Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

Automated Summarization of Bug Reports to Speedup Software Development/Maintenance Process by Using Natural Language Processing (NLP)

Syed Mohammed Furqan Ishaqui, Dr. Mohd.Abdul Bari, Dr..L.K Suresh Kumar
PG Scholar, Department of CSE, ISL Engineering College, Hyderabad, India
Professor, Department of CSE , ISL Engineering College , Hyderabad, India
Associate Professor & BOS; Department of Computer Science; UCE, Osmania University, Hyderabad, India

(Received: 04 August 2023 Revised: 12 September Accepted: 06 October)

KEYWORDS
Software Development, Software Maintenance, Automated Summarization

ABSTRACT:
Developers may benefit much from bug reports as they work on new features. However, it might be challenging to make use of these artifacts in the given time owing to the massive size of bug repositories. One strategy to aid developers is to give concise summaries of these reports, focusing on the most relevant information. After deciding if this report is what's needed, you may go into the specifics. With the development of text mining tools, several considerable methods have been developed to produce efficient summaries for bug reports. In this research, we present an extractive-based technique that makes use of language embedding to generate summaries of bug reports. In comparison to prior state-of-the-art methods, our rouge-1 and rouge-2 outcomes for bug report summarization are far better.

1. INTRODUCTION

Software quality assurance is significantly impacted by defect correction. It's the meat and potatoes of software engineering's post-release support phase. The software engineering business has been growing rapidly in recent years, leading to an increase in the size and complexity of software systems' architecture and code bases [1]. This pattern causes a great deal of errors to be made during the creation of software programs. Developers should review the bug report [2] to find out how to address these issues. The content of the bug report, which includes several tags like ID, Description, and Impact, outlines the system's flaws. In the past, managers used tags to categorize problem reports before assigning them to the most qualified engineers to resolve the issues. There are too many problem complaints, however, to verify each one individually. In addition, each reporter brings their own unique set of skills and expertise to the table, increasing the likelihood that the tags they assign in the Bug Tracking System report will be wrong [3]. When a bug report is incorrectly tagged, it may not be sent to the right people, which may make fixing the problem more challenging [4, 5]. Accurate and automatic categorization techniques for bug reports are needed in the software engineering industry to lessen this effect and hasten the pace at which defects are fixed. Many scientists in recent years have investigated the possibility of automatically categorizing bug reports. Others have used textmining techniques to categorize problem reports, such as Antoniol et al. [6]. It demonstrated the effectiveness and feasibility of automatically classifying reports into bug and other sorts using training models. To identify whether a new bug report is legitimate, Zhou et al. [7] suggested a hybrid approach that combines text mining and data mining approaches. This technique takes into account the report's structural data (such as severity and priority) by mining the textual description alone [5]. Lamkanfj et al. [8] used machine learning to categorize bug reports as critical or noncritical. To anticipate the severity of the bug report, Tian et al. [9] suggested an information

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

retrieval-based closest neighbor method. They zeroed in on figuring out which of the five possible report severity levels—Blocker, Critical, Major, Minor, and Trivial—would be encountered.

Furthermore, some academics worry about the accuracy of bug reports [10], as well as the skewed representation of certain groups in statistics [11, 12]. Reports are submitted by those who have good intentions. The purpose of the summary text content may be categorized into two types: explanation or recommendation, based on our examination of the summaries of a significant number of open source software bug reports. The Bug Tracking System, however, does not provide a "intent" label. When identifying reported bugs, many previous studies haven't taken the reporter's purpose into account, leading to subpar results. The strategy presented here takes into account the report's goals since they have an impact on how the report is categorized. In this context, "explanation" is a detailed description of the defect (such as a problem or its root cause) and "suggestion" means a proposed remedy for the deficiency. The following table provides instances of actual bug reports submitted by users in four distinct software environments.

As software systems get larger and more intricate, the emergence of Big Code has become an increasingly important trend in the industry [1]. Big Code is the term for the massive amount of software-related artifacts that may be found in places like bug databases, code snippet collections, and online source code repositories. It's a treasure trove of information and wisdom that other researchers may use to enhance the results of their own work. The mission of Big Code is to provide scalable and efficient methods to help software developers evaluate, comprehend, and forecast on enormous codebases. By centralizing so much information in one place, Big Code might potentially significantly advance AI research and development. Advanced programming languages, potent machine learning methods like large language models (LLMs), and NLP approaches based on the software naturalness hypothesis are all used in the creation of statistical programming systems [2, 3]. This theory proposes that, just how NLP treats human natural languages, a wide variety of programming languages may be understood and handled by a computer.

Table 1: Comparison of surveys on language models in software naturalness[1].

Title	Year	Focus Area
A Survey of Machine Learning for Big Code and Naturalness	2019	Big Code and Naturalness
Software Vulnerability Detection Using Deep Neural Networks: A Survey	2020	Security
A Survey on Machine Learning Techniques for Source Code Analysis	2021	Code Analysis
Deep Security Analysis of Program Code: A Systematic Literature Review	2022	Security
A Survey on Pretrained Language Models for Neural Code Intelligence	2022	Code Summarization and Generation, and Translation
Deep Learning Meets Software Engineering: A Survey on Pre-trained Models of Source Code	2022	Software Engineering

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

Title	Year	Focus Area
Software as Storytelling: A Systematic Literature Review	2023	Storytelling
Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing	2023	Prompt-based Learning

This analysis, however, is narrowed down to software-based naturalness in its language models. In Table 1, we present a comprehensive comparison of previous assessments that have covered similar ground.

2. RELATED WORK

Software engineers are aided in their quest to repair bugs when reports are categorized. Manual categorization has become tedious and time-consuming due to the ever-increasing volume of problem reports. Automatic bug report categorization is something that has been studied for quite some time [18]. Some previous studies will be reviewed here.

In 1992, IBM's Chillarege et al. [19] presented Orthogonal Defect Classification (ODC) as the first approach to classifying bugs. There are 13 distinct types (such as functions, interfaces, documents, etc.) included in this technique that bridges the gap between qualitative and quantitative approaches. Using vector space technology to extract characteristics and then training Decision Trees (DT), Naive Bayes (NB), and Logistic Regression (LR) classifiers to determine whether or not a report is a problem, Antoniol et al. [6] suggested an automated classification technique for bug reports in 2008. The results provide a categorization accuracy of 77% to 82% across the Mozila, Eclipse, and JBoss projects. To determine whether bugs are real, Pingclasai et al. [20] suggested a categorization scheme in 2013. Accuracy for HTTP-Client, Jackrabbit, and Lucene respectively ranged from 66% to 76%, 65% to 77%, and 71% to 82% when they used the topic model of Latent Dirichlet Allocation (LDA) in conjunction with NB and Linear Logistic Regression (LLR) classifiers. Similarly, kukkar et al. [13] used a hybrid approach that incorporates TM, NLP, and ML technologies to determine if the report is a problem or not in 2019. On five distinct data sets (Mozilla, Eclipse, JBoss, Firefox, OpenFOAM), they evaluated the efficacy of Term Frequency- Inverse Document Frequency (TF-IDF), feature selection, and K-NN classifiers. Experiments reveal that the K-NN classifier's performance varies among datasets, with an F-measure of between 78% and 96%. Scientists also categorize the severity of reported bugs. In 2008, Menzies et al. [21] introduced an innovative automated approach dubbed SERVERS. This technique utilizes TF-IDF, InfoGain, and Rule Learning to categorize the severity of reported bugs into five levels, from highest to lowest. From a total of 14 characteristics included in the bug report for serious and non-serious categorization, only 5 were considered legitimate by Sari et al. [22] in 2011. They are "component," "qa_contact," "summary," and "cc_list," respectively. When used together, they can improve the SVM model's accuracy to 99.83%. Improved REP (i.e. REP theme) and K-NN method were used by Zhang et al. [23] to find comparable bug reports from the past, extract characteristics to forecast problem severity, and categorize reported issues as Blocker, Trivial, Critical, Minor, or Major. The results demonstrate that their suggested approach may successfully enhance the precision with which the severity of bug reports can be predicted. In 2019, Kukkar et al. [24] suggested a Deep Learning-based categorization approach for bug reports since they felt that existing Machine Learning classifiers were unable to capture certain potentially crucial information. To address the challenge of predicting the relative severity of many bug reports, the model employs a Convolutional Neural Network (CNN), a Random Forest, and a Boosting algorithm. The average accuracy across their five open source projects is 96.34 percent, thus their efforts have paid off.

Researchers have suggested a wide variety of categorization schemes, not only based on bugs or severity. In 2017, Du et al. [25] created an automated

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

categorization system based on word2vec that separated bug reports into Bug/Non-Bug, BOH/MAN, ARB/NAM, and NAM/ARB categories according to four granularities. Tan et al. [26] from 2014 thought that software systems are intimately involved in semantic, security, and concurrency issues. On the basis of these hypotheses, they analyzed the frequency with which each category occurred in popular open source projects like Apache, Mozilla, and Linux and used machine learning to automatically categorize bug reports into the three categories listed above. Approximately 70% is the mean F-measure. Catolino et al. [27] recently defined a new bug report classification pattern for 2019 that includes 9 defect types (Configuration problem, Network problem, Database related, GUI related, Performance problem, Permission/deprecation problem, Security problem, Program anomaly problem, Test code related). Catolino et al.'s approach of categorizing bug reports is more explicit and thorough than that of Tan et al. [26]. The automated model they developed also had better results in terms of F-Measure (64%) and AUC-ROC (74%).

From this and other similar studies, it is clear that many academics have made significant progress toward an automatically accurate categorization of bug reports. In this paper, we build on previous studies to automatically categorize bug reports while also taking into account the reporter's purpose. Boosting this variable, we think, will lead to better categorization results.

3. METHODOLOGY

In this subsection, we describe the proposed bug report categorization scheme in detail. Figure 1 depicts this structure. Bug reports are gathered from the public repository, carefully annotated, and then pre-processed. Then, we extract features using the BERT and TF-IDF techniques. In addition, the frequency feature is normalized and combined with the text feature. The characteristics are then sent into five different classifiers (K-NN, NB, LR, SVM, and RF). We conclude by distinguishing between bug reports and non-bug reports.

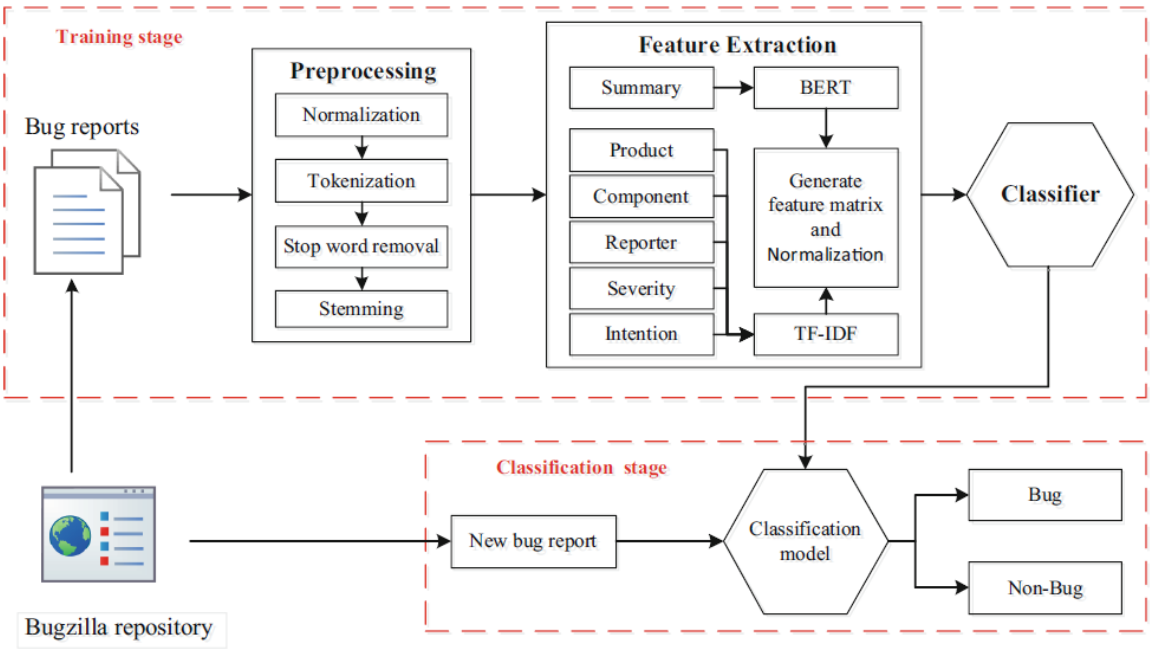


Figure 1: Framework

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

Language Models on Software Naturalness

Some of the most effective language models based on transformers are discussed here. Figure 2 shows how LLMs have changed over time starting in 2018.

Figure 2: Language Models

Table 3: Summary of language models using transformers [1]

Model	Type	AI-Assisted Programming Tasks
Encoder-only	Understanding	Code Summarization, Code Translation
Decoder-only	Generation	Code Generation, Code Completion
Encoder–decoder	Generation and understanding	Code Generation, Code Refinement, Defect Detection, Clone Detection

Sequence-to-sequence models, also known as encoder-decoder models , use both halves of the transformer design . At each level, the attention layers in the encoder have access to all words in the input phrase, whereas the attention layers in the decoder have access only to the words preceding a specific word in the input. Code generation, code refinement, defect detection, and clone detection are all examples of AI-assisted programming tasks that benefit from sequence-to-sequence models like BART , T5 (Text-to-Text Transfer Transformer) , and TreeGen.

Encoder-only models, also known as autoencoders, rely only on an encoder network to encode data. They are widely used in unsupervised learning applications, especially those involving dimensionality reduction and anomaly detection in natural language processing. In the past, code embedding methods like Neural Network Language Model , Code2Vec , ELMo , TextRank , and GGNN may be used to derive the representation from the input data. The BERT and RoBERTa are used for understanding tasks in AI-assisted programming to learn usable representations of data in an unsupervised way; these representations may then be utilized as features in downstream tasks like code translation and code summarization.

Natural language processing tasks including GPT-2 , GPT-3 , GPT-J , Reformer , and GPT-Neo employ decoder-only models, also known as autoregressive models, to predict the next token output given all prior tokens. Only a decoder network, which predicts the next token based on the distribution of previous tokens, is used to produce any text at all. However, jobs that need a more nuanced understanding of the input-output sequence connection may not fare as well with these models, despite their simplicity and efficiency. Despite this, they have shown outstanding performance in a number of benchmarks and continue to see widespread usage in a variety of natural language processing applications for AI-assisted programming, such as code generation and code completion.

Measurement of Language Models with Entropy

By performing a maximum-likelihood estimation (MLE) of the parameter of a properly selected parametric distribution given a corpus C of programs CS, an estimated language model is produced, known as

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

a pre-trained language model . In Section 2.2, we go out the steps involved. Given the above context, the programming language defines the tokenization of the code to estimate the probability distribution of code tokens. This data is then put to use in software engineering choices and forecasts. The models are educated to estimate the likelihood of subsequent words in a sequence given the words that came before them . N-gram models, which have been utilized extensively for estimating the probability distribution of words or characters in a text sequence , are often employed in the construction of the language model. This was the accepted strategy until the advent of RNN-based distributed word vectors and linguistic representations . N-gram models can predict the probability of a token following another token given a system s with a series of tokens W1,W2,...Wn. By multiplying a string of conditional probabilities, the model may arrive at an estimate of the likelihood of s:p(s)=p(W1)p(W2|a1)p(W3|W1W2)…p(Wn|W1…Wn−1).

Word or character co-occurrence patterns in a text may be captured by using an N-gram model. A mathematical representation of an N-gram model is a collection of N-grams, where each N-gram is a tuple of n elements and their respective probabilities. The MLE can estimate the likelihood of an N-gram given its frequency of occurrence in a specific training corpus. Similarly, this presumes a Markov property, whereby the occurrences of tokens are affected by just a small prefix length of n. For instance, in a model with three grams (n=3):

$$p(Wi|W1\ldots Wi-1)\cong p(Wi|Wi-2Wi-1). \quad (3)$$

The probability of a word Wi given its preceding word Wi−1 can be estimated:

$$p(Wi|Wi-1)=count(Wi-1,Wi)/count(Wi-1), \quad (4)$$

where count(Wi1,Wi) is the total number of occurrences of the 3-gram (Wi1,Wi) in the training corpus, and count(Wi1) is the total number of occurrences of the word Wi1. Recent advances in natural language processing may be directly attributed to the effectiveness of these models. The effectiveness of the method is determined by the accuracy with which the language model represents the target data's patterns and structures. Many studies have been conducted to enhance the quality of language models for various tasks by expanding training methods, enlarging training corpora, and refining assessment measures.

4. EXPERIMENTS

This study uses a split of 8:2 across the training and test sets to isolate features from the report summary, other fields (product, component, reporter, severity), and intent. We superimpose and merge these three characteristics in succession, and then feed them into five different machine learning classifiers (K-NN, NB, SVM, LR, RF) to see which one works best with the suggested approach.

These scientific inquiries were answered by the experiments:

Does the automated categorization of bug reports increase in accuracy if the purpose of the report is also included? Question 2: How well does our strategy work with five distinct classifiers?

4.1 Dataset

For this research, we gathered a total of 2,230 bug reports from Bugzilla, with contributions coming from Apache [14], Eclipse [15], Gentoo [16], and Mozilla [17]. We only choose reports with a "FIXED" resolution or a "RESOLVED" status. And then get the product, component, reporter, severity, and summary labels out of them. We used this information to manually categorize the purpose, source, and nature of these reports. information statistics are shown in Table 3.

Table 3. Type statistics of our dataset

Ecosystem	Total	Bug	Non-Bug
Apache	446	296	150
Eclipse	658	419	239
Gentoo	511	294	217
Mozilla	615	425	190

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

4.3 Results

Does Including the Reason for the Report Help Automatically Classify Bug Reports?

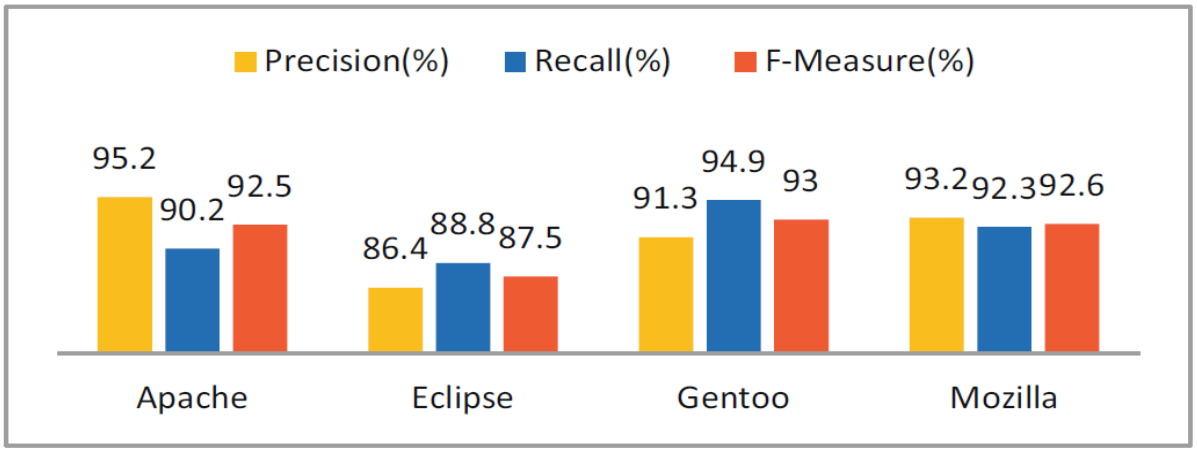
Table 4 displays the average accuracy of the ten-fold cross-validation, which we employ to train the classifier using three different kinds of features that are fused and overlaid successively. Summary text (represented by Text) reflects the written content of the bug report, whereas word frequency (represented by Freq) indicates the content of the other fields (product, component, reporter, severity). The table's numbers are expressed as percentages. Our proposed approach combines text frequency analysis with intent.

Table 4. Average accuracy of all datasets[1]

Ecosystem	Features	Classifier				
		K-NN	NB	LR	SVM	RF
Apache	Text	60.5	65.5	65.6	66.4	63.0
	Text+Freq	70.6	80.0	70.9	70.9	85.7
	Text+Freq+Intention	**90.4**	**89.2**	**90.8**	**91.0**	**91.7**
Eclipse	Text	61.5	63.7	65.0	64.6	61.0
	Text+Freq	66.4	66.1	65.2	64.4	73.1
	Text+Freq+Intention	**83.9**	**84.0**	**84.8**	**84.8**	**84.8**
Gentoo	Text	67.7	61.8	57.3	62.8	67.3
	Text+Freq	83.2	73.6	71.2	72.8	87.3
	Text+Freq+Intention	**91.8**	**85.1**	**86.1**	**87.7**	**94.5**
Mozilla	Text	65.2	66.8	65.0	69.4	67.5
	Text+Freq	75.3	70.4	72.0	72.3	78.2
	Text+Freq+Intention	**89.9**	**87.5**	**87.8**	**88.0**	**87.8**

How Effective Is Our Suggested Approach Across Five Distinct Classifiers?

Using the five classifiers shown in Figs. 3, 4, 5, 6, and 7, we evaluate the efficacy of our proposed technique, which integrates text, frequency, and intention variables (Text+Freq+Intention). Data origin is shown along the x-axis, and the mean of 10 independent validations is shown along the y-axis.

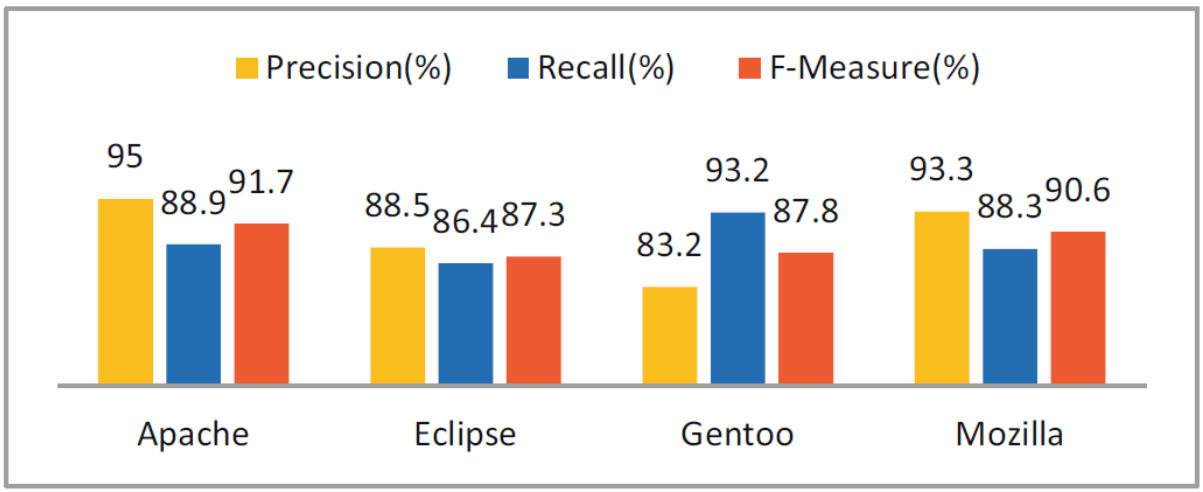


K-NN classifier performance

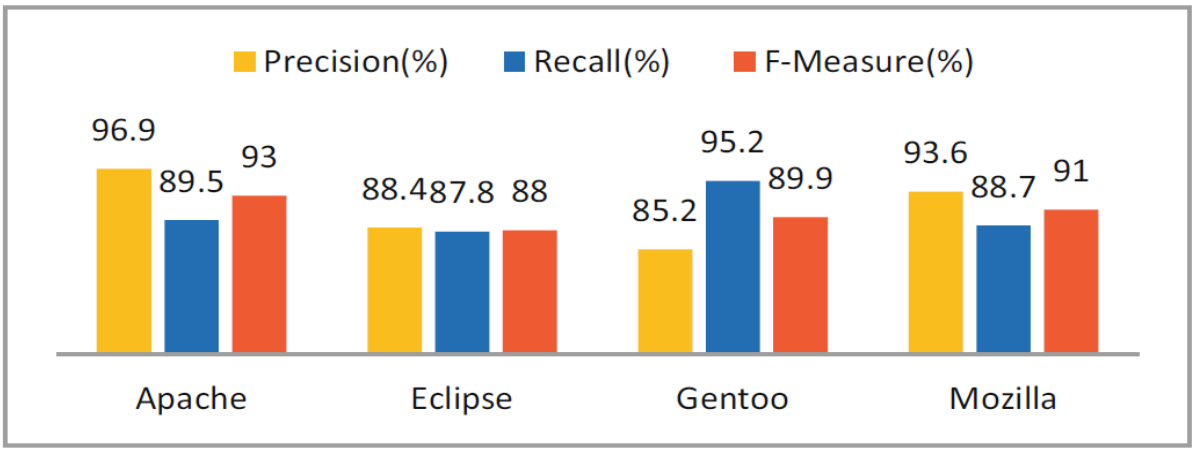
Journal of Chemical Health Risks

www.jchr.org

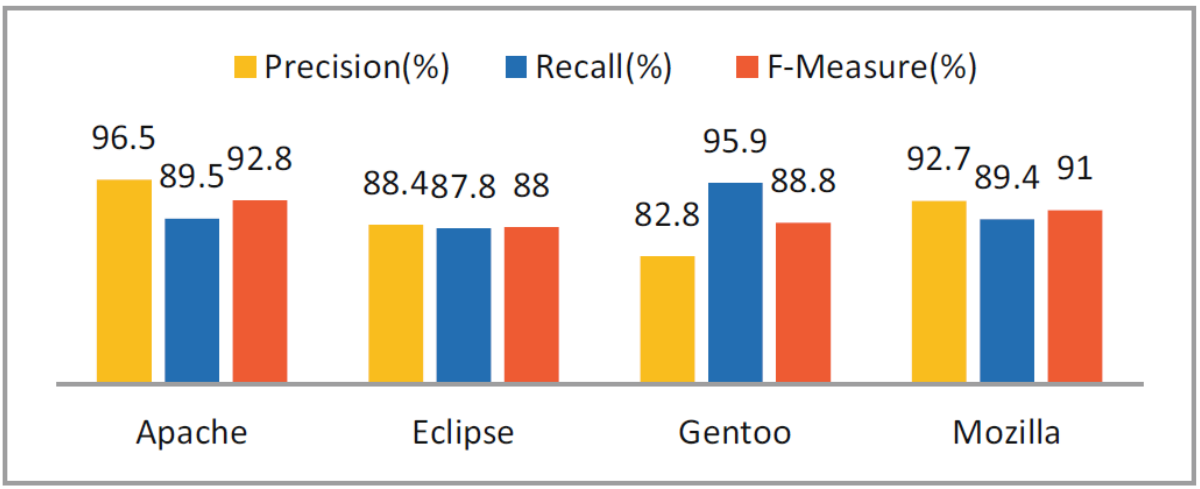
JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727



NB classifier performance



SVM classifier performance

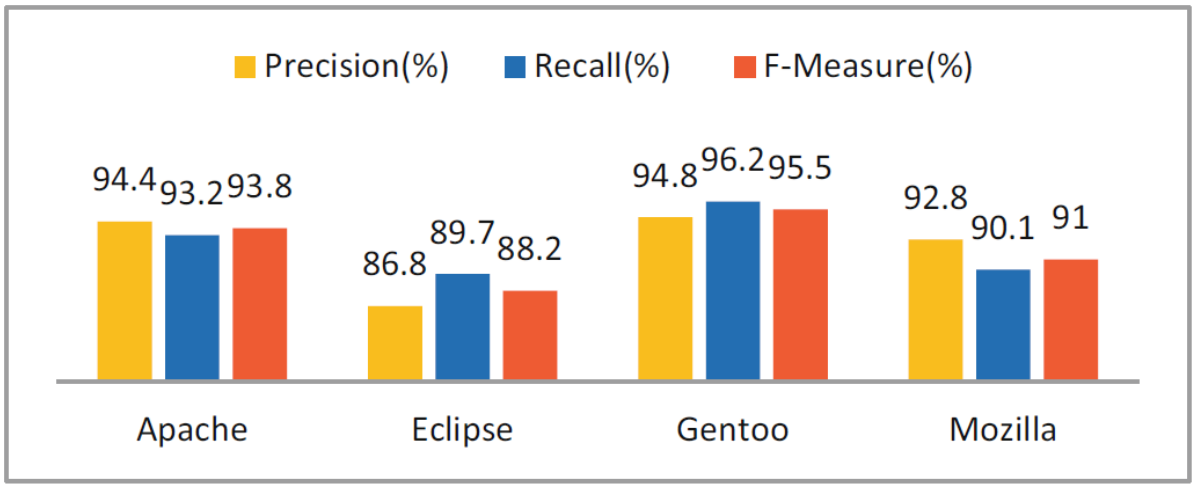


LR classifier performance

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727



RF classifier performance

5. DISCUSSIONS

5.1 Experiment Analysis

We test our strategy on the Apache, Eclipse, Gentoo, and Mozilla datasets, combining the suggested methodology with five different machine learning classifiers. Table 4 summarizes the typical Accuracy achieved by various data sets while using various classifiers. The highest percentages among these collections are as follows: 91.7% for the Apache data set; 84.8% for the Eclipse data set; 94.5% for the Gentoo collection; and 89.9% for the Mozilla collection. Table 4 shows that our suggested addition of the intention element of the report greatly increased the accuracy of the data sets of the four ecosystems on the five classifiers, compared to only examining the text field of the report. We analysed the distribution of intention characteristics and their connection with labels (i.e., bug or non-bug) in the experimental dataset to better understand how adding the binary feature of reporting intention might boost classification performance.

6. CONCLUSIONS AND FUTURE WORK

In this research, we present a novel automated categorization strategy for bug reports, with the goal of better understanding the report's motivation from its textual content. Our method integrates tools from the fields of Text Mining, NLP, and ML. To begin building the data set necessary for the study, we gathered 2,230 reports from the bug repository across the four ecosystems (Apache, Eclipse, Gentoo, Mozilla). Then, we supplement the report's purpose characteristics with the text features extracted from the summary field and the word frequency features of the other fields. Then, we feed this merged set of features through one of five classifiers (K-NN, NB, SVM, LF, RF). The last step is to distinguish between legitimate bugs and false positives while reviewing reported issues. The findings demonstrate that our suggested enhancements to the report's intention features may greatly boost the efficiency of bug report classification compared to merely extracting text information features for classification. In the future, we want to test the suggested method on other open-source projects and integrate Deep Learning tools to enhance the efficiency with which bug reports may be automatically classified.

REFERENCES

[1] Fanqi Meng, Xuesong Wang(B), Jingdong Wang(B), and Peifang Wang, Automatic Classification of Bug Reports Based on Multiple Text Information and Reports' Intention, © Springer Nature Switzerland AG 2022 Y. Aït-Ameur and F. Crˇaciun (Eds.): TASE 2022, LNCS 13299, pp. 131–147, 2022.

[2] M. Irtaza Nawaz Tarar; Faizan Ahmed; Wasi Haider Butt, 2020 15th International

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

Conference on Computer Science & Education (ICCSE), 18-22 August 2020, IEEE.

[3] Meng, F., Cheng, W., Wang, J.: Semi-supervised software defect prediction model based on tri-training. KSII Trans. Internet Inf. Syst. **15**(11), 4028–4042 (2021)

[4] Guo, S., Chen, R., Li, H.: Using knowledge transfer and rough set to predict the severity of android test reports via text mining. Symmetry **9**(8), 144–161 (2017)

[5] Yang, G., Min, K., Lee, J.W.: Applying topic modeling and similarity for predicting bug severity in cross projects. KSII Trans. Internet Inf. Syst. **13**(3), 1583–1589 (2019)

[6] Kim, S., Zhang, H.,Wu, R., Gong, L.: Dealing with noise in defect prediction. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 481–490. ACM (2011)

[7] Kochhar, P.S., Le, T.D.B., Lo, D.: Dealing with noise in defect prediction. In: 2014 11th Working Conference on Mining Software Repositories (MSR), pp. 296–299. IEEE (2014)

[8] Antoniol, G., Ayari, K., Di, P.M., Khomh, F., Guéhéneuc, Y.G.: Is it a bug or an enhancement? A text-based approach to classify change requests. In: 2008 Conference of the Centre for Advanced Studies on Collaborative Research: Meeting of Minds, pp. 304–318 (2008)

[9] Zhou, Y., Tong, Y., Gu, R., Gall, H.: Combining text mining and data mining for bug report classification. J. Softw.: Evol. Process **28**(3), 150–176 (2016)

[10] Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: 2010 7th IEEE/ACMWorking Conference on Mining Software Repositories (MSR), pp. 1–10. IEEE (2010)

[11] Tian, Y., Lo, D., Sun, C.: Information retrieval based nearest neighbor classification for finegrained bug severity prediction. In: 2012 19thWorking Conference on Reverse Engineering, pp. 215–224 (2012)

[12] Mrs. Manga Geethanjali, SK Kusheeda Bee, Syed Abdul Muqhit, CS MD Azeemuddin**,** Traffic Priority For Ambulance, International Journal of Multidisciplinary Engineering in Current Research - IJMEC Volume 8, Issue 2, February-2023, http://ijmec.com/, ISSN: 2456-4265.

[13] Feng, Y., Chen, Z., Jones, J., Fang, C., Xu, B.: Test report prioritization to assist crowdsourced testing. In: 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 225–236 (2015)

[14] Zhang, T., Chen, Y., Yang, X., Zhu, H.: Approach of bug reports classification based on cost extreme learning machine. J. Softw. **30**(5), 1386–1406 (2019)

[15] Syed Shehriyar Ali, Mohammed Sarfaraz Shaikh, Syed Safi Uddin, Dr. Mohammed Abdul Bari, "Saas Product Comparison and Reviews Using Nlp", Journal of Engineering Science (JES), ISSN NO:0377-9254, Vol 13, Issue 05, MAY/2022

[16] Ms. Vrushali Pawar, Mr. Syed Zaker Hussain, Dr.Tushar Rathod**,** Wearable Biosensors For Healthcare Monitoring, International Journal of Multidisciplinary Engineering in Current Research - IJMEC Volume 8, Issue 2, February-2023, http://ijmec.com/, ISSN: 2456-4265.

[17] Hafsa Fatima, Shayesta Nazneen, Maryam Banu, Dr. Mohammed Abdul Bar," Tensorflow-Based Automatic Personality Recognition Used in Asynchronous Video Interviews", Journal of Engineering Science (JES), ISSN NO:0377-9254, Vol 13, Issue 05, MAY/2022

[18] Mohammed Shoeb, Mohammed Akram Ali, Mohammed Shadeel, Dr. Mohammed Abdul Bari, "Self-Driving Car: Using Opencv2 and Machine Learning", The International journal of analytical and experimental modal analysis (IJAEMA), ISSN NO: 0886-9367, Volume XIV, Issue V, May/2022

[19] Mr. Pathan Ahmed Khan, Dr. M.A Bari,: Impact Of Emergence With Robotics At Educational Institution And Emerging Challenges", International Journal of Multidisciplinary Engineering in Current

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

Research(IJMEC), ISSN: 2456-4265, Volume 6, Issue 12, December 2021,Page 43-46

[20] Dr Altaf C, Syed Muzammil Ahmed,Mir Mohammed Asad Ali , Rayan Razvi, Automatic Railway Gate Controlling Using IR Sensors And Microcontroller**,** International Journal of Multidisciplinary Engineering in Current Research - IJMEC Volume 8, Issue 3, March-2023, http://ijmec.com/, ISSN: 2456-4265.

[21] Yang, X.L., Lo, D., Xia, X., Huang, Q., Sun, J.L.: High-impact bug report identification with imbalanced learning strategies. J. Comput. Sci. Technol. **32**(1), 181–198 (2017)

[22] Kukkar, A., Mohana, R.: A supervised bug report classification with incorporate and textual field knowledge. Proc. Comput. Sci. **132**, 352–361 (2018)

[23] Zhang, T., Jiang, H., Luo, X., Chen, A.T.: A literature review of research in bug resolution: tasks, challenges and future directions. Comput. J. **59**(5), 741–773 (2016)

[24] Chillarege, R., et al.: Orthogonal defect classification-a concept for in-process measurements IEEE Trans. Softw. Eng. **18**(11), 943–956 (1992)

[25] Pingclasai, N., Hata, H., Matsumoto, K.I.: Classifying bug reports to bugs and other requests using topic modelling. In: 2013 20thAsia-Pacific Software EngineeringConference (APSEC), vol. 2, pp. 13–18 (2011)

[26] Raj Kumar D bhure, K. Saisrikar, Ch. Devayani, D. Shivareddy, Energy Efficiency Routing For Manet Using Residual Energy, International Journal of Multidisciplinary Engineering in Current Research - IJMEC Volume 8, Issue 4, April-2023, http://ijmec.com/, ISSN: 2456-4265.

[27] Mr. Touseef Sumeer, Shahzada Salim, Md Abdul Jabbar, Shaik Rehmath, Home Appliances Control Using Remote Control System**,** International Journal of Multidisciplinary Engineering in Current Research - IJMEC Volume 8, Issue 4, April-2023, http://ijmec.com/, ISSN: 2456-4265.

[28] Menzies, T., Marcus, A.: Automated severity assessment of software defect reports. In: 2008 IEEE International Conference on Software Maintenance (ICSM), pp. 346–355. IEEE (2008)

[29] Sari, G.I.P., Siahaan, D.O.: An attribute selection for severity level determination according to the support vector machine classification result. In: 1st International Conference on Information Systems for Business Competitiveness (ICISBC) (2012)

[30] Zhang, T., Chen, J., Yang, G., Lee, B., Luo, X.: Towards more accurate severity prediction and fixer recommendation of software bugs. J. Syst. Softw. **177**(10), 166–184 (2016)

[31] Kukkar, A., Mohana, R., Nayyar, A., Kim, J., Kang, B.G., Chilamkurti, N.: A novel deeplearning- based bug severity classification technique using convolutional neural networks and random forest with boosting. Sensors **19**(13), 2943–2964 (2019)

[32] Du, X., Zheng, Z., Xiao, G., Yin, B.: The automatic classification of fault trigger based bug report. In: 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 259–265. IEEE (2017)

[33] Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., Zhai, C.: Bug characteristics in open source software. Empir. Softw. Eng. **19**(6), 1665–1705 (2013). https://doi.org/10.1007/s10664-013- 9258-8

[34] Catolino, G., Palomba, F., Zaidman, A., Ferrucci, F.:Not all bugs are the same: understanding, characterizing, and classifying bug types. J. Syst. Softw. **152**(10), 165–181 (2019).

[35] D. Cubrani c and G.C. Murphy, "Hipikat: Recommending Pertinent Software Development Artifacts", *Proc. 25th Int'l Conf. Software Eng. (ICSE '03)*, pp. 408-418, 2003.

[36] C. Sun, D. Lo, S.-C. Khoo and J. Jiang, "Towards More Accurate Retrieval of Duplicate Bug Reports", *Proc. 26th Int'l Conf. Automated Software Eng. (ASE '11)*, pp. 253-262, 2011.

Journal of Chemical Health Risks

www.jchr.org

JCHR (2023) 13(3), 1226-1237 | ISSN:2251-6727

[37] A. Nenkova and K. McKeown, "Automatic Summarization", *Foundations and Trends in Information Retrieval*, vol. 5, no. 2/3, pp. 103-233, 2011.

[38] K. Zechner, "Automatic Summarization of Open-Domain Multiparty Dialogues in Diverse Genres", *Computational Linguistics*, vol. 28, no. 4, pp. 447-485, 2002.

[39] X. Zhu and G. Penn, "Summarization of Spontaneous Conversations", *Proc. Ninth Int'l Conf. Spoken Language Processing (Interspeech '06- ICSLP)*, pp. 1531-1534, 2006.

[40] O. Rambow, L. Shrestha, J. Chen and C. Lauridsen, "Summarizing Email Threads", *Proc. Human Language Technology Conf. North Am. Chapter of the Assoc. for Computational Linguistics (HLT-NAACL '04)*, 2004.

[41] R.J. Sandusky and L. Gasser, "Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management", *Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work (GROUP '05)*, pp. 187-196, 2005.

[42] R. Lotufo, Z. Malik and K. Czarnecki, "Modelling the 'Hurried' Bug Report Reading Process to Summarize Bug Reports", Proc. IEEE 28th Int'l Conf. SoftwareMaintenance (ICSM'12).

[43] S. Mani, R. Catherine, V.S. Sinha and A. Dubey, "AUSUM: Approach for Unsupervised Bug Report Summarization", *Proc. ACM SIGSOFT 20th Int'l Symp. The Foundations of Software Eng. (FSE '12)*, 2012.

[44] S. Rastkar, G.C. Murphy and G. Murray, "Summarizing Software Artifacts: A Case Study of Bug Reports", *Proc. 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10)*, pp. 505-514, 2010.

[45] R. Lotufo, Z. Malik and K. Czarnecki, "Modelling the 'Hurried' Bug Report Reading Process to Summarize Bug Reports", *Empir. Softw. Eng.*, vol. 20, no. 2, pp. 516-548, April 2015.