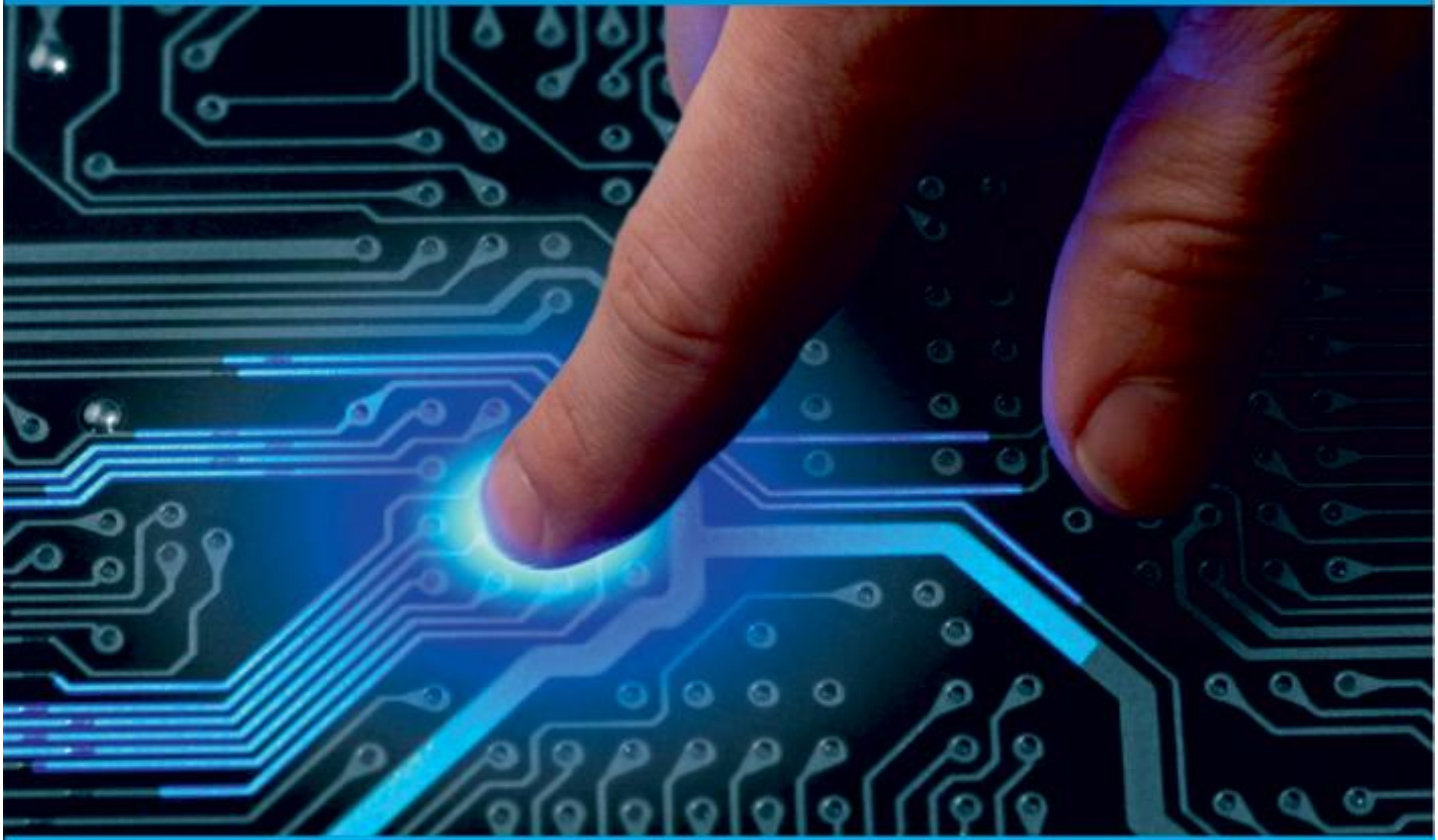
e-ISSN: 2320-9801 | p-ISSN: 2320-9798



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 8, Issue 8, August 2020

Impact Factor: 7.488

e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | Impact Factor: 7.488 |

|| Volume 8, Issue 8, August 2020 ||

Machine Learning Based Software Bug Prediction Model

Patil Rekha [1], Waheeda Anjum [2]

Associate Professor, Department of Computer Science and Engineering, Poojya Doddappa Appa College of Engineering, Kalburagi, Karnataka, India[1]

P.G student, Department of Computer Science and Engineering, Poojya Doddappa Appa College of Engineering, Kalburagi, Karnataka, India[2]

ABSTRACT: Software Bug Prediction (SBP) is an issue in software development and process maintenance, which concerns with the overall software successes. This is because of the predicting device malfunctions earlier boosts quality of applications, reliability , performance and low application cost. The development of a reliable bug prediction model is therefore a challenging task and a number of techniques were proposed in the literature. Here we have proposed a software bug prediction model based on machine learning (ML) algorithms. Based on historical data, three Supervised Machine Learning algorithms have been used to predict future software faults. The classifiers are Naïve Bayes (NB), Linear Regression(LR) and Decision Tree algorithm(DT). The evaluation process suggested that Machine Learning algorithms can be used effectively at a high precision scale. Tests gathered show that the Machine Learning approach works better. They also achieve higher accuracy compared to the other models.

KEYWORDS: Naïve Bayes (NB), Decision Tree (DT), Linear Regression (LR), Software Bug.

I. INTRODUCTION

The presence of defects in software significantly affects the cost of software reliability , efficiency, and maintenance. It is hard to achieve bug-free software, including the software applied carefully because secret bugs occur much of the time. Additionally, a major challenge in software engineering is designing software bug prediction model that could predict the defective modules in the early phase. Bug prediction software is an important task in the production of software. This is because anticipating the faulty modules before deploying applications improves user satisfaction, enhancing overall performance of the program. In addition, detecting the software bug early enhances the adaptation of software to various conditions and increases the utilization of resources. Several strategies were introduced to tackle the issue of Software Bug Prediction (SBP). Machine Learning (ML) techniques are the most known. In Software Bug Prediction, the Machine Learning techniques are extensively used to predict unstable modules based on historical faulty data, critical metrics and various computing software techniques. Three supervised Machine Learning classifiers are used here to test the Machine Learning capabilities of Software Bug Prediction. The research dealt with classifier Naïve Bayes (NB), Linear Regression (LR). The Machine Learning classifiers mentioned here refers to the datasets obtained from the works. Besides that, here it distinguishes between classifier Naive Bayes (NB), classifier Linear Regression(LR). The distinction is based on various evaluation criteria such as the classifier's accuracy , precision , recall, F-measures, and ROC curves.

II. RELATED WORK

The Chidamber and Kemerer (CK) metrics [36] have been widely used in the context of bug prediction. Basili et al. [1] investigated the usefulness of the CK suite for predicting the probability of detecting faulty classes. They showed that five of the experimented metrics are actually useful in characterizing the bug-proneness of classes. The same set of metrics has been successfully exploited in the context of bug prediction by El Emam et al. [26] and Subramanyam et al. [27]. Both works reported the ability of the CK metrics in predicting buggy code components, regardless of the size of the system under analysis. Still in terms of product metrics, Nikora et al. [28] showed that measuring the evolution of structural attributes (e.g., number of executable statements, number of nodes in the control flow graph, etc.) it is possible to predict the number of bugs introduced during the system development. Later, Gyimothy et al. [2] performed a new investigation on the relationship between CK metrics and bug proneness. Their results showed that the coupling between object metric is the best in predicting the bug-proneness of classes, while other CK metrics are untrustworthy. Ohlsson et al. [3] focused the attention on the use of design metrics to identify bug-prone modules. They performed a study on an Ericsson industrial system showing that at least four different design metrics can be used with equivalent results. The metrics performance are not statistically worse than those achieved using a model based on the project size. Zhou et al. [29] confirmed their results showing that size-based models seem to perform as well as those based on CK metrics except than the Weighted Method per class on some releases of the Eclipse system. Thus, although Bell et al. [35] showed that more complex metric-based models have more predictive power with respect to size-based models, the latter seem to be generally useful for bug prediction. Nagappan and Ball [4] exploited two static analysis tools to early predict the pre-release bug density. The results of their study, conducted on the windows server system, show that it is possible to perform a coarse grained classification between high and low quality components with a high level of accuracy. Nagappan et al. [14] analyzed several complexity measures on five Microsoft software systems, showing that there is no evidence that a single set of measures can act universally as bug predictor. They also showed how to methodically build regression models based on similar projects in order to achieve better results.

III. **METHODOLOGY**

Machine Learning (ML) techniques are the most known. In Software Bug Prediction, the Machine Learning techniques are extensively used to predict unstable modules based on historical fault data, critical metrics and various computing software techniques. Three supervised Machine Learning classifiers are used to test the Machine Learning capabilities of Software Bug Prediction. The research dealt with classifier such as Naïve Bayes (NB), Linear Regression (LR).

The results showed that our Software Bug Prediction model main comprises of the follows modules namely:

1. Load dataset : This helps to load dataset from .csv file which consists of the parameters such as loc, v(g),ev(g).......etc. The dataset used here is named as Jm1.

2. Data Pre Processing : This mainly cleans the dataset. It also finds records containing the null values. If it is present then it also finds it. Dataset is pre processed by a proposed clustering technique. The pre processing also divides the data into test dataset and trained dataset using clustering technique.

3. Clustering : This is the task of dividing the data points or population into a number of groups such that data points are more similar in the same group to other data points in the same group than those in other groups. The proposed clustering technique marks the data with class labels such as Successful and Redesign.

4. Classification : This is often a method of categorizing given set of knowledge into classes. This could be performed on each structured and unstructured data. The method starts with predicting the category of given knowledge points. The categories are nothing however said as target, label or classes. The labels are set to classify the amount of faults into 2 different categories (one with fault and alternate while not fault).

5. Bug Prediction : This helps us to seek out bugs from the program code. In order to evaluate the performance, we are using Machine Learning algorithms such as Linear Regression, Naive Bayes, Decision Tree in software bug prediction.

Naive Bayes is one of the Supervised Machine learning algorithm/method which is preferred here as it helps to reduce the complexity of the model and also achieve better performance which helps in easy detection of the bugs or faults in the system and also provides the better accuracy(which is approx 0.99%) when compared to the other models.

Here a set of well known measures based on the generated confusion matrixes such as follows:

	Precision	Recall	F1 - score	Support
Redesign	0.93	0.94	0.94	319
Successful	0.99	0.99	0.99	1858
Accuracy			0.98	2177
Macro avg	0.96	0.97	0.96	2177
Weighted avg	0.98	0.98	0.98	2177

[[301 18]

[22 1836]]
The accuracy is 0.991626090
The intercept is -0.0935996864
The Co- efficient value is [0.00761893]
The Mean squared error (MSE) is 0.063444130693
The Root Mean Squared Error (RMSE) is 0.2518811836827

IV. EXPERIMENTAL RESULTS

Fig 1: Home Screen

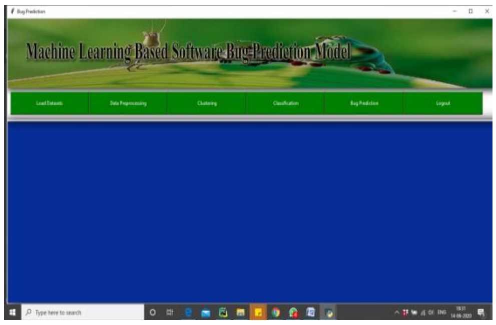
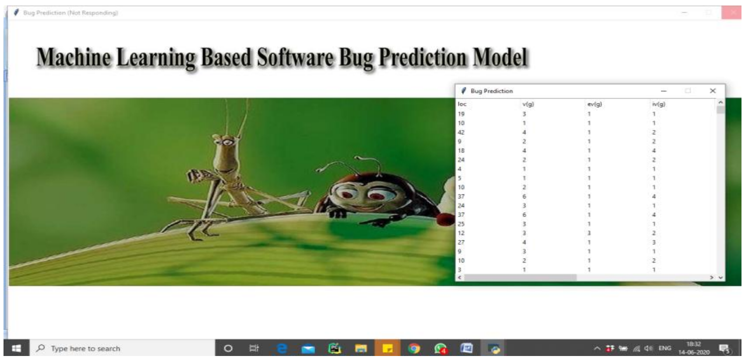


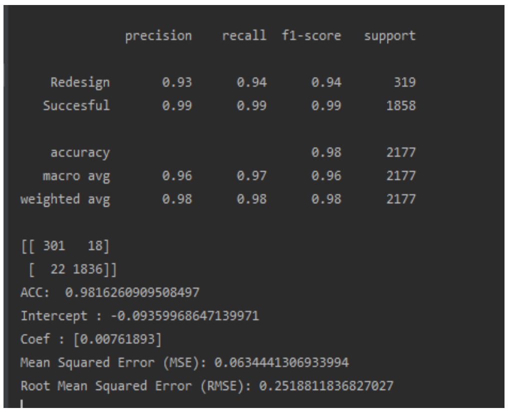
Fig 2: Menu options


Bug Prediction (Not Responding)
Machine Learning Based Software Bug Prediction Model
Bug Prediction

loc	v(g)	ev(g)	iv(g)
19	3	1	1
10	1	1	1
42	4	1	2
9	2	1	2
18	4	1	4
24	2	1	2
4	1	1	1
5	1	1	1
10	2	1	1
37	6	1	4
24	3	1	1
37	6	1	4
25	3	1	1
12	3	3	2
27	4	1	3
9	3	1	1
10	2	1	2
3	1	1	1

Fig 3: Load Dataset

A dataset is a collection of data. In machine learning projects, we need a training dataset. It is the actual dataset used to train the model for performing various actions. In the above fig 3 bug dataset is used which loads in this module. This dataset consists of the attributes such as loc, v(g), ev(g), iv(g)....etc.



```
              precision    recall  f1-score   support

    Redesign       0.93      0.94      0.94       319
   Succesful       0.99      0.99      0.99      1858

    accuracy                           0.98      2177
   macro avg       0.96      0.97      0.96      2177
weighted avg       0.98      0.98      0.98      2177

[[ 301   18]
 [  22 1836]]
ACC:  0.9816260909508497
Intercept : -0.09359968647139971
Coef : [0.00761893]
Mean Squared Error (MSE): 0.0634441306933994
Root Mean Squared Error (RMSE): 0.2518811836827027
```

Fig 4: Using NB Classifier Accuracy

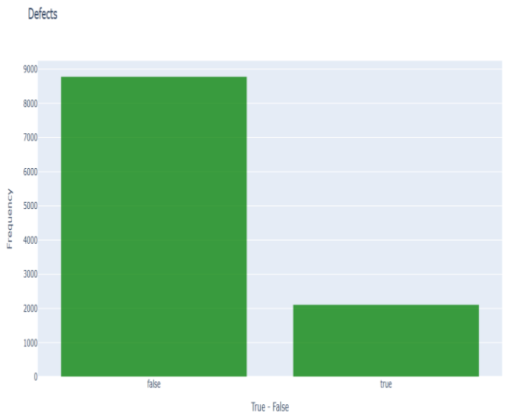


Fig 5: Defects

A software system defect is a blunder, bug, flaw, fault, malfunction or mistakes in software that causes it to make associate inaccurate or unheralded outcome. Faults are essential properties of a system. Software system flaws are programming errors that cause different performance compared with anticipation. The majorities of the faults are from source code or design, some of them are from incorrect code generating from compilers. It displays the software defects in terms of false and true. It is displaying defect frequency in terms of false and true. The graph implements that there are 8900 false defect frequency and about 2000 true defect frequency present.

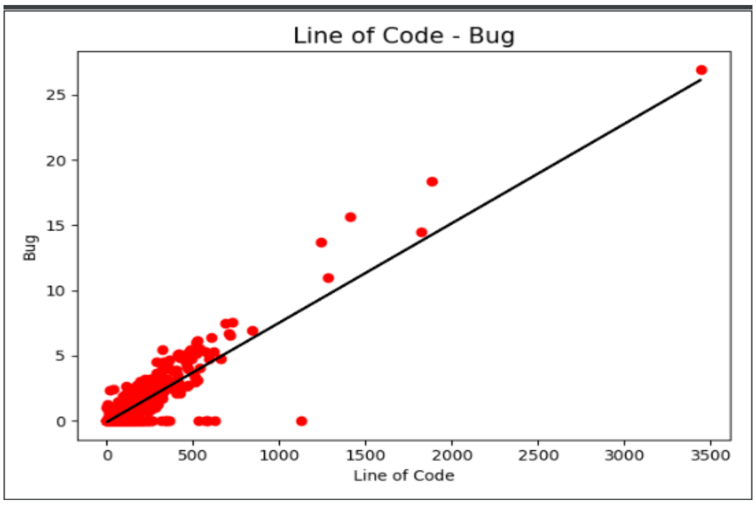


Fig 6: Line of Code Vs Bug

Prediction of software bugs is a technique in which a prediction model is created to predict the future software failures based on historical data. Different methods were suggested using different datasets, Metrices, and performance indicators. Here it makes use of the Machine Learning algorithms in software bugging problem. Three methods in machine learning were used namely Naive Bayes, Linear Regression, Decision Tree. The assessment process is carried out using three real test/debugging datasets. Experimental results are gathered based on measures of accuracy, accuracy, Recall, F-measurement, and RMSE. Results shows that Machine Learning method namely Naive Bayes classifier showed best performance when compared to others. The above Fig 6 represents the graph for the bugs verses the line of code.

Algorithm	Accuracy
Naive Bayes	0.99
Linear Regression	0.93

Datasets	NB
DS1	0.956
DS2	0.989
DS3	0.990
Average	0.978

V. CONCLUSION

Prediction of software bugs is a technique in which a prediction model is created to predict the future software failures based on historical data. Different methods were suggested using different data sets, different metrics and various performance indicators. This paper assessed the use of machine learning algorithms in software bugging problem. Three methods in machine learning were used which are NB, LR. The assessment process is carried out using three real test / debugging datasets. Experimental results are gathered based on measures of accuracy, accuracy, recall, F-measurement, and RMSE. Results show that the Machine Learning strategies are effective approaches for predicting possible defects in the program. The Confrontation results showed that the NB classifier performs best over the others.

REFERENCES

[1] V. Basili, L. Briand, and W. Melo, “A validation of object-oriented design metrics as quality indicators,” Software Engineering, IEEE Transactions on, vol. 22, no. 10, pp. 751–761, Oct 1996.
[2] T. Gyimothy, R. Ferenc, and I. Siket, “Empirical validation of ´ object-oriented metrics on open source software for fault prediction,” IEEE Transactions on Software Engineering (TSE), vol. 31, no. 10, pp. 897–910, 2005.
[3] N. Ohlsson and H. Alberg, “Predicting fault-prone software modules in telephone switchess,” Software Engineering, IEEE Transactions on, vol. 22, no. 12, p. 886894, 1996.
[4] N. Nagappan and T. Ball, “Static analysis tools as early indicators of pre-release defect density,” in Proceedings of the 27th International Conference on Software Engineering, ser. ICSE ’05. New York, NY, USA: ACM, 2005, pp. 580–586. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062558
[5] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in Proceedings of the Third International Workshop on Predictor Models in Software Engineering, ser. PROMISE ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10
[6] A. N. Taghi M. Khoshgoftaar, Nishith Goel and J. McMullan, “Detection of software modules with high debug code churn in a very large legacy system,” in Software Reliability Engineering. IEEE, 1996, pp. 364–371.
[7] J. S. M. Todd L. Graves, Alan F. Karr and H. P. Siy, “Predicting fault incidence using software change history,” Software Engineering, IEEE Transactions on, vol. 26, no. 7, pp. 653–661, 2000.
[8] A. E. Hassan, “Predicting faults using the complexity of code changes,” in ICSE. Vancouver, Canada: IEEE Press, 2009, pp. 78–88.
[9] R. Bell, T. Ostrand, and E. Weyuker, “The limited impact of individual developer data on software defect prediction,” Empirical Software Engineering, vol. 18, no. 3, pp. 478–505, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9178-4
[10] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Programmer-based fault prediction,” in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ser. PROMISE ’10. New York, NY, USA: ACM, 2010, pp. 19:1–19:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868357
[11] R. Moser, W. Pedrycz, and G. Succi, “Analysis of the reliability of a subset of change metrics for defect prediction,” in Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ser. ESEM ’08. New York, NY, USA: ACM, 2008, pp. 309–311. [Online]. Available: http://doi.acm.org/10.1145/1414004.1414063
[12] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, “Does measuring code change improve fault prediction?” in Proceedings of the 7th International Conference on Predictive Models in Software Engineering, ser. Promise ’11. New York, NY, USA: ACM, 2011, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/2020390.2020392
[13] W. P. Raimund Moser and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” in International Conference on Software Engineering (ICSE), ser. ICSE ’08, 2008, pp. 181–190.
[14] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” in Proceedings of the 28th International Conference on Software Engineering, ser. ICSE ’06. New York, NY, USA: ACM, 2006, pp. 452–461. [Online]. Available: http: //doi.acm.org/10.1145/1134285.1134349
[15] M. DAmbros, M. Lanza, and R. Robbes, “Evaluating defect prediction approaches: a benchmark and an extensive comparison,” Empirical Software Engineering, vol. 17, no. 4, p. 531577, 2012.
[16] J. Sliwerski, T. Zimmermann, and A. Zeller, “Don’t program on fridays! how to locate fix-inducing changes,” in Proceedings of the 7th Workshop Software Reengineering, May 2005. [17] L. T. Jon Eyolfso and P. Lam, “Do time of day and developer experience affect commit bugginess?” in Proceedings of the 8th Working Conference on Mining Software Repositories, ser. MSR ’11, 2011, pp. 153–162.
[18] F. Rahman and P. Devanbu, “Ownership, experience and defects: a fine-grained study of authorship,” in Proceedings of the 33rd International Conference on Software Engineering, ser. ICSE ’11, 2011, pp. 491–500.
[19] E. J. W. J. Sunghun Kim and Y. Zhang, “Classifying software changes: Clean or buggy?” IEEE Transactions on Software Engineering (TSE), vol. 34, no. 2, pp. 181–196, 2008.
[20] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, “Don’t touch my code!: Examining the effects of ownership on software quality,” in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE ’11. ACM, 2011, pp. 4–14.