IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-5, May 2019 E-ISSN: 2347-2693

A Machine Learning Based Bug Severity Prediction using Customized Cascading Weighted Majority Voting

Prachi Pundir [1], Satwinder Singh [2*], Gurpreet Kaur [3]

[1,2] Department of Computer Science and Technology, Central University of Punjab, Bathinda, India
[3] Assistant Professor, Department of Law, Bathinda College of Law, Bathinda, India

Corresponding Author: satwinder.singh@cup.edu.in , Tel.: +919300064064

DOI: https://doi.org/10.26438/ijcse/v7i5.13451350 | Available online at: www.ijcseonline.org

Accepted: 21/May/2019, Published: 31/May/2019

Abstract— As open source software systems are becoming bigger and more complex, the bug detection task and fixing it to improve the performance of the software is also getting complex, time taking, and inefficient. Users are permitted by the developers to report bugs that are found by them using a bug tracking system such as Bugzilla to improve the quality and efficiency of the software. In Bugzilla, users identify clearly the details of the bug, such as the description, the component, the version, the product, and the severity. Depending on this information, the priority levels to the reported bugs are assigned by the developers according to their severity. In this research, the model is proposed that is a customized version of a classification technique called "Customized Cascading Randomized Weighted Majority Voting". This technique will include an ensemble of two base classifiers: Naïve Bayes classifier and Random Forest classifier with different proposed weights in case of textual datasets.

Keywords— Eclipse, Priority Prediction, Severity Prediction, Machine Learning, Textual Analysis, Bugzilla, Jupyter Notebook.

I. INTRODUCTION

As the rapid increase in dependence on software systems, the importance of software quality is becoming more necessitous. There are different methods to ensure quality in software such as code reviews and rigorous testing so that bugs can be removed as quickly as possible to prevent the loss it may cause.

The bug is commonly described as the presence of a fault in a software system which results in it to act differently from its specified behavior. Due to system complexity and incomplete testing, many software systems are often released with defects. Bug reporting is a standard practice which involves the integration of source software with bug tracking/feedback system such as Bugzilla, Jira to keep track of the reported bug by the end user to improve the next releases. The number of responses is very large that it becomes nearly difficult to remove all the bugs (due to time constraints) and the software remains the same for many releases. In order to come over this, the process called bug triaging came into the picture. The collaboration of bug priority on the basis of bug severity is called bug triaging. The prioritization on the basis of severity forms the basis of the important attribute that describes the impact of a bug on the successful execution of the software product.

II. LITERATURE REVIEW

Bug fixing time is one of the important prospects in this research area. Kim & Whitehead et al. [1] did an experimental study to report important metrics such as bug fixing time, fixing time distribution, files with the highest bug fixing time and likewise. (2006)

Gujral and Sharma et al. [2] have explored the usage of the dictionary-based approach using text mining technique and Naïve Bayes Multinomial classifier in the classification of Bug severity. They have further employed an approach using a dictionary of bug terms to make the task more efficient. They have created the local dictionary of particular terms using TF-IDF with which severity can be predicted. The precision and accuracy level of 72% and 69% respectively have been calculated using the NBM algorithm. (2015)

Cubranic et al. [3] went beyond the prediction of bug severity and automated the process of assigning the bug to the concerned developer. He trained a Naıve Bayes classifier with the history of the developers who resolved the bugs as the category and the corresponding explanations of the bug reports as the data. This classifier is subsequently used to predict the most appropriate developer for a newly reported bug. It was found to be over 30 % accurate with assigning of the incoming bug reports of the Eclipse project to a correct developer using this approach (2004).

Similarly, Anvik et al. [13] used supervised learning algorithms and labeling heuristic approach on different datasets of Eclipse and Firefox. The precision level was found to be 57% and 64% respectively. (2006)

Yang and Zhang et al. [11] proposed a novel method for bug triage and bug severity prediction. They tried extracting topic(s) from historical bug reports in the bug repository using NLP and tokenization and find bug reports related to

each topic. When a new bug report arrives, they decide the topic(s) to which the report belongs. Then utilizing the multi-feature to identify corresponding reports that have the same multi-feature (e.g., component, product, priority, and severity) with the new bug report. Thus, given a new bug report, we are able to recommend the most appropriate algorithm (depending upon the frequency of topic) to fix each bug and predict its severity. To evaluate the approach, they had measured the effectiveness by using about 30,000 golden bug reports extracted from three open source projects (Eclipse, Mozilla, and Netbeans). The results show that the defined approach is effective to recommend the appropriate algorithm to fix the given bug and predict its severity. (2014)

Pushpalatha and Mrunalini et al. [9] used a bagging ensemble method for predicting the severity of Bugzilla bug reports repository. They further analyzed the results and compared with C4.5 classifier. Bagging, known as Bootstrap aggregating technique creates the composite model N* which reduces the variance of the individual classifier by combining a series of N learned models, N1, N2…Nn. Whereas, on the other hand, C4.5, also known as J48 on WEKA data mining tool, is based on the Decision Tree algorithm. The results have shown that bagging ensemble method gives better accuracy (81%) as compared to C4.5 general classifier (76%) on the given dataset. (2016)

Menzies and Marcus et al. [8] used textual analysis methods like tokenization, stop word removal and stemming. Important tokens are then identified using term frequency-inverse document frequency (TF-IDF).These tokens are then used as features for technique named SEVERIS (Severity Issue assessment) on NASA datasets to predict the bug severity.(2008)

Ahmed Lamkanfi et al. [7] proposed a new method for classifying bugs based on the basis of severity. The usage of text Mining Algorithms along with Machine Learning Algorithm is effective in increasing the efficiency of the Bug Severity prediction model. Bug reports from Eclipse, GNOME, and Mozilla were preprocessed with text mining algorithms (tokenization, stop word removal, stemming). After that, Naïve Bayes was applied, and the average precision and recall of Eclipse and Mozilla was 0.65-0.75 and 0.70-0.85 respectively in case of GNOME. (2011)

Israel Herraiz et al. [12] analyzed the bug reports of Eclipse. It was concluded that this bug report has too many options for severity and priority field. Severity levels can be reduced to three levels as important, non-important and request for enhancement based on the time taken. Similarly, the priority field in bug reports was grouped into high, medium, low, according to mean time taken to close the bug. (2008)

Lamkanfi et al. [6] provided a comparison of several classifiers to classify bugs issues as severe, non-severe. He compared Naive Bayes (NB), Naive Bayes Multinomial (NBM) and Support Vector Machines (SVM) to evaluate the performance of the classifiers within Eclipse and GNOME projects. Moreover, he reduced the lower bound for the number of bug reports for accurate prediction to 250 reports, necessary for NB and NBM to start providing accurate predictions. Overall, the best classifier selected for each component was NBM with an average precision to vary within 0.59 to 0.93. NBM was generally faster and more efficient in the prediction process. (2010)

Ahsan S, Ferzund J, Wotawa F et al. [5] employed Naïve Bayes, RBF network, Random Forest, SVM and J48 algorithms on Mozilla datasets and compared precision values in the bug severity prediction.(2009)

Sharma and Rana et al. [10] proposed a feature selection and classification approach for categorizing the bug reports into severe and non-severe class. Further, they used the output to create a dictionary of critical terms of severity indicator. Top 125 dictionary terms are selected and used as dictionary terms to train a classifier using feature selection methods such as info – gain and Chi-square. Further, the author compared the performance of NBM and KNN. It was found that accuracy, precision is in the range of 64 % to 75 % and 66 % to 74 % using Naïve Bayes classifier and using KNN classifier, it is in the range of 87 % to 91 and 79 % to 95 % respectively. (2015)

III. OBJECTIVE

The objectives of this research are as follows:

1. To study more than a thousand bug reports over various versions of Eclipse and to apply the data pre-processing techniques such as tokenization, Stop-words removal process and stemming on the dataset.
2. To study various machine learning algorithms and finalize two (Naïve Bayes and Random Forest) to create a prediction model by using these algorithms
3. To compare and analyze the accuracy of these models.

IV. METHODOLOGY

4.1 BUG DATA COLLECTION:

The bug reports of Eclipse project are used for the experiments. Eclipse projects focused on various metrics such as building extensible frameworks, tools, and runtimes for building and managing software across the life cycle phase. In this research work, Eclipse bug data reports from Bugzilla (bug repository) have been extracted. It has many products and components. The versions taken into account are 2.0, 2.1, 3.0,and 3.1. There are 5 main products of Eclipse of which we focus on 3 which are JDT, PDE, and Platform. The task of bug data collection was recorded during the research work manually. The factors that were taken into interest during the data collection are bug id, summary, long description, product, component, resolution, status, opened time, close time, severity, author and priority.

4.2 DATA PRE-PROCESSING:

Text processing is a major stage while working with textual data in order to test and evaluate the accuracy of performance

of the proposed model. It is accomplished to extract textual attributes from the bug report attributes such as summary and long description. Because the summary of bug report gives a better description of a bug report, it is included for priority and severity prediction. There are a number of words in the text attributes that contain meaningless information and increase dimensionality, thus decreasing the performance of the model. Therefore, standard text categorization approach is used to transform the text data into meaningful representation. It is done by applying the text pre-processing techniques, which include tokenization, stop-word removal, and stemming.

Various code is being written in Jupyter notebook to carry out Text-Preprocessing.

4.3 CUSTOMIZED CASCADING WEIGHTED MAJORITY VOTING:

This technique depends on having an ensemble of multiple base classifiers that fed their individual results into a number of learners; a learner per each severity category. For each learner, each base classifier votes with different weight, i.e., a particular bug can be critical for one of the classifier, but major for the other classifier and so on. Thus, the final output is concluded using Majority Voting model. The higher the combined weightage for the bug as a particular type of severity, the more the chances of declaring it the same at the end of the experiment. We would be using Random Forest and Naive Bayes Multinomial classifiers on textual datasets. The final contribution of this study is to design a generic classification framework that can perform well with textual datasets and also compare the result between NBM, RF, and CCWMV.

4.4 EVALUATION TECHNIQUE:

Evaluation of the proposed technique is based on two performance metrics: F-measure and Accuracy. Traditionally, the accuracy rate has been the most commonly used empirical measure. However, in the framework of imbalanced datasets (used here), accuracy is no longer a proper measure. Hence, it may lead to erroneous conclusions. Therefore, F-measure is used.

F-measure can be defined as the weighted harmonic mean of precision and recall and can be computed as follows.

$$F\ measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (1)$$

$$Precision = \frac{TP(True\ Positive)}{TP+FP(False\ Positive)} \quad (2)$$

$$Recall = \frac{TP}{TP+FP(False\ Positive)} \quad (3)$$

Where

F-measure is also used to define the weightage to the particular severity type of the bug.

V. RESULTS AND DISCUSSIONS

As already discussed, the experiments were conducted with different settings in Weka. The experiment accomplished by using 70 percent of the dataset for training and 30 percent for testing purpose. The results which are obtained are listed in Table 1 and Table 2

Table 1 show precision, recall and F-measure for each bug severity class for different versions of Eclipse projects and their corresponding calculated accuracy for the bug report features for Naïve Bayes classifier.

Experiments were performed on 4 versions of Eclipse using textual datasets and different algorithms, namely Naïve Bayes, Random Forest & proposed technique (CCWMV) with 2 base classifiers (RF & NB). It is easily deducible from the Table 1 that the Random Forest model predicts the severity levels with an accuracy of 86.5% for version 2.0, 72% for version 2.1, 64.1% for version 3.0, 84.7% for version 3.1. If we go by severity level, then Naïve Bayes is efficient in predicting minor severity with the utmost accuracy and average precision of about 86%.The reason behind it is the presence of minor severity level in abundance in our bug reports dataset with a maximum number of instances. Given the imbalanced textual dataset, the Naïve Bayes based classifier is also performing efficiently in classifying different levels of severity with f-measure values (combined metric of precision & recall) ranging between 0.71 to 0.86.

Table 1. Severity prediction models of Naïve Bayes for Eclipse version 2.0, 2.1, 3.0, and 3.1

Version	Algorithm		Severity	Precision	Recall	F-measure	Accuracy
Eclipse 2.0	Naïve Bayes		Major	0.952	0.833	0.889	86.597%
			Minor	0.786	0.688	0.733	
			Normal	0.947	0.857	0.900	
			Blocker	1.000	0.947	0.973	
			Critical	0.680	1.000	0.810	
		Wt. Avg		0.884	0.866	0.868	
Eclipse 2.1			Major	0.125	0.286	0.174	72%
			Minor	0.895	0.850	0.872	
			Normal	0.000	0.000	0.000	
			Blocker	0.286	0.235	0.258	
			Critical	0.000	0.000	0.000	
		Wt. Avg		0.754	0.720	0.735	
Eclipse 3.0			Major	0.745	0.798	0.771	64.1026%
			Minor	0.957	0.643	0.769	
			Normal	0.349	0.526	0.420	
			Blocker	0.046	0.692	0.087	
			Critical	0.143	0.182	0.160	
		Wt. Avg		0.851	0.641	0.717	
			Major	0.923	0.781	0.846	
			Minor	0.821	0.636	0.717	
Eclipse 3.1	Naïve Bayes		Normal	0.883	0.807	0.833	84.705 %
			Blocker	0.941	0.884	0.841	
			Critical	0.671	0.990	0.962	
		Wt. Avg		0.847	0.819	0.836	

Table 2. Severity prediction models of Random Forest

Version	Algorithm	Severity	Precision	Recall	F-measure	Accuracy
Eclipse 2.0	Random Forest	Major	0.557	0.348	0.430	70.9%
		Minor	0.519	0.507	0.516	
		Normal	0.867	0.911	0.877	
		Blocker	0.553	0.126	0.202	
		Critical	0.360	0.839	0.509	
		Wt. Avg	0.712	0.709	0.691	
Eclipse 2.1	Random Forest	Major	0.911	0.842	0.875	84.2051%
		Minor	0.802	0.561	0.660	
		Normal	0.838	0.829	0.833	
		Blocker	0.750	0.958	0.841	
		Critical	0.935	0.990	0.962	
		Wt. Avg	0.845	0.842	0.836	

Eclipse 3.0	Random Forest	Major	0.000	0.000	0.000	81.0588%
		Minor	0.545	0.862	0.670	
		Normal	0.943	0.962	0.954	
		Blocker	0.163	0.056	0.087	
		Critical	0.667	0.313	0.421	
		Wt. Avg	0.789	0.811	0.691	
Eclipse 3.1	Random Forest	Major	0.915	0.741	0.819	82.5%
		Minor	0.673	0.614	0.642	
		Normal	0.745	0.844	0.792	
		Blocker	0.840	0.932	0.883	
		Critical	0.934	0.966	0.950	
		Wt. Avg	0.827	0.825	0.823	

The Severity report, as the name guessed, is suggestive that the model is capable of differentiating ranging between major severity bugs to normal severity bugs efficiently. If the bugs with normal severity do not distinguish from are not major or blocker severity level with high accuracy by the classifier, the fundamental purpose of time efficiency of the developers will not be served, and the developers will not be able to comprehend as to solve which bug and which bug can be kept aside. As a result, the time and man-hours of the developers will be utilized in the wrong direction, thus decreasing efficiency and productivity. Solving blocker, major severity level bugs is critical from the viewpoint of any company too as the whole product will be corrupted and thereby decreasing the market brand of that company.

Table 3. Severity Prediction Model for CCWMV

Eclipse Version	CCWMV Accuracy	F-Measure
Version 2.0	87.4	0.7095
Version 2.1	83.8	0.7355
Version 3.0	78.5	0.6495
Version 3.1	86.1	0.8045

This categorization is somewhat solved as our classifier can distinguish between major, minor & normal with quite a good accuracy and developers can decide which bug to be addressed.

The algorithm shows lesser accuracy while classifying normal and critical severity types of bugs. One of the causes behind it is the less number of instances for the above-mentioned types may be due to 70-30% division of dataset. There may be miss-classification of severity types which may be one of the causes of less accurate performance.

Table 2 shows Random Forest classifier for predicting severity levels with an accuracy of 70.9% for version 2.0, 84.2

% for version 2.1, 81.05% for version 3.0, 82.5% for version 3.1 respectively. Again, if we talk of the f-measure (harmonic mean of precision and recall), the F-measure for Minor, Normal & Critical severity types of version 2.0 is comparably better than Major and Blocker severity types. Similarly, the F-measure for Major, Normal, Blocker, Critical severity types of version 2.1 is comparable among themselves and is better than Minor severity types. The F-measure of version 3.0 for Normal severity types is excellent and good for comparison with Major, Minor, Blocker, and Critical. Similarly, in this version, Minor can be easily distinguished from Major as well as Blocker. The f-measure in this version is poor for Blocker as well as Major severity types. The F-measure of version 3.1 is comparable between Critical and Blocker with the maximum value for Critical severity type while f-measures for all others such as Major, Minor & Normal severity types are good.

This research paper hovers around the prediction of bug severity on the basis of various severity types (Major, Minor, Normal, Blocker & Critical) based on a CCWMV (Customized Cascading Randomized Weighted Majority Voting) technique. The proposed model has been assessed and compared with related works by other classification algorithms such as Random forest & Naïve Bayes. It can be easily inferred from Table 3 that CCWMV technique has better accuracy in predicting the types of severity in Eclipse version 2.0, 2.1, 3.0, 3.2 over the Naïve Bayes but Random Forest has the better performance in predicting the bug severity in eclipse version 2.1 and version 3.0 with the accuracy of 84.2% & 81.05% as compared to 83.8% & 78.5% respectively in case of CCWMV model. However the percentage accuracy in versions 2.0 & 3.1 is high in case of CCWMV with 87.4% & 86.1% over 70.9% & 82.5% in case of Random forest model.

On the same note, the f-measure (0.70) in version 2.0 in case of CCWMV is better than that of Random forest (0.691), thereby CCWMV is more accurate in distinguishing between various types of severity than Random forest. The f-measure in version 3.1 is comparable to that of Random forest; thus, both of the models are equivalent in classifying bug into various severity types. The main reason for low performance in version 2.1, version 3.0 might contribute to the small-size dataset in the respective cases. It can be enhanced by using

the different set of base classifiers with the usage of statistical classifiers too.

Graphical representation of evaluated Accuracies for various versions of Eclipse for Naïve Bayes, Random Forest, and CCWMV:

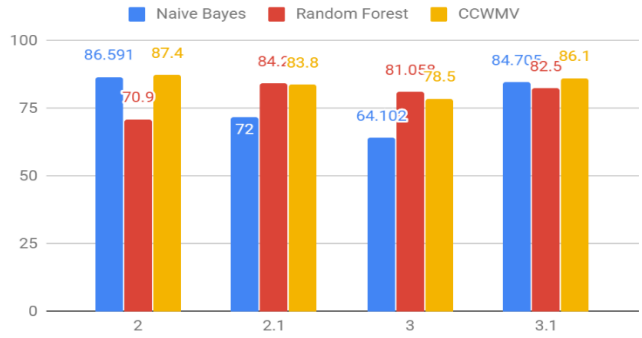

Figure 1. Accuracy of each version for Naïve Bayes, Random Forest and Customized Cascading Weighted Majority Voting Algorithm

VI. CONCLUSIONS AND FUTURE SCOPE

In this paper, the experiments were performed to predict the severity levels of bug reports of Bugzilla. These bugs affect the trustworthiness and quality of software. Bug tracking systems allow users to report bugs that are introduced when they use the Eclipse platform that is an open source software. However, predicting the priority and severity level of these bug reports is an arising issue. The factors that were considered for the experiments include temporal, textual, author-related, severity and product. All these features are fed to the Naïve Bayes and Random Forest successively using the CCWMV after textual analysis to classify the issues with bug reports and to predict the severity levels to bug reports. The text preprocessing techniques refine the most useful terms from datasets.

The prediction models for chosen techniques are developed and compared with the CCWMV results.

REFERENCES

[1] S. Kim, E. J. Whitehead, *"How long did it take to fix bugs?"*, MSR '06 Proceedings of the 2006 international workshop on Mining software repositories, Shanghai, **China**, pp.**173-174**, **2006**.

[2] S. Gujral, G. Sharma, *"Classifying Bug Severity Using Dictionary Based Approach"*, International Conference on Futuristic trend in Computational Analysis and Knowledge (ABLAZE 2015), Noida, **India**, pp.**632-639**, **2015.**

[3] D. Cubranic, G. C. Murphy, *"Automatic bug triage using text categorization"*, 16th International Conference on Software Engineering, **Italy**, pp.**92-97**, **2004**.

[4] V. Challagulla, F. Bastani, I.-L. Yen, R. Paul, *"Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques."*, 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems., Arizona, **USA**, pp.**263-270**, **2005**.

[5] S N Ahsan , J. Ferzund , F. Wotawa, *"Automatic Software Bug Triage System (BTS) Based on Latent Semantic Indexing and Support Vector Machine"*, Proceedings of the 2009 Fourth International Conference on Software Engineering Advances, **Portugal**, pp.**216-221**, **2009.**

[6] Lamkanfi, S. Demeyer, E. Giger, B. Goethals, *"Predicting the severity of a reported bug."*, 7th IEEE Working Conference Mining Software Repositories (MSR), **South Africa**, pp.**1-10, 2010.**

[7] Lamkanfi, Ahmed, et al. *"Comparing mining algorithms for predicting the severity of a reported bug"*, 15th European Conference on Software Maintenance and Reengineering (CSMR), **Germany**, pp.**249-258**, **2011**.

[8] T. Menzies, A. Marcus, *"Automated severity assessment of software defect reports,"* in IEEE International Conference on Software Maintenance, **China**, pp.**346–355, 2008.**

[9] M.N. Pushpalatha, M. Mrunalini, *"Predicting the Severity of Bug Reports using Classification Algorithms"*, 2016 International Conference on Circuits, Controls, Communications and Computing (I4C), Bangalore, **India**, pp.**520-525**, **2016**.

[10] S. Sharma, P. Rana, *"Implementing Bug Severity Prediction through Information Mining using KNN Classifier*", International Journal of Science Technology & Engineering, Vol. **2**, Issue **4**, pp.**333 – 340**, **2015**

[11] G. Yang, T. Zhang, *"Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-Feature of Bug Reports"*, 2014 IEEE 38th Annual International Computers, Software and Applications Conference, **Sweden**, pp.**97-106**, **2014**.

[12] Herraiz, D. German, J. Gonzalez-Barahona, G. Robles, *"Towards a Simplification of the Bug Report Form in Eclipse,"* in 5th International Working Conference on Mining Software Repositories, **Germany**, pp.**145-148,** May **2008**.

[13] J. Anvik, L. Hiew, and G. Murphy, *"Who should fix this bug?"* in Proc 28th International Conference on Software Engineering. ACM, **China**, pp.**361–370**, **2006**.

Authors Profile

Prachi Pundir pursued Bachelor of Technology in 2015 from University Institute of Engineering and Technology. She is currently persuing Masters of Technology in Computer Science and Technology from Central University of Punjab, Bathinda,Punjab, India. She is currently working in the area of Data Science.

Dr. Satwinder Singh had completed his Ph.D in 2014 from Guru Nanak Dev University, Amritsar. He is currently working as an Assistant Professor at Department of Computer Science and Technology, Central University of Punjab, Bathinda, Punjab, India, He has 15 years teaching experience. He has published research papers in reputed journals and conferences. His research interests include Re-engineering of Software System , Maintenance and Fault prediction of Object Oriented Systems, Big data analytics and Text Data Analytics

Mrs. Gurpreet Kaur had completed her Ph.D in 2016 Punjabi University, Patiala. She is currently working in Department of Law, Bathinda College of Law, Bathinda, Punjab, India. She has 11 years of teaching experience. She has expertise in Criminal Law and International Law.

