Developing Classifiers by Considering Sentiment Analysis of Reported Bugs for Priority Prediction

Anisha Singh
Jawaharlal Nehru University
P.K. Kapur
Amity University
V.B Singh (vbsingh@mail.jnu.ac.in)
Jawaharlal Nehru University https://orcid.org/0000-0002-9601-7039

Research Article

Keywords: Bug reports, classification, machine learning techniques, priority prediction, software maintenance, sentiments

Posted Date: May 25th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2933283/v1

Version of Record: A version of this preprint was published at International Journal of System Assurance Engineering and Management on November 23rd, 2023. See the published version at https://doi.org/10.1007/s13198-023-02199-2.

Developing Classifiers by Considering Sentiment Analysis of Reported Bugs for Priority Prediction

Anisha singh[1], P.K. Kapur[2], V.B Singh[1]

[1]School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, India
[2]Centre for Interdisciplinary Research, Amity University, Noida, UP

Singh2012anisha@gmail.com
pkkapur1@gmail.com
vbsingh@mail.jnu.ac.in

Abstract

Software systems behave abnormally due to bugs when it comes into the operational phase. Lack of proper understanding of customer requirements, implementation, knowledge, wrong algorithmic designing, and other issue is also the reason for bug production. To fix those flaws, developers request to the users for feedback. Users have had issues with the software systems that have been released. Users are encouraged to submit their issues to issue-tracking systems such as Bugzilla, Mantis, Google Code Issue Tracker, GitHub Issue Tracker, and Jira to improve the next version of the product and meet user needs. Manual prioritization is time-consuming and inconvenient. In this research paper, we propose using sentiment analysis to anticipate the report's priority. This is the first time the sentiment-based approach used for a bug report to prioritize prediction on open-source projects. First, we take the bug report summary and use natural language pre-processing techniques to clean the text and pre-process the bug report. Second, sentiment analysis is applied to clean texts that contain sentiments of terms. Third, we use TF-IDF to construct a feature vector for bug reports, fourth, we used resampling techniques to balance the dataset, and then we used different machine learning classifiers to train historical data namely Bugzilla open-source projects to forecast their priority. The proposed method improves the performance of the classifier in comparison with the methods available in the literature.

KEYWORDS: Bug reports, classification, machine learning techniques, priority prediction, software maintenance, sentiments

1 Introduction

Software maintenance is most important for software development, and it takes 70% to 80% cost and time [23][24].Bugzilla [17], Mantis [18], Google Code Issue Tracker [19], GitHub Issue Tracker [20], and Jira [21] are the open-source issue-tracking systems. Users can submit their issues in these open-source issue tracking systems. Nowadays, resolving bug reports has become more and more challenging and expensive [22]. Bug reports are increasing at an exponential rate [24] as the size and complexity of software are increasing. Bug priority prediction helps developers to prioritize and fixed first as per their priority. The manual prioritization of bug reports can take more time. In manual process, the assignee who has been assigned to solve the given bug report, their workload and bug fixing time increases [28]. Numerous methods have been used for automatically ranking the bugs according to priority and for predicting bug report priorities using machine learning techniques [1][3][29].
According to the frequency and entropy of the crashes, Chaudhary's [49] model for priority prediction using SVM assigns precedence to reports of Firefox crashes in the Mozilla Socorro server. In an effort to improve performance, Chaturvedi and Singh [36] used a variety of classifiers, including nave bayes, multiple nave bayes, support vector machines, k-nearest neighbour, j48, and ripper techniques, to assess the severity of reported bugs for the NASA dataset's closed source bug repository. By leveraging the closest neighbours to determine fine-grained bug report labels, Tian et al. [50] predicted the priority of bug reports. A broader group of bug reports were subjected to the purposed approach made up of more than 65000 Bugzilla reports. Alenezi and Banitaan[13] used a brand-new naive Bayes, decision tree, and random forest to carry out the priority prediction. The two feature steps were employed namely TF-weighted terms followed by the attributes used to classify bug reports. According to the evolution results, the second feature set, which included decision trees and random forests, outperformed naive Bayes and performed better than the first feature set. To prioritize bug reports, Tian et al. [3]

presented the automatic classification approach DRONE. Priority classification was done using linear regression (LR). In order to prioritize and fix bugs based on their meantime, Abdelmoeg [37] employed a naive Bayes classifier to create a model that could anticipate bug reports. Using multi-nominal naive Bayes, Dommati et al. [38] concentrated on feature extraction and noise reduction of bug reports. From 2000 to 2015, a survey and automated article utilizing prioritizing were undertaken by Uddine et al. [12]. To automatically assign priorities to reported defects, Kanwal and Maqbool [1] created a recommender system based on machine learning. On the basis of reports of eclipse bugs, they used SVM to train their classification-based methodology. Precision, recall, and F1-score are used in the evolution of purpose recommenders to assign priority to bugs automatically. For determining the priority of bug reports, many methods are available. Singh and Chaturvedi [46] used BTRAS to provide centralized system for bug tracking, management and statistical analysis to identify pattern and trends in bug reports to identify common issues and prioritize bug and fixes them. By giving each bug report an emotion value based on the language used to express those emotions in the bug report description, Qasim Umer et al. [6] expand upon a drone and enhance its functionality. Sharma et al. [47] models to determine the priority of a reported bug were developed and evaluated using data collecting, feature engineering, machine learning algorithms, cross-project validation, and performance measures, receiver operating characteristics (ROC), and area under the curve (AUC). open office and eclipse projects' five data sets were cross validated for 76 instances. Except for the Naive Bayes technique, all machine learning techniques have accuracy levels of above 70% when it comes to forecasting the priority of a reported bug both inside and between projects. To improve the results (f-measure and accuracy), Singh et al. [52] automate the bug severity prediction using the two ways of ensemble by using two operators accessible in rapid minor vote and bagging. Singh and Chaturvedi [48] compared the results of the proposed models with those found in the literature, they discovered that the proposed models are effective bug predictors because they have demonstrated a significant improvement in their performance. They proposed an entropy-based bug prediction approach using support vector regression (SVR). The software code becomes complex as a result of frequent updates brought on by bugs. Entropy measurements have already been used to determine the complexity of the code changes.

The motivation of the proposed work is to decrease the workload of the assignee and time, cost also and to determine which bug report is necessary to fix first. We want to learn a model that can predict the priority labels of every new bug report. Our proposed sentiment-based approach is used for the bug report to prioritize prediction on open-source projects. The classifiers have been built using sentiment analysis and machine learning techniques for bug priority prediction.

This is the first time; a sentiment-based approach has been used for bug priority prediction. Thus, when a new bug report is received, the model can be used to recommend the priority label of the new bug report this bug and outline weight are newly proposed. The classification and regression techniques used include Ordinal regression, SVM (Support vector machine), kNN(k-nearest neighbor), NB(Naive bayes), Multi linear regression to build models[6]. SVM gives best performance for 5 datasets out of 8 in terms of precision for severity prediction[6]. We discovered many techniques that were suggested or utilized in research papers as a result of our study. They also use different datasets from Gnome, Bugzilla, JIRA, Mozilla, Eclipse etc. We explain the most common techniques that are used to predict bugs in fix-time.

2 PROPOSED APPROACHES

Firstly, we have decided which attributes are considered for priority prediction of bug reports. priority is considered as a dependent attribute, or we can say that the target attribute rather than priority others are considered as an independent attribute like severity, severity quantity, and summary of a bug report [17][18][19][20][21]. In summary of the bug report as an independent attribute, we have to apply the pre-processing [2] technique on the summary of the bug report tokenization (break sentences into text named as tokens) lemmatization [51], stop removal, POS tagging, and after that clean texting. Then we applied sentiment analysis and after that feature extractions by TFIDF and then applied the resampling techniques smote synthetic minor over-sampling technique. The data set we have taken is an imbalanced dataset because of the imbalanced dataset we use an over-sampling technique smote which is creating a bar chart that shows priority levels P1 P2 P3 P4 p5 and their imbalanced distribution. In this research paper, we have applied different machine learning techniques like support Vector machine, decision tree, KNN, and naïve Bayes. According to the result, the SVM gives better results other than those else.

3 DATASETS

Data set which chooses four well-known products are general, calc, build tool, and base for each of access and generalization we have collected those data sets from Bugzilla open source in the given table, so these are the following products their reports gathered from all the respective four products which we took into consideration.

Products used	General	Calc	Build tool	Base

Products are taken from Bugzilla.

4 PREPROCESSING

Pre-processing is a natural language processing technique. standard the pre-processing used natural language processing tool kit and NLTK [2] includes tokenization, lemmatization [51], stop word removal, and pos tagging.

5 SENTIMENT CALCULATION FOR BUG REPORT

Data set which is in form of text widely used to perform how user and developer interact with software examining text based on feedback is just a superior option to compare the actual response. we can use the NLP approach to detect patterns in the text to evaluate it and determine the sentiment behind it after pre-processing and cleaning the text calculating sentiment scores. The different type of sentiment which we have got is useful for this research paper. The score of the text is positive which means it reflects positive sentiment and vice versa. There are numerous ways available for determining sentiment scores we have applied in this paper the Vader sentiment intensity analyzer using the Vader lexicon sentiment analyzer used in this manner sentiment of text calculated[4][5][32] by Vader also evaluates whether a text is favorable, neutral, or negative other than compound the compound score will be used in the cases, where the sum of all the word in the lexicon a string lied between –1 to 1 made the compound score which is the sum of each word score when added together the result of weather could be impacted if generic test preparation procedure is used for calculation and further organization. The sentiment score is called polarity.
The formula is used in the Vader sentiment intensity analyzer to find out the score of sentiments.
Sentiment score = polarity = [round (sent.polarity scores(i)['compund'],2]].

6 FEATURE EXTRACTION

Machine learning cannot access text directly. Before applying a machine learning algorithm, we have to convert text data into a numerical feature vector. There is two widely used technique are available for extracting features. TFIDF (term frequency-inverse document frequency) [6] and BOG (bag of words) [25][26]. There is some issue with this bag of words such as some less important words but most frequently used means the number of counts is high that is the possibility other than less frequent but important words known as find not found exception and runtime exception which the number of counts is low. In this paper, we have used TFIDF. TFIDF is strong and concentrated [6] on words that are frequently used in the text but not in every document TFIDF. Here documents are transferred into a matrix of TFIDF features by the TF-IDF vectorizer. Collection of a word that we got after preprocessing. TFIDF has correct characteristics to count every single word and how many times appeared in the collection based on characteristics TFIDF [8][9]. Tfidfvectorizor creates a large dimension sparse matrix. SKlearn library is used for sparse matrix format. it stores only non-zero values of words basically TFIDF turns text data into numerical form.
This SKlearn formula used. tfidf = $\log\left(\frac{N}{df}\right) * \ln\left(\frac{1+M}{T}\right)$,

Where N number of occurrence words in the data set. df total number of documents in the collection. M number of times words occur in this document. T total number of words in this particular document. SKlearn TF-IDF vectorizer formula applied Iteratively on each document. For balancing the data set resampling technique [30][31] is used a machine learning algorithm that is based on classification as well as regression they are very sensitive across imbalance or uneven data sets a small range of data sets disgraded due to uneven data set as per situation data balancing is most needed technique. Recently technique is a widely approved method for balancing the date of the set. There are many recent techniques for balancing the data set under-sampling oversampling and hybrid sampling. Here in this paper priority distribution is uneven way. So, we have applied SMOTE (synthetic minority over-sampling technique) [30][31] for balancing the uneven priority distribution.

7 TRAINING AND TESTING

There are two main steps training and testing used in machine learning algorithms in the classification model. The feature extraction model generates feature vectors. In this suggested approach we continue with SVM, KNN, Decision Tree, Naive Bayes algorithms, and interaction between extracted features and priority level of bug reports. Now we started from SVM first the following justification that how we classify data using support vector machine the kernel method also allows here for a scale to high dimensional data sets for completed in the first place. Second, there is explicit controllability over the trade-off between classifier complexity and error. Third, we apply resampling methodology for presenting unbalanced dataset means for balance distribution of imbalance distribution of priority level. When we get the bug report feature set then apply it to train the data by SVM classifier. The trained model is used to forecast the priority of problem reports. The second popular supervised

machine learning technique KNN predicts the class level of a new data point by using the idea of near proximity here if we consider x as a training data set with an n number of data points. And y is the test dataset, and we use the Manhattan or equilibrium distance function to determine the distance between each row of the data set between the test and train data set. The test data set will receive based on a class level like closest neighbors k data points. Thirdly we used a decision tree also known as a rooted directed tree. And one more machine learning classifier naive Bayes used naïve Bayes based on the Bayes theorem and total probability theorem.

$(\boldsymbol{p}(\frac{\boldsymbol{n}}{\boldsymbol{m}})$ = **P (n)**$\boldsymbol{p}(\frac{\boldsymbol{m}}{\boldsymbol{n}}))$

For the effective evolution of different machine learning classifiers, we have to calculate the confusion matrix, so we have to need to calculate precision, recall, F1 score, and accuracy. For good performance of the imbalance dataset, we have used the F1 score value rather than accuracy. The F1 score is calculated by the harmonic mean of precision and recall. F1 Score gives better results concerning accuracy. In the following table, we have got the result of four different machine learning classifier with sentiment analysis and without sentiment analysis applied on four different bug report data set. Easily see the difference and improvement in results with sentiments and without sentiments.

8 RESULTS AND DISCUSSION

Results and comparisons with sentiments analysis and without sentiment analysis are shown in the given table.the problem which we are define in this paper it is a multiclass classification problem five priority levels p1,p2,p3,p4,and p5 are multiclasses here.Therefore we also perform microanalysis,weightedanalysis and macroanalysis for all priority levels to evaluate the performance of multiclass classification problem. Macroanalysis combine precision and recall of all priority levels by averaging their values. similarly microanalysis computes precision recall from the sum of true positive , true negative, false positive, and false negative values of all priorioties level. While in weightedanalysis averaging of total number of true positive of all levels devided by total number of objects in all levels.

Performance metric for proposed classifer(Sentiment analysis)

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision: 81% Recall:79% F1Score:79.98%	Precision: 82% Recall: 82% F1Score:82%	Precision: 80% Recall:80% F1Score:80%	Precision: 79% Recall:79% F1Score:79%
Weighted analysis	Precision: 81% Recall:79% F1Score:79.98%	Precision: 85% Recall:82% F1Score:83.47%	Precision:85% Recall:82% F1Score:83.47%	Precision: 81% Recall:79% F1Score:79.98%
Macro analysis	Precision:81% Recall:78% F1Score:79.47%	Precision:85% Recall:85% F1Score:85%	Precision:80% Recall:80% F1Score:80%	Precision:81% Recall:80% F1Score:80.49%

F1-score, accuracy, precision, recall with sentiment **using KNN classifier**

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision:79% Recall:79% F1Score:79%	Precision:86% Recall:86% F1Score:86%	Precision:81% Recall:82% F1Score:81.49%	Precision:80% Recall:80% F1 Score:80%
Weighted analysis	Precision:79% Recall:79% F1Score:79%	Precision:86% Recall:86% F1Score:86%	Precision:81% Recall:82% F1Score:81.49%	Precision:79% Recall:80% F1Score:79.49%
Macro analysis	Precision:79% Recall:79% F1Score:79%	Precision:87% Recall:86% F1Score:86.49%	Precision:82% Recall:82% F1Score:82%	Precision:80% Recall:80% F1Score:80%

F1-score, accuracy, precision, recall with sentiment **using DECISION TREE classifier**

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision:51% Recall:25% F1Score:33.35%	Precision:66% Recall:66% F1Score:66%	Precision:56% Recall:56% F1Score:56%	Precision:57% Recall:57% F1Score:57%
Weighted analysis	Precision:40% Recall:51% F1Score:44.83%	Precision:69% Recall:66% F1Score:67.46%	Precision:56% Recall:56% F1Score:56%	Precision:55% Recall:57% F1Score:55.98%
Macro analysis	Precision:21% Recall:25% F1 Score:22.82%	Precision:48% Recall:25% F1Score:32.87%	Precision:57% Recall:56% F1Score:56.49%	Precision:50% Recall:28% F1Score:35.89%

F1-score, accuracy, precision, recall with sentiment **using NAÏVE BAYES classifier**

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision: 74% Recall: 60% F1Score:66.65%	Precision: 75% Recall: 68% F1Score:71.32%	Precision: 66% Recall: 58% F1Score:61.74%	Precision: 77% Recall: 66% F1Score:71.07%
Weighted analysis	Precision: 75% Recall: 59% F1Score:60.04%	Precision:76% Recall:70% F1score:72.87%	Precision: 60% Recall: 59% F1Score:59.49%	Precision:77% Recall: 68% F1Score:72.22%
Macro analysis	F1 Score: 59% Precision: 59% Recall: 59%	F1 Score: 68% Precision: 68% Recall: 68 %	F1 Score: 58% Precision: 58% Recall: 58 %	F1 Score: 66% Precision: 66% Recall: 66 %

F1-score, accuracy, precision, recall with sentiment **using SVM classifier**

These tables show the results with sentiments using different machine learning classifier KNN, DECISION TREE, NAÏVE BAYES and SVM, and the following tables which are given below shows the result without sentiments.

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision: 50% Recall: 50% F1 Score: 50%	Precision: 66% Recall: 66% F1 Score: 66%	Precision: 55% Recall: 55% F1 Score: 55%	Precision: 56% Recall: 56% F1 Score: 56%
Weighted analysis	Precision: 40% Recall: 50% F1 Score:44.44%	Precision: 66% Recall: 66% F1 Score: 66%	Precision: 55% Recall: 56% F1Score: 55.98%	Precision: 54% Recall: 56% F1 Score: 54.98%
Macro analysis	Precision: 20% Recall: 25% F1 Score:22.82%	Precision: 48% Recall: 25% F1 Score: 25%	Precision: 56% Recall: 56% F1 Score: 56%	Precision: 49% Recall: 26% F1 Score: 33.97%

F1-score, accuracy, precision, recall without sentiment **using NAÏVE BAYES classifier**

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision:77% Recall:57% F1score:65.50%	Precision:75% Recall:68% F1score:71.32%	Precision:57% Recall:57% F1score:57%	Precision:66% Recall:58% F1score:61.74%
Weighted analysis	Precision:70% Recall:59% F1score:64.03%	Precision:75% Recall:68% F1score:71.32%	Precision:57% Recall:59% F1score:57.98%	Precision:66% Recall:59% F1score:62.30%
Macro analysis	Precision:57% Recall:57% F1score:57%	Precision:66% Recall:66% F1score:66%	Precision:57% Recall:57% F1score:57%	Precision:58% Recall:58% F1score:58%

F1-score, accuracy, precision, recall without sentiment **using SVM classifier**

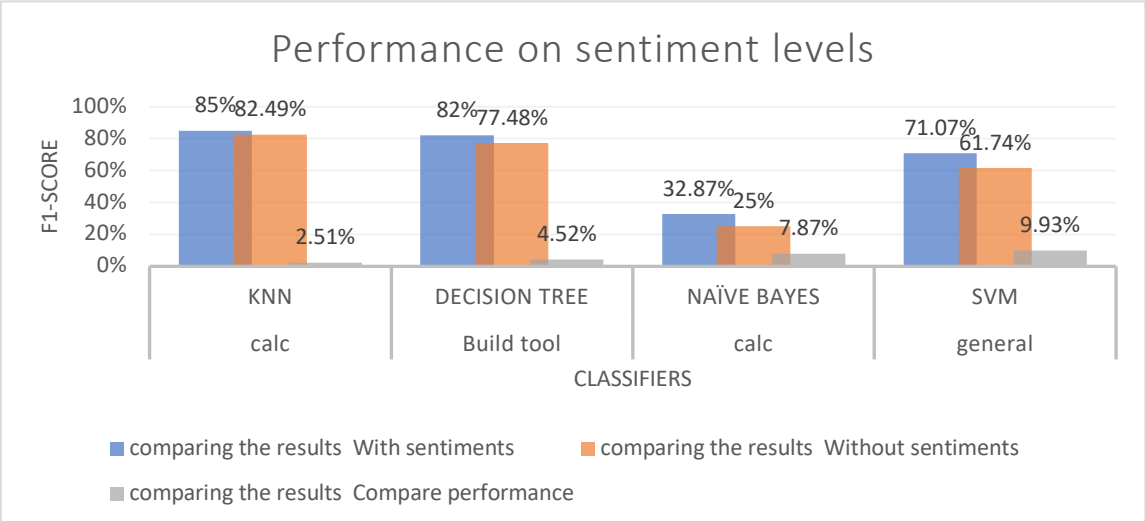
	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision:78% Recall: 78% F1 Score: 78%	Precision: 82% Recall: 82% F1 Score: 82%	Precision:80% Recall: 80% F1Score: 80%	Precision: 79% Recall: 79% F1Score: 79%
Weighted analysis	Precision: 79 Recall: 78 F1Score:78.49%	Precision: 82% Recall: 81% F1 Score:81.49%	Precision: 82% Recall: 82% F1 Score:82%	Precision: 80% Recall: 78% F1 Score: 78.95%
Macro analysis	Precision: 79% Recall: 78% F1Score:78.49%	Precision: 82% Recall: 83% F1Score: 82.49%	Precision: 78% Recall: 80% F1Score:78.98%	Precision: 80% Recall: 79% F1Score: 79.49%

F1-score, accuracy, precision, recall without sentiment **using KNN classifier**

	Dataset: Product-base	Dataset: Product-calc	Dataset: Product-build tools	Dataset: Product-general
Micro analysis	Precision: 77% Recall: 77% F1Score: 77%	Precision: 85% Recall: 85% F1Score: 85%	Precision: 79% Recall: 79% F1Score: 79%	Precision: 78% Recall: 78% F1Score: 78%
Weighted analysis	Precision: 78% Recall: 77% F1Score: 78%	Precision: 85% Recall: 85% F1Score: 85%	Precision: 78% Recall: 77% F1Score: 77.49%	Precision: 78% Recall: 78% F1Score: 78%
Macro analysis	Precision: 78% Recall: 78% F1Score: 78%	Precision: 86% Recall: 86% F1Score: 86%	Precision: 77% Recall: 78% F1Score: 77.49%	Precision: 78% Recall: 78% F1Score:78%

F1-score, accuracy, precision, recall without sentiment **using DECISION TREE classifier**

Product	F1-measure of classifiers	With sentiments	Without sentiments	Compare performance
calc	KNN	85%	82.49%	2.51%
Build tool	DECISION TREE	82%	77.48%	4.52%
calc	NAÏVE BAYES	32.87%	25%	7.87%
general	SVM	71.07%	61.74%	9.93%



Comparing all table's results according to with sentiments and without sentiments, we can see easily improvement in results. In the KNN classifier, we have got improvement from 0.98 % to 2.51%. for

example, the calc dataset in macro analysis with sentiments F1-score is 85% and without sentiments is 82.49% and the improvements are 85%-82.49%=2.51%. we have used three different analyzers to find out the desired results Micro analyzer, weighted analyzer, and Macro analyzer. In the DECISION TREE classifier, we have got improvement from 0.49% to 4.51%. for example, the build tool dataset in macro analysis with sentiments F-1-score is 82% and without sentiments is 77.49% and the improvements are 82%-77.49%=4.51%. In the NAÏVE BAYES classifier, we have got improvement from 0% to 7.87%. for example, the calc dataset in macro analysis with sentiments F1-score is 32.87% and without sentiments is 25% and the improvements are 32.87%-25%=7.87%.but in naïve Bayes we got zero percent result so we cannot say that naïve Bayes is better than other two classifiers which we have used previously, the KNN classifier and decision tree classifier. In the SVM classifier, we have got improvement from 1% to 9.93%. for example, the general dataset in micro analysis with sentiments F1-score is 71.07% and without sentiments is 61.74% and the improvements are 71.07%-61.74%=9.93%. According to the results and the above discussion, SVM work better than the other three classifiers which we mentioned in this paper.

9 CONCLUSIONS

Many times, bug reports are submitted either with the wrong priority level or even without knowledge and expertise of the priority of each specific bug report. The manual process of prioritizing bug reports calls for knowledge and resources (e.g., time and expertise). In order to achieve this, we put forward a classifier that takes sentiment analysis into account when prioritizing problem reports. The suggested method used a machine learning technique as well as sentiment analysis, natural language processing, resampling techniques, and feature extraction techniques on the provided dataset to estimate the priority of bug reports. The suggested method—classifiers that take sentiment analysis into account when a bug is reported—saved the developers the necessary time and work in the priority prediction process. On the historical information of the four open-source products from Bugzilla, we conducted a cross-project review. The evaluation's findings revealed that the proposed strategy performed better than cutting-edge methods. The main takeaway from our research is that bug reports' textual details may be a valuable source of data for prioritizing them for prompt resolution. Our findings should inspire more studies into how to prioritize bugs. In the future, we'd like to look into the justification for the suggested strategy. In the future, we would like to find out the cause of the occurrence using cutting-edge neural network approaches. By incorporating additional bug reports from various domains, such as information systems, we would also like to examine the priority of issue reports according to their respective domains. The proposed method's generalizability will be confirmed by the ranking of bug reports according to their respective domains.

Declarations

• The authors have no relevant financial or non-financial interests to disclose.
• The authors have no competing interests to declare that are relevant to the content of this article.
• All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.
• The authors have no financial or proprietary interests in any material discussed in this article**.**

REFERENCES

[1]. Kanwal, J., & Maqbool, O. (2012). Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, *27*(2), 397-412

[2] Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.

[3] Tian, Y., Lo, D., & Sun, C. (2013, September). Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance* (pp. 200-209). IEEE.

[4] Denecke, K. (2008, April). Using sentiwordnet for multilingual sentiment analysis. In *2008 IEEE 24th international conference on data engineering workshop* (pp. 507-512). IEEE.

[5] Asghar, M. Z., Khan, A., Kundi, F. M., Qasim, M., Khan, F., Ullah, R., & Nawaz, I. U. (2014). Medical opinion lexicon: an incremental model for mining health reviews. *International Journal of Academic Research*, *6*(1), 295-302.
[6] Umer, Q., Liu, H., & Sultan, Y. (2018). Emotion based automated priority prediction for bug reports. *IEEE Access*, *6*, 35743-35752.
[7] Malhotra, R., Dabas, A., Hariharasudhan, A. S., & Pant, M. (2021, January). A study on machine learning applied to software bug priority prediction. In *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)* (pp. 965-970). IEEE.
[8] Xuan, J., Jiang, H., Ren, Z., & Zou, W. (2012, June). Developer prioritization in bug repositories. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 25-35). IEEE.
[9] Bhattacharya, P., & Neamtiu, I. (2010, September). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-10). IEEE.
[10] Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, *6*(5), 429-449.
[11] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D.(2019). Text classification algorithms: A survey. *Information*, *10*(4), 150.
[12] Uddin, J., Ghazali, R., Deris, M. M., Naseem, R., & Shah, H. (2017). A survey on bug prioritization. *Artificial Intelligence Review*, *47*(2), 145-180.
[13] Alenezi, M., Magel, K., & Banitaan, S. (2013). Efficient Bug Triaging Using Text Mining. *J. Softw.*, *8*(9), 2185-2190.
[14] Tamrawi, A., Nguyen, T. T., Al-Kofahi, J., & Nguyen, T. N. (2011, May). Fuzzy set-based automatic bug triaging (NIER track). In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 884-887).
[15] Zhang, T., Yang, G., Lee, B., & Shin, I. (2012). Role analysis-based automatic bug triage algorithm. *Information Processing Society of Japan*, 4.
[16] Zou, W., Hu, Y., Xuan, J., & Jiang, H. (2011, July). Towards training set reduction for bug triage. In *2011 IEEE 35th Annual Computer Software and Applications Conference* (pp. 576-581). IEEE.
[17] (Jan. 2018). (Bugzilla Issue Tracker). [Online]. Available: https://www. bugzilla.org/
[18] (Jan.2018). Mantis Bug Reporting System. [Online]. Available: https:// code.google.com/archive/p/support/wikis/IssueTracker.wiki
[19] (Jan. 2018). Google Issue Tracker. [Online]. Available: https://www. mantisbt.org/
[20] (Jan. 2018). Github Issue Tracker. [Online]. Available: https://github. com/features
[21] (Jan. 2018). Jira Issue Tracker. [Online]. Available: https://www.atlassian. com/software/jira
[22] Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., & Poshyvanyk, D. (2015, August). Auto-completing bug reports for android applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 673-686).
[23] Xie, X., Zhang, W., Yang, Y., & Wang, Q. (2012, September). Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering* (pp. 19-28).
[24] Kemerer, C. F. (1995). Software complexity and software maintenance: A survey of empirical research. *Annals of Software Engineering*, *1*(1), 1-22.
[25] Zhang, Y., Jin, R., & Zhou, Z. H. (2010). Understanding bag-of-words model: a statistical framework. *International journal of machine learning and cybernetics*, *1*(1), 43-52.
[26] Bahmanyar, R., Cui, S., & Datcu, M. (2015). A comparative study of bag-of-words and bag-of-topics models of EO image patches. *IEEE Geoscience and Remote Sensing Letters*, *12*(6), 1357-1361.
[27] Anvik, J. (2007). *Assisting bug report triage through recommendation* (Doctoral dissertation, University of British Columbia).
[28] Alenezi, M., & Banitaan, S. (2013, December). Bug reports prioritization: Which features and classifier to use?. In *2013 12th International Conference on Machine Learning and Applications* (Vol. 2, pp. 112-116). IEEE.
[29] Yu, L., Tsai, W. T., Zhao, W., & Wu, F. (2010, November). Predicting defect priority based on neural networks. In *International Conference on Advanced Data Mining and Applications* (pp. 356-367). Springer, Berlin, Heidelberg.

[30] Bhatia, A., Chug, A., Singh, A. P., & Singh, D. (2022). Fractional mega trend diffusion function-based feature extraction for plant disease prediction. *International Journal of Machine Learning and Cybernetics*, 1-26.
[31] Branco, P., Ribeiro, R. P., & Torgo, L. (2016). UBL: an R package for utility-based learning. *arXiv preprint arXiv:1604.08079*.
[32] Asghar, M. Z., Khan, A., Ahmad, S., Khan, I. A., & Kundi, F. M. (2015). A unified framework for creating domain dependent polarity lexicons from user generated reviews. *PloS one*, *10*(10), e0140204.
[33] Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007, May). How long will it take to fix this bug?. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)* (pp. 1-1). IEEE.
[34] Menzies, T., & Marcus, A. (2008, September). Automated severity assessment of software defect reports. In *2008 IEEE International Conference on Software Maintenance* (pp. 346-355). IEEE.
[35] Lamkanfi, A., Demeyer, S., Soetens, Q. D., & Verdonck, T. (2011, March). Comparing mining algorithms for predicting the severity of a reported bug. In *2011 15th European Conference on Software Maintenance and Reengineering* (pp. 249-258). IEEE.
[36] Chaturvedi, K. K., & Singh, V. B. (2012, September). Determining bug severity using machine learning techniques. In *2012 CSI sixth international conference on software engineering (CONSEG)* (pp. 1-6). IEEE.
[37] Abdelmoez, W., Kholief, M., & Elsalmy, F. M. (2012, October). Bug fix-time prediction model using naïve Bayes classifier. In *2012 22nd International Conference on Computer Theory and Applications (ICCTA)* (pp. 167-172). IEEE.
[38] Dommati, S. J., Agrawal, R., & Kamath, S. S. (2013). Bug Classification: Feature Extraction and Comparison of Event Model using Na\" ive Bayes Approach. *arXiv preprint arXiv:1304.1677*.
[39] Khan, A., Baharudin, B., Lee, L. H., & Khan, K. (2010). A review of machine learning algorithms for text-documents classification. *Journal of advances in information technology*, *1*(1), 4-20.
[40] Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software*, *117*, 166-184.
[41] Yang, G., Zhang, T., & Lee, B. (2014, July). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference* (pp. 97-106). IEEE.
[42] Iliev, M., Karasneh, B., Chaudron, M. R., & Essenius, E. (2012, June). Automated prediction of defect severity based on codifying design knowledge using ontologies. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)* (pp. 7-11). IEEE.
[43] Zhang, T., Yang, G., Lee, B., & Chan, A. T. (2015, April). Predicting severity of bug report by mining bug repository with concept profile. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (pp. 1553-1558).
[44] Xuan, J., Jiang, H., Ren, Z., & Zou, W. (2012, June). Developer prioritization in bug repositories. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 25-35). IEEE.
[45] Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008, May). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering* (pp. 461-470).
[46] Singh, V. B., & Chaturvedi, K. K. (2011). Bug tracking and reliability assessment system (btras). *International Journal of Software Engineering and Its Applications*, *5*(4), 1-14.
[47] Sharma, M., Bedi, P., Chaturvedi, K. K., & Singh, V. B. (2012, November). Predicting the priority of a reported bug using machine learning techniques and cross project validation. In *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)* (pp. 539-545). IEEE.
[48] Singh, V. B., & Chaturvedi, K. K. (2012, November). Entropy based bug prediction using support vector regression. In *2012 12th international conference on intelligent systems design and applications (ISDA)* (pp. 746-751). IEEE.
[49] Choudhary, P. A., & Singh, S. (2017). Neural Network Based Bug Priority Prediction Model using Text Classification Techniques. *International Journal of Advanced Research in Computer Science*, *8*(5).
[50] Tian, Y., Lo, D., Xia, X., & Sun, C. (2015). Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, *20*(5), 1354-1383.

[51] Schapire, R. E., & Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine learning*, *39*(2), 135-168.
[52] Singh, V. B., Misra, S., & Sharma, M. (2017). Bug severity assessment in cross project context and identifying training candidates. *Journal of Information & Knowledge Management*, *16*(01), 1750005.