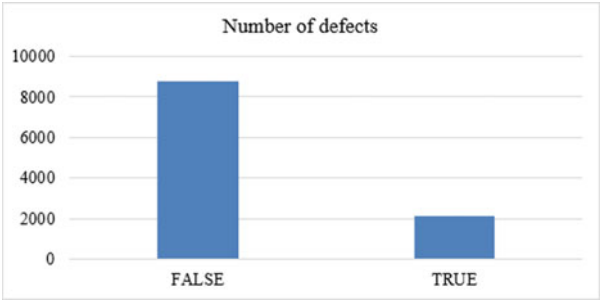

Fig. 5 Number of defects

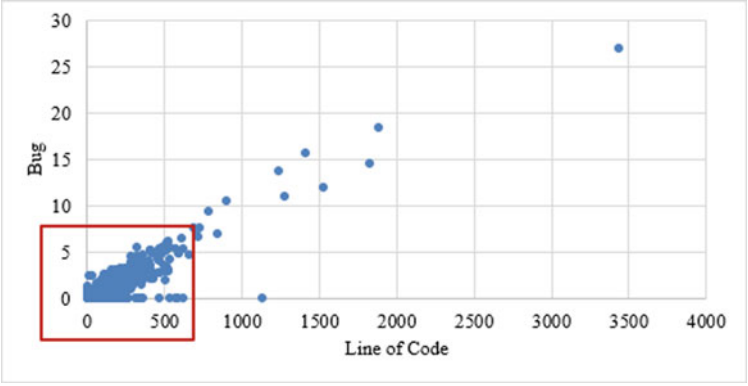

Fig. 6 Bug - line of code

The diagram below shows bug per line of code. As we can see, 0–600 lines of code approximately there are around 6 bugs. The rest above 600 lines of code can be considered as outliers (Fig. 6).

4 Results and Interpretation

4.1 Software Defect Model Evaluation

Score Model: For scoring the model, the testing dataset is being used which contains 2176 rows of records. It shows that there are 1756 elements which represents 81% of the data that contain no defects and 420 elements which represents 19% of data that contain defects.

Table 1 Evaluate defect model

Classifiers	Accuracy	Precision	Recall	AUC
Two Class Decision Forest (Random Forest)	0.810	0.514	0.255	0.730
Two Class Boosted Decision Tree (Decision Tree)	0.790	0.447	0.374	0.741
Two Class Bayes Point Machine (Naïve Bayes)	0.809	0.529	0.086	0.719
Two Class Neural Networks (Artificial Neural Networks)	0.812	0.558	0.126	0.716

Table 2 Evaluate maintainability

Classifiers	Accuracy	Precision	Recall	AUC
Two Class Decision Forest (Random Forest)	0.925	0.444	0.200	0.790
Two Class Boosted Decision Tree (Decision Tree)	0.924	0.457	0.350	0.836
Two Class Bayes Point Machine (Naïve Bayes)	0.930	0.667	0.033	0.782
Two Class Neural Networks (Artificial Neural Networks)	0.933	0.750	0.100	0.865

Evaluate Model: Each model has been evaluated and wrap up together in the table below. For evaluating these classification models, Accuracy, Precision, Recall and Area Under the Curve (AUC) have been used (Table 1).

The AUC has been chosen as the evaluating parameters because it is best to define the accuracy of the classifiers. Artificial Neural Networks (ANN) tends to be the least reliable prediction model according to the AUC which is 0.716. Naïve Bayes performed slightly better that ANN with an AUC of 0.719. Random Forest has a higher AUC of 0.730 as compared to Naïve Bayes and ANN. Decision Tree had the best result of a high AUC value of 0.741 as compared to other experimented classifiers.

4.2 Software Maintainability Model Evaluation

Score Model: For scoring the model, the testing dataset is being used which contains 837 rows of records. It shows that there are 777 elements which represents 93% of the data do not contain changes and 60 elements which represents 7.2% of data that contain changes.

Evaluate Model: Each model has been evaluated and consolidated in the table below. For evaluating these classification models, Accuracy, Precision, Recall and Area Under the Curve (AUC) have been used (Table 2).

The AUC has been selected as the evaluating factor because it is best to define the accuracy of the classifiers. Naïve Bayes had the least reliable prediction model

according to the AUC which is 0. 782. Random Forest performed slightly better that Naïve Bayes with an AUC of 0. 790. Decision Tree has a higher AUC of 0. 836 as compared to Naïve Bayes and Random Forest and a recall of 0.350 and precision of 0.457. ANN had the best result of a high AUC value of 0.865 as compared to other experimented classifiers but with a low recall of 0.100. However, Decision Tree tends to perform much better as compared to the other classifiers for both defect and maintainability prediction.

5 Conclusion

The aim of this research was to build two models which is Software Defect Prediction Model (SDPM) and Software Maintainability Prediction Model (SMPM) and finally to devise a framework based on a set of guidelines that can be used to improve software quality. Both models have successfully being built and tested with datasets available online. Several classifiers, namely Two Class Neural Networks, Two Class Bayes Point Machine, Two Class Boosted Decision Tree and Two Class Decision Forest whose equivalent is Artificial Neural Network (ANN), Naive Bayes, Decision Tree and Random Forest have been considered for the purpose of this research and then eventually compared. To evaluate the models, several metrics have been used. These include Accuracy, Precision, Recall and Area Under the Curve (AUC). The two models (SDPM and SMPM) have successfully been evaluated and Decision Tree has been chosen as compared to other classifiers since it tends to perform much better in both models' prediction. Software Quality is indeed is becoming an essential condition in the Software Industry. The results of this research is interesting since it addresses the issue of Software Quality Assurance by making use of Machine Learning, ensuring that this is done in a more reliable and effective way.

References

1. R. Aliverdi, L.M. Naeni, A. Salehipour, Monitoring project duration and cost in a construction project by applying statistical quality control charts. Int. J. Project Manag. **31**(3), 411–423 (2013)
2. H. Alsolai, M. Roper, Application of ensemble techniques in predicting object-oriented software maintainability, in *EASE '19: Proceedings of the Evaluation and Assessment on Software Engineering* (ACM, 2019) ISBN 978-1-4503-7145-2/19/04
3. N.E. Fenton, M. Neil, Software metrics: roadmap, in *Proceedings of the Conference on The Future of Software Engineering* (ACM, Limerick, Ireland), pp. 357–370
4. S. Jayawarna, A.T. Fonseka, Factors affecting product quality in the software development industry of Sri Lanka. Sri Lankan J. Manag. **1** (2011)
5. N. Kalaivani, R. Beena, Overview of software defect prediction using machine learning algorithms. Int. J. Pure Appl. Math. **118**(20), 3863–3873 (2018)

6. A. Khan, N. Nuzhat, K. Aihab, Survey to improve software quality assurance in developing countries. Int. J. Technol. Res. Islamabad 3.1 1–6, 3–5 (2015)
7. L. Madeyski, M. Jureczko, Which process metrics can significantly improve defect prediction models? An empirical study. Softw. Qual. J. **23**(3), 393–422 (2015)
8. J. Menezes, C. Gusmão, H. Moura, Indicators and metrics for risk assessment in software projects: a mapping study. in *Proceedings 5th Experimental Software Engineering Latin American Workshop (ESELAW 2008)* (2008)
9. F.J. Molina-Castillo, R.J. Calantone, M.A. Stanko, J.L. Munuera-Aleman, Product quality as a formative index: evaluating an alternative measurement approach. J. Prod. Innov. Manag. **30** (2), 380–398 (2013)
10. K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, S. Ducasse, Software quality metrics aggregation in industry. J. Softw. Evol. Process. **25**(10), 1117–1135 (2013)
11. F. Rahman, P. Devanbu, How, and why, process metrics are better, in *2013 35th International Conference on Software Engineering (ICSE)* (IEEE), pp. 432–441
12. M.S. Rawat, A. Mittal, S.K. Dubey, Survey on impact of software metrics on software quality. (IJACSA) Int. J. Adv. Comput. Sci. Appl. **3**(1)
13. F. Selnes, An examination of the effect of product performance on brand reputation, satisfaction and loyalty. Eur. J. Mark. **27**(9), 19–35 (1993)
14. M. Sharma, G. Singh, A. Arora, P. Kaur, A comparative study of static object-oriented metrics. Int. J. Adv. Technol. **3**(1), 25–34 (2012)
15. D. Sharmal, N. Kumar, A review on machine learning algorithms, tasks and applications. Int. J. Adv. Res, Comput. Eng. Technol. (IJARCET), **6**(10), 1548–1552 (2017)