ICIC Express Letters
Volume **17**, Number **12**, December **2023**
 ISSN 1881-803X
pp. **1365–1373**

COMPARATIVE STUDY OF SUPERVISED MACHINE LEARNING MODELS FOR MULTICLASSIFICATION IN BUG REPORT DOMAIN

Jantima Polpinij[1], Manasawee Kaenampornpan[1,*], Umaporn Saisangchan[1]
Samruan Wiangsamut[1] and Bancha Luaphol[2]

[1]Faculty of Informatics
Mahasarakham University
Kantaravichai District, Mahasarakham 44150, Thailand
{ jantima.p; umaporn.s; samruan.w }@msu.ac.th
*Corresponding author: manasawee.k@msu.ac.th

[2]Department of Digital Technology
Faculty of Administrative Science
Kalasin University
Namon District, Kalasin 46230, Thailand
bancha.lu@ksu.ac.th

Received March 2023; accepted May 2023

Abstract. *The challenge in this study was to multiclassify bug reports, and the proposed method attempted to assign bug reports into three categories: real-bug, enhancement, and task. The dataset that is used in this study was obtained from the Bugzilla system and was connected to the opensource Firefox browser. Our approach began with bug report pre-processing. It was driven by replacing contractions, tokenization, spelling correction, punctuation and stop-word removal, CamelCase processing, and stemming and lowercase conversion, in that order. We compared two features of bug reports (i.e., unigram words only and unigram together with CamelCase words). The pre-processed bug reports were afterwards formatted in a vector space model format, with each term weighed using a term weighting scheme. In addition, term frequency (tf) and term frequency-inverse gravity moment (tf-igm) used to assign weight for each term were examined in this research. Following that, the vector of bug reports was utilized to build the multi-classifier models. Logistic Regression, Multinomial Naïve Bayes, eXtreme Gradient Boosting, Linear Support Vector Machines, Random Forest, and Neural Networks were all evaluated. Finally, it was determined that the Linear Support Vector Machine classifier was the most suitable model for our dataset.*
Keywords: Multiclassification, Bug report, Real-bug report, Enhancement report, Task report, *tf*, *tf-igm*, Supervised machine learning

1. **Introduction.** It is common that all software contains faults, defects, and errors, which are colloquially known as "*bugs*". When software users detect software defects, they will notify them to the software developer team in the form of "*bug report*", which contains critical error, defect, or bug information for software maintenance [1-4]. Most of software projects employ bug reports as guidelines for maintaining and enhancing the functionality and quality of software. When software world-wide end users are the primary source of knowledge about software issues, the main challenge is how to effectively gather bug reports from those software end users. To effectively manage bug reporting and issue triaging, there have been various bug tracking systems (BTS) established and proposed, where end-users become the key source for identifying and reporting software bugs and they can simply report software defects discovered using a BTS [1-4]. As a result, various BTSs have been proposed to manage bug reporting and triaging, including Bugzilla,

DOI: 10.24507/icicel.17.12.1365

Trace, FogBugz, Mantis, Backlog, and Jira. Software end users continuously generate and upload many reports about software issues to BTSs. Unfortunately, many reports submitted to the BTS system do not concern software bugs. Therefore, before using substantial information from them for software quality improvement or maintenance, all reports must be examined to identify real-bug reports. However, previous research revealed that 39% of bug reports that were first designated as real-bug reports did not contain any software defect or bug information, known as non-bug reports [4,5]. This issue was called misclassification issue between real-bug and non-bug reports [4,5]. As a result, an automatic process of identifying and filtering out non-bug reports is required to reduce software cost and bias analysis, where information in real-bug reports can be used to fix bugs. Several methods based on binary classification have been proposed for filtering out non-bug reports from the repositories before analysis [4-8].

Bug reports that are not considered as real-bug reports are often overlooked. However, those overlooked bug reports can also include enhancement and task bug reports [9,10]. Enhancement bug reports describe new software features or user interface (UI) software performance that should be improved [9,10]. Therefore, information concerning enhancement bug reports can be utilized to improve software products without engineering change. Meanwhile, task bug reports consider refactoring, removal, replacement, enabling or disabling of functionality and any other engineering tasks [9,10]. This issue becomes a multiclassification task [11].

As previously stated, there are more types of bug reports than just real-bug and non-bug reports. These are real-bug, enhancement, and task reports. This is known as multiclassification issue and this issue for bug reports has not yet been completely studied in the domain of bug report analysis, where it includes other study domains of bug reports, namely severity and priority analysis [12-15]. Chaturvedi and Singh [12] aimed to demonstrate the usefulness of machine learning techniques, including Naïve Bayes, k-Nearest Neighbor (KNN), Multinomial Naïve Bayes (MNB), Support Vector Machine (SVM), J48, and RIPPER, for classifying the bug severity of NASA bug report data from the PROMISE repository. Meanwhile, Kumar and Singla [13] proposed a comparative analysis of Decision Tree, Bagging and Naïve Bayes approach for the bug severity classification. Kukkar et al. [14] proposed a revolutionary deep learning model for multiclass severity classification that uses a Convolutional Neural Network and Random forest with Boosting to address these issues (BCR). The text of the bug report was preprocessed using natural language processing methods, and then the *n-gram* technique was utilized to extract features from bug reports. Afterwards, the Convolutional Neural Network was employed to identify the significant feature patterns of each severity class. Finally, the random forest with boosting was utilized to classify the multiple bug severity classes. Meng et al. [15] applied BERT transformer and TF-IDF to leveraging the features of the intention and the multiple text information. Later, the features were used to train the classifiers by KNN, MNB, SVM, Logistic Regression (LR) and Random Forest (RF). However, Kaewnoo and Senivongse [16] mentioned that there are a few studies for identifying types of bug reports based on multiclassification problem domain. This aspect presented a problem for the purpose of this study. Even after completely fixing bugs in the software, quality improvement and maintenance are still required, with the added necessity of enhancement and task reports.

The purpose of this study was to examine machine learning algorithms for developing multiclassification models used to assign bug reports to the three types of real-bug, enhancement, and task reports. Our method compared two bug report features, unigram only and unigram, together with CamelCase, and compared two term weighting schemes (i.e., *tf* and *tf-igm*). By utilizing a modestly sized dataset, we then benchmarked six popular machine learning algorithms, i.e., Logistic Regression (LR), Multinomial Naïve Bayes (MNB), eXtreme Gradient Boosting (XGBoost, XGB), Linear Support Vector Machines

(Linear SVM), Random Forest (RF), and Neural Networks (NN) in order to identify the most applicable classifier model for our dataset.

The structure of this paper is as follows. In Section 2, it provides the materials and dataset. The research method is provided in Section 3. Section 4 presents the experiment's results and analysis. Finally, Section 5 gives the study's conclusion.

2. Materials and Dataset.

This section has described the toolkits and datasets utilized for the research.

Materials: In this study, we trained a text categorization model using Python's NLTK (Natural Language Toolkit) and Scikit-Learn package. However, we have also developed certain methods ourselves using Python, such as *tf-igm*.

Dataset: The Bugzilla system was used to download our dataset. It was connected to Firefox's open source. The structure of a bug report is depicted in Figure 1. It consists of predefined fields, free-form text, attachments, and dependencies [17,18]. The predefined fields give categorical information about the bug report. In addition, they consist of product component, operating system, version, priority, and severity. The free-form text contains the report's title, a complete explanation of the defect, and extra remarks; attachments related to non-textual supplementary material (e.g., a screenshot of erroneous behavior). In this study, we used the title of the report (also known as the 'summary' part) for the free-form text.

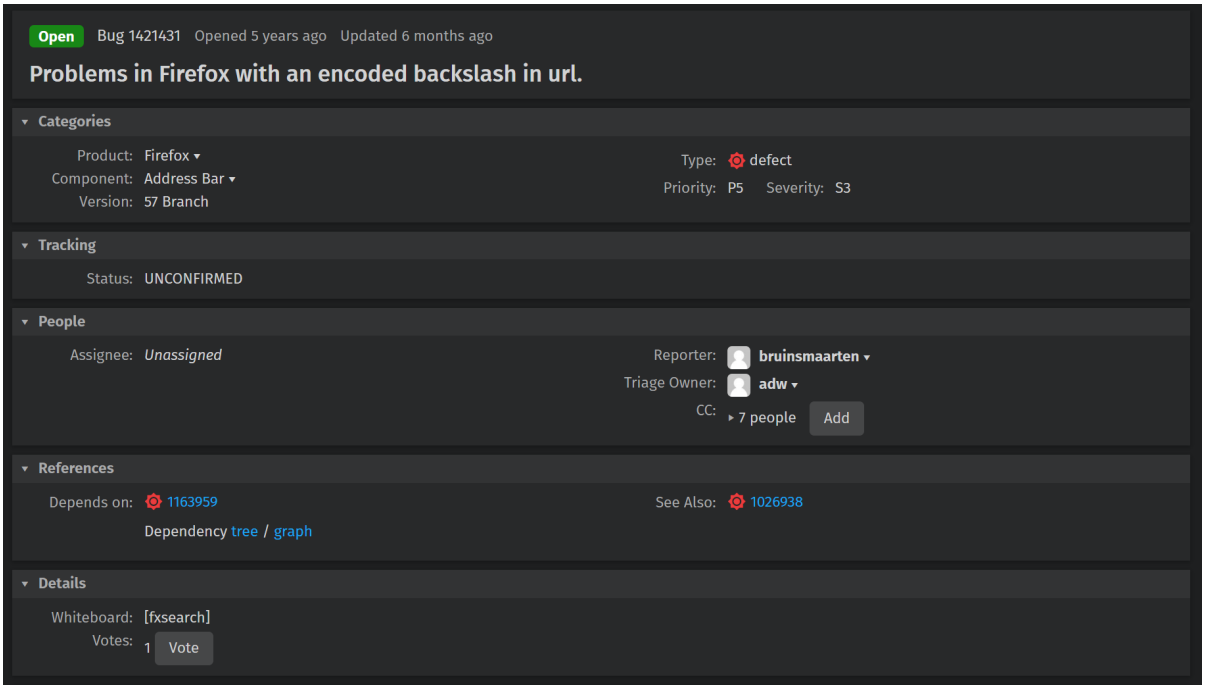

FIGURE 1. Example of bug report from the Bugzilla system

The bug reports were submitted to the Bugzilla system between 1 October 2018 and 30 September 2021, and our dataset was obtained on September 30, 2021. The dataset included 21,920 reports that were divided into three categories: defects, enhancement, and tasks. There were 14,849 reports in the defect category, 4,242 in the enhancement category, and 2,829 in the task category.

We chose 2,500 reports per class for the first step, also known as the bug report classifier modeling stage, in order to lessen the effects of the unbalanced data brought on by the excessively skewed class distribution. Additionally, the data partitioning method known as k-fold cross-validation divides an entire dataset into k groups. We employ $k-1$ folds for model training and the remaining fold (known as a validation set) is used for performance

assessment. The value of k in this study is 10. Finally, the remaining bug reports that were not used in the first step would be used as test set for the second stage, also known as the experiment stage.

3. **Research Methods.** This section has described the research method used in the study.

3.1. **Bug report pre-processing.** Many previous researches on bug reports uses only the summary part because this section has been proven to have less noise [2,3,18-20]. Therefore, we also utilize only the summary part for our study. Two features, i.e., unigrams and CarmelCases, are used in this study. Unigram refers to single words [20], while CamelCase refers to the use of two (or more) words or abbreviations without punctuation or spaces between them to form a new term (e.g., browser_tools, and UrlBar) [20]. CamelCase words can assist to demonstrate the software's specificity.

Each step of our bug report pre-processing in this study is detailed as follows.

- Replacing contractions: This pre-processing step replaces contractions with their full forms, e.g., "doesn't" with "does not".
- Tokenization: This pre-processing step is performed to break down a bug report into smaller units known as tokens. A token in this study is "*word*".
- Spelling corrections: It is probable that many of the words in bug reports contain misspellings. As a result, this pre-processing step is performed to correct those words in order to reduce linguistic ambiguity.
- Punctuation and stop-word removal: This preprocessing step removes punctuation and non-informative terms (e.g., "*so*", "*and*", and "*or*") from the bug report. This assists in eliminating unnecessary words or noise.
- CarmelCase Processing: This pre-processing step splits a CarmelCase word into single words; both the original CamelCase words and their single words are subsequently utilized as features. This can assist in reducing the problem of short text.
- Stemming and lowercase conversion: This pre-processing step is conducted to reduce inflected words to their '*word stem*' using the Python NLTK's snowball algorithm. This may help to reduce noise in bug reports.

Table 1 shows an example of pre-processing for a bug report summary.

TABLE 1. An example of each pre-processing step for a bug report's summary

Pre-processing steps	**Results**
An original bug report summary	The AutoComplete function cannot use. Its work is wrong.
Replacing contractions	The AutoComplete function cannot use. Its work is wrong.
Tokenization	The / AutoComplete / function / / cannot / use / Its / work / is / wrong
Spelling corrections	The / AutoComplete / function / / cannot / use / Its / work / is / wrong
Punctuation and stop-word removal	AutoComplete / function / cannot / use / work / wrong
CarmelCase processing	AutoComplete / Auto / Complete / function / cannot / use / work / wrong
Stemming and lowercase conversion	autocomplet / auto / complet / function / can / not / use / work / wrong

3.2. **Bug report representation and term weighting.** After the pre-processing stage, the pre-processed bug reports were represented in *Vector Space Model* (*VSM*) format, where each input bug report D is represented in the VSM format as an n-dimensional vector, and n is the total number of distinct words used in the bug report. Afterwards, each bug report feature (or word) was given a weight using a term weighting method. This study examined *tf* and *tf-igm* [21]. Many previous studies on bug reports noted that *tf* returned good results for bug report analysis [5]. *tf* is the number of times that a certain term t appears within a bug report, denoted as d. Meanwhile, *tf-igm* proposed by Chen et al. [21] is a supervised term weighting. It may be able to accurately calculate a word's distinguishing class, especially in multiclass cases as per the following equations.

$$tf\text{-}igm(t_i, d_j) = tf(t_i, d_j) \times (1 + \lambda \times igm(t_i)) \tag{1}$$

$$igm(t_i) = \frac{f_{i1}}{\sum_{r=1}^{M} f_{ir} \times r} \tag{2}$$

$tf(t_i, d_j)$ represents the measurement of how frequently a term t_i occurs within a document d_j. For $igm(t_i)$, f_{ir} $(r = 1, 2, \ldots, M)$ denotes the total number of bug reports in the rth class that include the term t_i. These bug reports are sorted in descending order. Thus, f_{i1} indicates the frequency of t_i in the class in which it appears the most frequently, while an adjustable coefficient (λ) is employed to maintain the relative balance of the global weight *igm* and the local weight *tf* in a term's weight. This study sets the coefficient value as 7.0, but this can be altered to a value between 5.0 and 9.0.

3.3. **Machine learning algorithms used for bug report classifier modeling.** After obtaining the training set of bug reports in the VSM format, this vector was used to model bug report classifiers based on multiclassification. In order to compare the performance of machine learning algorithms [20,22], six supervised machine learning algorithms were applied to creating the bug report classifier models. These algorithms are described as follows.

***Logistic Regression* (*LR*):** This algorithm can be utilized to solve classification problems by establishing thresholds for the probability predicted for each class [23]. Although this algorithm is commonly used for binary classification, it can easily adapt to multiple classes. In LR, given N bug reports x_i, $i = 1, \ldots, N$, the m features of the input bug report $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$, are linearly integrated using coefficient β_0 and $\beta = (\beta_0, \ldots, \beta_{im})$ to predict the classification outcome y_i. Specifically, given an input bug report x_i, the probability that $y_i = 1$ is indicated by $P(x_i)$ and is modelled with the conventional logistic regression model as follows:

$$P(x_i) = \frac{e^{(\beta_0 + \beta * x_i)}}{1 + e^{(\beta_0 + \beta * x_i)}} \tag{3}$$

The LR classifiers employ a sigmoid function to process the weighted combination of input features. By using the sigmoid function, any real value can be converted to a number between 0 and 1.

***Multinomial Naïve Bayes* (*MNB*):** The MNB algorithm is based on Bayes' theorem and assumes that each feature is independent [24]. It is useful for classification tasks based on natural language processing and can be used to multinomially distributed datasets. It considers a feature vector in which a certain term denotes how frequently it appears. This algorithm first determines the percentage of documents in each class, denoted as $P(c)$, and then determines the likelihood of each word for a certain class, denoted as $P'(w|c)$, to create the classifier model. These formulas can be written as follows:

$$P(c) = \frac{N_{class}}{N} \tag{4}$$

where N represents number of all of bug reports in the training set, and N_{class} is the total number of bug reports detected in each class, and

$$P'(w_i|c) = \frac{count(w_i, c) + \alpha}{count(c) + |V| + 1} \tag{5}$$

where $count(w_i, c)$ shows how many times the term w_i is found in the class c. In the meantime, $count(c)$ denotes the total number of training set classes, and $|V|$ denotes the total number of distinct words inside the training set. Since some words have zero counts, Laplace smoothing is performed with a low value of $\alpha = 0.001$. Finally, the Bayes' rule is used to calculate an estimate of $P'(c|d)$ for the test documents. The MNB prediction formula is written as follows:

$$P'(c|d) = \arg\max P(c) \prod_{i=1}^{n} P'(w_i|c_j) \tag{6}$$

Support Vector Machines (SVM): The SVM algorithm classifies data by constructing a hyperplane and a decision boundary [24]. This algorithm computes the margin between a line and the support vectors. The points closest to the hyperplane are chosen as the support vector. The goal is to maximize the margin to the maximum degree possible. When the maximum margin is reached, the decision boundary is used to divide classes as large as possible, enabling classes to be distinguished more clearly. In this study, the linear kernel was used for the SVM technique since previous research had shown that this kernel function produced appropriate results.

Random Forest (RF): The RF algorithm is an ensemble learning approach based on decision trees that consists of many decision trees [25]. This approach utilizes bagging and feature randomization to create an uncorrelated forest of trees that improves prediction accuracy over a single tree. The RF algorithm might be effective in preventing overfitting. In this experiment, 100 decision trees are developed for our forest.

eXtreme Gradient Boosting (XGBoost, XGB): The XGB method is similar to the RF algorithm, but it additionally employs a gradient boosting approach to improve performance [26]. This algorithm has demonstrated to be extremely efficient, versatile, and portable. Also, the XGB algorithm might be effective in preventing overfitting. For our XGB, we created 100 decision trees.

Neural Networks (NN): The NN algorithm is a network of linked neurons (or nodes) composed of a number of node layers [27]. This algorithm has three levels: an input layer, one or more hidden layers, and an output layer. Each neuron has its own weight and threshold and is connected to other neurons. If the output of a certain node exceeds the set threshold, that node is activated and transfers data to the subsequent network layer. Otherwise, no data is sent to the following layer. In this work, we applied the Adam algorithm to optimizing adaptive learning rate.

4. **Experimental Results and Discussion.** This study used the metrics of Accuracy (Acc), F1, the Area Under Curve (AUC), and the Matthews Correlation Coefficient (MCC) to evaluate the efficacy of our approaches. The accuracy and F1 formulae are shown here.

$$recall = \frac{TP}{TP + FN} \tag{7}$$

$$precision = \frac{TP}{TP + FP} \tag{8}$$

$$F1 = 2 \times \frac{recall \times precision}{recall + precision} \tag{9}$$

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

where *TP* indicates the number of bug reports that were accurately recognized as real-bug reports and *TN* represents the number of bug reports that were accurately identified as other. *FN* denotes the number of false-positive bug reports, whereas *FP* reflects the number of false-positive bug reports that were misclassified as other.

AUC is used to assess classification quality by examining the area under the *Receiver Operating Characteristic* (*ROC*) *Curve*. The ROC curve is displayed against the *True Positive Rate* (*TPR*) and the *False Positive Rate* (*FPR*), with the TPR on the y-axis and the *FPR* on the x-axis. Meanwhile, the MCC is used to assess the classifier model's performance [1]. In general, MCC is best suited for binary classes; however, numerous researches have used it for multiclassification.

Table 2 shows the experimental results of the proposed method. Our test set includes 12,349 reports for the real-bug category, 1,742 reports for the enhancement category, and 329 reports for the task category. Because the *tf-igm* is a Supervised Term Weighting (STW) scheme, the results in Table 2 demonstrate that utilizing *tf-igm* with every algorithm produced better results than using *tf* term weighting scheme. The STW system is distinguished by its capacity to generate class distinguishing power by assessing the word significance in a bug report of a certain class. Simply speaking, the STW scheme shows variances in word weights for terms in different classes, while unusual words appear in a few manuscripts. This is the primary reason why *tf-igm* outperformed *tf*.

Table 2. The experimental results of six supervised machine learning models for multiclassification in bug report domain

Algorithms	***tf***				***tf-igm***			
	Acc	**F1**	**AUC**	**MCC**	**Acc**	**F1**	**AUC**	**MCC**
LR	0.709	0.709	0.781	0.563	0.712	0.713	0.784	0.568
MNB	0.687	0.688	0.765	0.531	0.711	0.712	0.783	0.567
SVM	0.709	0.709	0.782	0.564	**0.722**	**0.723**	**0.797**	**0.585**
RF	0.690	0.691	0.768	0.536	0.689	0.690	0.767	0.543
XGB	0.628	0.632	0.721	0.448	0.626	0.630	0.719	0.445
NN	0.655	0.654	0.741	0.482	0.677	0.676	0.757	0.515

When all of the algorithms used for modeling multi-classifiers for bug reports were considered, the experimental results in Table 2 show that the SVM multi-classifier with the *tf-igm* term weighting scheme produced the best results for accuracy, F1, AUC, and MCC at 0.722, 0.723, 0.797, and 0.585, respectively, while the LR, MNB, RF, XGB, and NN classifiers produced poorer results. This occurred as a result of utilizing only the title portion of the bug report, which decreased the number of features. SVM operates well on smaller datasets and outliers have less of an influence, whereas NN requires a large training set. As a result, when the training set is limited, the NN classifiers frequently produce inferior results. However, as additional data is used, the performance of the NN classifier improves, but it still performs worse than the SVM classifier.

Considering the LR results in Table 2. The LR classifiers produced poorer results than the SVM classifiers. This is due to the fact that the number of our features was fairly minimal, but perhaps still too much for LR. This might lead the LR classifiers to over-fit on the training set, inflating prediction accuracy while decreasing model performance on the test set. In simple words, the LR classifiers get underdetermined when the number of features exceeds the number of data points. When it comes to the performance, the MNB classifiers performed worse than the SVM classifiers. This is because the basic idea behind MNB is to keep an error rate as low as possible while assuming class conditional independence. Unfortunately, this is not often the case in practice, and the MNB classifier performance is frequently low.

The RF and XGB algorithms have several decision trees, which may have an influence on unimportant features. The RF, XGB and NN multi-classifier models require longer computational time for modeling and testing than the other models.

5. **Conclusions.** While binary classification has long been researched to distinguish between real-bug reports and non-bug reports, bug reports may really be divided into three groups (i.e., real-bug, enhancement, and task reports). The multiclassification of bug reports thus becomes difficult. The information that was acquired from the Bugzilla system consisted of 21,920 bug reports pertaining to the Firefox open source project. 14,849 reports in the real-bug class, 4,242 reports in the enhancement class, and 2,829 reports in the task class are included in our dataset. Our proposed method consisted of six main processing steps as pre-processing, bug report representation and term weighting, modeling multi-classifiers and evaluation. The first step is bug report pre-processing. It includes the processes of replacing contractions, tokenization, spelling correction, punctuation and stop-word removal, CamelCase processing, and stemming and lowercase conversion, respectively. This study used two bug report features (unigram only and unigram with CamelCase words), where the CamelCase words might identify the specificity of problem areas in certain program. Following that, the pre-processed bug reports were expressed in the VSM format, and each term's weight was assigned using *tf* and *tf-igm*. The vector of bug complaints was then utilized to model the multi-classifiers using six supervised machine learning methods (LR, MNB, SVM, RF, XGB, and NN) to find the best multi-classifier. After testing these multi-classifier models using accuracy, F1, AUC and MCC, the Linear SVM classifier model produced the best results. This research will be utilized in our future work to determine the priority or severity of bug reports.

Acknowledgment. This research project was financially supported by Mahasarakham University.

REFERENCES

[1] 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.

[2] P. Bhattacharya and I. Neamtiu, Bug-fix time prediction models: Can we do better?, *Proc. of the 8th Working Conference on Mining Software Repositories*, pp.207-210, 2011.

[3] N. Jalbert and W. Weimer, Automated duplicate detection for bug tracking systems, *2008 IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN)*, pp.52-61, 2008.

[4] J. Polpinij, A method of non-bug report identification from bug report repository, *Artificial Life and Robotics*, vol.26, pp.318-328, 2021.

[5] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh and Y. G. Guéhéneuc, Is it a bug or an enhancement? A text-based approach to classify change requests, *Proc. of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pp.304-318, 2008.

[6] N. Limsettho, H. Hata, A. Monden and K. Matsumoto, Automatic unsupervised bug report categorization, *The 6th International Workshop on Empirical Software Engineering in Practice*, pp.7-12, 2014.

[7] P. Terdchanakul, H. Hata, P. Phannachitta and K. Matsumoto, Bug or not? Bug report classification using N-Gram IDF, *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp.534-538, 2017.

[8] K. Herzig, S. Just and A. Zeller, It's not a bug, it's a feature: How misclassifcation impacts bug prediction, *The 35th International Conference on Software Engineering (ICSE)*, pp.392-401, 2013.

[9] N. Pingclasai, H. Hata and K. Matsumoto, Classifying bug reports to bugs and other requests using topic modeling, *The 20th Asia-Pacific Software Engineering Conference (APSEC)*, pp.13-18, 2013.

[10] Firefox, 2016, *Bug Types*, https://firefox-source-docs.mozilla.org/bug-mgmt/guides/bug-types.html, Accessed on October 25, 2021.

[11] X. Liu, Y. Si, D. Wang and Y. Liang, Heartbeat multi-classification algorithm based on hierarchical support vector machine, *ICIC Express Letters*, vol.11, no.7, pp.1221-1228, 2017.

[12] K. K. Chaturvedi and V. B. Singh, Determining bug severity using machine learning techniques, *2012 CSI 6th International Conference on Software Engineering (CONSEG)*, pp.1-6, 2012.
[13] R. Kumar and S. Singla, Multiclass software bug severity classification using Decision Tree, Naïve Bayes and Bagging, *Turkish Journal of Computer and Mathematics Education*, vol.12, no.2, pp.1859-1865, 2021.
[14] 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, pp.1-22, 2019.
[15] F. Meng, X. Wang, J. Wang and P. Wang, Automatic classification of bug reports based on multiple text information and reports' intention, *International Symposium on Theoretical Aspects of Software Engineering*, pp.131-147, 2022.
[16] P. Kaewnoo and T. Senivongse, Identification of software problem report types using multiclass classification, *Proc. of the 2019 3rd International Conference on Software and e-Business*, pp.104-109, 2019.
[17] T. Menzies and A. Marcus, Automated severity assessment of software defect reports, *2008 IEEE International Conference on Software Maintenance*, pp.346-355, 2008.
[18] J. Anvik, Automating bug report assignment, *Proc. of the 28th International Conference on Software Engineering*, pp.937-940, 2006.
[19] N. Pandey, A. Hudait, D. K. Sanyal and A. Sen, Automated classification of issue reports from a software issue tracker, *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, pp.423-430, 2017.
[20] B. Luaphol, B. Srikudkao, T. Kachai, N. Srikanjanapert and J. Polpinij, Feature comparison for automatic bug report classification, *International Conference on Computing and Information Technology*, pp.69-78, 2019.
[21] K. Chen, Z. Zhang, J. Long and H. Zhang, Turning from TF-IDF to TF-IGM for term weighting in text classification, *Expert Systems with Applications*, vol.66, no.30, pp.245-260, 2016.
[22] J. A. Widjaja and A. Wibowo, Sentiment analysis with slang dictionary in Indonesian social media using machine learning approach, *ICIC Express Letters*, vol.16, no.11, pp.1169-1177, 2022.
[23] W. P. Ramadhan, S. T. M. T. Astri Novianty and S. T. M. T. Casi Setianingsih, Sentiment analysis using multinomial logistic regression, *2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC)*, 2017.
[24] O. Chantamuang, J. Polpinij, V. Vorakitphan and B. Luaphol, Sentence-level sentiment analysis for student feedback relevant to teaching process assessment, *Multi-Disciplinary Trends in Artificial Intelligence*, pp.156-168, 2022.
[25] M. Z. Islam, J. Liu, J. Li, L. Liu and W. Kang, A semantics aware random forest for text classification, *Proc. of the 28th ACM International Conference on Information and Knowledge Management*, pp.1061-1070, 2019.
[26] U. Salamah and D. Ramayyanti, Supervised classification of indonesian text document using Extreme Gradient Boosting (XGBoost), *International Journal of Computer Technique*, vol.5, no.5, pp.79-84, 2018.
[27] P. L. Prasanna and D. R. Rao, Text classification using artificial neural networks, *International Journal of Engineering & Technology*, vol.7, no.1, pp.603-606, 2018.