IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 1 March 2024, accepted 28 April 2024, date of publication 8 May 2024, date of current version 15 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3398732

RESEARCH ARTICLE

Accurate Information Type Classification for Software Issue Discussions With Random Oversampling

BOBURMIRZO MUHIBULLAEV AND JINDAE KIM, (Member, IEEE)
Department of Computer Science and Engineering, Seoul National University of Science and Technology, Nowon-gu, Seoul 01811, Republic of Korea
Corresponding author: Jindae Kim (jindae.kim@seoultech.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by Korean Government [Ministry of Science and ICT (MSIT)] under Grant 2022R1F1A1070464.

ABSTRACT An Issue Tracking System (ITS) plays a crucial role in software development and provides valuable information for understanding issue management. In an ITS, software developers often discuss issues that are reported during software development. Recent studies analyzed such issue discussions and identified information types of issue comments that appeared in the discussions. Automatic classification of the information types can help developers understand and locate required information more easily, but existing techniques cannot provide accurate classification. In this study, we propose a more accurate technique to classify information types of issue comments. The key to increasing classification performance is employing random oversampling to deal with imbalances among training instances of different information types. With random oversampling, we trained a classifier using logistic regression with hyperparameter tuning and achieved an average 0.95 F1-score, which was much higher than 0.53 of the compared existing technique. We also considered two other key aspects of the technique to fully investigate the potential performance improvement. We expanded an existing issue comment dataset by adding 4,098 more instances, almost double the size of the dataset. We analyzed the influence of hyperparameters on classification performance and found that using values within an appropriate range is important to achieve high performance.

INDEX TERMS Issue discussion analysis, issue management, issue tracking systems, open source software, random oversampling.

I. INTRODUCTION

In software development, an Issue Tracking System (ITS) plays an important role in issue management, and any issues during the development can be reported, tracked, and discussed in the system. Because of its crucial roles, many software engineering studies and techniques have targeted ITSs. Some of the studies investigated the ITSs and tried to provide a better understanding of issues and activities that occurred during the software life cycle [1], [2], [3], [4], [5], [6]. Another line of work tries to help software developers by leveraging the information obtained from ITSs, such as localizing faults causing reported bugs [5], [7], [8], [9], [10], [11], [12], or recommend developers suitable for handling reported issues [13], [14], [15], [16].

One issue with using ITSs is that it is difficult to find useful information if an issue has a very long issue discussion. FIGURE 1 shows an example of such issue discussions that we collected from the issue #13353 of the Keras software project.[1] The first box is the actual issue report, and the other three boxes show a part of the issue discussion extracted from the issue. There are 101 comments for this issue, and each comment may consist of several sentences. In FIGURE 1, red lines represent solution discussion, and blue

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

[1] https://github.com/keras-team/keras/issues/13353

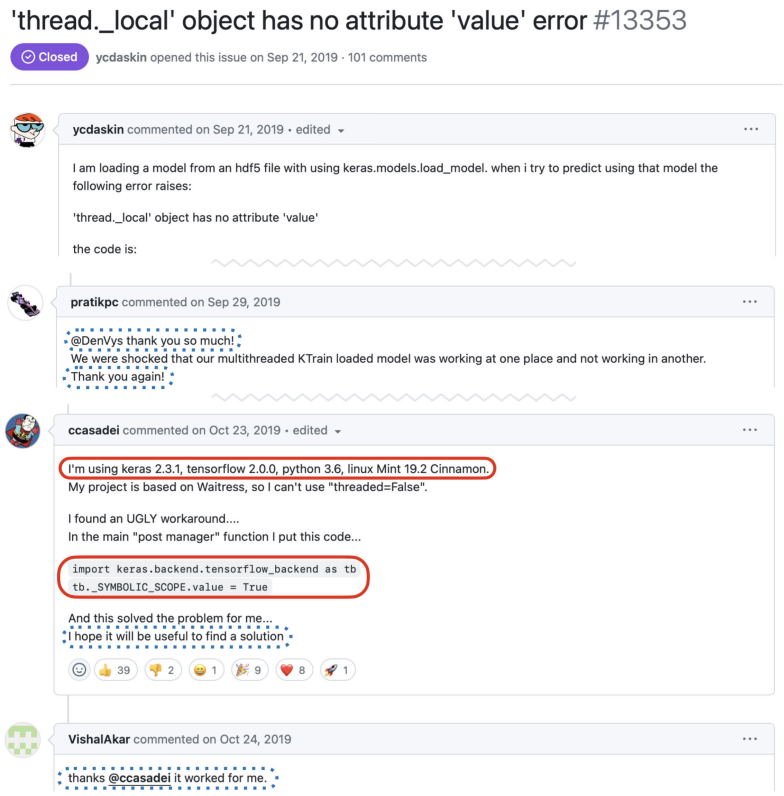

FIGURE 1. An Example of issue discussion from keras issue #13353.

dotted lines indicate social conversions. To solve the problem, a developer should read all the sentences and identify the sentences that are closely relevant to solutions. This is not an easy task, especially since many sentences exist in the issue discussion. For instance, the red-lined sentences in FIGURE 1 appear only after scrolling down one-third of the long web page. In such situations, identifying information types of an individual comment - i.e., each sentence - can be helpful to find necessary information. In this paper, we will use "issue comment" and "sentence" interchangeably to indicate an individual sentence that belongs to a specific information type.

To solve this problem, Arya et al. analyzed and identified 16 information types in an issue discussion dataset consisting of more than 4K issue comments collected from GitHub [17]. Then, they trained classifiers using logistic regression and random forest techniques to identify information types of issue comments automatically. Using the techniques, we can focus on the required information directly while filtering out other unnecessary types of information. For example, a developer assigned to solve an issue may want to see comments related to bug reproduction only, and the technique can be used to list up bug reproduction-type comments.

However, the previous classifiers were still insufficient to solve the problem since their classification performance was not sufficiently high. The classification performance of the existing techniques is measured by the weighted average F1-score using the dataset consisting of the collected issue comments. Although the values varied for different configurations, the overall F1-score ranged from 0.42 to 0.61. Mehder and Aydemir employed deep learning models and tried to improve the classification performance [18], but in their experiments with the same dataset, Arya et al.'s logistic regression model obtained 0.51 F1-score, and the proposed models using BERT [19] only achieved 0.54 F1-score. These results indicate that the existing techniques do not reliably produce classification outcomes. We need to improve their performance to truly resolve the problem caused by long issue discussions.

In this study, we propose a new solution to significantly increase the classification performance for information types of issue comments. The core idea of the proposed technique is employing *Random Oversampling* [20] to mitigate the class imbalance problem. During logistic regression classifier training, we identified a significant class imbalance issue. The two largest classes (information types) comprised over half of the training instances, potentially introducing biases to the classifier. To address this problem, we applied random oversampling, which copies the instances from minority classes to balance the number of instances among the classes. In the previous study [17], this problem was handled by the Synthetic Minority Oversampling Technique (SMOTE) [21]. While random oversampling simply copies existing instances, SMOTE synthesizes similar instances

from numeric vectors representing the sentences. Since the instances were short sentences, we thought that synthesized vectors might not represent other sentences belonging to the same class, which eventually decreased the performance.

Surprisingly, the simple change significantly increased the classification performance. In our experiments, we compared the classification performance of the two logistic regression classifiers with different oversampling methods: random oversampling and SMOTE. With random oversampling, the average weighted values of precision, recall, and F1-score were 0.95, 0.95, and 0.95, respectively. On the other hand, using SMOTE yielded only 0.54, 0.54, and 0.53 for these same metrics. By simply changing the oversampling method, we achieved significantly higher classification performance, demonstrating that improvements do not always require the adoption of new, complex, and costly training methods.

Although employing random oversampling significantly improved the performance, we also considered two other key aspects: dataset expansion and hyperparameter tuning. Increasing the number of training instances can enhance classification performance. Hence, we collected and labeled 4,098 issue comments and combined them with the existing dataset, nearly doubling its size. With the combined dataset, the average weighted F1-score was improved to 0.56 for SMOTE but was not changed for random oversampling. While more instances improved the performance slightly, we observed that the class imbalance persisted in the combined dataset, and the performance improvement was limited without proper handling of the imbalance. We also analyzed the influence of hyperparameters - `LogReg C` (regularization strength), `max_df` (maximum document frequency), and `ngram_range`- on classification performance by experimenting with various values. Our findings indicate that `LogReg C` influences classification performance most, while the impact of `max_df` and `ngram_range` is more limited. However, selecting values within an appropriate range for these hyperparameters remains important to achieve high performance.

Here are the key contributions of this study.

- We propose an accurate and reliable technique to classify information types of issue comments from issue discussions.
- We expand the issue comment dataset and double its size. The dataset is publicly available.
- We present an analysis of the influence of hyperparameters on the classification performance.

The rest of the paper is organized as follows. First, we explain the details of our approach in Section II-A. Then, we provide our empirical evaluation design of the proposed technique's performance in Section III. Next, we present the evaluation results and discuss the influence of oversampling methods, the dataset, and hyperparameters on classification performance (Section IV). After that, we discuss studies related to this paper (Section V) as well as threats to the validity of the study (Section VI), and finally we conclude with future work (Section VII).

II. INFORMATION TYPE CLASSIFICATION OF ISSUE COMMENTS

This section will explain our approach for classifying the information types of issue comments.

A. OVERALL PROCESS

To improve the classification performance of the existing approach [17], we considered three aspects of the technique: oversampling method, dataset, and hyperparameter tuning. Among the three aspects, the oversampling method is the most important part of the proposed technique. By using a different oversampling method, we can expect to mitigate the imbalance of training instances from different information types. In addition, we expanded the dataset with more collected and labeled issue comments to provide more training instances for the classification. Finally, we analyzed the influence of hyperparameters on the classification performance.

FIGURE 2 shows the overall process of the proposed technique. The process begins with a dataset containing collected issue comments. Next, the issue comments, expressed in natural language sentences, are converted into numeric vectors during the pre-processing step. Once the numerical vectors are obtained, a logistic regression technique is applied to the vectors to train a classifier. During this training step, we used grid search to tune the hyperparameters with three hyperparameters, `LogReg C`, `ngram_range`, and `max_df`, to use optimized values for the hyperparameters. Finally, the trained classifier can be used to classify the information type of comments on new issues.

In the following sections, we will explain the key aspects of the proposed information type classification technique in more detail.

B. RANDOM OVERSAMPLING

We use *Random Oversampling* [20] in the proposed approach to mitigate class imbalance in the dataset. We found that the number of instances in each class differed significantly; hence, mitigating class imbalance was a key issue in more accurate classification. In the previous study, SMOTE and class weights were used, and SMOTE showed slightly better performance when only textual features - sentences - were considered [17]. However, the performance was not high - around 0.5 F1-score- hence, we thought employing another oversampling method could solve this issue.

In the problem of classifying information type of issue comments, we believe that random oversampling is more appropriate than SMOTE. We are dealing with issue comments that are originally natural language sentences and are converted into numerical vectors for training. SMOTE helps to balance the class difference by adding more data in the minority class with fewer issue comments, resulting in a balanced class ratio. On the other hand, random oversampling simply copies existing instances randomly so that all new instances are actual sentences written by humans. In this way,

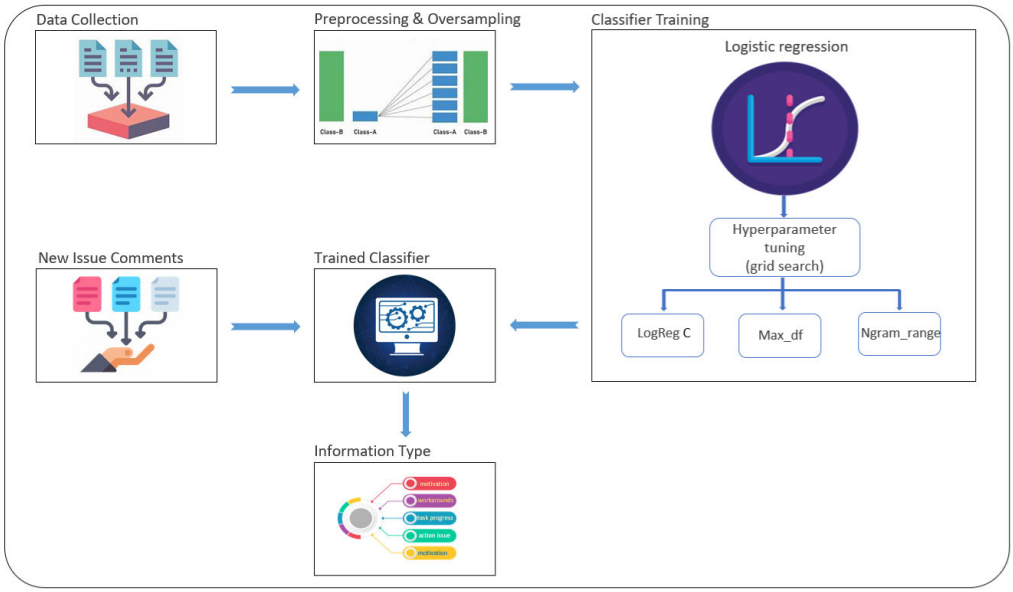

FIGURE 2. The overall process of the information type classification technique.

there is no risk of newly generated sentences belonging to different classes, and yet more instances are added for the minority classes to solve the class imbalance problem.

In addition, random oversampling is a simple method to be applied, compared to SMOTE. Although the dataset currently has only a few thousand instances, it would be better if we could apply lower-cost pre-processing steps and obtain the same performance. As we already explained, random oversampling only requires copying randomly selected instances, which is a relatively cheaper operation than SMOTE synthesis. Hence, if we can achieve the same performance, employing a simple, low-cost method is a reasonable choice.

C. DATASET EXPANSION

Expanding the dataset enhances machine learning accuracy by allowing the model to capture a broader range of patterns, reducing overfitting and improving its understanding of diverse scenarios.

To this end, we collected more issue comments and expanded the dataset used for our information type classification technique. In the previous study [17], Arya et al. collected about 4K issue comments from three OSS projects and labeled their information types to present a dataset, which we will refer to the Arya2019 dataset from now on.

We selected five prominent open-source software (OSS) projects for our study, incorporating three from the Arya2019 dataset and adding two additional projects to broaden our research scope. Subsequently, we gathered an additional 4K issue comments from these five OSS projects. Each comment was categorized according to 16 pre-defined information types as outlined in the Arya2019 analysis [17]. These newly collected and categorized comments were then combined with the existing Arya dataset to enrich our research foundation. We also excluded three classes (*Testing*, *Future Plan*, and *Issue Content Management*) with a small number of occurrences and considered 13 information types in total to be consistent with the existing dataset. With the combined dataset, we can expect more accurate classifiers by using additional training instances to represent the features of sentences in each information type.

We employed a simple multi-step labeling process to label the collected issue comments. First, one annotator read each sentence and assigned one of the information types. Then, two other inspectors verified the assigned information type for each sentence and labeled it if they thought the label was incorrect. Finally, the original annotator re-examined the marked sentences again, discussed their labels with inspectors, and decided which labels were more appropriate.

In this way, we can obtain more labeled issue comments relatively quickly. A more thorough labeling process could be used to obtain more carefully verified, reliable training instances like the previous study [17]. However, we found that most of the issue comments were quite plain and simple; hence, they could be labeled without much difference of opinions. Among 4,098 collected issue sentences, we disagreed on only 129 sentences (3.15%) and scrutinized them to label them correctly. This approach allowed us to focus more on difficult cases for correct labeling while we quickly obtained many easily labeled issue comments.

D. HYPERPARAMETER TUNING

Since we used grid search to optimize the hyperparameters, we must also specify a set of values for each considered hyperparameter. TABLE 1 shows the option values of hyperparameters used for a grid search.

TABLE 1. Option values used for `LogReg C`, `ngram_range`, and `max_df` hyperparameters.

Hyperparameter	Option Values
`LogReg C`	0.01, 0.1, 1, 10, 100
`ngram_range`	(1,1),(1,2),(1,3),(2,2)(2,3)
`max_df`	0.1, 0.5, 0.9, 1

The hyperparameter `LogReg C` controls the strength of regularization in the model. Smaller `LogReg C` values result in stronger regularization, preventing overfitting. In TABLE 1, five different `LogReg C` values from 0.01 to 100 were tested, with higher `LogReg C` values generally leading to better precision, recall, and F1-score.

`ngram_range` defines the range of word n-grams considered as features in text data processing. We tested five different values for `ngram_range`, as shown in TABLE 1. Larger ranges capture more context but may increase computational complexity, hence it is important to adjust this hyperparameter and find appropriate values.

`max_df` is a hyperparameter to filter out words with document frequency higher than the given value when we are using the `TfIdfVectorizer` of `scikit-learn` library. If we set the value higher, we can include more words in the vocabulary, even if the words may not be closely related to the key characteristic of the sentence. On the other hand, low values of `max_df` may also lower the dimension of the converted vectors by excluding more words, even if they are shown in other sentences. For `max_df`, we tested four values from 0.1 to 1, which contain very low to extremely high values.

Although the employed grid search method tested all combinations of values and provided the best outcome, we also investigated the influence of the hyperparameters. One downside of grid search is that it takes quite a long time if there are many different combinations of option values. In our case, with the option values shown in TABLE 1, there are 100 combinations for the three hyperparameters. If we can find a sweet spot or at least narrow down the range, it will be helpful for future training since we can use appropriate option values directly.

Hence, we additionally trained classifiers with different combinations of option values and analyzed the influence of hyperparameters.

III. EVALUATION

This section explains our empirical evaluation design for the proposed technique, including research questions, subjects and data collection, and experimental setup to answer the questions.

A. RESEARCH QUESTIONS

In this section, we provide Research Questions (RQ) to be answered about the proposed technique's key aspects and performance.

TABLE 2. The Subjects for data collection.

Project	Selected Issues#	Average Comments#	Closed Issues#
Tensorflow	7	54	35,654
Scikit-learn	7	88	8,682
SpaCy	2	49	5,371
Pandas	9	82	21,481
Keras	4	70	11,473

- **RQ1: Does random oversampling improve the classification performance?** We employ random oversampling instead of SMOTE to balance categories of the training data. To analyze its effect, we need to verify whether random sampling increases the classification performance of the information type.
- **RQ2: Can using more data improve classification performance?** We also expanded the dataset by collecting and labeling more issue comments. Hence, we need to analyze whether the additional issue comments are indeed helpful in improving classification performance.
- **RQ3: How do hyperparameters affect the classification performance?** We used grid search to find the optimal result with various hyperparameter ranges. However, understanding the influence of hyperparameters is necessary to adjust the classification technique further and more efficiently.

B. DATA COLLECTION

As we explained in Section II-C, we used five OSS projects as the subjects of our study. TABLE 2 shows brief information about the subjects. The project column shows each project's name. The selected Issues# and Average Comments# columns show the number of issues we collected and the average number of comments on the collected issues. The Closed Issues column indicates the total number of closed issues that were part of the data collected for each project.

The five OSS projects used for this study are all Python projects related to machine learning. Tensorflow, Scikit-learn, and SpaCy were also subjects from the previous study [17], but we collected additional comments from more recent issues. We have included two other OSS projects - Pandas and Keras - indicated by the bold font in TABLE 2 as our subjects as well. All of these OSS projects are popular and have been developed and managed for more than seven years, so there is a high probability that they contain issue discussions.

For evaluation, we used two datasets: one is Arya2019 dataset from the previous study [17], and the other is the dataset combined Arya2019 with issue comments collected from the OSS projects. This data was used to evaluate the proposed technique's effectiveness without additional training instances. The *Combined* dataset is obtained by combining Arya2019 and the 4,098 labeled issue comments we collected. In total, the combined dataset contains 8,428 labeled issues from five different OSS projects. We employed

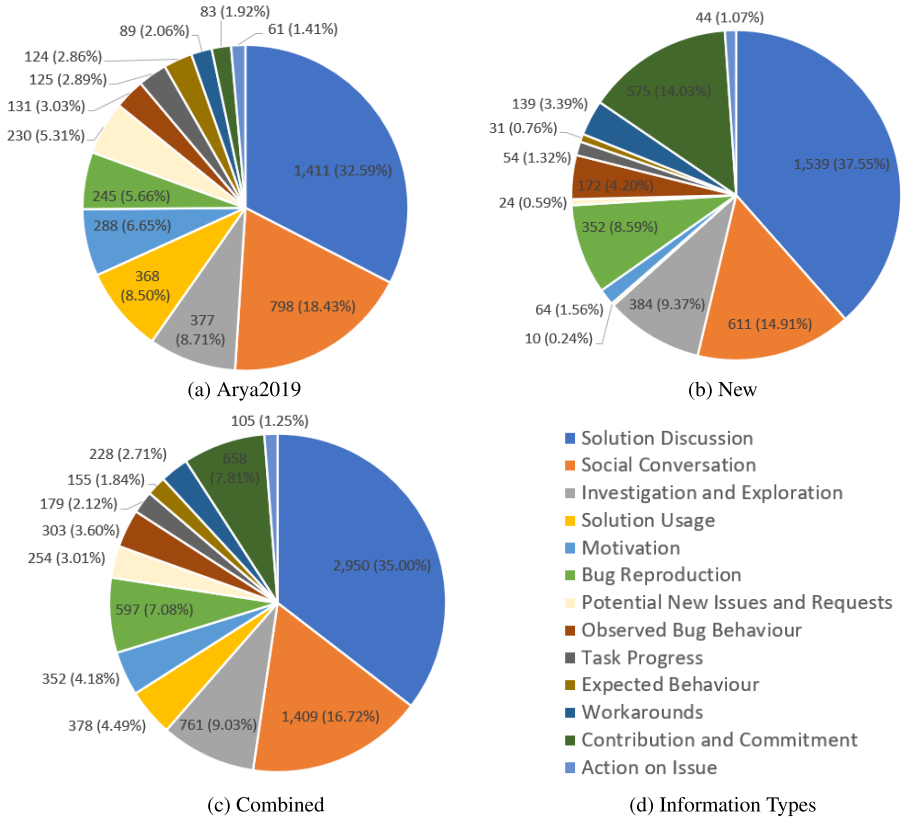

FIGURE 3. The ratio of information types in Arya2019, new, and the combined datasets.

the combined dataset for our empirical evaluation and observed whether additional training instances improved the classification performance. The combined dataset and other resources related to this study are publicly available.[2]

FIGURE 3 shows the ratio of information types in each of the three datasets: Arya2019, New, and Combined. Arya2019 is the 4,330 issue comments that were collected in a previous study [17]. The new dataset contains the 4,098 issue comments that were newly collected and labeled for this study. The combined dataset contains 8,428 issue comments from both the Arya2019 and New datasets.

Overall, all three datasets show a similar distribution of each information type in a certain way. Although there are some differences in actual ratio, the three most common information types are *Solution Discussion*, *Social Conversation*, and *Investigation and Exploration*. In all three datasets, these top-3 information types account for more than half of the total data.

However, there were also changes in the ratio of minority classes. In the newly collected dataset (FIGURE 3b), *Contribution and Commitment* and *Bug Reproduction* types have more instances compared to the Arya2019 dataset (FIGURE 3a). As a result, these types are the new third and fourth largest types from the combined dataset (FIGURE 3c).

[2]https://github.com/Bobur98/OSS_information_type_detection

The information type distribution in FIGURE 3c indicates that the class imbalance is still an important issue even after we added more instances to the dataset. Some information types, such as *Contribution and Commitment*, became more frequent after being supplied with more instances from newly collected data. However, still, the biggest information type (*Solution Discussion*) contains almost thirty times the instances from the smallest type (*Action on Issue*).

C. EXPERIMENTAL SETUP

To answer the research questions, we designed three experiments.

Firstly, we compared the classification performance using random oversampling and SMOTE. We used the Arya2019 dataset for this experiment to rule out the influence of additional training instances we collected. We employed 5-fold cross-validation and computed average precision, recall, and F1-score to evaluate the classification performance for the 13 information types except for the three excluded types as explained in Section II-C. For the measurements, weights were used to compensate for the size of classes when computing the average values. Since the only difference is an oversampling method, we can answer RQ1 by comparing the measurements of the two setups.

Since both techniques employed grid search for hyper-parameter tuning, the option values needed to be provided.

TABLE 3. The classification performance comparison for random oversampling and SMOTE.

Information Types	Random Oversampling			SMOTE			Ratio
	Precision	Recall	F1-score	Precision	Recall	F1-score	
Solution Discussion	0.89	0.66	0.76	0.56	0.66	0.61	32.59%
Social Conversation	0.95	0.88	0.91	0.76	0.70	0.73	18.43%
Investigation and Exploration	0.93	0.95	0.94	0.50	0.46	0.48	8.71%
Contribution and Commitment	0.99	1.00	1.00	0.51	0.39	0.44	1.92%
Bug Reproduction	0.96	0.97	0.96	0.49	0.52	0.50	5.66%
Motivation	0.93	0.99	0.96	0.39	0.36	0.37	6.65%
Observed Bug Behaviour	0.89	0.99	0.93	0.25	0.18	0.21	3.03%
Solution Usage	0.92	0.95	0.94	0.48	0.56	0.52	8.50%
Workarounds	0.99	0.97	0.98	0.38	0.16	0.22	2.06%
Potential New Issues and Requests	0.96	0.98	0.97	0.32	0.23	0.26	5.31%
Task Progress	0.93	1.00	0.97	0.40	0.34	0.36	2.89%
Expected Behaviour	0.98	0.99	0.98	0.42	0.30	0.34	2.86%
Action on Issue	1.00	1.00	1.00	0.75	0.51	0.59	1.41%
Total	0.95	0.95	0.95	0.54	0.54	0.53	100.00%

Due to the lack of information on which hyperparameter values would provide the best performance during experiment design, it was a conservative choice to use the originally reported values for the existing technique [17].

Secondly, we analyzed the combined dataset and repeated the first experiments with it. We first checked the ratio of information types in each of the Arya2019, newly collected and combined datasets, and observed whether the class imbalance issue can be mitigated with the newly collected instances. Then, we applied the classification techniques and compared the results with the first results to verify further whether additional instances could be helpful in improving performance.

Lastly, we investigated the influence of hyperparameters on the classification performance. We applied the proposed technique to the combined dataset, showing all possible hyperparameter values in TABLE 1. Then, we analyzed how the different combinations of hyperparameter values affected the classification performance. In particular, we checked the precision, recall, and F1-score change when one of the hyperparameters changed while the other two had fixed values.

IV. RESULTS

In this section, we present our empirical evaluation results for the classification performance of the proposed approach.

A. OVERSAMPLING METHODS

TABLE 3 shows the result of the comparison of classification performance between random oversampling and SMOTE. The columns Random Oversampling and SMOTE showed the classification performance when we used random oversampling and SMOTE, respectively. Performance was measured as precision, recall, and F1-score with the Arya2019 dataset using 5-fold cross-validation. The ratio column provides the ratio of each information type to the entire Arya2019 dataset. Each row separately shows the precision, recall, and F1-score values for each information type. The Total row shows the same values for all instances without distinguishing information types.

The results show that using random oversampling improves classification performance for all information types significantly in all aspects. When using random oversampling, we had an overall precision of 0.95, recall of 0.95, and F1-score of 0.95, which are much higher than 0.54, 0.54, and 0.53 of SMOTE. When we consider each information type separately, the performance (F1-score) of random oversampling ranges from 0.76 to 1.00, which is much higher than the 0.21 to 0.73 of SMOTE. Even in the worst case, using random oversampling shows better performance than the best case of using SMOTE. Therefore, random oversampling significantly outperforms SMOTE in precision, recall, and F1-score for each information type.

Moreover, random oversampling provides more reliable results regardless of information type. When we employed random oversampling, the F1-score did not vary too much, dropping to only 0.76 even for the worst case. However, using SMOTE causes significant degradation of performance for some information types such as *Observed Bug Behaviour* (0.21) or *Workarounds* (0.22), which take smaller portions - 3.03% and 2.06% respectively - in the dataset. Note that using SMOTE provided much better performance than the average for *Social Conversation* (0.73) or *Solution Discussion* (0.61), which are the two largest information types from the dataset. Hence, we can count on the proposed technique more and expect to obtain more reliable results for every information type.

As we have seen in the evaluation results, random oversampling is a better solution than SMOTE to the problem of class imbalance in information types of issue comments and provides better classification performance. Random oversampling directly replicates instances from the minority classes, giving the model sufficient examples to learn from. This ultimately leads to better classification

TABLE 4. The classification performance for the combined dataset.

Information Types	Random Oversampling			SMOTE		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Solution Discussion	0.91 (0.02↑)	0.64 (0.02↓)	0.75 (0.01↓)	0.61 (0.05↑)	0.67 (0.01↑)	0.64 (0.03↑)
Social Conversation	0.93 (0.02↓)	0.91 (0.03↑)	0.92 (0.01↑)	0.70 (0.06↓)	0.73 (0.03↑)	0.72 (0.01↓)
Investigation and Exploration	0.93	0.95	0.94	0.46 (0.04↓)	0.38 (0.08↓)	0.42 (0.06↓)
Contribution and Commitment	0.96 (0.03↓)	0.99 (0.01↓)	0.97 (0.01↓)	**0.78 (0.21↑)**	**0.72 (0.33↑)**	**0.75 (0.31↑)**
Bug Reproduction	0.93 (0.03↓)	0.97	0.95 (0.01↑)	0.51 (0.02↑)	0.54 (0.02↑)	0.52 (0.02↑)
Motivation	0.95 (0.02↑)	0.99	0.97 (0.01↑)	0.34 (0.05↓)	0.31 (0.05↓)	0.33 (0.04↓)
Observed Bug Behaviour	0.97 (0.08↑)	0.98 (0.01↓)	0.98 (0.05↑)	0.32 (0.07↑)	0.23 (0.05↑)	0.26 (0.05↑)
Solution Usage	0.89 (0.03↑)	0.99 (0.04↑)	0.94	0.41 (0.07↓)	0.59 (0.03↑)	0.48 (0.04↓)
Workarounds	0.97 (0.02↓)	0.99 (0.02↑)	0.98	0.32 (0.06↓)	0.20 (0.04↑)	0.24 (0.02↑)
Potential New Issues and Requests	0.97 (0.01↑)	0.99 (0.01↑)	0.98 (0.01↑)	0.26 (0.06↓)	0.20 (0.03↓)	0.23 (0.03↓)
Task Progress	0.96 (0.03↑)	0.98 (0.02↓)	0.97	0.35 (0.05↓)	0.23 (0.11↓)	0.28 (0.08↓)
Expected Behaviour	0.98	0.99	0.98	**0.30 (0.12↓)**	**0.17 (0.13↓)**	**0.21 (0.13↓)**
Action on Issue	1.00	1.00	1.00	0.70 (0.05↓)	0.63 (0.12↑)	0.66 (0.07↑)
Total	0.95	0.95	0.95	0.56 (0.02↑)	0.57 (0.03↑)	0.56 (0.03↑)

performance across all classes. On the other hand, SMOTE synthesizes new instances close to existing instances, which are represented by numeric vectors. However, it is possible that a close distance in numeric vector representations does not guarantee the similarity of the sentences they represent. This might explain why SMOTE was less effective for the issue comment classification problem, although we need a more thorough investigation to prove this point. Since the main purpose of this study is to provide an accurate information type classification, we leave further investigation on this matter as future work.

B. DATASETS

TABLE 4 provides the classification performance when we use the combined dataset. We tried both oversampling methods for the combined dataset and measured overall precision, recall, and F1-score with all instances and information types, which are shown as columns (measurements) and rows (information types). Underneath the numbers for each measurement, the increments (with ↑) or the decrements (with ↓) from the same measurement obtained with the Arya2019 dataset (TABLE 3) are presented. Bold-faced numbers indicate the information type that has shown the most changes - increment and decrement - after we added more instances to the dataset.

The effect of additional instances varies depending on the oversampling techniques used. In the case of using random oversampling, F1-score increases for five information types and decreases for two information types. However, the increments and the decrements only range from 0.01 to 0.05; eventually, the weighted average does not change. On the other hand, when we used SMOTE, six information types showed higher F1-score, and seven information types had lower F1-score, which means all information types have shown changes in performance. The amount of increments and decrements are 0.02 to 0.31 and 0.01 to 0.13, respectively, which leads to the increment of the weighted average F1-score to 0.56 (0.03↑). Therefore, adding more instances significantly affects classification performance when we use SMOTE, but the influence is less significant when we use random oversampling.

We observed that the changes in classification performance are, to some extent, linked to the class imbalance issue. Notably, the most substantial improvement was observed in the *Contribution and Commitment* information type, with its ratio soaring from 1.92% to 7.81%. Despite starting as the second smallest information type in the Arya2019 dataset, it ascended to the fifth largest in the combined dataset. Consequently, the trained classifier exhibited enhanced performance on this information type, resulting in a notable increase of 0.31 in the F1-score. In contrast,

TABLE 5. The classification performance of different hyperparameter value combinations.

Target	Fixed	Performance for Option Values					
`LogReg C`	`ngram_range`: (1,2) `max_df`: 0.9	Values	0.01	0.1	1	10	100
		Precision	0.70	0.80	0.93	0.95	0.95
		Recall	0.70	0.81	0.93	0.95	0.95
		F1-score	0.67	0.80	0.92	0.95	0.95
`ngram_range`	`LogReg C`: 10 `max_df`: 0.9	Values	(1,1)	(1,2)	(1,3)	(2,2)	(2,3)
		Precision	0.92	0.95	0.95	0.95	0.95
		Recall	0.92	0.95	0.95	0.94	0.94
		F1-score	0.92	0.95	0.95	0.94	0.94
`max_df`	`ngram_range`: (1,2) `LogReg C`: 1	Values	0.1	0.5	0.9	1	
		Precision	0.92	0.93	0.93	0.49	
		Recall	0.93	0.93	0.93	0.15	
		F1-score	0.93	0.92	0.92	0.13	

the *Expected Behaviour* information type experienced the most pronounced negative impact. The F1-score dropped by 0.13, which was almost 30% of the original score. Since there were very few instances of this type, its ratio decreased from 2.86% to 1.84%, making it the second-smallest information type. Such changes did not affect the performance significantly when we used random oversampling. However, using SMOTE was not effective in handling such class imbalance, hence classification performance varied for different information types.

C. HYPERPARAMETER TUNING

TABLE 5 shows the classification performance of different combinations of the hyperparameter values. The Target column indicates the hyperparameter whose values are changed, and the Fixed column shows the other two hyperparameters whose values are fixed. The other columns contain Precision, Recall, and F1-score for each value of the target hyperparameter. Apart from using specific values for hyperparameters, all other parts of the proposed approach were left unchanged - i.e., the use of random oversampling with the combined dataset.

We tested all possible combinations of the values we used for the grid search (TABLE 1), but due to the limitation of the space, we only report a part of the results that show the influence of hyperparameters more clearly. Although we do not list all cases in the paper, the full results of all the combinations are publicly available.[3]

Overall, the classification performance changes slightly if we use the values in a certain range but drops quickly outside of the range. For `LogReg C`, using values from 1 to 100 continuously provides a high F1-score, which is at least 0.92, while using smaller values decreases F1-score significantly to 0.67. `max_df` does not affect the performance much, as long as we do not use the extreme value, 1. Change `ngram_range` values also influence the performance slightly, just like `LogReg C` in the appropriate range.

The results in TABLE 5 may give an impression that using hyperparameter values in the appropriate ranges is okay, but it is not entirely true. For instance, it looks like `ngram_range` values have only a small impact on the classification performance. However, the option values (1, 2) and (1, 3) give the best performance, and compared to the worst case (1, 1), the overall performance increases by 0.03. This is the same increment we obtained by doubling the instances of the dataset, which cannot be ignored.

Moreover, using inappropriate combinations of option values may decrease the performance significantly. Note that we only report a part of the results for all combinations of option values. Even if we use an appropriate option value for one hyperparameter, out-of-range values for the other hyperparameters can significantly drop the performance. For example, using `ngram_range` value (1, 1) and `max_df` value 0.1 with `LogReg C` 1 value only achieves an F1-score of 0.88, which is still high but about 7% lower than the best case.

Therefore, although `LogReg C` has the most influence on the classification performance, we cannot ignore the other hyperparameters since they may also affect the performance greatly if we use a wrong combination of values.

In conclusion, our empirical evaluation results show that the proposed random oversampling approach remarkably improves classification performance. The weighted average F1-score of the technique is 0.95, which is 79.25% higher than the compared technique. With such high precision and recall, we can develop more reliable applications that accurately identify information types from issue discussions. In addition, we also provide an extended dataset with more instances that can further improve performance, although the effect can be limited. Finally, we analyzed the influence of hyperparameters on classification performance, which can be considered for training classifiers for issue comment information types.

V. RELATED WORK

A. INFORMATION TYPE CLASSIFICATION IN ISSUE TRACKING SYSTEMS

Issue management is an important part of software development and evolution, and issue-tracking systems are a great source of information to help with related tasks. Many techniques have been proposed to identify

[3] Analysis Results of All Hyperparameter Combinations.

the types of information obtained from an ITS and help developers understand and extract more useful and actionable information for issue management.

Arya et al. conducted a study to address information retrieval in long discussion threads [17]. They identified 16 information types from issue comments and proposed and investigated techniques to automatically classify the information types. To classify issue comments based on their information types, they extracted conversation features and text features and then applied random forest and logistic regression to obtain a trained classification. Although Arya et al. pioneered classifying information types to more easily retrieve valuable information from issue discussions, an important limitation of their work is the low accuracy of the proposed techniques. They achieved the best overall precision and recall by using a random forest with conversational features, but the values were about 0.6; thus, the resulting information type is unreliable. Our proposed technique improved the classification performance significantly to 0.95, hence information retrieval from issue discussions could be much more reliable with our technique.

Similarly, Mehder and Aydemir tried to improve the classification performance using deep language models [18]. This research first proposes an Arya et al.'s dataset comprising 4,656 labeled sentences annotated with 16 information types extracted from discussions in three AI libraries on GitHub. After preprocessing, the dataset is reduced to 4,330 sentences across 13 classes due to duplicates and class imbalance. A benchmark is created by splitting the dataset into training and testing sets, ensuring the preservation of label ratios. The replication study focuses on logistic regression models, replicating Arya et al.'s experiments with textual features. Results showed comparable or improved performance over the original work due to differences in validation methods. Then, the research extends to classification using transformer-based deep language models (BERT, RoBERTa, and DistilBERT). Two evaluation settings are designed: (1) using the benchmark dataset and (2) Leave-One-Group-Out cross-validation. Performance comparison demonstrates that BERT variants outperform logistic regression through statistical testing. Finally, the study examines the models' performance when a new issue arises, showing RoBERTa's suitability for such scenarios, although some classes have zero F1-scores due to data sparsity. Even though the replicated technique and the newly proposed models did not perform well, they were all slightly over 0.5.

Many other studies have tried to classify information obtained from ITSs. McMillan et al. introduced an SVM-based method for classifying software applications using API call features, effective on large Java repositories [22]. Our research complements this by introducing an advanced technique for classifying issue comments in ITSs, enhancing automated software analysis, and improving bug localization. Their model achieved an 85% hit rate when tested on a GitHub dataset. Ferreira et al. compared BERT with classic models for incivility detection in open-source code discussions. BERT, integrating context and SMOTE, outperformed traditional models with an F1-score of 0.87 [23]. Our research advances the classification of issue comments in ITSs, utilizing random oversampling to achieve a high F1-score. While we focus on classifying issue-related information for better management, Ferreira et al. demonstrate the effectiveness of BERT, especially with data augmentation and SMOTE balancing. Pan et al. developed a method for extracting crucial information from OSS developer chat rooms, employing a two-level taxonomy to categorize discussions and achieving an accuracy of 0.81 in classifying new threads, outperforming existing methods [24]. While their project succeeded in categorizing discussions, insights from our method could potentially enhance their approach by providing strategies for improving classification performance, such as oversampling techniques and dataset expansion. Additionally, Siddiq et al. utilized a BERT-based technique to classify GitHub issue types, achieving an F1-score of 0.8571 on a dataset of over 800,000 labeled issues [25]. Kallis et al.'s Ticket Tagger program autonomously categorizes software issue tickets on GitHub, achieving an F-measure score of 0.83 [26]. Kim and Lee studied labels assigned to issue reports and how such labeled issues have been managed during software development. In this study, they identified the categories of the labels, which represent information types of the labels [27]. Nadeem et al. introduced a RoBERTa-based method, achieving high F1-scores for bug reports, enhancements, and questions, implemented in an industry tool called Automatic Issue Classifier (AIC) [28]. Merten et al. explored issue types and information distribution in unstructured natural language text, emphasizing the presence of prioritization and scheduling information across various open-source projects [29]. Our approaches, like handling class imbalance, augmenting the dataset with more labeled instances, and applying hyperparameter tuning techniques, can potentially enhance the accuracy and effectiveness of their classification model. Krasniqi and Rrezarta developed an automated method for extracting relevant bug-fixing comments from large discussion threads [30]. They combined Sentiment Analysis, Text Rank Model, and Vector Space Model to identify comments based on query relevance, positivity, and semantic relevance, outperforming the baseline Vector Space Model. However, classifying software issue discussions faces challenges such as data imbalance and noise. Pattaramon et al. identified data imbalance issues [31], while our study addresses these challenges by enhancing issue comment classification using random oversampling in ITSs.

B. ISSUE TRACKING SYSTEMS

One line of work tries to locate a bug using the ITS's information. Information-Retrieval (IR) based bug localization tries to identify faulty locations by using information retrieved

from issue reports [9], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43]. Although our proposed technique does not classify issue reports directly, it can be used to add more information by classifying issue report comments. Also, since issue reports themselves contain sentences related to the issues, our technique might be used to filter out less relevant sentences.

Software bug prediction is challenging, but it is important for prioritizing bug fixes and improving software quality. Machine learning is effective for bug prediction, and several studies have proposed various machine learning models and features for this task. Malhotra et al. compare five machine learning classifiers (Multinomial Naive Bayes, Decision Tree, Logistic Regression, Random Forest, and AdaBoost) for predicting fault priorities using data from six open-source projects [44]. The authors found that all five classifiers performed well, but Multinomial Naive Bayes achieved the best overall performance. Barah et al. propose a machine-learning system for predicting the severity of software bug reports in closed-source projects [45]. Just as we experimented with different oversampling methods and hyperparameter tuning techniques to optimize our classification model, researchers comparing eight different machine learning algorithms (Naive Bayes, Naive Bayes Multinomial, Support Vector Machine, Decision Tree (J48), Random Forest, Logistic Model Trees, Decision Rules (JRip), and K-Nearest Neighbor) and compare their performance against a dataset of bug reports from a closed-source project. The authors found that Logistic Model Trees performed the best, with an accuracy of 86.31%. Samant et al. explore the use of machine learning to predict bug priority in software projects [46]. They propose a machine-learning model that uses various features, including bug report text, developer information, and project history. The authors evaluate their model on a dataset of bug reports from two open-source projects and find that it achieves 80% accuracy. Oliveira et al. propose a deep-learning approach for predicting bugs [47]. They utilize a convolutional neural network to learn features from the text of bug reports and then use a recurrent neural network to predict bug priority. The model's evaluation on a dataset of bug reports from two open-source projects shows it achieves 85% accuracy. Wong et al. delve into fault localization techniques in software development, which involves identifying and fixing bugs or faults efficiently [7]. They can apply similar techniques like classification of information types to identify the type of bugs or faults and categorize them into several groups, which can help fix bugs more efficiently. Clemente et al. studied the use of machine learning to predict the severity of software bug reports [48]. Their proposed model incorporates diverse features such as bug report text, developer details, and project history. Upon evaluating a dataset comprising bug reports from two open-source projects, the model demonstrates an accuracy of 82%. By applying similar text processing techniques, they can effectively represent textual data for machine learning models, facilitating more accurate predictions of fault priorities or bug severity.

Previous research has described several methods for automatically classifying issues in bug-tracking systems. For example, Antoniol et al. have demonstrated the potential of machine learning algorithms to distinguish bugs from other issue types [49]. Herzig et al. team defined six issue categories– bugs, feature requests, and documentation requests [50]. They found that developers often mislabeled these categories. In response, Zhou et al. integrated structured and free-text data to develop a classifier [51]. This model was highly adept at identifying whether a report was a bug or a different issue type. Murphy et al. address bug triage in open-source projects by using machine learning for predicting developer assignments based on bug description [16]. Our method can enhance the analysis of bug descriptions in the bug triage process. By applying similar classification methods to categorize information within bug reports, the bug triage system can better understand and extract relevant details from bug descriptions. Notably, given GitHub's minimalist issue tracking structure, GitHub issues don't typically contain structured data. Our solution proficiently automates the tagging of GitHub issues, basing its classification solely on the text in the issue titles and details. This is crucial for developers, as it expedites the process of addressing new issues [52] by instantaneously sorting them upon submission. In summary, our work on information type classification of issue comments offers valuable insights and methodologies that can be leveraged by researchers working on predicting fault priorities, bug severity, or bug priority in software projects. By incorporating similar approaches and techniques, they can enhance the accuracy and efficiency of their classification models.

VI. THREATS TO VALIDITY

Several points might affect the validity of this study.

First of all, this study was conducted on subjects that were all open-source Python projects related to machine learning libraries. Hence, it is possible that the proposed technique might not show enhanced performance if issue comments were collected from other sources. For instance, issue discussions could be more formal in commercial software product development; hence, we may observe less social conversation information. Also, other information types, such as Bug Reproduction or Workarounds, could be more complicated and make it less easy to capture the patterns in other types of software. However, by employing random oversampling, we resolved the class imbalance issue and achieved very high performance. Although there is a limit to representing all software with our subjects, software development tends to share common characteristics, so there is a good chance that our high-performance technique can also show promising performance for other subjects.

Also, labeling the collected issue comments could be subjective and not entirely free from human errors. We employed

a more simplified method to collect more issue comments in a relatively short time period. However, that does not mean that the provided results are not trustworthy. As we explained in Section II-A, we verified labels in a multi-step process with multiple human examiners. Only 3.15% of the labels disagreed, and we also solved the issues for them. Moreover, we evaluated the proposed technique with another independent dataset from the previous study, and the technique continuously showed better performance, which does not diminish the study's implications. In addition, the resources, including code and the datasets we used in this study, are publicly available for further verification, as we mentioned in Section III-B.

VII. CONCLUSION

In this study, we proposed an improved automatic information type classification technique for issue comments in software development. We primarily implemented random oversampling to enhance performance alongside two supplementary techniques: data extension and hyperparameter experimentation. We utilized random oversampling during preprocessing to address the class imbalance, and our evaluation demonstrates a notable performance boost simply by switching oversampling methods. Our proposed approach yields more precise and dependable classification results for information types within issue comments than existing techniques.

Furthermore, we augmented the dataset with additional labeled issue comments and observed that employing random oversampling yielded consistent results for both the original data from Arya2019 [17] and the extended dataset.

In addition, we investigated the influence of hyperparameters on the classification of information types in issue comments. We found that we need to choose the hyperparameters `LogReg C`, `max_df` carefully, and `ngram_range` since the classification performance may significantly decrease if out-of-range values are selected for these hyperparameters. This information can be helpful when future studies require the application of machine learning techniques to classify issue comments.

We can leverage more accurate and reliable classification results with the proposed technique. To develop a new technique to help software engineers with issue management, such a high-performance technique is an essential part of a successful application. Since we achieved quite a high performance for the classification, our future work will focus more on the applications of the proposed technique's outcome and further investigations on this matter, such as which kinds of approaches are more effective for issue comment-related problems.

REFERENCES

[1] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, Feb. 2010, pp. 291–300.

[2] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *Int. J. Appl. Sci. Eng.*, vol. 17, no. 4, pp. 331–344, Jan. 2020.

[3] F. Tu, J. Zhu, Q. Zheng, and M. Zhou, "Be careful of when: An empirical study on time-related misuse of issue tracking data," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 307–318.

[4] C. Catal and B. Diri, "Software defect prediction using artificial immune recognition system," in *Proc. 25th Conf. IASTED Int. Multi-Conf., Softw. Eng.* Anaheim, CA, USA: ACTA Press Anaheim, 2007, pp. 285–290.

[5] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 31–41.

[6] T. Merten, D. Kramer, B. Mager, P. Schell, S. Bürsner, and B. Paech, "Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data?" in *Requirements Engineering: Foundation for Software Quality*. Gothenburg, Sweden: Springer, 2016, pp. 45–62.

[7] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, and D. Li, "Software fault localization: An overview of research, techniques, and tools," in *Handbook of Software Fault Localization: Foundations and Advances*, 2023, pp. 1–117.

[8] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2014, pp. 803–814.

[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] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *Proc. Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2015, pp. 190–197.

[11] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 14–24.

[12] A. Zakari, S. P. Lee, K. A. Alam, and R. Ahmad, "Software fault localisation: A systematic mapping study," *IET Softw.*, vol. 13, no. 1, pp. 60–74, Feb. 2019.

[13] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 361–370.

[14] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2008, pp. 308–318.

[15] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Aug. 2009, pp. 111–120.

[16] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, 2004, pp. 1–6.

[17] D. Arya, W. Wang, J. L. C. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 454–464.

[18] S. Mehder and F. Basak Aydemir, "Classification of issue discussions in open source projects using deep language models," in *Proc. IEEE 30th Int. Requirements Eng. Conf. Workshops (REW)*, Aug. 2022, pp. 176–182.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[20] A. Moreo, A. Esuli, and F. Sebastiani, "Distributional random oversampling for imbalanced text classification," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2016, pp. 805–808.

[21] D. Elreedy and A. F. Atiya, "A comprehensive analysis of synthetic minority oversampling technique (SMOTE) for handling class imbalance," *Inf. Sci.*, vol. 505, pp. 32–64, Dec. 2019.

[22] C. McMillan, M. Linares-Vásquez, D. Poshyvanyk, and M. Grechanik, "Categorizing software applications for maintenance," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 343–352.

[23] I. Ferreira, A. Rafiq, and J. Cheng, "Incivility detection in open source code review and issue discussions," *J. Syst. Softw.*, vol. 209, Mar. 2024, Art. no. 111935.

[24] S. Pan, L. Bao, X. Ren, X. Xia, D. Lo, and S. Li, "Automating developer chat mining," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2021, pp. 854–866.

[25] M. L. Siddiq and J. C. Santos, "Bert-based GitHub issue report classification," in *Proc. 1st Int. Workshop Natural Lang.-Based Softw. Eng.*, 2022, pp. 33–36.
[26] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on GitHub," *Sci. Comput. Program.*, vol. 205, May 2021, Art. no. 102598.
[27] J. Kim and S. Lee, "An empirical study on using multi-labels for issues in GitHub," *IEEE Access*, vol. 9, pp. 134984–134997, 2021.
[28] A. Nadeem, M. U. Sarwar, and M. Z. Malik, "Automatic issue classifier: A transfer learning framework for classifying issue reports," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2021, pp. 421–426.
[29] T. Merten, B. Mager, P. Hübner, T. Quirchmayr, B. Paech, and S. Bürsner, "Requirements communication in issue tracking systems in four open-source projects," in *Proc. REFSQ Workshops*, 2015, pp. 114–125.
[30] R. Krasniqi, "Extractive summarization of related bug-fixing comments in support of bug repair," in *Proc. IEEE/ACM Int. Workshop Automated Program Repair (APR)*, Jun. 2021, pp. 31–32.
[31] P. Vuttipittayamongkol, E. Elyan, and A. Petrovski, "On the class overlap problem in imbalanced data classification," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106631.
[32] M. M. Rahman and C. K. Roy, "Improving IR-based bug localization with context-aware query reformulation," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 621–632.
[33] Y. Wang, Y. Yao, H. Tong, X. Huo, M. Li, F. Xu, and J. Lu, "Bug localization via supervised topic modeling," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 607–616.
[34] C. Mills, E. Parra, J. Pantiuchina, G. Bavota, and S. Haiduc, "On the relationship between bug reports and queries for text retrieval-based bug localization," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3086–3127, Sep. 2020.
[35] M. Pradel, V. Murali, R. Qian, M. Machalica, E. Meijer, and S. Chandra, "Scaffle: Bug localization on millions of files," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 225–236.
[36] M. Rath and P. Mäder, "Structured information in bug report descriptions—Influence on IR-based bug localization and developers," *Softw. Quality J.*, vol. 27, no. 3, pp. 1315–1337, Sep. 2019.
[37] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *Proc. IEEE/ACM 25th Int. Conf. Program Comprehension (ICPC)*, May 2017, pp. 218–229.
[38] M. Rath, D. Lo, and P. Mäder, "Analyzing requirements and traceability information to improve bug localization," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories (MSR)*, May 2018, pp. 442–453.
[39] Z. Yang, J. Shi, S. Wang, and D. Lo, "IncBL: Incremental bug localization," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2021, pp. 1223–1226.
[40] K. E. E. Swe and H. M. Oo, "Bug localization approach using source code structure with different structure fields," in *Proc. IEEE 16th Int. Conf. Softw. Eng. Res., Manag. Appl. (SERA)*, Jun. 2018, pp. 159–164.
[41] X. Xia, D. Lo, X. Wang, C. Zhang, and X. Wang, "Cross-language bug localization," in *Proc. 22nd Int. Conf. Program Comprehension*, Jun. 2014, pp. 275–278.
[42] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin, "Improving bug localization with an enhanced convolutional neural network," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2017, pp. 338–347.
[43] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 345–355.
[44] R. Malhotra, A. Dabas, A. S. Hariharasudhan, and M. Pant, "A study on machine learning applied to software bug priority prediction," in *Proc. 11th Int. Conf. Cloud Comput., Data Sci. Eng.*, Jan. 2021, pp. 965–970.
[45] 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.
[46] N. Samant, H. Limaye, A. Bapat, S. Shinde, and A. Nerurkar, "Optimizing issue tracking systems using deep learning-based issue classification," in *Proc. 2nd Int. Conf. Paradigm Shifts Commun. Embedded Syst., Mach. Learn. Signal Process. (PCEMS)*, Apr. 2023, pp. 1–6.
[47] P. Oliveira, R. M. C. Andrade, I. Barreto, T. P. Nogueira, and L. Morais Bueno, "Issue auto-assignment in software projects with machine learning techniques," in *Proc. IEEE/ACM 8th Int. Workshop Softw. Eng. Res. Ind. Pract.*, Jun. 2021, pp. 65–72.
[48] C. J. Clemente, F. Jaafar, and Y. Malik, "Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?" in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2018, pp. 95–102.
[49] 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," in *Proc. Conf. Center Adv. Stud. Collaborative Res. Meeting Minds*, 2008, pp. 304–318.
[50] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 392–401.
[51] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *J. Softw., Evol. Process*, vol. 28, no. 3, pp. 150–176, Mar. 2016.
[52] J. L. Cánovas Izquierdo, V. Cosentino, B. Rolandi, A. Bergel, and J. Cabot, "GiLA: GitHub label analyzer," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, Mar. 2015, pp. 479–483.

BOBURMIRZO MUHIBULLAEV received the bachelor's degree in computer science and business administration from Sejong University, in 2021. He is currently pursuing the master's degree in computer science and engineering with Seoul National University of Science and Technology. His research interests include machine learning, software engineering, and web development. He is researching to improve information types' analysis in open-source software issue discussions. Additionally, he practices web development by creating useful websites.

JINDAE KIM (Member, IEEE) received the B.S. degree in physics and computer science and the M.S. degree in computer science and engineering from Seoul National University, South Korea, in 2009 and 2011, respectively, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, in 2019. He is currently an Assistant Computer Science and Engineering Professor with Seoul National University of Science and Technology. His research interests include automatic program repair and mining software repositories and software evolution.