IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 10, 2021, accepted March 19, 2021, date of publication March 29, 2021, date of current version April 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3069248

CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms

HAFIZA ANISA AHMED[1], NARMEEN ZAKARIA BAWANY[1], AND JAWWAD AHMED SHAMSI[2]

[1]Department of Computer Science and Software Engineering, Jinnah University for Women, Karachi 74600, Pakistan
[2]Department of Computer Science, National University of Computer and Emerging Sciences (NUCES), Karachi 75400, Pakistan

Corresponding author: Narmeen Zakaria Bawany (nshawoo@gmail.com)

ABSTRACT Bug reports facilitate software development teams in improving the quality of software. These reports include significant information related to problems encountered within a software, possible enhancement suggestions, and other potential issues. Bug reports are typically complex and are too detailed; hence a lot of resources are required to analyze and process them manually. Moreover, it leads to delays in the resolution of high priority bugs. Accurate and timely processing of bug reports based on their category and priority plays a significant role in improving the quality of software maintenance. Therefore, an automated process of categorization and prioritization of bug reports is needed to address the aforementioned issues. Automated categorization and prioritization of bug reports have been explored recently by many researchers; however, limited progress has been made in this regard. In this research, we present a novel framework, titled CaPBug, for automated categorization and prioritization of bug reports. The framework is implemented using Natural Language Processing (NLP) and supervised machine learning algorithms. A baseline corpus is built with six categories and five prioritization levels by analyzing more than 2000 bug reports of Mozilla and Eclipse repository. Four classification algorithms i.e., Naive Bayes, Random Forest, Decision Tree, and Logistic Regression have been used to categorize and prioritize bug reports. We demonstrate that the CaPBug framework achieved an accuracy of 88.78% by using a Random Forest classifier with a textual feature for predicting the category. Similarly, using the CaPBug framework, an accuracy of 90.43% was achieved in predicting the priority of bug reports. Synthetic Minority Over-Sampling Technique (SMOTE) has been applied to address the class imbalance issue in priority classes.

INDEX TERMS Bug reports, natural language processing, machine learning, bug report categorization, bug report prioritization.

I. INTRODUCTION

Software testing and maintenance are the most critical phases of software development. Bug reports play a vital role in these stages of development activities [1], [2]. A bug report is generated by the software quality assurance team while testing software modules. It contains detailed information about a specific component or problem that is needed to be fixed [3]–[5]. The information in a bug report includes many attributes such as feature request, functionality enhancement request, code errors, logical errors, and compatibility issues. The report consists of several headings including priority, summary, description of the affected component, and open/close status [6], [7]. However, the major problem encountered during the analysis of bug reports is that the information is in natural language. Therefore, it is very difficult to process and extract information from it. It requires a tedious effort from the development team to understand and address the reported issues [8]–[12]. Many studies exist that address the issues related to bug reports [13]. These include bug categorization [14]–[19], bug prioritization [20]–[23], bug localization [24], bug assignment [25], bug classification [10], [26], [27], bug severity prediction [28], and bug report summarization [29]–[31].

Bug categorization and bug prioritization remain the most important element of information that is required from any bug report. Most of the studies used supervised machine learning algorithms to automate the information extraction

The associate editor coordinating the review of this manuscript and approving it for publication was Ikramullah Lali.

process [32]. Through these algorithms, a classification model is constructed by training the manually labeled data of bug reports, which are then used to automatically categorize and prioritize new bugs with pre-defined labels. Supervised learning techniques need a large labeled dataset, which is not easily available. In most of the available datasets, category and priority information is missing. Furthermore, most of the available research is focused on one problem independently, i.e., either automating bug categorization [11], [33] or bug prioritization [34], [35]. Consequently, very limited work has been done in the area of categorization and prioritization of bug reports simultaneously [36]. Therefore, there is a need for a capable framework that automates both bug categorization and bug prioritization at the same time.

This research is motivated to address the above-mentioned requirements. To this end, the key objective of this research is to develop a framework that categorizes and prioritizes each issue in the bug reports automatically. We propose and implement an automated framework for categorizing and prioritizing bug reports called CaPBug. The framework – CaPBug uses NLP and machine learning algorithms to categorize and prioritize bug reports based on their textual and categorical features. Baseline corpus is built by using the XML files of Mozilla and Eclipse repository. Different NLP techniques have been applied in bug reports' textual descriptions to create a feature vector-set. Afterwards, the Term Frequency–Inverse Document Frequency (TF-IDF) feature extraction method has been used to extract relative and important words from the feature vector-set. Due to the imbalanced nature of priority classes, Synthetic Minority Over-sampling Technique (SMOTE) is used for oversampling the records. Finally, four machine learning algorithms i.e., Naive Bayes (NB), Random Forest (RF), Decision Tree (DT) and Logistic Regression (LR) have been used to train the models that predict the category and priority of bug reports.

Below are the major contributions of this research.

- We created a baseline corpus with six labeled categories of bugs and five priorities by using two online bug repositories of Eclipse and Mozilla that are available on Bugzilla. Labeled dataset with predefined categories for bug reports from year 2016 to year 2019 is not available publically.
- We proposed and implemented a framework named CaPBug for categorization and prioritization of bug reports using NLP and supervised machine learning. The novel contribution of this research is that it addresses the need for both automated bug categorization and prioritization.
- We applied SMOTE to address class imbalance problem and to improve the model's accuracy that prioritizes the bug reports. Limited work has been done using SMOTE to adjust the number of bug reports for each priority level so that the model can accurately predict the priority of bug reports.
- We performed extensive experiments with the most recent dataset comprising of reports from year 2016 to year 2019. It includes a comparison of textual and categorical features for categorizing and prioritizing bug reports using four machine learning algorithms.

TABLE 1. List of experiments conducted.

S.No.	Experiments and Results	Section No.
	Using Textual Features	
1	Predicting Category and Priority from Textual Feature	IV.A
2	Category Wise Results from Textual Feature	IV.A.1
3	Priority Wise Results from Textual Feature	IV.A.2 V.A.2
	Using Categorical Features	
4	Predicting Category and Priority from Categorical Features	IV.B
5	Category Wise Results from Categorical Features	IV.B.1
6	Priority Wise Results from Categorical Features	IV.B.2
	After Applying SMOTE	
7	Predicting Priority from Textual Feature with SMOTE	IV.C
8	Priority Wise Results from Textual Feature with SMOTE	IV.C.1
9	Predicting Priority from Categorical Features with SMOTE	IV.D
10	Priority Wise Results from Categorical Features with SMOTE	IV.D.1

Table 1 summarizes our experiments and the corresponding section number in which results are presented.

We anticipate that our work will be useful for the community in automating categorization and prioritization of bug reports. This will be beneficial in maintenance and debugging of large software projects.

The remainder of this paper is organized as follows. Section II presents literature review of bug categorization as well as bug prioritization. Section III introduces proposed methodology for the CaPBug framework. Next, Section IV discusses the results after training and testing of the CaPBug framework. Finally, Section V concludes the research.

II. LITERATURE REVIEW

Researchers have addressed various aspects of automated software bug management, classification and prioritization. These include automation of bug assignment, duplicate or similar bug detection, bug fixing time prediction, bug localization, bug categorization, bug severity and priority predictions etc. Z. Weiqin *et al.* [36] conducted a survey of 327 participants to gain insight into bug management techniques and confirmed that these techniques play an important role in improving the automatic management of bug reports.

Y. Tan *et al.* [37] proposed a novel approach for predicting severity. They linked the bug repositories post on stack overflows to the contents of Mozilla, Eclipse, and GCC bug reports. Three classification algorithms of K-Nearest Neighbor algorithm (KNN), Naive Bayes and Long Short-Term Memory (LSTM) were used to predict the severity of bugs. The results of the experiments showed an increase of

23.03%, 21.86%, and 20.59% in the average F-measurement of Mozilla, Eclipse, and GCC in the proposed method.

R. Chen *et al.* [38] implemented an improved SMOTE technique called Rectangle SMOTE (RSMOTE) to avoid the poor performance for severity prediction. Due to class imbalance problem in bug reports dataset, RSMOTE was used to balance the size of datasets. Furthermore, a technique of repeated sampling was used to avoid indeterminate results due to over-sampling of records and to acquire multiple balance datasets. Further, ensemble approach, named Fusion of Multi-RSMOTE with Fuzzy Integral (FMR-FI), was used to integrate the trained classifiers that were built on multiple balanced datasets. Four evaluation metrics were used to evaluate the performance of the FMR-FI algorithm, namely accuracy, precision, recall and f1-score. The results show that the FMR-FI algorithm with RSMOTE worked well to improve the classifier's performance for severity prediction.

Y. Xiao *et al.* [39] proposed an enhanced Convolutional Neural Networks (CNN) based model called DeepLoc for automated bug localization. Based on CNN, DeepLoc replaced the features of bug reports and source files with word embedding techniques. The experiments were performed on 18,500 bug reports from 2001 to 2014 that have been extracted from five projects of Aspect J, Eclipse UI, JDT, SWT and Tomcat. They compared DeepLoc's performance from the four bug localization approaches of BugLocator, LR+WE, HyLoc and DeepLocator. The results of the experiments show that with the use of DeepLoc, Mean Average Precision (MAP) has improved from 10.87% to 13.4% for bug localization compared to traditional CNN.

To improve the automatic bug assignment, R. Shakripur *et al.* [25] suggested a time-based approach named ABA-TF-IDF using the Time TF-IDF weighting technique. The data was collected from the software repository of the Version Control System (VCS) where changes to the source codes are managed and other project facts are documented. Four machine learning algorithms i.e., Support Vector Machine (SVM), Naive Bayes, Vector Space Model (VSM) and Smooth Unigram Model (SUM) were used to train the model. The results show that the proposed approach performed well with a Mean Reciprocal Rank (MRR) up to 11.8% and 8.94%.

The focus of this research is to automate the process of bug categorization as well as bug prioritization. Therefore, the literature review has been presented in two parts. The first part presents a comprehensive overview of studies related to bug categorization. The next part explores the work that has been conducted on the bug priority. The previous studies of each group are discussed below.

A. BUG CATEGORIZATION

Bug categorization is the process of automatically labeling bug reports with its relevant category. N. Limsettho *et al.* [14] proposed a model to automatically categorize bug reports using clustering and Hierarchical Dirichlet Process (HDP) techniques with NLP chunking. The clustering algorithms of X-means and Expectation Maximization (EM) were used and implemented using Weka 3.6. Two experiments were conducted on the online bug reports of Lucene, Jackrabbit (JCR) and HttpClient projects and evaluated by using cluster purity/accuracy and f1-score. The clustering result was compared with two classification methods of J48 and Logistic Regression. Results demonstrated that the approach of X-means performed well and cluster purity/accuracy and f1-score were high. Also, comparable results recommend that the algorithm of logistic regression may perform better with a supervised learning approach.

Labeled Latent Dirichlet Allocation (LLDA) based topic modeling was implemented by M. F. Zibran [15] for classifying bug reports. These reports were collected from online projects of Eclipse, GNOME and Python. The dataset comprises 1,138 bug reports from which 428 reports were selected. The results show that the accuracy in terms of precision, recall, and f1-score improved considerably when the LLDA is trained using the larger corpus.

N. Limsettho *et al.* [16] extended their work [14] and proposed an automated framework without labeled data and used topic modeling and clustering technique to categorize bug reports. Also, a new technique of NLP Chunking was used to automatically label a cluster and top words of that cluster. To solve the labeling problems of terms in previous studies, a weighted-reduction algorithm was chosen to provide a variety of words. Five experiments were conducted and the dataset comprises three online projects from Lucene, Jackrabbit (JCR) and HTTPClient. The results showed that the topic model performed well with a higher average of f1-score. The performance of their proposed framework with no labeled datasets is better than the labeled projects which are trained using the training models. The result of Phrase-level labeling by NLP chunking provided the high-quality labels that are related to the bug.

C. Zhou *et al.* [17] proposed a new approach called Bug Named-Entity Recognition (BNER). Three features i.e., description phrases, solid distribution, and Parts of Speech (POS) of bug reports' entities were summarized and the category method was created to categorize bugs into a predefined set of 16 categories based on these features. A baseline corpus was built with all related information and a semi-supervised system of BNER. To extract features from the bug repository, an embedded technique was used. The two online software bug repositories of Mozilla and Eclipse were used to train and evaluate the proposed approach. The result showed that it is very useful to design a baseline corpus in initial phases and their approach increased the accuracy by 70% to 80%. Also, BNER can be effective for entities of cross-projects' bug recognition.

B. BUG PRIORITIZATION

The process of bug prioritization involves automatically prioritizing highly influenced bugs so that the critical bug is identified immediately. An automated approach to prioritize bug reports named Drone was proposed by Y. Tian *et al.* [20].

For handling imbalanced data of bug reports, a new classification engine called GREY was built by merging linear regression and their thresholding approach. Different dimensions i.e., author, product, related-report, severity, textual and temporal were reviewed to predict bug reports' priority. The dataset was collected from the Eclipse project with 100,0000 bug reports and divided them into three-set: REP−training data (for identifying similar reports), training data of Drone and testing data of Drone. The proposed approach was compared with the baseline solution of previous studies and the result showed that their approach performed well comparatively up to the f1-score of 209%.

Another study for prioritizing bug reports was proposed by P. A. Choudhary and D. S. Singh [40]. The research focused on five priority levels with six features i.e., temporal, textual, author, related-report, severity and product, to predict the priorities of bug reports using the Artificial Neural Network and Naive Byes classifier. Five versions of the Eclipse project, i.e., 2.0, 2.1, 3.0, 3.1, and 3.2, with three products, i.e., JDT, PDE and Platform are collected from Bugzilla that have been used to train and test the model. To evaluate the model, Receiver Operating Characteristic (ROC) curve and f1-score measures were used. The results showed that the model predicated priority level P3 with 82.7% precision and 80.9% recall measures in Eclipse 2.0. Furthermore, the model performed more efficiently by using Naïve Bayes with ROC ranging from 89% to 98% for different priority levels.

To automate bug prioritization, Y. Wang *et al.* [22] introduced methods of feature selection by using the classification models on the two most popular bug prioritization projects: WordPress and Trac. By accompanying the two main feature selection methods of wrapper and filter, seven techniques of feature selection: Correlation, CfsSubset, OneR, InfoGain, SymmetricalUncert, GainRatio and ReliefF were considered. Two classification algorithms of Naive Bayes and SVM were also used like previous studies, to train the set of feature vectors. Results were evaluated using precision, recall, and accuracy measures and it shows that the GainRatio, InfoGain and Correlation performed better for bug prioritization.

Q. Umer *et al.* [23] proposed a new Emotion-based approach for predicting the priority of bug reports. The dataset consists of bug reports from four online projects of JDT, Eclipse, CDT and PDE. The effectiveness of different classification algorithms of Naive Bayes, SVM, Linear Regression, and Multi-Nomial Naive Bayes was investigated. To prioritize bug reports, five class labels were used. To identify and analyze emotion words from the bug reports, the feature vector set was compared with the emotion-word corpus available online. For performance evaluation, Recall, Precision, and f1-score measures were used. Experimental results showed that the proposed approach has improved with f1-score of more than 6%. Also, Pearson correlation coefficient (r=0.405) showed that there is a strong correlation between priority and emotions.

An automated approach for predicting bug priority and severity using machine learning classification algorithms was investigated by H. Manh *et al.* [41]. The performance of different classifiers: SVM, Naive Bayes, Artificial Neural Network (ANN), K-Nearest Neighbors and DT was compared. Random Forest and Decision Tree classifiers were selected to conduct experiments on the datasets of open-source Bug Tracking Systems: Bugzilla, Launchpad, Mantis and Debian. The proposed model used four classes of priority i.e., urgent, high, normal and low and for severity, it was classified into critical, normal and minor. The performance of both the classifiers was evaluated by using time consumption, Mean-Squared Error (MSE) and Median-Absolute Error (MAE) and the result showed that Random Forest outperforms DT with the accuracy of 0.75, which is average.

Existing research is mostly focused on either automating bug categorization [11], [33] or bug prioritization [34], [35]. Limited work has been found in the area of categorization and prioritization of bug reports simultaneously [36] and therefore we present CaPBug framework that automates both bug categorization and bug prioritization. The summary of existing studies which includes datasets, methodology, and results is shown in TABLE 2 whereas, the comparison of the existing studies with the proposed framework is shown in TABLE 3.

III. METHODOLOGY

We now explain the methodology of the CaPBug framework. It includes six phases: 1. Data collection, 2. Pre-Processing, 3. Feature extraction, 4. Class imbalance 5. Classification, and 6. Performances' evaluation.

In the first phase, data has been collected by using the two online software bug repositories of Mozilla[1] and Eclipse[2] from Bugzilla. In the next phase, pre-processing NLP techniques have been applied to bug reports' content. This phase converts the bug reports' textual feature into topic vector sets which is helpful for machine learning algorithms to easily train the model and predict categories and priorities of bug reports correctly. In the third phase, the topic vector set which has been projected in the second phase is evaluated and important words are extracted based on their similar textual structure by using the TF-IDF approach. Afterward, the class imbalance problem has been resolved for priority levels. In the fifth phase, textual and categorical features are trained by using machine learning algorithms for future inference to automate bug prioritization and categorization. Finally, performance of different algorithms has been analyzed to measure the accuracy of the proposed framework. Fig. 1 shows the overall framework of this research.

We now describe each phase of the CaPBug framework in detail.

A. DATA COLLECTION PHASE

The dataset used in this research is collected from the two online software bug repositories of Mozilla and Eclipse from

[1]https://bugzilla.mozilla.org/
[2]https://bugs.eclipse.org/bugs/

TABLE 2. Summary of existing studies related to bug categorization and prioritization.

Year	Authors	Dataset	Methodology	Results
Bug Categorization				
2014	N. Limsettho et al. [14]	Lucene, Jackrabbit (JCR) and HTTPClient	Unsupervised machine learning approach using topic modeling and clustering techniques with NLP chunking. • Topic Modeler: Hierarchical Dirichlet Process (HDP) • Clustering Algorithms: Expectation-Maximization (EM), J48, Logistic regression and X-means. • Measurements: Recall, Precision, f1-score and cluster's purity.	EM and X-means both performed well with a small difference. The approach of X-means is preferable because of faster runtime and comparable outcomes. EM performs well when the data structure is imbalanced and complex. Also, Logistic regression is suggested when data is available for training.
2016	M. F. Zibran [15]	Eclipse, Python 3.1 and Gnome	Supervised machine learning approach using topic modeling technique of Labeled Latent Dirichlet Allocation (LLDA) with predefined categories. • Measurements: Recall, Precision and f1-score.	When training on the large corpus by using the Labeled Latent Dirichlet Allocation (LLDA) technique, accuracy is increased.
2016	N. Limsettho et al. [16]	Lucene, Jackrabbit (JCR) and HTTPClient	Unsupervised machine learning approach using topic modeling and clustering techniques with NLP chunking. Cluster labeling using a new weighted-reduction algorithm and different clustering labeling algorithms. • Topic Modeler: Hierarchical Dirichlet Process (HDP) • Clustering Algorithms: Expectation-Maximization (EM), J48, Logistic regression and X-means. • Cluster labeling Algorithms: Title of the closest instance, Adjusted Jensen-Shannon Divergence • Measurements: Recall, Precision, f1-score and cluster's purity.	NLP chunking with weight-reduction algorithm performed well with phrase-level of label extraction as compared to JSD with word-level and title of the closest instance with sentence level.
2018	C. Zhou et al. [17]	Mozilla and Eclipse	Supervised machine learning approach using BNER (Bug Specific Named Entity Recognition) with word embedding techniques and CRF (Conditional Random Fields) model and with predefined categories. • Features: Contextual (Basic), Gazetteer, Orthographic, Vector and Cluster. • Measurements: Precision, Recall and f1-score.	• BNER performed well when using the Orthographic feature as compared to others. • For cross-projects' bug-specific NER, BNER is effective and the word-embedding feature played a major role in this. • The results of accuracy and recall can achieve up to 70 to 80% by using the CRF model.
Bug Prioritization				
2015	Y. Tian et al. [20]	Eclipse	Supervised machine learning approach using a classification engine called GREY, a component of a proposed framework: DRONE by using the linear regression and thresholding. • Measurements: Precision, Recall and f1-score.	DRONE performed well with a relative improvement of 209% in terms of f1-score.
2017	P. A. Choudhary and D. S. Singh [40]	Eclipse	Supervised machine learning approach using text classification method and Neural Network MLP (Multilayer Perceptron). • Features: temporal, textual, and author-related, severity, product and component. • Classification Algorithms: Naive Bayes • Measurements: Precision, Recall, f1-score and ROC	• The model performed well for predicting the priority level P3 with 82.7% precision and 80.9% recall measures in Eclipse 2.0. • Also, the model performed efficiently with Naïve Bayes with ROC about 89% to 98% at different priority levels.
2017	Y. Wang et al. [22]	WordPress and Trac	Supervised machine learning approach using feature selection techniques and classification algorithms. • Features: Textual features • Feature Selection Methods: CfsSubset (CFS), Correlation (CO), GainRatio (GR), InfoGain (IG), OneR (OR), ReliefF (RF) and SymmetricalUncert (SU) • Classification Algorithms: SVM and Naive Bayes • Measurements: Precision and Recall	• GainRatio, InfoGain and Correlation performed better than other methods of feature-selection for bug prioritization. • It is enough to use 1/3rd to half of the features to obtain high recall and precision results.
2018	Q. Umer et al. [23]	CDT, Java development tool, Eclipse, PDE and platform from Bugzilla.	Supervised emotion-based machine learning approach using NLP and classification algorithms. • Emotion-word corpus has been compared with the feature sets. • Classification Algorithms: Naive Bayes, SVM, Linear Regression and Multi-Nomial Naive Bayes • Measurements: Precision, Recall and f1-score.	The result r=0.45 showed that there is a strong correlation between priority and emotions.
2020	H. Manh et al. [41]	Bugzilla, Launchpad, Mantis and Debian	Supervised machine learning approach using classification algorithms. • Classification Algorithms: Decision Trees and Random Forest. • Measurements: time consumption, MSE (Mean-Squared Error) and MAE (Median-Absolute Error)	Random Forest outperforms DT with an accuracy of 0.75, which is average.

TABLE 3. Comparison of existing studies with proposed-framework.

Year	Paper Reference	Bug Categorization	Bug Prioritization	Dataset	Comments
2014	N. Limsettho et al. [14]	✓	×	Bug repositories of Lucene, Jackrabbit (JCR) and HTTPClient till 2013	This study used bug reports up to 2013 and only performed automatic bug categorization. Labels suggested by the labeling algorithm have many flaws. As a result, algorithms did not perform well when categorizing bug reports into bugs and other requests within a single project.
2015	Y. Tian et al. [20]	×	✓	Bug repository of Eclipse from 2001 to 2007	This study used an older dataset of bug reports from 2001 to 2007 and only performed automatic bug prioritization. The proposed approach only worked better with the priority level P3 while other priority levels did not achieve good results.
2016	M. F. Zibran [15]	✓	×	Bug repositories of Eclipse, Python 3.1 and Gnome till 2009	This study used an older dataset of bug reports up to 2009 and only performed automatic bug categorization. The dataset of 428 bug reports was categorized into 22 categories. It is not appropriate to train a model with small datasets that have assigned a large number of categories. Hence, research results are not good.
2016	N. Limsettho et al. [16]	✓	×	Bug repositories of Lucene, Jackrabbit (JCR) and HTTPClient till 2013	This study used bug reports up to 2013 and only performed automatic bug categorization. The cluster labeling algorithm used in the research did not work well because the labels suggested by labeling algorithm, i.e., six out of ten labels, were usable. The other four labels that suggest labeling algorithm were not feasible. Furthermore, all approaches used in research for categorizing bug reports into bugs and other requests within the project yielded poor results.
2017	Y. Wang et al. [22]	×	✓	Bug repositories of Trac Open-Source Project and Wordpress from 2003 to 2013	This study used bug reports from 2003 to 2013 and only performed automatic bug prioritization. Their proposed strategy did not yield good results using the feature selection method to find the relevant features related to priority. Furthermore, the research did not evaluate the performance of each priority level.
2018	C. Zhou et al. [17]	✓	×	Bug repositories of Eclipse and Mozilla till 2017	This study used bug reports up to 2017 and only performed automatic bug categorization. Their proposed approach worked well with the use of the word cluster and word vector embedding features.
2018	Q. Umer et al. [23]	×	✓	Bug repository of Eclipse from 2001 to 2007	This study used an older dataset of bug reports from 2001 to 2007 and only performed automatic bug prioritization. The results of the proposed approach were compared with the previous approach in their research. Although the proposed approach worked better than the other, it did not yield good results.
2020	H. Manh et al. [41]	×	✓	Bug repositories of Mozilla, Launchpad, Mantis and Debian till 2017	This study used bug reports up to 2017 and only performed automatic bug prioritization. Experimental results show that his approach was achieved an accuracy of 75%, which is not enough to train the model for automatic prioritization of bug reports. Furthermore, the research did not provide a detailed analysis of each level of priority.
2020	CaPBug	✓	✓	Bug repositories of Eclipse and Mozilla from 2016 to 2019	Bug reports of recent years between 2016 and 2019 have been used in our research. We performed simultaneous bug categorization and prioritization with SMOTE to improve the model performance. Our framework achieved the highest accuracy of 88.78% for category prediction and 90.43% for priority prediction.

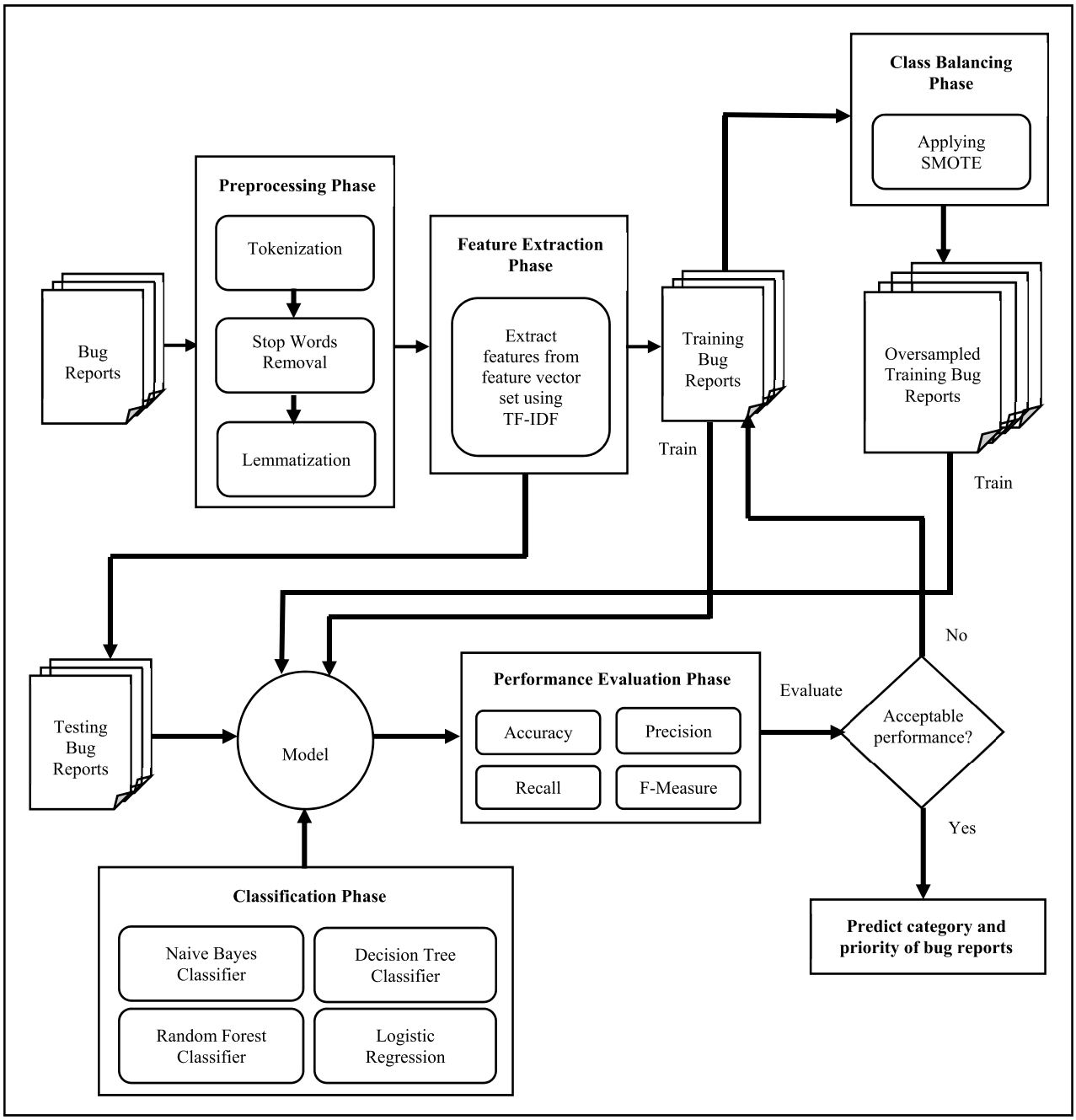

FIGURE 1. Overall framework to categorize and prioritize Bug Reports.

the Bugzilla issue tracking system. Eclipse and Mozilla are authentic and open-source datasets available for reporting software bugs and have only real flaws. These are available from Bugzilla's system, which contains a large number of the latest bug reports, including the entire bug life cycle, which stores all the actions and information to solve bugs [42], [43]. Although a large number of bug reports are available in the Bugzilla system, the category of bugs is not mentioned in the bug reports for recent years. We randomly selected bug reports and labeled them manually after a thorough investigation, based on the 6 categories. Around 2000 bug reports from both the repositories within the time period of 2016 to 2019 have been selected for this research. We've used keywords in the selection process to identify bug reports in each category. For example: For GUI type reports, we used keywords i.e., font, color, alignment, view, layout, etc. During this process, we tried to ensure that the number of bug reports in each category of our dataset is almost equal.

Both the textual and the categorical features are used for predicting the category and priority of bug reports.

TABLE 4. Description of bug report features used in capbug framework.

Feature	Description
Summary	Summary of the descriptive contents related to the bug. Such as, 'High CPU on https://www.typeform.com/product/ with video loaded' is one of the summaries of the problem related to performance.
Product	The affected product of the software system due to a bug. Some of the products are Core, Firefox, JDT, Toolkit, UX systems, etc.
Component	The component of the software project is influenced by the bug. A few of them are related to JavaScript Engine, Widget, Security, Networking, Storage, Graphics, Inspector, Source Code, etc.
Assignee	One of the developers from the development team who is responsible for resolving the bug
Status	The status of the report shows the current state. When the new bug is reported in the bug tracking system, its default status is new. The other status includes: unconfirmed, resolved, fixed, closed, verified, etc.
Classification	The classification shows which component of the project the bug report belongs to. For example, there is a bug report of the widget component, is it related to Eclipse or whether it is related to Mozilla?
Priority	Priority Levels differ from each other in each project. The most common priority levels that nearly incorporate in all of the projects are P1 for higher priority, P2 for normal and P3 for lower priority levels.
Category	The category defines the types of bugs based on their characteristics. Some of them are related to Graphical user interface, some related to program anomalies, privacy or security, or related to database, etc.

The summary attribute of the dataset is included as a textual feature on which NLP techniques are applied. We have chosen the summary feature as the textual feature because it provides us detailed information about the problem. Whereas, the categorical features include: product, component, assignee, status, classification, priority and category attributes of the dataset. We have chosen these features because of their impact on categorizing and prioritizing bug reports. The description of each feature is given in TABLE 4.

The dataset comprises of six categories: Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test-Code. To label the dataset, a category was assigned to each record manually by the developer after carefully reading the summary. TABLE 5 summarizes the number of records in each category in the dataset.

These categories of bug reports are explained below with examples.

1) PROGRAM ANOMALY

This category refers to the issues that occur due to problems in source code. Examples of such problems include exceptions, logical errors, return value problems, and syntax errors [44], etc. An example in TABLE 6 shows the summary contents of a bug report in which code for the next line is automatically assigned in if-else condition due to incorrect indentation.

2) GUI

This category refers to the issues that are related to the design and event handling of user interfaces. It involves potential bugs related to widget and text colors, layouts, CSS styles, widgets appearance, visibility [45], [46], etc. An example of the GUI related problem mentioned in the bug report is given in TABLE 6. The report highlights the text unreadability issue that occurred due to the side-scrolling in the text editor.

3) NETWORK OR SECURITY

This category refers to those bugs that are related to network problems or security issues. Bugs related to the network category include connection or server problems such as improper usage of communication protocols, unexpected shutdowns of server [47], [48], etc. The network related issue raised due to sending larger files in one request and leading to xmlHttpRequest hang up, is exemplified in TABLE 6.

Bugs related to security involves those bugs that are related to vulnerabilities, deletion of unused permission, reloading of certain parameters [49], etc. The example shown in TABLE 6, is the summary content of a bug report related to the security issue in which permission is denied when the user wants to access windowUtils property.

4) CONFIGURATION

This category belongs to bugs in which the problem occurs due to the integration of configuration files. Problems in this bug category are caused by wrong file paths or directory paths in XML, updating in external libraries, fixing external libraries, manifest artifacts, plug-in failures [50], etc. An example in TABLE 6 shows that a bug is reported when updating the application and because of this update, the shared configuration area is missing.

5) PERFORMANCE

This category refers to those problems that are concerned with memory, which include infinite loops that cause memory to hang up, energy leaks, extra memory usage [51], etc. An example in TABLE 6 shows the performance-related bug in which during the debugging process, the Eclipse project is very slow and consumed 100% of CPU usage.

6) TEST CODE

This category belongs to those problems that emerge in the test code. When looking at the dataset, it is observed that the bugs related to test-code occurred due to (1) intermittent tests, (2) updating, repairing and running test-cases, and (3) test failures when searching for de-localized bugs [52], etc. A sample report summary in TABLE 6 shows that the bug is reported due to the failure of intermittent JUnit testing in API tools.

Software developers spend a lot of time resolving bug reports that have been reported by their quality assurance team. Sometimes, the number of bug reports for software bug fixes exceeds the generally available resources. As a result, critical bugs are not resolved at all or are handled very slowly. Severity and priority both can be used to mark the level of urgency with which the bug has to be resolved. Severity is defined as the level of impact that a defect has on the product.

TABLE 5. Class distribution of bug categories in dataset.

Bug Repository	Bug Categories						Total No. of Bug Reports
	Program Anomaly	GUI	Network or Security	Configuration	Performance	Test-Code	
Eclipse	314	294	131	211	204	237	1391
Mozilla	184	171	111	70	112	99	747
Total	498	465	242	281	316	336	2,138

TABLE 6. Example summary of bug categories.

Category	Sample Report Summary
Program Anomaly	Wrong indentation after 'else if ' with newline in condition [Eclipse Project] – Bug Report: 548910
GUI	Side scrolling in text editor results in unreadable text [Eclipse Project] – Bug Report: 549296
Network or Security	[Network] xmlHttpRequest is hanging up the page when sending files larger than 10mb in a single request. [Mozilla Project] – Bug Report: 1642459 [Security] Intermittent Mn-fis SecurityError: Permission denied to access property "windowUtils" on cross-origin object (chrome://marionette/content/listener.js, line 378) [Mozilla Project] – Bug Report: 1625410
Configuration	Updating application in UNC path does not find shared configuration area [Eclipse Project] – Bug Report: 508348
Performance	Eclipse turns too slow and using 100% of CPU during debug [Eclipse Project] – Bug Report: 531988
Test Code	Intermittent test failure in test010 in API tools junits [Eclipse Project] – Bug Report: 537642

As severity is typically reported by the user or customer after the system has been deployed [21], we have worked on predicting priority only. Priority is assigned by the development and quality assurance team during the product development [20], [21].

We have used five priority levels in this research i.e., P1 (Very High), P2 (High), P3 (Medium), P4 (Low) and P5 (Very Low). These levels are assigned by the developers in Bugzilla bug reports. The bug reports which are assigned as P1 should be fixed on high priority [53], [54]. Priority level three is more frequently used by the development and testing team as common software bugs fall in this category. Therefore, datasets available at Mozilla and Eclipse have more records of priority level three that is P3. Furthermore, if the development team is unsure about prioritizing any defect or if it is a minor bug, they configure it as a P3 level priority. They can proceed to fix this level of bugs after all critical and high priority bugs are fixed. To ensure, that the ratio of records of each priority level remains similar to the actual dataset, we also selected more records of P3 as shown in TABLE 7.

B. PRE-PROCESSING PHASE

In this phase, the textual feature of the bug report i.e., summary feature is converted into vectors of topics by using Python Natural Language Processing Toolkit (NLTK). Three NLTK processing methods: Tokenization, Stop Words Removal and Lemmatization [55] are applied in this phase.

- Tokenization: To easily understand the bug reports' content, the text is transformed into a series of tokens (or words) without unnecessary punctuation and special symbols [56]–[58].
- Stop Words Removal: In this step, frequently used words of any natural language with no useful meaning (like articles, prepositions, etc. in English) are removed because when transformed into a feature vector set, these words do not contribute as a meaningful word and do not provide any useful information [59], [60].
- Lemmatization: The final step of the pre-processing phase is Lemmatization. A bug report may contain similar words in numerous forms i.e., 'performing', 'performed', 'performs' have the same meaning. So, this process converts various forms of a single word into a meaningful base form [61]. The research uses the Wordnet Lemmatizer package to perform this step as it is the most frequently used lemmatizer.

TABLE 7. Class distribution of bug prioritization in dataset.

Priority Levels	P1	P2	P3	P4	P5
No. of Bug Reports (%)	13.75%	25.21%	43.26%	7.24%	10.52%
No. of Bug Reports after SMOTE (%)	20.60%	21.64%	27.08%	16.25%	14.40%

C. FEATURE EXTRACTION PHASE

After the pre-processing phase, important words are extracted from the feature vector set by examining each dimension of features. To perform this process, the feature extraction technique of Term Frequency–Inverse Document Frequency (TF-IDF) is used. TF-IDF is the information retrieval technique and numerical statistical measure to find those words which are relevant and important to a document in a corpus [62], [63]. These words are calculated as follows.

$$TF\_IDF = TF * IDF$$

where,

$$TF = \frac{Number\ of\ times\ the\ word\ occurs\ in\ the\ text}{Total\ number\ if\ words\ in\ text}$$

and

$$IDF = \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ word\ t\ in\ it}$$

D. CLASS IMBALANCE

Mozilla and Eclipse datasets are highly imbalanced with respect to priority levels as there are abundant records for priority level P3, while there are very few records for other priority levels. As shown in TABLE 7, priority levels P1, P4 and P5 have very few records. There are a number of common issues that normally occur in software and the development team labels these types of issues as P3. Besides that, the dataset also has many P2 level bug reports but not equivalent to P3. However, the bug reports of other priority levels i.e., P1, P4 and P5 are very rare and as a result, a class imbalance problem occurs in a dataset.

This class imbalance problem makes it difficult to train the model and to accurately predict the priority of bug reports [64]. To overcome this issue, Synthetic Minority Oversampling Technique (SMOTE) is used in this research. SMOTE adjusts the class distribution by oversampling and under-sampling classes of the dataset [65].

The distribution of each priority level before and after applying SMOTE is shown in TABLE 7.

E. CLASSIFICATION PHASE

This phase consists of two steps: training and testing. We have divided the datasets of both Mozilla and Eclipse projects into 80% for training and 20% for testing. Four machine learning classification algorithms i.e., Naive Bayes (NB), Random Forest (RF), Decision Tree (DT) and Logistic Regression (LR) were applied to train the model.

1) NAIVE BAYES CLASSIFIER

A Naive Bayes classifier is a classification algorithm that relies on Bayes' Theorem. It has the concept of independence between features i.e., the presence of one feature does not affect the other. It is a very simple and fast algorithm that is highly sophisticated for large datasets [66]. Bayes' Theorem describes the equation in which the probability of label is found according to the observed features and this can be written as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Here, the probability of a target class A can be found if B (predictors or features) are given. This shows that the features or predictors are independent of one other [67], [68].

2) DECISION TREE CLASSIFIER

Decision Tree is a classification algorithm that has a tree-like structure in which data is constantly distributed according to certain parameters. A tree consists of (i) nodes for testing the value of each feature, (ii) edges or branches that are connected to the next node or leaf and have the results of a test and (ii) leaf nodes that predict the target label. This research uses the ID3 algorithm that has measures of Entropy and Information Gain (IG) for building a decision tree [69].

3) RANDOM FOREST CLASSIFIER

Random Forest is an ensemble classification-based algorithm that contains a set of decision trees. These decision trees give classifications and are created from data samples that decide which classification has the most votes [70]. Then, the algorithm selects the trees that give the best prediction result. The combination of different trees makes this classifier ensemble as it gives the best result and reduces overfitting [71].

4) LOGISTIC REGRESSION

Logistic regression is a statistical and classification-based machine learning algorithm that assigns prediction to a number of distinct classes. It is built-up upon the concept of probability and applied when the target class's value has categorical data. The algorithm uses the cost function i.e. Sigmoid function to map the probability of predicting values between 0 and 1 by using the following equation [72].

$$f(x) = \frac{1}{1 + e^{-x}}$$

F. Evaluation Metrics

Classifiers' performance is evaluated using a confusion matrix which is used to create different metrics in conjunction with True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) values [73]. Accuracy, Precision, Recall, and f1-score are also used to evaluate the performance of the CaPBug framework. Accuracy shows how correctly the algorithm classifies the target class. It is the portion of correctly predicted values out of the total number of input samples [38], [74]. Below is the formula for calculating the accuracy of the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

Precision is the proportion of positive predictions which is positive and how accurate the proposed model is [75]. It computes the accuracy of minority classes. The following formula shows how to calculate precision.

$$Precision = \frac{TP}{TP + FP}$$

A recall is the ratio of correct instances among all related instances [76]. It is calculated by looking into a number of False Negatives (FN) in the confusion matrix. It is sometimes referred to as True Positive Rate (TPR) or sensitivity. The formula which is used to compute the recall is:

$$Recall = \frac{TP}{TP + FN}$$

F1-score is used to find the weighted average of precision and recall [77]. When the dataset has a large number of actual negatives or uneven distribution of classes, it finds out the

TABLE 8. Results of classification algorithms from textual feature.

Method	Accuracy%		Precision%		Recall%		f1-score%	
	Category	Priority	Category	Priority	Category	Priority	Category	Priority
Naive Bayes	67.05	57.24	66.00	59.00	65.50	61.60	65.16	59.20
Decision Tree	83.87	68.22	84.50	70.20	83.16	68.80	83.50	68.80
Random Forest	88.78	68.22	90.00	77.20	87.16	65.60	86.66	69.00
Logistic Regression	85.51	55.37	89.00	66.80	81.66	37.50	84.83	40.00

balance between precision and recall [78]. The following formula shows how to calculate f1-score.

$$f1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

IV. RESULTS AND DISCUSSION

We randomly selected more than 2000 reports of two projects i.e., Eclipse and Mozilla from Bugzilla. The baseline corpus has been built by manually labeling the category in these reports to be used for training and testing algorithms. We have predicted categories and priorities of bug reports using four classification algorithms i.e., Naive Bayes, Decision Tree, Random Forest and Logistic Regression. Both textual and categorical features have been used for training and testing the CaPBug framework. Finally, results have been evaluated using evaluation metrics and conducting a comparative analysis between textual and categorical features.

A. PREDICTING CATEGORY AND PRIORITY FROM TEXTUAL FEATURE

The textual feature of the dataset i.e., a summary has been used to predict the category and priority of bug reports. NLP techniques and TF-IDF in textual feature is used to create a feature vector set and extract important features. TABLE 8 shows the results obtained using four classification algorithms.

For predicting category, it is observed from TABLE 8 that the Random Forest classifier achieved the highest accuracy of 88.78% with the highest precision, recall, and f1-score measures of approximately 90.00%, 87.16% and 86.66%. Whereas, Naive Bayes classifier showed the lowest accuracy of 67.05% with the lowest precision of 66.00%, recall of 65.50%, and f1-score of 65.16%. Decision Tree and Logistic Regression classifiers performed moderately as their accuracy is in between 83% to 86%. Both the classifiers recorded precision from 84% to 89%, recall from 81% to 83%, and f1-score from 83% to 85% for category prediction. However, in predicting the priority of bug reports, none of the algorithms performed satisfactorily with a textual feature. Decision Tree and Random Forest classifier achieved the highest accuracy among four algorithms with lower precision, recall and f1-score measures. Both the algorithms did not perform well as their accuracy is 68.22% which is not good enough to train the model. The other two algorithms i.e., Naive Byes and Logistic Regression, both have very low-performance measures as their accuracy is below 60%.

Hence, it is concluded that the category prediction of bug reports performed well with a textual feature. However, no algorithm with a textual feature achieved a good accuracy for predicting priority as there is a class imbalance issue in the dataset. The records of priority classes P1, P4 and P5 are very rare. Most of the bug reports are prioritized with immediate and normal priority levels of P2 and P3. As a result, algorithms did not achieve good performance measures.

TABLE 9. Category wise results of classification algorithms from textual feature.

Method	Bug Categories (Accuracy %)					
	Program Anomaly	GUI	Network or Security	Configuration	Performance	Test-Code
Naive Bayes	71.71	85.71	50.94	68.51	61.84	53.22
Decision Tree	87.87	84.52	73.58	77.77	86.84	87.09
Random Forest	94.94	95.23	75.47	79.62	89.47	88.70
Logistic Regression	94.94	97.61	67.92	66.66	84.21	87.09

1) CATEGORY WISE RESULTS FROM TEXTUAL FEATURE

TABLE 9 presents the accuracy of classification algorithms for each category of bug reports that have been predicted from a textual feature. It is observed that each of the categories performed differently in classification algorithms. Using the Naive Bayes classifier, only a 'GUI' category achieved a better accuracy of 85.71% whereas, 'Program Anomaly' worked fine with 71.71%. Other four categories did not acquire good results as their accuracies were in between 50% to 69% with Naïve Bayes. Decision Tree, Random Forest and Logistic Regression classifiers worked well with the category of Program Anomaly, GUI, Performance and Test-Code. The accuracy of these four categories falls within 84% to 98%. The other two categories of Network or Security and Configuration achieved better results with the classifiers of Decision Tree and Random Forest as their accuracies are from 73% to 80%. However, both the categories achieved the

TABLE 10. Priority wise results of classification algorithms from textual feature.

Method	Priority Levels (Accuracy %)				
	P1	P2	P3	P4	P5
Naive Bayes	52.38	62.37	50.56	84.84	58.18
Decision Tree	49.20	61.38	74.43	90.90	69.09
Random Forest	41.26	55.44	84.09	87.87	60.00
Logistic Regression	23.80	49.50	90.90	06.06	18.18

accuracy of 66% to 68% using the Logistic Regression classifier which is not good enough. Overall, the GUI category is more accurately predicted by all algorithms as compared to other categories and achieved the highest accuracy of 97.61% using the Logistic Regression algorithm.

2) PRIORITY WISE RESULTS FROM TEXTUAL FEATURE

TABLE 10 presents the accuracy of algorithms at each priority level that has been predicted with a textual feature. Using Naive Bayes, Decision Tree and Random Forest classifiers, the P4 priority level acquired the accuracy of 84% to 91%, whereas, it achieved the lowest accuracy of 6.06% with Logistic Regression. Priority level P3 obtained good results using Random Forest and Logistic Regression with the accuracy of 84.09% and 90.90% respectively. It achieved an accuracy of 74.43% by using a Decision Tree classifier. The other three priority levels of P1, P2 and P5 are not accurately predicted by the classifiers. These levels obtained the lowest accuracies i.e., P1 achieved the accuracy of 23% to 53%, P2 with 49% to 63% and P5 acquired 18% to 69%.

Hence, we conclude from the above results that only P3 and P4 priority levels acquired a moderate level of accuracy with machine learning algorithms. However, other priority levels did not acquire good results using a textual feature of the dataset.

B. PREDICTING CATEGORY AND PRIORITY FROM CATEGORICAL FEATURES

The categorical features have also been used in this research for the automatic prediction of categories and priorities of bug reports. These features include product, component, assignee, status, classification, priority and category. TABLE 11 shows the results obtained after applying four classification algorithms with the above-mentioned features.

When predicting the category of bug reports, we observed that no classifiers worked well with categorical features. The performance measures of Naive Bayes and Logistic regression are from 28% to 42%, which is inadequate to train the model correctly. Furthermore, the remaining two classifiers of Decision Tree and Random Forest also did not produce good results as they achieved the accuracy of 53.74% and 54.43% respectively. Other performance measures of these classifiers achieved accuracy in between 51% to 53%. However, when we predicted the priority with categorical features, it has been noted that all the classifiers worked satisfactorily as compared to the results obtained with a textual feature. Random Forest achieved the highest accuracy among all classifiers as their accuracy increased from 68.22% to 77.33%. Furthermore, the recall and f1-score measure of Random Forest performed slightly better as compared to those results when we predicted the priority with a textual feature. Such as, recall has increased from 65.60% to 71.40% and f1-score from 69.00% to 72.20%. Decision Tree obtained the accuracy of approximately 73.36%, but with low-performance measures i.e., precision recorded 67.80%, recall with 67.80% and f1-score with 56.66%. However, the performance measures of Naive Bayes and Logistic Regression acquired low results with the accuracy of approx. 61% to 66%, and, precision, recall and f1-score of 55% to 63% respectively.

It is evident from the results in TABLE 11 that the category prediction is not working well with categorical features as compared to those results when category prediction was done with a textual feature. Furthermore, priority prediction has also been affected by the class imbalance problem. Records of P3 class are abundant while those of other classes are also disproportionate, thereby creating a class imbalance.

1) CATEGORY WISE RESULTS FROM CATEGORICAL FEATURES

TABLE 12 presents the accuracy of classification algorithms for each category of bug reports using categorical features. None of the algorithms have succeeded in achieving good results by using categorical features. Only Random Forest achieved a better accuracy of 71.71% in the category of Program Anomaly which is also not good enough. The other categories did not acquire good results using Random Forest. The categories of Test-Code and Configuration achieved the lowest accuracy i.e., 4.83% and 9.24%, by using Logistic Regression classifier. The categories of Network or Security, Configuration, Performance and Test-Code acquired very low results using Naive Bayes and Logistic Regression classifiers. Only the categories of GUI and Program Anomaly achieved better results with all the algorithms.

A satisfactory level of results was not achieved using categorical features to predict the category of bug reports. We, therefore, conclude that the textual features should be used to predict the category of the bug reports. Since textual features provide detailed information about the bug and help to train the model with the right category.

2) PRIORITY WISE RESULTS FROM CATEGORICAL FEATURES

The performance of each priority level is specified in TABLE 13. We have evaluated that no classifiers succeeded in accurately predicting the priority of bug reports using categorical features. At most, P2 and P3 priority levels obtained better results with all the algorithms. Both priority levels achieved the highest accuracy of 82.96% and 81.81% respectively using the Random Forest classifier. They acquired the

TABLE 11. Results of classification algorithms from categorical features.

Method	Accuracy%		Precision%		Recall%		f1-score%	
	Category	Priority	Category	Priority	Category	Priority	Category	Priority
Naive Bayes	37.38	61.58	42.16	62.80	34.66	55.92	33.66	56.20
Decision Tree	53.73	73.36	52.83	67.80	52.33	67.80	52.33	56.66
Random Forest	54.43	77.33	52.50	73.60	51.83	71.40	51.83	72.20
Logistic Regression	34.81	65.88	35.33	61.84	30.33	60.62	28.00	60.74

TABLE 12. Category wise results of classification algorithms from categorical features.

Method	Bug Categories (Accuracy %)					
	Program Anomaly	GUI	Network or Security	Configuration	Performance	Test-Code
Naive Bayes	61.61	48.80	45.28	14.81	21.05	16.12
Decision Tree	60.60	61.90	58.49	46.29	55.26	32.25
Random Forest	71.71	63.09	54.71	35.18	55.26	30.64
Logistic Regression	63.63	58.33	28.30	9.25	18.42	4.83

TABLE 13. Priority wise results of classification algorithms from categorical features.

Method	Priority Levels (Accuracy %)				
	P1	P2	P3	P4	P5
Naive Bayes	48.57	78.51	80.21	42.35	58.33
Decision Tree	81.63	77.68	77.47	48.27	67.56
Random Forest	64.40	81.81	82.96	51.72	75.67
Logistic Regression	53.96	67.25	73.62	47.05	61.11

minimum accuracy of 73.63% and 67.25% with Logistic Regression. P1 priority level also acquired better results with the Decision Tree classifier as it achieved an accuracy of 81.63%. However, the remaining two priority levels of P4 and P5 were not predicted accurately by the classifiers, except P5 that achieved better results with Random Forest. The accuracy of both priority levels is between 42% to 53% in P4 and 58% to 76% in P5.

Considering the above low-performance measures of priority prediction, we have done an in-depth investigation of each priority level results using both textual and categorical features. After a detailed evaluation, we have concluded that due to the class imbalance problem, no classifiers performed well in priority prediction of bug reports. For this reason, SMOTE has been used in this research to address the class imbalance problem. In our initial experiments, the percentage of P4 and P5 was kept same that is P4 had the smallest percentage before and after the SMOTE. However, we could not attain the satisfactory level of accuracy. It was only when the records of P4 were increased to the level that bypassed P5 as shown in TABLE 7, we reached the desired level of accuracy.

C. PREDICTING PRIORITY FROM TEXTUAL FEATURE WITH SMOTE

After applying SMOTE, all the classifiers performed well for predicting the priority of bug reports using a textual feature. It is evaluated from TABLE 14 that the framework achieved the highest accuracy of 90.43% using the Random Forest classifier with the precision and recall of 91.60% and f1-score measure of 91.50%. Decision Tree acquired an accuracy of 88.94% which is closer to the Random Forest classifier with other performance measures i.e., 89.80% precision, 90.40% recall and 90.20% f1-score. Moreover, the other two classifiers Naive Bayes and Logistic Regression achieved an accuracy of 83.29% and 84.33% respectively. The other performance measures of these two classifiers achieved the best results as well.

TABLE 14. Results of classification algorithms from textual feature with SMOTE.

Method	Accuracy %	Precision %	Recall %	f1-score%
Naive Bayes	83.29	84.00	86.00	84.40
Decision Tree	88.94	89.80	90.40	90.20
Random Forest	90.43	91.60	91.60	91.50
Logistic Regression	84.33	85.60	85.40	85.40

When we compared the results of priority prediction using SMOTE with the results before class imbalance, it is observed that the performance measures have been improved using SMOTE with a textual feature. The performance of all the classifiers succeeded in achieving good results. Hence, it is concluded that the framework performed well with a textual feature after applying SMOTE for priority prediction.

1) PRIORITY WISE RESULTS FROM TEXTUAL FEATURE WITH SMOTE

TABLE 15 presents the accuracy of each priority level with a textual feature after applying SMOTE. Priority level P1

TABLE 15. Priority wise results of classification algorithms from textual feature with SMOTE.

Method	Priority Levels (Accuracy %)				
	P1	P2	P3	P4	P5
Naive Bayes	88.38	84.45	70.61	98.56	97.32
Decision Tree	92.92	83.93	79.20	98.56	86.60
Random Forest	93.43	81.86	85.84	98.56	97.32
Logistic Regression	84.34	78.23	80.08	97.84	86.60

obtained good results with all the classifiers and achieved accuracy in between 84.34% to 93.43%. Furthermore, priority level P4 and P5 also predicted well with all the classifiers. P4 achieved the highest accuracy of 98.56% against all priority levels. P5 achieved accuracy in between 86.60% to 97.32%. Priority level P2 acquired better results with all the classifiers except Logistic Regression. It achieved the maximum accuracy of 84.45% using Naive Bayes and minimum accuracy of 78.23% using Logistic Regression. However, priority level P3 did not obtain good results with Naive Bayes and achieved an accuracy of 70. 61%. It obtained the maximum accuracy of 85.84% using the Random Forest classifier.

We have concluded that after applying SMOTE, almost each priority level succeeded in acquired good results with a textual feature. Textual features give detailed information of the bug reports. Based on these features, we might train the model to accurately predict the priority of bug reports after handling the class imbalance problem.

D. PREDICTING PRIORITY FROM CATEGORICAL FEATURES WITH SMOTE

When predicting the priority of bug reports after applying SMOTE by using categorical features, results shown in TABLE 16 were obtained.

TABLE 16. Results of classification algorithms from categorical features with SMOTE.

Method	Accuracy %	Precision %	Recall%	f1-score%
Naive Bayes	43.31	41.20	42.40	42.20
Decision Tree	87.32	87.20	88.40	87.60
Random Forest	88.47	88.20	89.00	88.40
Logistic Regression	41.24	40.80	38.80	38.40

After handling the class imbalance problem, the framework performed well with Decision Tree and Random Forest classifier as their accuracy is 87.32% and 88.47% respectively. The other performance measures of these classifiers are: 87.20% precision, 88.40% recall and 87.60% f1-score in Decision Tree whereas, 88.20% precision, 89.00% recall and 88.40% f1-score in Random Forest classifier. The other two classifiers of Naive Bayes and Logistic Regression didn't work well with categorical features after applying SMOTE. Naive Bayes achieved the accuracy of 43.31% with 41.20% precision, 42.40% recall and 42.20% f1-score measure. Logistic Regression acquired the lowest accuracy of 41.24% as well as with the lowest performance measures i.e., 40.80% precision, 38.80% recall and 38.49% f1-score.

When we compared these results with the results before class balancing, it has been shown that there is an improvement only in Decision Tree and Random Forest classifiers after applying SMOTE using categorical features. Whereas, the performance measures are decreased in Naive Bayes and Logistic Regression. They didn't perform well after handling the class imbalance problem. Therefore, we have concluded that after applying SMOTE, the framework performed well with categorical features by using only Decision Tree and Random Forest classifiers.

1) PRIORITY WISE RESULTS FROM CATEGORICAL FEATURES WITH SMOTE

After applying SMOTE with categorical features, the accuracy of each priority level is shown in TABLE 17.

TABLE 17. Priority wise results of classification algorithms from categorical features with SMOTE.

Method	Priority Levels (Accuracy %)				
	P1	P2	P3	P4	P5
Naive Bayes	25.25	70.46	44.24	31.65	41.07
Decision Tree	93.93	87.04	75.22	94.96	91.07
Random Forest	93.43	89.11	80.53	91.36	91.07
Logistic Regression	32.82	65.28	47.34	17.26	32.14

It is observed that Decision Tree and Random Forest classifiers achieved good results in all priority levels except P3, which obtained 75.22% accuracy using Decision Tree. The Decision Tree classifier achieved the highest accuracy of 94.96% in priority level P4 among all the priority levels. Naive Bayes and Random Forest classifiers could not attain a satisfactory level of accuracy in predicting priority levels, even after applying SMOTE. Naive Bayes classifier acquired the lowest accuracy of 25.25% in P1 and a maximum of 70.46% in P2. The model worked poorly with Logistic Regression as it acquired the lowest accuracy of 17.26% in P4 among all priority levels while the maximum accuracy is achieved in P2 which is 65.28%. Therefore, we have concluded that Decision Tree and Logistic Regression classifiers obtained better results in each priority level after applying SMOTE by using categorical features.

V. CONCLUSION

Bug reports play a crucial role in software development and maintenance activities. They allow software developers,

quality assurance team, and customers to identify and report related issues. These reports entail extensive details, hence manual extraction of this information is infeasible due to large time complexity. Therefore, an automated mechanism is needed for their categorization and prioritization. The focus of this research is to automate the process of categorization and prioritization of bug reports. We propose CaPBug, a machine learning based framework that recommends a category and priority level to each issue based on information available in the bug reports. The approach is based on supervised machine learning that uses textual and categorical features of the dataset. We conducted an experimental study on 2,138 Eclipse and Mozilla bug reports from the Bugzilla dataset. We manually labeled the dataset with bug categories and applied NLP techniques and machine learning classifiers for predicting category and priority of bug reports.

The CaPBug framework utilizes both textual and categorical features to predict the category and priority of bugs. The summary feature was taken as a textual feature for training the model. Since, the summary feature includes detailed information about the bug reports, therefore, it was more significant in predicting the category and priority. Moreover, as the dataset was highly imbalanced with respect to priority class, we applied SMOTE to correctly train the model with each priority level. Categorical features worked well with SMOTE in predicting priority, but only with few classifiers. Nonetheless, it is evident from the above results that the framework is giving better results with a textual feature for predicting both the category and the priority of bug reports.

The research concluded that the framework achieved the highest accuracy of 88.78% for category prediction and 90.43% for priority prediction by using the textual feature with Random Forest classifier.

We intend to enhance the CaPBug framework by adding more records in the training dataset. Consequently, the framework can be used to predict both the priority and category of new bug reports. Additionally, deep learning techniques can be applied in future to further improve the performance and results.

REFERENCES

[1] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," *Empirical Softw. Eng.*, vol. 20, no. 2, pp. 516–548, 2015.

[2] C.-Z. Yang, C.-M. Ao, and Y.-H. Chung, "Towards an improvement of bug report summarization using two-layer semantic information," *IEICE Trans. Inf. Syst.*, vol. 101, no. 7, pp. 1743–1750, 2018.

[3] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for Android applications," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, Aug. 2015, pp. 673–686.

[4] W. Zhang, S. Wang, and Q. Wang, "KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity," *Inf. Softw. Technol.*, vol. 70, pp. 68–84, Feb. 2016, doi: 10.1016/j.infsof.2015.10.004.

[5] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *J. Syst. Softw.*, vol. 121, pp. 223–233, Nov. 2016.

[6] X. Xia, D. Lo, E. Shihab, and X. Wang, "Automated bug report field reassignment and refinement prediction," *IEEE Trans. Rel.*, vol. 65, no. 3, pp. 1094–1113, Sep. 2016.

[7] D.-G. Lee and Y.-S. Seo, "Improving bug report triage performance using artificial intelligence based document generation model," *Hum.-Centric Comput. Inf. Sci.*, vol. 10, no. 1, pp. 1–22, Dec. 2020, doi: 10.1186/s13673-020-00229-7.

[8] H. Zhong and Z. Su, "An empirical study on real bug fixes," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 913–923, doi: 10.1109/ICSE.2015.101.

[9] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Inf. Softw. Technol.*, vol. 82, pp. 177–192, Feb. 2017.

[10] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," in *Proc. Int. Conf. Comput. Intell. Data Sci. (ICCIDS)*, vol. 132, 2018, pp. 352–361, doi: 10.1016/j.procs.2018.05.194.

[11] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *J. Syst. Softw.*, vol. 152, pp. 165–181, Jun. 2019.

[12] M. R. Karim, "Key features recommendation to improve bug reporting," in *Proc. IEEE/ACM Int. Conf. Softw. Syst. Processes (ICSSP)*, May 2019, pp. 1–4, doi: 10.1109/ICSSP.2019.00010.

[13] Y. Fan, X. Xia, D. Lo, and A. E. Hassan, "Chaff from the wheat: Characterizing and determining valid bug reports," *IEEE Trans. Softw. Eng.*, vol. 46, no. 5, pp. 495–525, May 2020, doi: 10.1109/TSE.2018.2864217.

[14] N. Limsettho, H. Hata, A. Monden, and K. Matsumoto, "Automatic unsupervised bug report categorization," in *Proc. 6th Int. Workshop Empirical Softw. Eng. Pract.*, Nov. 2014, pp. 7–12.

[15] M. F. Zibran, "On the effectiveness of labeled latent Dirichlet allocation in automatic bug-report categorization," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, May 2016, pp. 713–715.

[16] N. Limsettho, H. Hata, A. Monden, and K. Matsumoto, "Unsupervised bug report categorization using clustering and labeling algorithm," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 7, pp. 1027–1053, Sep. 2016, doi: 10.1142/S0218194016500352.

[17] C. Zhou, B. Li, X. Sun, and H. Guo, "Recognizing software bug-specific named entity in software bug repository," in *Proc. IEEE/ACM 26th Int. Conf. Program Comprehension (ICPC)*, May/Jun. 2018, pp. 108–119.

[18] M. Hammad, R. Alzyoudi, and A. F. Otoom, "Automatic clustering of bug reports," *Int. J. Adv. Comput. Res.*, vol. 8, no. 39, pp. 313–323, Nov. 2018, doi: 10.19101/IJACR.2018.839013.

[19] X. Ren, Q. Huang, X. Xia, Z. Xing, L. Bao, and D. Lo, "Characterizing common and domain-specific package bugs: A case study on ubuntu," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 426–431.

[20] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Softw. Eng.*, vol. 20, no. 5, pp. 1354–1383, Oct. 2015, doi: 10.1007/s10664-014-9331-y.

[21] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, pp. 145–180, Feb. 2017, doi: 10.1007/s10462-016-9478-6.

[22] Y. Wang, T. He, W. Zhang, C. Fang, and B. Luo, "Exploring the influence of feature selection techniques on bug report prioritization," in *Proc. 29th Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2017, pp. 179–184.

[23] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018, doi: 10.1109/ACCESS.2018.2850910.

[24] T.-D. B. Le, F. Thumg, and D. Lo, "Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools," *Empirical Softw. Eng.*, vol. 22, no. 4, pp. 2237–2279, 2017, doi: 10.1007/s10664-016-9484-y.

[25] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "A time-based approach to automatic bug report assignment," *J. Syst. Softw.*, vol. 102, pp. 109–122, Apr. 2015, doi: 10.1016/j.jss.2014.12.049.

[26] N. Pandey, D. Kumar, S. Abir, and H. Amitava, "Automated classification of software issue reports using machine learning techniques: An empirical study," *Innov. Syst. Softw. Eng.*, vol. 13, no. 4, pp. 279–297, 2017.

[27] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in *Proc. 9th Int. Conf. Inf. Commun. Manage. (ICICM)*, 2019, pp. 17–21, doi: 10.1145/3357419.3357424.

[28] Y. Tian, N. Ali, D. Lo, and A. E. Hassan, "On the unreliability of bug severity data," *Empirical Softw. Eng.*, vol. 21, no. 6, pp. 2298–2323, Dec. 2016, doi: 10.1007/s10664-015-9409-1.

[29] I. Ferreira, E. Cirilo, V. Vieira, and F. Mourão, "Bug report summarization: An evaluation of ranking techniques," in *Proc. X Brazilian Symp. Softw. Compon., Archit. Reuse (SBCARS)*, Sep. 2016, pp. 101–110, doi: 10.1109/SBCARS.2016.17.

[30] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *Proc. IEEE/ACM 26th Int. Conf. Program Comprehension (ICPC)*, May/Jun. 2018, pp. 144–14411.

[31] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 2–22, Mar. 2019.

[32] Z. Ge, Z. Song, S. X. Ding, and B. Huang, "Data mining and analytics in the process industry: The role of machine learning," *IEEE Access*, vol. 5, pp. 20590–20616, 2017, doi: 10.1109/ACCESS.2017.2756872.

[33] R. R. Panda and N. K. Nagwani, "Software bug categorization technique based on fuzzy similarity," in *Proc. IEEE 9th Int. Conf. Adv. Comput. (IACC)*, Dec. 2019, pp. 1–6.

[34] M. Kumari and V. B. Singh, "An improved classifier based on entropy and deep learning for bug priority prediction," in *Proc. 18th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, 2018, pp. 571–580.

[35] Q. Umer, H. Liu, and I. Illahi, "CNN-based automatic prioritization of bug reports," *IEEE Trans. Rel.*, vol. 69, no. 4, pp. 1341–1354, Dec. 2020.

[36] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Trans. Softw. Eng.*, vol. 46, no. 8, pp. 836–862, Aug. 2020.

[37] Y. Tan, S. Xu, Z. Wang, T. Zhang, Z. Xu, and X. Luo, "Bug severity prediction using question-and-answer pairs from stack overflow," *J. Syst. Softw.*, vol. 165, Jul. 2020, Art. no. 110567, doi: 10.1016/j.jss.2020.110567.

[38] R. Chen, S.-K. Guo, X.-Z. Wang, and T.-L. Zhang, "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 12, pp. 2406–2420, Dec. 2019.

[39] Y. Xiao, J. Keung, K. E. Bennin, and Q. Mi, "Improving bug localization with word embedding and enhanced convolutional neural networks," *Inf. Softw. Technol.*, vol. 105, pp. 17–29, Jan. 2019, doi: 10.1016/j.infsof.2018.08.002.

[40] P. A. Choudhary and D. S. Singh, "Neural network based bug priority prediction model using text classification techniques," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 5, pp. 1315–1320, 2017.

[41] H. M. Tran, S. T. Le, S. V. Nguyen, and P. T. Ho, "An analysis of software bug reports using machine learning techniques," *Social Netw. Comput. Sci.*, vol. 1, no. 1, p. 4, Jan. 2020, doi: 10.1007/s42979-019-0004-1.

[42] A. Lamkanfi, J. Perez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 203–206, doi: 10.1109/MSR.2013.6624028.

[43] S. Banerjee, J. Helmick, Z. Syed, and B. Cukic, "Eclipse vs. Mozilla: A comparison of two large-scale open source problem report repositories," in *Proc. IEEE 16th Int. Symp. High Assurance Syst. Eng.*, Jan. 2015, pp. 263–270, doi: 10.1109/HASE.2015.45.

[44] R. M. Karampatsis and C. Sutton, "How often do single-statement bugs occur? The ManySStuBs4J dataset," in *Proc. 17th Int. Conf. Mining Softw. Repositories*, Jun. 2020, pp. 573–577, doi: 10.1145/3379597.3387491.

[45] V. Lelli, A. Blouin, and B. Baudry, "Classifying and qualifying GUI defects," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2015, pp. 1–10, doi: 10.1109/ICST.2015.7102582.

[46] Z. Wan, D. Lo, X. Xia, and L. Cai, "Bug characteristics in blockchain systems: A large-scale empirical study," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*, May 2017, pp. 413–424, doi: 10.1109/MSR.2017.59.

[47] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5803–5833, Dec. 2016, doi: 10.1002/sec.1737.

[48] K. Sangeetha and K. Ravikumar, "Defense against protocol level attack in tor network using deficit round robin queuing process," *Egyptian Informat. J.*, vol. 19, no. 3, pp. 199–205, Nov. 2018, doi: 10.1016/j.eij.2018.03.005.

[49] N. Munaiah, F. Camilo, W. Wigham, A. Meneely, and M. Nagappan, "Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project," *Empirical Softw. Eng.*, vol. 22, no. 3, pp. 1305–1347, Jun. 2017, doi: 10.1007/s10664-016-9447-3.

[50] W. Wen, T. Yu, and J. H. Hayes, "CoLUA: Automatically predicting configuration bug reports and extracting configuration options," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2016, pp. 150–161.

[51] X. Han, T. Yu, and D. Lo, "PerfLearner: Learning from bug reports to understand and generate performance test frames," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 17–28.

[52] P. E. Strandberg, T. J. Ostrand, E. J. Weyuker, W. Afzal, and D. Sundmark, "Intermittently failing tests in the embedded systems domain," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 337–348, doi: 10.1145/3395363.3397359.

[53] D. G. Lee and Y. S. Seo, "Systematic review of bug report processing techniques to improve software management performance," *J. Inf. Process. Syst.*, vol. 15, no. 4, pp. 967–985, 2019, doi: 10.3745/JIPS.04.0130.

[54] C. Sun, V. Le, Q. Zhang, and Z. Su, "Toward understanding compiler bugs in GCC and LLVM," in *Proc. 25th Int. Symp. Softw. Test. Anal.*, Jul. 2016, pp. 294–305, doi: 10.1145/2931037.2931074.

[55] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, vol. 7, pp. 46846–46857, 2019, doi: 10.1109/ACCESS.2019.2909746.

[56] A. Baarah, A. Aloqaily, Z. Salah, M. Zamzeer, and M. Sallam, "Machine learning approaches for predicting the severity level of software bug reports in closed source projects," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 8, pp. 285–294, 2019, doi: 10.14569/IJACSA.2019.0100836.

[57] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors*, vol. 19, no. 13, p. 2964, Jul. 2019, doi: 10.3390/s19132964.

[58] A. A. Mohamed, "An effective dimension reduction algorithm for clustering arabic text," *Egyptian Informat. J.*, vol. 21, no. 1, pp. 1–5, Mar. 2020, doi: 10.1016/j.eij.2019.05.002.

[59] X. Du, Z. Zhou, B. Yin, and G. Xiao, "Cross-project bug type prediction based on transfer learning," *Softw. Qual. J.*, vol. 28, pp. 39–57, Sep. 2020, doi: 10.1007/s11219-019-09467-0.

[60] T. Uçkan and A. Karci, "Extractive multi-document text summarization based on graph independent sets," *Egyptian Informat. J.*, vol. 21, no. 3, pp. 145–157, Sep. 2020, doi: 10.1016/j.eij.2019.12.002.

[61] Z. A. Nizamani, H. Liu, D. M. Chen, and Z. Niu, "Automatic approval prediction for software enhancement requests," *Automated Softw. Eng.*, vol. 25, no. 2, pp. 347–381, Jun. 2018, doi: 10.1007/s10515-017-0229-y.

[62] S. Qaiser and R. Ali, "Text mining: Use of TF-IDF to examine the relevance of words to documents," *Int. J. Comput. Appl.*, vol. 181, no. 1, pp. 25–29, Jul. 2018, doi: 10.5120/ijca2018917395.

[63] S. W. Kim and J. M. Gil, "Research paper classification systems based on TF-IDF and LDA schemes," *Hum.-Centric Comput. Inf. Sci.*, vol. 9, no. 1, p. 30, 2019, doi: 10.1186/s13673-019-0192-7.

[64] N. K. Singha Roy and B. Rossi, "Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results," in *Proc. 43rd Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2017, pp. 426–429, doi: 10.1109/SEAA.2017.71.

[65] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," 2019, *arXiv:1905.06872*. [Online]. Available: https://arxiv.org/abs/1905.06872

[66] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Proc. IEEE 23rd Int. Requirements Eng. Conf. (RE)*, Aug. 2015, pp. 116–125, doi: 10.1109/RE.2015.7320414.

[67] R. Punnoose and P. Ajit, "Prediction of employee turnover in organizations using machine learning algorithms," *Int. J. Adv. Res. Artif. Intell.*, vol. 5, no. 9, pp. 22–26, 2016, doi: 10.14569/ijarai.2016.050904.

[68] S. Delphine Immaculate, M. Farida Begam, and M. Floramary, "Software bug prediction using supervised machine learning algorithms," in *Proc. Int. Conf. Data Sci. Commun. (IconDSC)*, Mar. 2019, pp. 1–7, doi: 10.1109/icondsc.2019.8816965.

[69] C.-C. Wu, Y.-L. Chen, Y.-H. Liu, and X.-Y. Yang, "Decision tree induction with a constrained number of leaf nodes," *Int. J. Speech Technol.*, vol. 45, no. 3, pp. 673–685, Oct. 2016, doi: 10.1007/s10489-016-0785-z.

[70] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Identifying gameplay videos that exhibit bugs in computer games," *Empirical Softw. Eng.*, vol. 24, no. 6, pp. 4006–4033, Dec. 2019, doi: 10.1007/s10664-019-09733-6.

[71] A. Alqudsi and A. El-Hag, "Application of machine learning in transformer health index prediction," *Energies*, vol. 12, no. 14, p. 2694, 2019, doi: 10.3390/en12142694.

[72] H. Kartal, A. Oztekin, A. Gunasekaran, and F. Cebi, "An integrated decision analytic framework of machine learning with multi-criteria decision making for multi-attribute inventory classification," *Comput. Ind. Eng.*, vol. 101, pp. 599–613, Nov. 2016, doi: 10.1016/j.cie.2016.06.004.

[73] S. Guo, R. Chen, M. Wei, H. Li, and Y. Liu, "Ensemble data reduction techniques and multi-RSMOTE via fuzzy integral for bug report classification," *IEEE Access*, vol. 6, pp. 45934–45950, 2018, doi: 10.1109/ACCESS.2018.2865780.
[74] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 368–410, Apr. 2016.
[75] U. Subbiah, M. Ramachandran, and Z. Mahmood, "Software engineering approach to bug prediction models using machine learning as a service (MLaaS)," in *Proc. 13th Int. Conf. Softw. Technol.*, 2018, pp. 879–887, doi: 10.5220/0006926308790887.
[76] J. P. Winkler, J. Gronberg, and A. Vogelsang, "Optimizing for recall in automatic requirements classification: An empirical study," in *Proc. IEEE 27th Int. Requirements Eng. Conf. (RE)*, Sep. 2019, pp. 40–50, doi: 10.1109/RE.2019.00016.
[77] H. Jiang, N. Nazar, J. Zhang, T. Zhang, and Z. Ren, "PRST: A pagerank-based summarization technique for summarizing bug reports with duplicates," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, no. 6, pp. 869–896, 2017, doi: 10.1142/S0218194017500322.
[78] K. Goseva-Popstojanova and J. Tyo, "Identification of security related bug reports via text mining using supervised and unsupervised classification," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2018, pp. 344–355, doi: 10.1109/QRS.2018.00047.

HAFIZA ANISA AHMED received the bachelor's degree in computer science from Jinnah University for Women, Karachi, Pakistan, in 2012, secure second position. She is currently pursuing the master's degree in computer science with the Virtual University of Pakistan, Karachi campus. She is currently working as a Lecturer with Jinnah University for Women. Her research interests include machine learning and data mining, natural language processing, algorithms, and programming languages.

NARMEEN ZAKARIA BAWANY received the Ph.D. degree in computer science from the National University of Computer and Emerging Sciences, Karachi. She is currently working as a Professor and the Dean of the Faculty of Science, Jinnah University for Women, Karachi. She has supervised many projects. She had received funding from Ignite (National Technology Fund, Pakistan) for her projects. She has more than 30 publications in journals and conferences. Her research interests include human–computer interaction, semantic web, cyber security, and software defined networking.

JAWWAD AHMED SHAMSI received the Ph.D. degree in computer science from Wayne State University, Detroit, MI, USA, in 2009. He is currently a Professor and the Director of the National University of Computer and Emerging Sciences, Karachi campus. His research has been funded by the NVIDIA Research Center and HEC NRPU grants. He has over 50 research publications in reputable journals and conferences. His research interests include distributed systems, networks, security, and high-performance computing.