Research Article

Multiclass Software Bug Severity Classification using Decision Tree, Naive Bayes and Bagging

Raj Kumar[1], Sanjay Singla[2]

[1]Research Scholar Department of CSE, IK Gujral Punjab Technical University, Jalandhar, India
[2]Professor Department of CSE, GGS College of Modern Technology, Kharar (Mohali), Punjab, India rajshira@gmail.com[1], dr.ssinglacs@gmail.com[2]

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: The software applications are experiencing the challenges of ever-growing complexity caused by the increase in the number of bugs. The software development process has been adversely affected due to the wastage of resources caused due to the bugs. It is imperative to identify and predict bugs to facilitate the software development process. Software bugs can be classified according to the severity of the bugs. In this paper a comparative analysis of Decision Tree, Naïve Bayes and Bagging approach is done for the bug severity classification. A comparative analysis of the Naïve Bayes, Decision Tree and Bagging approach is done for the accuracy, precision, recall and F-measure parameters

Keywords: Software Bugs, Bug Prediction, Decision Tree, Naïve Bayes, Bagging

1. Introduction

Quality Assurance (QA) is an activity that performs an essential role in software engineering. QA administers the software project to ensure that the final product or the software application meets the expectations of the end users. To judge the software quality, testing (quality assurance activity) is required. Testing is a time and efforts taken process. Several bugs remain uncovered even after rigorous testing using test cases. Main reason of these uncovers bugs is lack of time. The huge measure of information is gathered day by day. The enormous issue with this huge measure of information is in dissecting the information. This huge volume of information is broke down utilizing the data mining (Kumar& Sharma, 2016; Kumar & Kumar, 2015). So, for saving cost, time and efforts, there is a need to focus on this area. (Kumar, Singla, Yadav, & Kumar, 2019) have noticed that detection of bugs using traditional ways is efforts intensive and costly. In this work, main focus was on awareness of important reasons of bugs in software application. The developer and the tester need to rationalize the coding efforts at an opportune time so that reasons causing bugs are essentially identified. Historical data is always for the prediction. The process of software development attaches high importance to prediction of software bugs. Several bug indicators as the model input can be used with prediction models for predicting the number of bugs. A comprehensive list of bug indicators using prediction models can significantly improve the quality of software products. There is a growing need for improving the quality of software applications with judicious utilization of bug prediction approaches. To avoid the possible harms of software products, bug severity needs to be identified. An attempt has been made for correlating levels of inheritance using the neural network with the severities of the bugs.

2. Objectives

1. To identify important reasons of software bugs that impact software quality.
2. To classify the software bugs w.r.t severity using Naïve Bayes, Decision Tree and Bagging approach.
3. To analyses the Bagging and Decision Tree approach for software bug classification.

3. Literature Review

First of all the in (Yuen, 1985) extended data over a time of 19 months and concerned bugs in a large software system have been gathered and, after classification in different ways, broke down utilizing both parametric and nonparametric factual strategies. Different tests were applied: auto-and cross-relationships between the announcing of bugs and the reactions to these were examined, time arrangement models were made, and so on Important experiences about the transformative conduct of large software during upkeep and about the support cycle itself were acquired. Expected value of the instruments utilized for checking and dealing with the cycle are talked about.

(Zhenmin , Lin, Xuanhui, Shan, Yuanyuan , & Chengxiang , 2006) successfully configuration instruments and backing for identifying and recuperating from programming disappointments requires a profound comprehension of bug qualities. As of late, programming and its advancement interaction have essentially changed from numerous points of view, including more assistance from bug identification apparatuses, move towards multi-stringing design, the open-source improvement worldview and expanding worries about security and easy to understand interface. Accordingly, results from past investigations may not be material to introduce programming. To explore the effects of the new factors on programming mistakes, investigate bug attributes by first testing many true bugs in two enormous, agent open-source projects. Author (Stefan, 2008) inferred that classifications are halfway comparable yet additionally contain varieties and none of which is acknowledged as a fundamental device in programming projects. The creator sums up the work on bug classifications up until now, proposes a bunch of difficulties and the course to an answer.

According to (Wahyudin, Ramler, & Biffl, 2011) various organizations have been using numerous statistical techniques for making the prediction. However, such type prediction has not been holistically examined, supported by different bug indicators and machine learning methods.

Mende and Koschke (Mende & Koschke, 2009) have stated that for increasing the accuracy, number of parameters to identify the bug reasons can be increased. In their study they have proved that increase in the number of parameters with data from relevant entities may illustrate statistically significant results. This study has included more number of parameters to identify main reasons of bugs and light is thrown on reasons attached to product, process and project. Author (Arya, Kumar , & Singh, 2020) proposed a model in which machine learning approach is used for bug prediction. (Q., Liu, & Illahi, 2020) stated that huge number of bug report frequently received by software systems. A methodology proposed in which initially apply common language preparing strategies to preprocess textual data of bug reports and transfer this data into vectors on the basis of semantic and syntactic associations. In the secondary phase, the software engineering area specific emotion analysis for bug report performed and using the software engineering domain repository emotion value is computed.

Chaturvedi et.al (Chaturvedi & Singh, 2012) had proposed text mining for the prediction of bugs. Machine learning algorithms SVM, J48, Naïve Bayes, rule based classifiers and Random Forest are compared for the purpose of bug detection.

Roy, N. K. S. et.al made an attempt to enhance the classification of bugs using the text mining approach . Feature selection and bi-grams are used and it has been observed the using the bi-gram, a little improvement in the performance can be achieved, however as the result also depend on the type of the project, so in some case use of the bi-gram may yield to bad results (Roy & Rossi, 2012). Neelofar et.al (Neelofar, Javed, & Javed, 2012) expressed that for a developer task of tracking the bug is very difficult and time consuming, as the developers are expert in a particular area. For example, some developers are expert in creating the interface for the end user and other functionality of any high level language. So it would be very helpful to for the developers if the particular bug is assigned to a developer, who is interested in it. Depending on the bug summary, bug have different labels.

The aim of bug tracking, according to Alenezi, M. et.al is to assigned a new bug report to the potentially skilled developers. An automated methodology to predict a developer with appropriate expertise to solve the new coming report was presented to minimize time and cost of bug triaging (Alenezi, Magel, & Banitaan, 2013).

Otoom, A. F. et.al (Otoom A. , Al-Shdaifat, Hammad, & Abdallah, 2016) focus on the issue of determining the seriousness of a defect repository. The primary goal was to create an intelligent framework that could predict the severity of a afresh submitted defect report using a bug tracking framework. To aid in predicting the magnitude of bugs, the proposed feature set was combined with powerful boosting classification algorithms. A new model has been proposed by Awad, M. A. et.al (Awad, Elnainay, & Abougabal, 2018) that incorporates classification strategies using Customized Cascading Weighted Majority Voting.

Otoom, A. F. et.al (Otoom A. , Al-Shdaifat, Hammad, , Abdallah, & Aljammal, 2019) said that the primary goal, according to the researchers, is to create an intelligent classifier adept of predicting the severity. Train a number of discriminative models based on this data that will be used for automated bug report labeling and severity prediction. When it comes to automatic marking, the precision is about 91 percent. Kumar, R. et. al (Kumar, Singla, Yadav, & Kumar, 2019) exposed the hidden information in the bug repository may be extracted using different data mining techniques. Ni, Z. et.al (Ni, Li, Sun, Chen, Tang, & Shi, 2020) aimed at spontaneously classifying defect into their root cause categories by using the associated connection among defect fixes and defect causes. To begin, the code-related bug classification criteria in terms of the bug's cause is defined. The results of the experiment show that there is an empirical connection between the bug fix and the cause of historical bugs, and that the prophecy is right.

Zhang, T. et.al (Zhang, Chen, Yang, & Zhu, 2020) expressed that data-driven and supervised models are not capable to provide the desired result if the sufficient training data is not available. An incremental learning based framework propsed to for tracking the unlabeled bug reports and for training augmentation method used.

Prabha, C. L., et.al (Prabha & Shivakumar, 2020) exposed that while contributing to industrial results the observable outcomes are provided by software defect prediction . The developers get the help in identifying bugs from development faults predicting defective code areas. Using the parameters precision, recall, accuracy etc., a research analysis is conducted and results are compared.

Most of the researches have worked for bug severity classification using machine learning, but still lot of work can be done in the for bug classification using machine learning

4. Need of Study

The researchers have highlighted the following needs after conducting an general review of existing research literature:

- The prevention of software failures needs to be achieved by paying more attention to potential bug- prone areas.
- The quality of the project needs to be improved through rigorous shortlisting of bug indicators.
- Almost one third of the total cost of software development is required in testing. So, with the help of the machine learning techniques for defect prediction cost may be decreased.

5. Naïve Bayes, Decision Tree and Bagging Classification Algorithms

5.1. Naïve Bayes:

This approach dependent on the Bayes' hypothesis. . Bayesian is normally known as Bayes Classifier. In this approach idea of likelihood is utilized to group the information things. It is essentially utilized for text characterization. In the arrangement issue there are number of traits and classes say, C1, C2, … ., CK. In this approach property estimations (a1,a2, … . an) portray each instance say x and from the arrangement of values qualities V, any worth can be taken by the objective capacity f(x). The Bayesian Classifier get the best plausible estimation of the objective capacity with the mean to characterize the new coming case (Kumar, Singla, Yadav, & Kumar, 2019).

5.2. Decision Tree:

A very simple method used for data mining is the decision tree method. Decision tree is appropriate for the exploratory knowledge discovery. The leaf node of the decision tree represents the class label. Decision tree in the above diagram shows the concept, how likely a customer buys a laptop from an electronic shop. Greedy approach is used for constructing decision trees in these algorithms (Kumar, Singla, Yadav, & Kumar, 2019). .

5.3. Bagging:

If dealing with a classification or regression problem, when we train a model, a function that takes an input, returns an output, and is represented with respect to the training dataset. To suit models that are nearly independent, we rely on the strong ‘approximate properties' of bootstrap samples. (Yates & Islam, 2021).

6. Analysis and Result

A comparative analysis of the Naïve Byes, Decision Tree and Bagging approach is done for the bug severity classification of the Jira bug repository. The data set of the bug is taken from the kaggle. A multiclass classification is done for the bug severity is done using Naïve Bayes, Decision Tree and Bagging approach. Different classes of the bug severity are mentioned in the below table.

Table 1. Severity Levels

Severity Level	Description
S5	Very High
S4	High

S3	Medium
S2	Low
S1	Trivial (may be ignored)

In the Table 1, five levels of the severity mentioned. In this paper bug severity classification is done for the severity levels S5, S4, S3 and S2 suing the Decision Tree, Naïve Bayes and Bagging approaches. Accuracy of 84%, 83% and 67% observed using Bagging, Decision tree and Naïve Bayes approach respectively as shown in the Figure 1.

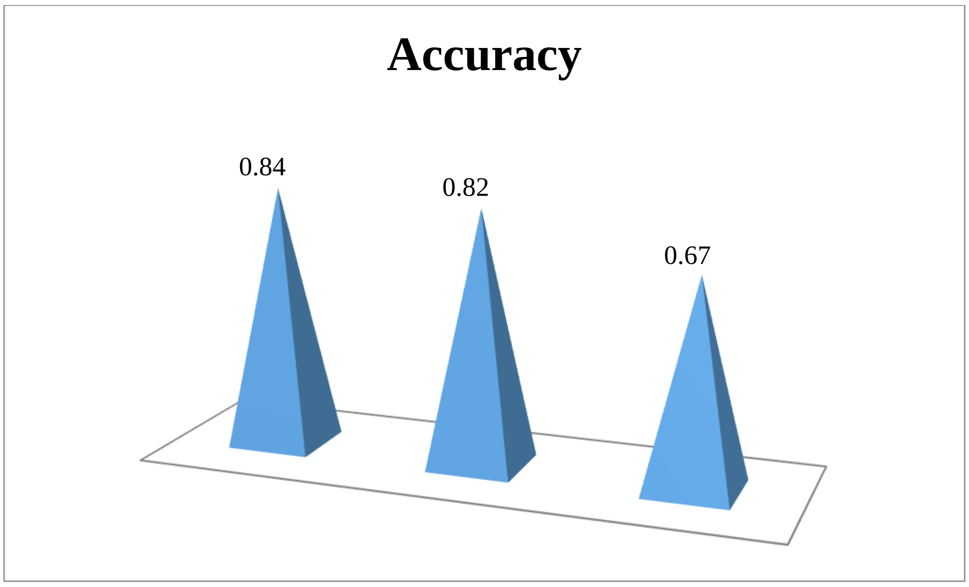

Figure 1. Comparison of Accuracy

A comparative analysis for the precision, recall and f-measure is also done as shown in the Table 2. It can be observed that precision of 83%, 81% and 52% has been observed for the Decision Tree, Bagging and naïve Bayes algorithms respectively. So with respect to the precision, it is observed that bagging approach is good.

Table 2. Precision, Recall and F-measure

	Decision Tree			**Bagging**			**Naïve Bayes**		
Severity	**Precision**	**recall**	**f-measure**	**Precision**	**recall**	**f-measure**	**Precision**	**recall**	**f-measure**
S2	0.7	0.88	0.78	0.69	0.88	0.78	0.56	0.66	0.61
S3	0.79	0.58	0.67	0.79	0.57	0.66	0.46	0.35	0.4
S4	0.97	0.94	0.95	0.97	0.95	0.96	0.95	0.85	0.9
S5	0.86	0.44	0.58	0.78	0.27	0.4	0.09	0.21	0.13
Average	**0.83**	**0.71**	**0.75**	**0.81**	**0.67**	**0.70**	**0.52**	**0.52**	**0.51**

A comparative analysis of precision and recall is shown in the Figure 6.Comparitive analysis of precision and recall is shown in Figure 1, Figure 2 and Figure 3 for the Decision Tree, Naïve Bayes and Bagging approaches respectively. It can be observed that with respect to the precision and recall the performance and decision tree and the Bagging approach is good as comparison of the Naïve Bayes approach.

All three algorithms also compared with respect to the f-measure parameter as shown in Figure 5, it can be clearly seen that Bagging algorithm can perform well bug severity classification.

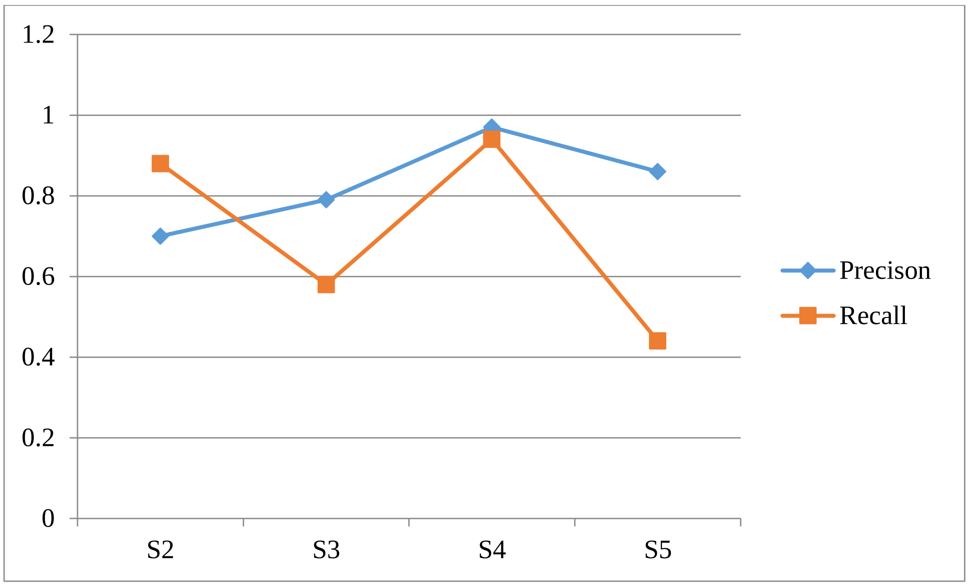

Figure 2. Precision and Recall for Decision Tree

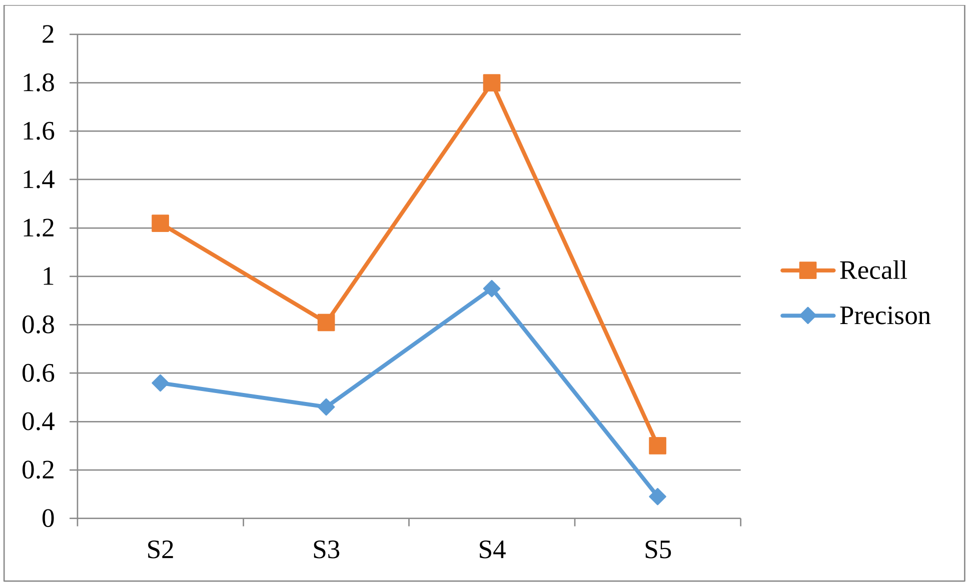

Figure 3. Precision and Recall for Naïve Bayes

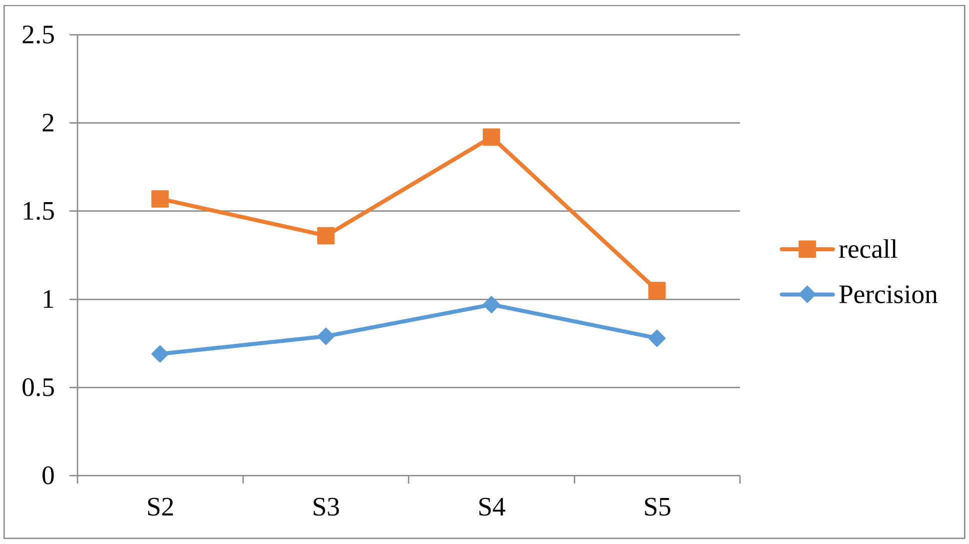

Figure 4. Precision and Recall for Bagging

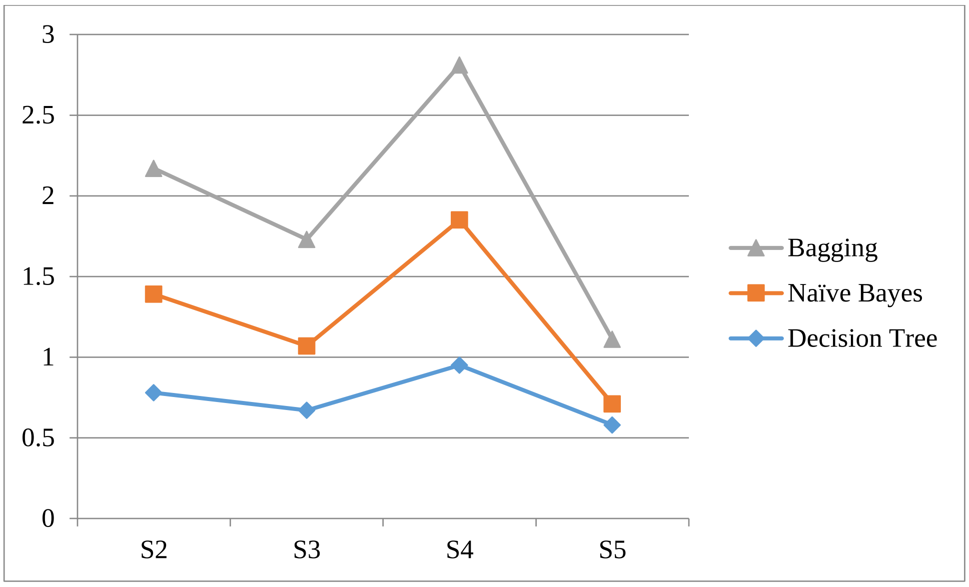

Figure 5. F-Measure for Bagging, Naïve Bayes and Decision Tree

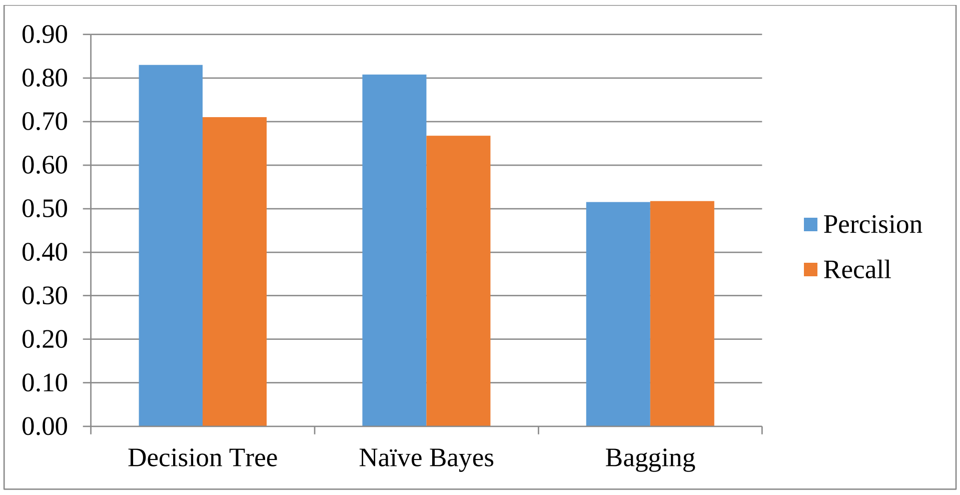

Figure 6. Comparison of Precision and Recall for Decision Tree, Naïve Bayes and Bagging

References

1. Alenezi, M., Magel, K., & Banitaan, S. (2013). Efficient bug triaging using text mining. *Journal of Software*, 2185-2190.
2. Arya, A., Kumar , S., & Singh, V. (2020). Prediction of defects in software using machine learning classifiers. *International Conference on Computational Methods and Data Engineering, ICMDE 2020*, (pp. 481-496).
3. Awad, M., Elnainay, M., & Abougabal, M. (2018). Predicting bug severity using customized weighted majority voting algorithms. *Proceedings of the Japan-Africa Conference on Electronics, Communications, and Computers, JAC-ECC 2017* (pp. 170-175). IEEE.
4. Chaturvedi, K., & Singh, V. (2012). An empirical comparison of machine learning techniques in predicting the bug severity of open and closed source projects. *International Journal of Open Source Software and Processes*, 32-59.
5. Kumar, R., & Kumar, M. (2015). Classification Rule Discovery for Diabetes Patients Using k-NN. *International Journal of Recent Research Aspects(IJRRA)*, 62-66.
6. Kumar, R., & Sharma, A. (2016). Comparative Analysis of SVM and kNN for Academic Prediction of Students. *International Journal of IT & Knowledge Management (IJITKM)*, 15-19.
7. Kumar, R., Singla, S., Yadav, R. K., & Kumar, D. (2019). An experimental analysis of various data mining techniques for software bug classification. *International Journal of Innovative Technology and Exploring Engineering*, 108-113.

8. Mende, T., & Koschke, R. (2009). Revisiting the evaluation of defect prediction models. *PROMISE '09: 5th International Conference on Predictor Models in Software Engineering*, (pp. 1-10).
9. Neelofar, Javed, M., & Javed, M. Y. (2012). An automated approach for software bug classification. *International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2012* (pp. 414-419). Palermo, Italy: IEEE.
10. Ni, Z., Li, B., Sun, X., Chen, T., Tang, B., & Shi, X. (2020). Analyzing bug fix for automatic bug cause classification. *Journal of Systems and Software*.
11. Otoom, A., Al-Shdaifat, D., Hammad, , M., Abdallah, E., & Aljammal, A. (2019). Automated labelling and severity prediction of software bug reports. *International Journal of Computational Science and Engineering*, 334-342.
12. Otoom, A., Al-Shdaifat, D., Hammad, M., & Abdallah, E. (2016). Severity prediction of software bugs. *International Conference on Information and Communication Systems, ICICS 2016* (pp. 92-95). Irbid, Jordan: IEEE.
13. Prabha, C., & Shivakumar, N. (2020). Software defect prediction using machine learning techniques. *4th International Conference on Trends in Electronics and Informatics, ICOEI 2020* (pp. 728-733). IEEE Xplore.
14. Q., U., Liu, H., & Illahi, I. (2020). CNN-Based Automatic Prioritization of Bug Reports. *IEEE Transactions on Reliability* (pp. 1341-1354). IEEE.
15. Roy, N., & Rossi, B. (2012). Towards an improvement of bug severity classification. *40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, (pp. 269-276).
16. Stefan, W. (2008). Defect classification and defect types revisited. *Workshop on Defects in Large Software Systems 2008, DEFECTS'08*, (pp. 39-40).
17. Wahyudin, D., Ramler, R., & Biffl, S. (2011). A framework for defect prediction in specific software project contexts. *3rd IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2008*, (pp. 261-274). Brno, Czech Republic.
18. Yates, D., & Islam, M. Z. (2021). FastForest: Increasing random forest processing speed while maintaining accuracy. *Information Sciences*, 130-152.
19. Yuen, C. (1985). EMPIRICAL APPROACH TO THE STUDY OF ERRORS IN LARGE SOFTWARE UNDER MAINTENANCE. *Conference on Software Maintenance* , (pp. 96-105). Washington, DC, USA.
20. Zhang, T., Chen, R., Yang, X., & Zhu, H. (2020). An uncertainty based incremental learning for identifying the severity of bug report. *International Journal of Machine Learning and Cybernetics*, 123-136.
21. Zhenmin , L., Lin, T., Xuanhui, W., Shan, L., Yuanyuan , Z., & Chengxiang , Z. (2006). Have things changed now?: An empirical study of bug characteristics in modern open source software. *ASID'06: 1st Workshop on Architectural and System Support for Improving Software Dependability*, (pp. 25-33). San Jose, California, USA.