Report on
'Predicting the Objective and Priority of Issue Reports'

from
Maliheh Izadi, Kiana Akbari and Abbas Heydarnoori
Empirical Software Engineering volume 27

Presented by Pragya Bhandari

Department of Computer Science, Mathematics, Physics and Statistics
Summer Term
The University of British Columbia
Okanagan
August 15, 2023

1 Introduction

Software bugs are an important element of the software engineering lifecycle, and their timely detection, analysis, and resolution can significantly impact the software development life cycle's overall cost and success. To manage these bugs, teams use a bug-tracking system, which helps track the bug's status from the moment it's reported to the point when it's closed. GitHub's bug tracker is called *Issues*. Once an issue is opened, the team decides its objective and priority and assigns it to a developer to be fixed. An issue can be labeled using one of the default GitHub issue labels such as bug, enhancement, documentation, support etc., or using any custom labels relevant for the project indicating the objective of the issue report. Issues can also be categorized into high-priority and low-priority types to ensure an efficient bug-triaging process and prioritize the high-risk issues. However, for larger projects with many unresolved issues, this process can be time-consuming and requires some knowledge about the project as well as the issue to select the most appropriate labels and priority.

Izadi et al. [10] found two main problem areas related to issue labeling and categorization. In a study conducted by Cabot et al. [4], a comprehensive analysis was conducted on approximately three million non-forked GitHub repositories. The objective was to investigate the usage of labels and its influence on issue resolution. The findings revealed that a mere 3% of the repositories had labeled issues, suggesting that developers rarely engage in issue labeling. Moreover, among the repositories that did implement issue labeling, only around 58% of the issues were actually labeled. The study also demonstrated a strong correlation between the number of labeled issues in a repository and both issue resolution and user engagement rates. These results suggest that incorporating issue labeling can potentially benefit project management. In a separate study by Herzig et al. [8], it was discovered that approximately 34% of bug reports submitted by different users were misclassified, meaning they had incorrect labels. Misclassified reports can be highly misleading and contribute to a lengthier issue resolution process. They can also lead to failed task assignments and negatively impact the performance of bug prediction models. This underscores the importance of properly labeling issue reports using an unbiased model in order to address these issues effectively.

Thus, Izadi et al. [10] sought to develop automatic bug classifiers for issue objective-based classification and issue priority-based classification, aiming to identify the most suitable approach. The present paper conducts a comparative analysis of the solutions proposed by various researchers, closely aligned with the problem addressed by Izadi et al. [10], pertaining to both objective classification and priority classification. This research provides valuable insights into the datasets and methodologies employed by these researchers, endeavoring to discern patterns that may offer generalizations for bug classification-related predicaments. The subsequent sections are organized as follows: The Methodology section elucidates the approach employed to curate the most pertinent research papers concerning our focal study, i.e., Izadi et al. [10]. Subsequently, the Results section presents comprehensive information about the diverse papers, their respective datasets, and strategies, juxtaposed with those of Izadi et al. Finally, the Discussion section entails key takeaways derived from the comparisons and reviews of the papers.

2 Methodology

The primary objective of our survey was to conduct a comparative analysis between the paper authored by Izadi et al. [10] and other closely aligned and pertinent research papers. Notably, the paper [10] addresses two key problem statements: objective-based classification, aimed at classifying GitHub issues using three distinct labels, and priority-based classification, designed to categorize issues into high or low-priority types. To comprehensively address both these problem statements and identify all relevant papers, we conducted a search for related works centered on bug classification-related problem statements without imposing specific criteria restrictions to begin with.

2.1 Approach

The paper collection process involved a meticulous search on Scopus[1], utilizing extensive search parameters. Scopus is a collection of scientific information that includes abstracts and citations from peer-reviewed research. I chose Scopus because it is one of the largest curated bibliographic databases, and it offers an effective search feature with multiple in-built criteria to filter the search results with. The inclusion criteria were established to ensure the relevance and quality of the selected papers. These criteria encompassed the keywords present in the research papers, the publication year, field of study, publication stage, and paper type. Each inclusion criterion will be explained in detail in the subsequent section.

The search for relevant papers was conducted using a combination of keywords in an "OR-ing" manner. The following expression was used for the search:

"bug classification" OR "software defect classification" OR "bug reports classification" OR "issue classification" OR "software bug classification" OR "bug tracker classification" OR "bug categorization"

The selection of papers for inclusion in this review encompassed the period from 2015 to the year 2023, thus spanning eight years of research endeavors focused on bug classification. This time range was chosen to ensure the relevance and currency of the selected literature, as papers published before 2015 were considered potentially outdated for the present study. It should be noted that we had completed the research paper collection by May 3rd, 2023, and any papers published in the year 2023 after this date were not included in this survey. In the interest of maintaining a specific focus, only papers from the Computer Science field were considered for inclusion. Furthermore, only papers in their final publication stage were included. The final publication stage of a paper refers to the point in the publishing process where the paper has undergone rigorous peer review, revisions, and editing, and it is accepted by a reputable academic journal or conference for formal publication. Lastly, the scope was limited to papers falling under the categories of Conference and Journal, aligning with the common types of academic publications in the field of Computer Science.

Following the initial collection of 133 papers, a subsequent refinement was conducted through the application of exclusion criteria to narrow down the list. To ensure the credibility of the selected papers, those not published in reputable conferences or journals were excluded from consideration. Objective benchmarks were set, requiring a minimum Conference ranking

[1]https://www.scopus.com

of B and a Journal impact factor of 2.5 as thresholds for inclusion. Conference ranking indicates that the conference is of reasonably high quality and impact whereas journal impact factor gives an idea that a journal that has a respectable level of impact and reach within the academic community. Any papers falling below these established thresholds were omitted from the final list. However, it is worth noting that some papers were kept in the survey even if they did not meet the criteria completely but had promising content relevant to the review. This was done based on my supervisor Dr. Gema Rodriguez-Perez's confirmations.

Additionally, a comprehensive assessment of the research objectives of each study in the list was undertaken. Papers that did not specifically address bug classification based on GitHub issue labels or bug priority were excluded from further consideration. Upon completion of the rigorous filtering process, six highly pertinent papers remained, which were deemed suitable for comparison with the work by Izadi et al. [10] in terms of their bug classification strategies.

3 Results

In this section, the findings will be provided through a comprehensive examination of multiple papers pertinent to two distinct bug classification models: the Objective-based classification model and the Priority-based classification model. For each model, a detailed analysis of the datasets employed and the optimal approach yielding the best results will be presented.

3.1 Objective-based Classification model

3.1.1 Dataset

The main paper in question by Izadi et al. [10] collected their dataset of 817,743 bug reports from GitHub API. Subsequently, they extracted the textual details of these issue reports, which included the titles and descriptions. Additionally, they also collected all the labels assigned to each issue. They selected three distinct categories, namely Bug Report, Enhancement, and Support/Documentation, which had 362K, 342K, and 112K preprocessed issues falling into the categories, respectively. If we look at other papers that have also classified bugs into one of the GitHub issue labels, naturally, the dataset would also be sourced from GitHub issue records as well. The paper by Kallis et al. [11] gathered their data from GitHub Archive[2] using Google BigQuery[3] from 12,112 heterogeneous projects. On the other hand, the paper by Bharadwaj and Kadam [3] used the data given to them by a third party, although the data seems to be sourced from the GitHub platform nonetheless.

While the papers by Izadi et al. [10] and Kallis et al. [11] used the textual features of issue reports, more specifically the title and description text, the paper by Bharadwaj and Kadam [3] used other non-textual features in addition to the issue title and description. The non-textual features utilized in the study [3] are Early Issue, Owner Opened, and Question Title. Early Issue indicated whether an issue is considered "early" based on its issue number. Early issues have smaller issue numbers and tend to have more enhancement requests, while

[2]https://gharchive.org

[3]https://cloud.google.com/bigquery

larger issue numbers are associated with more questions and bug reports. The threshold used to define "early" was set at 98. The owner-opened feature was designed to identify if the issue was created by the project owner. It was observed that project owners are the primary authors of enhancement-related issues. Question Title feature served to address the scarcity of data in the question class, a binary feature was created using simple natural language processing techniques. This feature indicates whether the sentences in the issue title are questions.

While the papers all classify the issues into the default labels from the GitHub platform, Kallis et al. [11] and Bharadwaj and Kadam [3] classify issues into Bug, Enhancement, and Question labels, whereas Izadi et al. [10] classifies issues into Bug, Enhancement and Support/Documentation labels from Github.

3.1.2 Classification Strategies

This section provides an overview of the classification models and strategies employed in three comparative papers. Kallis et al. [11] utilized FastText [4], a text classification and word embedding library developed by Facebook's AI Research (FAIR) lab, for direct implementation of their classifier. However, Kallis et al. [11] did not conduct a comparative analysis with any baseline models. The results obtained through their approach yielded F1-scores of 83.1%, 82.3%, and 82.5% for the respective labels Bug, Enhancement, and Question.

In contrast, Izadi et al. [10] employed various methods to compare their fine-tuning approach with RoBERTa [12] against other strategies, such as TicketTagger [11] using Fasttext, Intention mining CNN [9], Bidirectional-Long Short-Term Memory (Bi-LSTM), Multinomial Naive Bayes (MNB), and a Keyword-based rule approach. Among these methods, TicketTagger, proposed by Kallis et al. [11], was chosen for the comparative study using the dataset collected by Izadi et al. [10]. The results reported by Izadi et al. [10] indicate that fine-tuning RoBERTa achieved the highest F1-scores of 85%, 84%, and 67% for the labels Bug, Enhancement, and Support/Documentation, respectively. On the other hand, TicketTagger [11] exhibited inferior performance with the dataset provided by Izadi et al. [10], resulting in F1-scores of 80%, 78%, and 54% for the classes Bug, Enhancement, and Support/Documentation, respectively.

Similarly, Bhardwaj and Kadam [3] conducted experiments with FastText, BERT [5], XLNet, CodeBERT [6], and RoBERTa. It is pertinent to mention that Bhardwaj and Kadam [3] augmented their neural network architecture by incorporating a final linear layer with the softmax function and using the cross-entropy loss function after exploring the aforementioned large language models. Their research demonstrated the best F1-scores using RoBERTa, achieving 89%, 88%, and 61% for the Bug, Enhancement, and Question labels, respectively.

3.2 Priority-based Classification model

3.2.1 Dataset

This section delves into the datasets utilized in each of the papers under consideration. With the exception of Izadi et al. [10], all relevant papers [2] [7] [1] that focused on priority

[4]https://fasttext.cc/

Table 1: Comparative juxtaposition of the best results obtained for each paper for Objective prediction.

Papers	Input Features	NLP	Classifier	Class-wise F1-Score
Kallis et al. [11]	Bug Title Bug Body	Bag-of-words	FastText	Bug - 83.1% Enhancement - 82.3% Question - 82.5%
Bhardwaj and Kadam [3]	Bug Title, Body Early Issue Owner opened Question title	–	RoBERTa with Softmax layer	Bug - 89% Enhancement - 88% Question - 61%
Izadi et al. [10]	Bug Title Bug Description	–	RoBERTa	Bug - 85% Enhancement - 84% Support - 67%

classification obtained their datasets from Bugzilla [5], a bug tracking system. On the other hand, Izadi et al. [10] collected their dataset for the priority classification model from the GitHub platform.

Regarding the classification of priorities, Izadi et al. [10] and Guo et al. [7] adopted binary classes, namely low and high priority, and non-severe and severe, respectively. In contrast, Alazzam et al. [2] employed three priority levels, namely P1, P2, and P3, while Ahmed et al. [1] used four priority levels, namely P1, P2, P3, and P4, as their classification classes.

3.2.2 Approach

Izadi et al. [10] adopted a sophisticated approach in implementing their bug priority classification model, incorporating a diverse set of features in addition to textual attributes extracted from issue titles and bodies. The authors [10] selected 28 features, which were categorized into five distinct groups: textual, discussion-related, events-related, developer-related, and sentiment-related features. The sentiment-related features were particularly noteworthy, as the authors [10] proposed that issue severity could be inferred from the sentiment expressed in the issue report. They suggested that more severe issues might contain specific emotional text indicators. For sentiment analysis, Izadi et al. [10] employed SentiStrength [6] to quantify sentiments and TextBlob [7] to obtain additional indicators, such as polarity and subjectivity of emotions. This emphasis on sentiment analysis to determine issue priority was not explored in the other reviewed papers, which primarily focused on textual and non-textual features like product information, issue resolution status, and comment count.

Subsequently, each paper selected a collection of machine learning (ML) algorithms to train their respective classification models. Izadi et al. [10] experimented with MNB, K-Nearest Neighbor (KNN), Logistic Regression (LG), and Random Forest (RF) algorithms. They observed that Random Forest, when combined with their chosen feature set, outperformed

[5]https://www.bugzilla.org/
[6]http://sentistrength.wlv.ac.uk/
[7]https://textblob.readthedocs.io/en/dev/

Table 2: Comparative juxtaposition of the best results obtained for each paper for Priority prediction.

Papers	Input Features	Labels	Classifier	Average F1-Score
Alazzam et al. [2]	Bug Summary Product Resolution No. of comments	P1 P2 P3	Random Forest and SVM	86.23%
Guo et al. [7]	Bug Summary	Severe Non-Severe	SVM	88%
Ahmed et al. [1]	Bug Summary	P1 P2 P3 P4	Random Forest	88.4%
Izadi et al. [10]	Bug Title Bug Description Sentiment Analysis	High Priority Low Priority	Random Forest	74.5%

all other baselines, achieving F1-scores of 72% for high-priority and 77% for low-priority classes. In comparison, Alazzam et al. [2] explored a range of algorithms, including RF, KNN, SVM, Decision Tree (DT), and Convolutional Neural Networks (CNN) - a Deep Learning model. They found that RF demonstrated superior performance, yielding F1-scores of 73.2%, 86.8%, and 94.1% for priority classes P1, P2, and P3, respectively. Similarly, Guo et al. [7] experimented with Naive Bayes, MNB, SVM, KNN, DT, and Random Tree. Among these, SVM was reported to outperform all other models, achieving an F1-score of 88%. Ahmed et al. [1] employed NB, DT, RF, and LG in their experiments, with RF yielding the best overall F1-score of 88.47%.

4 Discussion

4.1 Objective-based classification

Upon comparing the findings from each of the papers, it becomes evident that Bhardwaj and Kadam [3] achieved the highest F1-scores for the Bug (89%) and Enhancement (88%) labels. However, their F1-score for the Question label (61%) was notably lower in comparison to Kallis et al.'s study [11], where they attained an F1-score of 82.5%. The discrepancy in the F1-scores for the Question label can be attributed to the class imbalance present in Kallis et al.'s research [11], as their dataset was meticulously formulated to encompass 10,000 records for each class. In contrast, Bhardwaj and Kadam [3] indicated that their dataset represented real-world scenarios more accurately, implying a lack of such balanced composition.

When juxtaposed with the aforementioned papers, Izadi et al. [10] reported scores that fell between those compared earlier for the Bug (85%) and Enhancement (84%) labels. The relatively lower scores reported by Izadi et al. [10] in comparison to Bhardwaj and Kadam [3] can be attributed to the features used in training their models. Both papers [10] and [3]

employed the text from issue titles and bodies; however, Bhardwaj and Kadam [3] further incorporated a range of non-textual features alongside the textual data. This additional feature augmentation might have contributed to their improved performance, despite both papers selecting RoBERTa as the best-performing model.

In regard to the labels Support/Documentation and Question, direct comparisons between the papers and their reported results cannot be drawn. However, Izadi et al. [10] also conducted experiments with TicketTagger [11] proposed by Kallis et al. using their dataset and reported lower results across all labels. Specifically, TicketTagger [11] yielded F1-scores of 80%, 78%, and 54% for the Bug, Enhancement, and Support/Documentation labels, respectively.

4.2 Priority-based classification

While acknowledging that direct comparisons of F1-scores among the four papers may not be definitive due to variations in datasets and classification classes, a notable insight emerges from their experiments: the exceptional performance of Random Forest and Support Vector Machine (SVM) in addressing bug classification based on bug priority. Although the papers employed diverse Natural Language Processing (NLP) techniques that are beyond the scope of this paper's elaboration, it is possible that these techniques influenced the reported results. Nonetheless, despite the divergence in NLP strategies, the machine learning (ML) models consistently exhibit a robust pattern in tackling this problem.

5 Conclusion

In conclusion, software bugs play a crucial role in the software engineering lifecycle, and their effective management can significantly impact the overall cost and success of software development projects. Bug-tracking systems, such as GitHub's Issues, are widely utilized to monitor and resolve bugs. However, the process of labeling and categorizing issues can be time-consuming and require substantial knowledge about the project and the issues.

This paper aimed to compare various solutions proposed by researchers in the context of bug classification, particularly focusing on objective-based classification and priority-based classification. Izadi et al. [10] conducted research to develop automatic bug classifiers for these two problem statements, and their work served as the focal point of comparison.

The review revealed that issue labeling was not consistently practiced by developers, with a significant portion of repositories lacking labeled issues. Incorporating issue labeling, as demonstrated by related studies, can enhance project management and user engagement rates. Misclassified bug reports were also identified as a prevalent concern, underlining the importance of accurate issue labeling to streamline the resolution process and optimize bug prediction models.

Regarding objective-based classification, the comparison highlighted Bhardwaj and Kadam [3] as achieving the highest F1-scores for Bug and Enhancement labels. However, their performance on the Question label was lower than that of Kallis et al. [11], which could be attributed to class imbalance in the datasets. Izadi et al. [10] reported F1-scores that fell between the two compared papers, with feature selection influencing the results.

For priority-based classification, Random Forest and SVM consistently stood out as top-performing algorithms, regardless of the NLP techniques employed. The comparison emphasized the effectiveness of these classifiers in addressing bug priority classification.

In conclusion, this research sheds light on the strategies and datasets utilized by researchers in bug classification-related studies. It provides valuable insights into the performance of various classifiers and the significance of accurate issue labeling. The paper serves as a comprehensive reference for bug classification-related predicaments, offering implications for future research in this domain.

A Questions and Answers

In this section, I will be addressing the questions or reviews given to me based on the presentation on the same paper.

1. Q: What criteria was used to gather the data used in the paper described in slide 7? Did they search for certain keywords, date range, particular issues, etc. Was there a particular reason that the priority data was gathered from only 70 repositories unlike the dataset for objectives?

   Answer: For objective-based classification dataset, Izadi et al. [10] gathered closed issues from GitHub's open-source repositories that predominantly employed Java as their primary programming language. According to Izadi et al. [10], the selection of Java was motivated by its extensive utilization in prior research, and it was also employed strategically to manage the volume of retrieved issues. This approach ensured a focused dataset while maintaining consistency with earlier studies. Additionally, the issues were gathered up to April 2021 to ensure relevance.

   On the other hand, for priority-based classification dataset, they only collected issues associated with at least one priority-related labels falling under either "blocker-priority," "critical-priority," "high-priority," or "low-priority." They aggregated issues tagged with "blocker," "critical," and "high-priority" labels into a single grouping of "high-priority". Conversely, the remaining issues were categorized under the "low-priority" grouping. In total, their dataset encompassed 47 synonyms that pertained to these two priority categories, namely "High" and "Low" priority.

   The datasets curation process seems to indicate a bottom-up approach such that they gathered issues that fell into their criteria instead of going for selective and fixed number of repositories.

2. Q: Also, how was RoBERTa "fine-tuned"? Why was Random Forest used for priority detection instead of other ML models?

   Answer: The paper by Izadi et al. [10] explains that they have fine-tuned RoBERTa by feeding the model vectorized textual features involving the issue report title and description and training it further on their specific context dataset with their labels of Bug Report, Enhancement and Support/Documentation.

   The paper by Izadi et al. [10] compared various ML models, K-Nearest Neighbor, Multinomial Naive Bayes, Logistic Regression and Random Forest. I was already comparing the best results obtained from each of the papers in consideration and hence chose to report only the best methods in the research.

3. Q: What was your search criteria for looking for other papers to compare with the main paper that you presented? in other words, on slide 11 and 12, what is the criteria you used to search for and select the first 2 papers on slide 11 and first 3 papers on slide 12.

   For slide 11, a simple Google scholar search shows more papers with similar study with bug title and bug body input and similar labels too. You only compared the main

paper to 2 papers (for objective of issues) but compared main paper to 3 papers (for priority of issues). Is there any reason why?

Answer: The search criteria have been elaborated more in section 2, which was part of a bigger literature review. All rationales, inclusion and exclusion criteria have been discussed in the same section.

4. Q: In the main paper that was presented, given that the second paper on slide 11 exists (it is very similar with the main paper and does better), what was mentioned as the main contribution of the authors?

   Answer: To answer this question it is important to clarify that the paper by Bhardwaj and Kadam [3] was a Workshop paper that was one of the exceptions we included in our inclusion criteria due to its relevance and approach. The report by Bhardwaj and Kadam [3] describes their implementation and project in a competition setting where they are provided with a fixed dataset given to them by the organizers. Therefore, even though we compare their methodologies, it may not be appropriate to compare their results as the dataset used by Bhardwaj and Kadam [3] was not described much in the paper.

5. Q: How will the results presented support your future research? This was not mentioned in the presentation.

   Answer: The results indicate and give hints for the kind of Large Language models and Machine Learning models that have been widely experimented with, and that give the best results in bug classification-related problem scenarios. My research is precisely around the automatic bug classification area. Therefore, this review provided me with a bird's eye perspective on how such classification problems have been tackled so far.

References

[1] Ahmed, H. A., Bawany, N. Z., and Shamsi, J. A. Capbug-a framework for automatic bug categorization and prioritization using nlp and machine learning algorithms. *IEEE Access 9* (2021), 50496–50512.

[2] Alazzam, I., Aleroud, A., Al Latifah, Z., and Karabatis, G. Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. *IEEE Transactions on Computational Social Systems 7*, 5 (2020), 1288–1303.

[3] Bharadwaj, S., and Kadam, T. Github issue classification using bert-style models. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)* (2022), pp. 40–43.

[4] Cabot, J., Izquierdo, J. L. C., Cosentino, V., and Rolandi, B. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (Mar. 2015), IEEE.

[5] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[6] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., and Zhou, M. Codebert: A pre-trained model for programming and natural languages, 2020.

[7] Guo, S., Chen, R., Wei, M., Li, H., and Liu, Y. Ensemble data reduction techniques and multi-rsmote via fuzzy integral for bug report classification. *IEEE Access 6* (2018), 45934–45950.

[8] Herzig, K., Just, S., and Zeller, A. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *2013 35th international conference on software engineering (ICSE)* (2013), IEEE, pp. 392–401.

[9] Huang, Q., Xia, X., Lo, D., and Murphy, G. C. Automating intention mining. *IEEE Transactions on Software Engineering 46*, 10 (Oct. 2020), 1098–1119.

[10] Izadi, M., Akbari, K., and Heydarnoori, A. Predicting the objective and priority of issue reports in a cross project context. *CoRR abs/2012.10951* (2020).

[11] Kallis, R., Di Sorbo, A., Canfora, G., and Panichella, S. Ticket tagger: Machine learning driven issue classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2019), pp. 406–409.

[12] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach, 2019.