

Machine Learning Approach for Reliability Assessment of Open Source Software

Ranjan Kumar Behera[1(✉)], Santanu Kumar Rath[1], Sanjay Misra[2], Marcelo Leon[3], and Adewole Adewumi[2]

[1] National Institute of Technology Rourkela, Rourkela, India
jranjanb.19@gmail.com, skrath235@gmail.com
[2] Covenant University, Ota 1023, Nigeria
{sanjay.misra,adewumi}@covenantuniversity.edu.ng
[3] Universidad Nacional De Loja, Loja, Ecuador
marceloleon11@hotmail.com

Abstract. Some of the quality parameters for any successful open source software may be attributed to affordability, availability of source code, re-distributability, and modifiability etc. Quality of software can be further improvised subsequently by either users or associated developers by constantly monitoring some of the reliability aspects. Since multiple users are allowed to modify the code there is a potential threat for security, which might degrade the reliability of software. Bug tracking systems are often considered to monitor various software faults, detected mostly in open source software projects. Various authors have made research in this direction by applying different techniques in order to improve the reliability of open source software projects. In this work, an various machine learning models have been implemented to examine the reliability of the software. An extensive numerical illustration has also been presented for bug data recorded on bug tracking system. The effectiveness of machine learning models for estimating the level of faults associated with the systems has been verified by comparing it with similar approaches as available in the literature.

Keywords: Software reliability · Open source software · Probabilistic neural network · Goodness of fit · Concave model

1 Introduction

The popularity of the software has been increasing at an exponential rate as it has become an essential part of human life. As the open source software are exposed to public domain, the assessment of failure rate for software is a major concern from developers point of view. Software reliability is an important attribute which determines the quality of the product. It can be defined as the ability of the software to handle both software and hardware faults [1].

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11622, pp. 472–482, 2019.
https://doi.org/10.1007/978-3-030-24305-0_35

Users not only want the higher reliability of the product, they also want the quantitative figure for reliability before buying any software product. Accurate measurement of software reliability is a challenging task as there are several factors that contributes to measuring software reliability. Some of the attributes that influence the reliability of software are dependent on two factors [14]:

- Total number of faults or number of failures associated with the software.
- The way user operates the system, also known as the operational profile. So, every time we develop a software, in turn we also maintain a complete profile for the software and there will be an associated documentation which mentions the operation profile.

There are several problems associated with software reliability assessment:

- Errors which are embedded in the software program may or may not the cause of failure. So, the error may not cause the failure with same degree of frequency.
- The failure rate is observer dependent.

A number of new technology along with methodologies have been emerged to improve the reliability of the software. Several models have been developed in order to improve the scalability and reliability at different phases of software development [2,3]. Waterfall model seems to be simple and elegant for development process as it deals with sequential phases of improvement. Reliability requirement analysis is realized at the software analysis phase only. However, the degree of reliability is strengthening at consecutive stages of the development process by drawing attention through full proof class diagram. The product design aspects are then reduced to executable form in implementation stage Error detection and recovery and robustness has been tested at the testing phase. In this stages the validation and verification process has been performed to check is the product meets both user and software requirement or not. The software is then finally process to minimize the future possibility of damages. From development point of view software can be broadly classified into two types [4]:

- Closed Source Software (CSS)
- Open Source Software (OSS).

CSSs are not distributed in the public domain. These type of software's are having some licensing agreement and can be distributed only as a commercial software. Unlike CSS, Open source software are made available publicly for accessing and modify by the user with few restrictions. They are designed to facilitate the users to have collaborative participation, rapid prototyping, idea exchange, transparency etc. However, they are often prone to security and reliability threats. The development approach to OSS mainly focuses on strengthen the reliability parameters which might provide uninterrupted service within a given period of time. SRGMs are mathematical models to predict the rate of failure. There are many types of SRGMs [5]. These SRGMs are grouped into 2 categories

– Concave Shaped Models
– S-Shaped Models

The best thing about these models is, the fault detection rate decreases as the amount of fault detected increases. Figure 1 presents the concave and s-shaped models, for software reliability growth. In concave shaped models, initially failure rate increases to its peak and after reaching the peak it starts decreasing. So, Concave models indicate that after reaching the peak value failure rate is expected to decrease exponentially. In S-shaped models: initially, when the testers don't have that much knowledge about the software, the fault removal rate will be less but once they get to know about the software there is an increase in fault removal rate. So basically, the fault removal rate depends upon the skills of the tester.

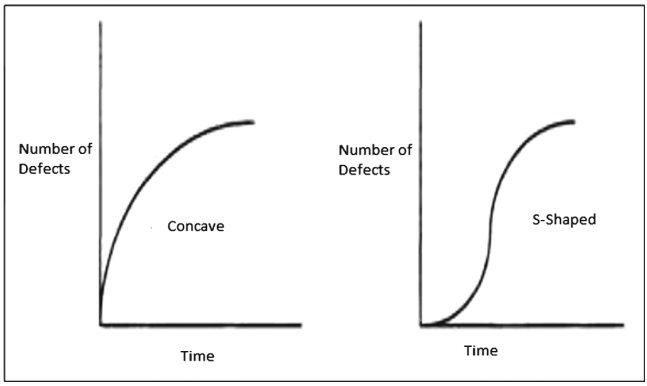

Fig. 1. Concave and S-shaped software reliability growth models

SRGMs don't care about the dependency between various components and consider the entire software as a single entity. So basically, SRGMs are also called as black box models. These models measure the value of reliability in the testing phase and on the basis of reliability value they decide whether to release the software or not [6]. Each SRGM imposes restrictions or assumptions. These assumptions are as follows:

– Once a fault is discovered in the system, it should be fixed immediately.
– Correction of a defect will not cause an introduction of a new defect in the system.
– During the quality assurance test no new code is introduced.
– Each fault is equally probable. They can lie into any severity class such as major, minor or critical.
– The failures are independent of each other.

Sometimes these models are not applicable to wide range of projects due having multiple constraints. Machine learning are found to be promising tools for

accessing the reliability of the network. In this paper, various neural models have been applied to validate the reliability parameters of the software. The functionality of the neural model is similar to the ways human brain works. It consists of several processing units known as neurons in layering fashion. Basically it consists of three layers such as input, hidden and output layer. There could be more than hidden layers where complex problem can be solved. Accessing the complex software requires high number of computational units for which we need more hidden layer. This approach leads to extension of several complex neural processing known as deep neural network. In this paper we have accessed the software reliability through several neural network models.

2 Background

In the past, many software reliability models have been proposed to assess the reliability of software. Among these models, the question is which model to select in a particular situation. Musa *et al.* have used various models and shown that some models are producing better results in comparison to others; for an example, the geometric family models are providing better prediction results in comparison to other models [7]. Goel *et al.* stated that the performance of different models depends on the type of dataset if one model is giving a good result for one dataset it may not perform well for another dataset [6]. So, for selecting the best model for a particular project we need to compare the results of various models. But it is also true that there is no way by which we can select a model a priori [8,9]. So, many authors have studied and proposed methods for selecting a model at an early stage of software development [10].

Brocklehurst *et al.* proposed that the model selection process basically depends upon the nature of faults presents in the system [11]. They have stated that the design flaws present in the software are the main reason for the failures. Abdel-Ghaly *et al.* analyzed ten models using five different approaches to measure the predictive quality of these models [12]. They observed that the criteria used for the evaluation of the model, choose a different model to be a good predictor. Khoshgoftaar also uses different criteria for selecting a model and he observed that Akaike Information Criteria (AIC) is the best criteria for selecting any model [13]. Lyu and Nikora also analyzed various software projects and they have developed a tool for selecting a model. In their paper, they have stated that Goodness-of-Fit (GOF) is the best criteria for selecting any particular model [14].

Shukla et al. have collected historical data from different software projects and with the help of this historical data they have proposed a method for selecting the best model for software reliability assessment [15]. But the flaw in this method is how to measure the similarity between different software projects. Stringfellow *et al.* have proposed a method for the selection of appropriate SRGM and it also helps in deciding to which extent we have to perform testing and what would be the appropriate time to release any software [16].

3 Fault Data Recorded in Bug Tracking System

Usually The fault data associated with a OSS are stored in separate file which can be reused for future development process. Bug-tracking systems are the tools that are usually adopted for mining fault data for both proprietary and open source software. Bugzilla and Buglister are widely used bug tracking system for OSS and proprietary software respectively. A number of parameters are associated with the datasets. Out of all the parameters only eight have been considered to predict the fault. Some of the parameters are date of capturing the fault, name of component, name of product, name of platform etc. The fault levels have been categorized into eight different groups. The categorical classification of fault levels is listed in Table 1.

Table 1. The indexed fault levels recorded on bug tracking system

Index number	Fault level
1	Trivial
2	Enhancement
3	Minor
4	Normal
5	Regression
6	Blocker
7	Major
8	Critical

4 Machine Learning Techniques Used

Machine learning algorithms are found to be suitable for classification and prediction task. In this work, we have considered various machine learning models for multi-level classification for fault identification. After the fault prediction, the reliability of the software has been accessed for measuring the feasibility of the software. Following are the list machine learning techniques used in this work

(1) Naive Bayes
(2) Decision Tree Classifier
(3) Random Forest Classifier
(4) Support Vector Machine (SVM)
(5) Artificial Neural Network (ANN)
(6) Probabilistic Neural Network (PNN)

5 Methodology Used

The Methodology used for identification of critical fault is shown in Fig. 2. Following are the steps used for the identification of critical fault present in the system

(i) The data of various OSS products have been collected from Bugzilla website [17].
(ii) The data collected from the website of Bugzilla is raw data, so it can't be directly processed into the models. So first, we have converted that data into fault count on the basis of occurrence.
(iii) After data conversion various machine learning techniques have been applied to the identification of critical fault present in the system.
(iv) Then various performance evaluation parameters such as accuracy, precision, recall and f-score have been calculated to compare the performance of each model.
(v) If the value of these parameters meets the objective then calculate the value of reliability factor and goto step (vii). Function for calculating reliability value is [34]

$$R(\Delta t|t) = e^{-a[e^{-bt} - e^{-b(t+\Delta t)}]} \tag{1}$$

where, a = Expected total number of defects to be eventually detected
b = Defect detection rate
t = Time interval
Δt = The increment of execution time

(vi) Otherwise modify the parameters and repeat the steps (iii) to (v) again.
(vii) If the value of reliability factor is acceptable then release the software otherwise it requires more testing effort.

6 Dataset

The software specification of open source software is associated with both functional and non-functional parameters. The fault data have been collected for four different OSS product namely Firefox, Core, Tomcat and Camino Graveyard [17]. The fault data extracted with the help of bug tracking system are

Table 2. Open source software used for experiment

Sl no.	Product name	Number of failures
1	Camino Graveyard	12502
2	Core	10000
3	Mozilla Firefox	10000
4	Tomcat	11025

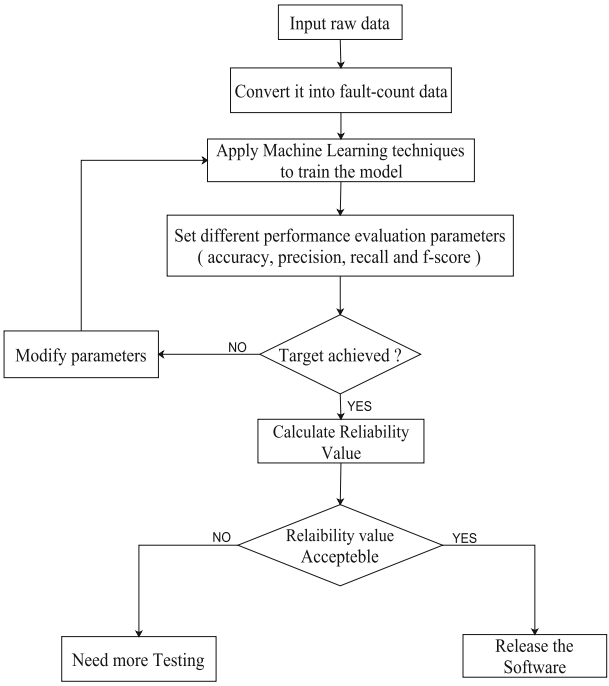

Fig. 2. Methodology used for software reliability assessment

not directly processed in machine learning model as it contains number of undesired features. Datasets have been preprocessed in order to filter and reduce the dimension. The dataset used in this paper is listed in Table 2.

7 Performance Evaluation Criteria

(i) **Precision**: Precision is defined as the total number of true positives divided by the total number of predicted positives [11].

$$Precision = \frac{T_p}{T_p + F_p} \tag{2}$$

(ii) **Recall**: Recall is defined as the total number of true positives divided by the total number of actual positives [11].

$$Recall = \frac{T_p}{T_p + F_n} \tag{3}$$

(iii) **F-score**: F_1 score is calculated by taking the harmonic mean of recall and precision, mathematically it can be defined as [11]

$$F\text{-}score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

(iv) **Accuracy**: Accuracy is defined as the total number of correct classifications divided by the total number of observations [11]. Mathematically accuracy can be defined as

$$Accuracy = \frac{T_p + T_n}{T_p + F_p + T_n + F_n} \tag{5}$$

where,

T_n = Observations that are actually negative and predicted negative (true negative).
T_p = Observations that are actually positive and predicted positive (true positive).
F_n = Observations that are actually positive and predicted negative (false negative).
F_p = Observations that are actually negative and predicted positive (false positive).

8 Results and Discussion

Bugzilla bug tracking system is being considered to obtained the raw dataset from different open source software. They have been preprocessed in order to make the dimensionality reduction for eradicating undesired features. Five different machine learning algorithms have been implemented to measure the reliability of the software. Eight fault levels have been considered on the objective variable that have been used in bug tracking system. The description of the fault levels has been presented in Table 1. The fault levels for instances of datasets have been estimated by using various machine learning algorithms. The machine learning models have been evaluated by using various performance parameters like accuracy, precision, recall and f-measure. The result of various models for multilevel classification have been presented in Tables 2, 3, 4 and 5.

Table 3. Performance analysis for Camino Graveyard dataset

Model	Accuracy	Recall	Precision	F-score
Naive Bayes	0.5984	0.5786	0.5984	0.5768
Decision Tree	0.6204	0.6074	0.6204	0.6374
Random Forest Classifier	0.6298	0.6242	0.6328	0.5718
SVM	0.6328	0.6512	0.6158	0.5678
ANN	0.6556	0.6086	0.6186	0.5968
PNN	**0.7016**	**0.6932**	**0.6434**	**0.6537**

It can be observed that the probabilistic based neural network provides better results as compared to other machine learning algorithms in most of the

Table 4. Performance analysis for core dataset

Model	Accuracy	Recall	Precision	F-score
Naive Bayes	0.7836	0.6828	0.7836	0.7972
Decision Tree	0.7933	0.6945	0.7836	0.7351
Random Forest Classifier	0.7944	0.6999	0.7944	0.7445
SVM	0.7984	0.6979	0.7982	0.7485
ANN	0.8104	0.7207	0.8104	0.7948
PNN	**0.8174**	**0.7256**	**0.8234**	**0.7988**

Table 5. Performance analysis for Firefox dataset

Model	Accuracy	Recall	Precision	F-score
Naive Bayes	0.7966	0.7086	0.7861	0.7459
Decision Tree	0.8264	0.7697	0.8164	0.7893
Random Forest Classifier	0.8272	0.7880	0.8172	0.8023
SVM	0.8352	0.7930	0.8271	0.8133
ANN	0.8476	0.6849	0.8376	0.8195
PNN	**0.8536**	**0.7349**	**0.8446**	**0.8235**

datasets. The reliability index for each software has been measured after identification of the critical faults. Probabilistic approach is found to be most suitable for predicting the fault levels. Naive Bayes algorithm has the less performance in all the datasets. The PNN has more than 85% accuracy in case of Firefox dataset where as in case of Tomcat dataset the accuracy is only 65%. Although accuracy is less, it is better than all other machine learning algorithms. The f-score in Tomcat dataset is found to be less for PNN model as it contained more unbalanced instances. However, the PNN model is found to be perform better in term of other parameters like accuracy, precision and recall (Table 6).

Table 6. Performance analysis for Tomcat dataset

Model	Accuracy	Recall	Precision	F-score
Naive Bayes	0.5755	0.5921	0.5355	0.5896
Decision Tree	0.5981	0.5607	0.5481	0.5818
Random Forest Classifier	0.5936	0.5692	0.5491	0.5312
SVM	0.6042	0.5804	0.5942	0.6280
ANN	0.6359	0.5981	0.5459	0.5856
PNN	**0.6559**	**0.6012**	**0.5989**	**0.5646**

9 Conclusion and Future Work

A number of bug tracking system are widely available for extracting the bug information from the open source software. In this work we have considered the Bugzilla bug tracking system for mining the bus from the software. The various characteristics of reliability parameters have been explored after identification of the fault level in order to predict whether the software is feasible for use or not. Machine learning algorithms have been implemented for predicting the fault level of the instances of the software. After the fault prediction the reliability of the software has been measures which help in identifying the best models based on both stability and fitting capabilities. From the experimental results, It has been found that probabilistic neural network is the most suitable algorithm to handle the fault prediction. This work can be extended by implementing deep learning models like Convolutional Neural Network (CNN), Recurrent neural network (RNN) in order to have better accuracy for fault prediction.

References

1. Lyu, M.R. (ed.): Handbook of Software Reliability Engineering. IEEE Computer Society Press, Los Alamitos (1996)
2. Yamada, S.: Software Reliability Modeling: Fundamentals and Applications. Springer, Tokyo/Heidelberg (2014). https://doi.org/10.1007/978-4-431-54565-1
3. Tamura, Y., Matsumoto, M., Yamada, S.: Software reliability model selection based on deep learning. In: Proceedings of the International Conference on Industrial Engineering, Management Science and Application, Korea, 23–26 May 2016, pp. 77–81 (2016)
4. Behera, R.K., Shukla, S., Rath, S.K., Misra, S.: Software reliability assessment using machine learning technique. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10964, pp. 403–411. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95174-4_32
5. Yamada, S., Tamura, Y.: OSS Reliability Measurement and Assessment. Springer, London (2016). https://doi.org/10.1007/978-3-319-31818-9
6. Goel, A.L.: Software reliability models: assumption, limitations, and applicability. IEEE Trans. Softw. Eng. **12**, 1411–1423 (1985)
7. Musa, J.D., Iannino, A., Okumoto, K.: Software Reliability, Measurement, Prediction and Application. McGraw-Hill, New York (1987)
8. Sharma, K., et al.: Selection of optimal software reliability growth models using a distance based method. IEEE Trans. Reliab. **59**(2), 266–276 (2010)
9. Schick, G.H., Wolverton, R.W.: An analysis of competing software reliability models. IEEE Trans. Softw. Eng. **2**, 104–120 (1978)
10. Sukert, A.N.: Empirical validation of three software errors predictions models. IEEE Trans. Reliab. **28**, 199–205 (1979)
11. Brocklehurst, S., Chan, P.Y., Littlewood, B., Snell, J.: Recalibrating software reliability models. IEEE Trans. Softw. Eng. **SE–16**(4), 458–470 (1990)
12. Abdel-Ghaly, A.A., Chan, P.Y., Littlewood, B.: Evaluation of competing software reliability predictions. IEEE Trans. Softw. Eng. **SE–12**(12), 950–967 (1986)
13. Khoshgoftaar, T.M.: On model selection in software reliability. In: 8th Symposium in Computational Statistics (Compstat 1988), pp. 13–14, August 1988

14. Lyu, M., Nikora, A.: CASREA-a computer-aided software reliability estimation tool. In: Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering, Montreal, pp. 264–275 (1992)
15. Shukla, S., Behera, R.K., Misra, S., Rath, S.K.: Software reliability assessment using deep learning technique. In: Chakraverty, S., Goel, A., Misra, S. (eds.) Towards Extensible and Adaptable Methods in Computing, pp. 57–68. Springer, Singapore (2018). https://doi.org/10.1007/978-981-13-2348-5_5
16. Stringfellow, C., et al.: An empirical method for selecting software reliability growth models. Empir. Softw. Eng. J. **7**, 319–343 (2002)
17. The Mozilla Software Foundation: The Mozilla project. https://bugzilla.mozilla.org/