International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Bug Severity and Priority Prediction using Machine Learning Techniques

Nithishka G
Department of Artificial Intelligence & DataScience
Rajalakshmi Institute of Technology
Chennai,India
nithishka2004@gmail.com

Priyadarshini A
Department of Artificial Intelligence & DataScience
Rajalakshmi Institute of Technology
Chennai,India
priyaarumug16@gmail.com

Priyadharshini K
Department of Artificial Intelligence & DataScience
Rajalakshmi Institute of Technology
Chennai,India
priyadharshini15022@gmail.com

Ms.Chithralekha T
Department of Artificial Intelligence & DataScience
Rajalakshmi Institute of Technology
Chennai,India
lekhathanigai15@gmail.com

***Abstract*— Manual bug triaging in large software projects is time-consuming, subjective, and difficult to scale as bug reports increase.** [1] **Accurate severity and priority assignment, fix-time estimation, and developer allocation are essential to avoid maintenance delays and missed releases.** [2] **This paper compares two machine learning approaches for automating bug triage using a real-world Mozilla Bugzilla dataset.** [3]**Both methods are evaluated on the same cleaned and balanced data for fair comparison. The first is a hybrid end-to-end system using TF–IDF features, LightGBM models, and semantic similarity to predict priority, estimate fix time, and recommend developers. The second is a simpler and more interpretable pipeline using TF–IDF with SVM and Logistic Regression to predict severity and priority. While the hybrid model achieves stronger overall performance, the lightweight approach offers competitive accuracy with lower complexity and easier deployment, highlighting key trade-offs between performance and practicality.**

Keywords— Bug Triage, Severity Prediction, Priority Prediction, Machine Learning, Software Maintenance

I. INTRODUCTION

Continuous software projects evolve and bugs occur constantly. Tools such as Bugzilla, Jira and GitHub Issues are used to track bugs. [4] As projects continue to develop, it has become increasingly complicating to manage bugs through manual triage because manual bug triage is extremely tedious and time consuming. [5] In addition, determining the level of severity, priority and complexity of each bug along with the appropriate developer to assign the bug to has relied significantly upon the experience of the individual triaging the bugs, and has taken significant amounts of time to complete. We believe that by utilizing machine learning in conjunction with historical bug data, we can automate the majority of the work associated with triaging bugs. [6] In this paper, we compare two methods of performing automated triage on bugs using machine learning. The first method triages multiple different aspects of triaging in one method, while the second method only focuses on triaging severity and priority via very simple models. The goal of this comparison is to illustrate the trade-offs between the complexity and ease of use between the two methods.

II. DATASET DESCRIPTION

A. Source Dataset

The BugRepo dataset that we used for this work comes from the EASE 2025 Dataset Track. You can find the BugRepo dataset on Zenodo. It has a CC BY 4.0 license. The BugRepo dataset is collected from the Mozilla Bugzilla platform. People mainly use the BugRepo dataset for research, on bug analysis and bug triaging. [7] We are looking at the file Bug_meta_data_230k_all.csv for this study. This file has a lot of bug reports around 230,000 of them. The Bug_meta_data_230k_all.csv file has all the details we need like how bad the bug is, how important it is to fix, when it happened, which parts of the system are affected if it is fixed or not and who is in charge of fixing it. We picked the Bug_meta_data_230k_all.csv file because it has clean and easy to understand information that's perfect, for automatically sorting through bug reports.

B. Selected Attributes

The original dataset has a lot of information with 299 attributes. To be honest most of these attributes are not really useful when we are trying to figure out bugs. So we pick the attributes that are really important, for bug triaging. This way the data's easy to understand and we do not have a lot of extra stuff that we do not need. We just focus on the bug triaging and the attributes that are relevant to bug triaging. [8] The things we look at in the end are: the summary of the bug how bad the bug's how important it is to fix the bug, when the bug was first found, when the bug was last worked on how many comments people made about the bug, what part of the system the bug affects the status of the bug how the bug was resolved who the bug was given to and the bugs ID. These fields together give us information that we can use to figure out how important something is, when it needs to be done and what kind of thing it is. We use this information to predict the priority of the fields estimate how long it will take to fix the fields and recommend which developers should work on the fields. The fields do not include details about the people who contribute to them so we do not have any bias, towards certain people when we are working with the fields.

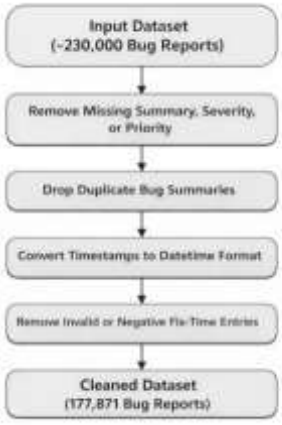

Fig. 1. Data cleaning and filtering workflow applied to the raw dataset.

C. Fix Time Computation

Bug fix time is calculated as the number of days taken to resolve a bug. It is computed as the difference between the last update time and the bug creation time. All timestamps are converted to a standard format before calculation. Bug reports with missing, invalid, or negative fix-time values are removed to ensure reliable results.

III. Data Preparation and Balancing

A. Data Cleaning and Filtering

Once the models have been trained, the first step has already been to clean the data, making operational improvements in both the quality and consistency of the data. Deleted bugs without a summary or without a severity or priority were eliminated; duplicate bug summaries were eliminated; timestamps were converted to a common datetime format and any bug reports without an acceptable fix-time or an invalid or negative fix-time value, were also removed. By cleaning the data, the overall data set has dropped from approximately 230,000 to 177,871 bug reports.

B. Severity and Priority Filter

To keep the dataset consistent, only standard labels will be used, which means severity will only have values of S1 to S4 and priority P1 to P5 will be the only values used for those labels. Any bug report missing those standard severity and/or priority will be deleted. Once this clean-up process was completed, the remaining dataset was left with 60,674 bug reports.

C. Class Imbalance Analysis

After the dataset has been filtered, it is easy to see that the filtered dataset has a clear class imbalance for the severity labels. [9] The number of critical (S1) or high impact (S2) bugs are much smaller than the number of S3 and S4 bugs and this class imbalance is likely to negatively impact model performance because the model will likely favor the majority classes.

D. Strategy for Increasing Data Sets

Data augmentation will only be used on the less frequent severity classes of data (S1 and S2) because that is the right way to address this issue. This will involve using simple alternate forms of words to change some words in sentences, making some sentences different by their context, and replacing certain words with alternate words that are commonly used in software failures. But the majority of the classes of data will stay the same so that there is no loss of balance in the data set. Once all the augmentations are complete, there will be 68,620 bug reports in the final data set which can be used for evaluating all models.

IV. Overview of Methodology

V. Approach A – Multi-task Bug Intelligence Framework

A. Framework Overview

Approach A consists of a complete framework that automates the main tasks associated with triaging bugs. In the initial stages of the process, the framework will receive raw bug summaries as input; from these raw summaries, it will be able to predict the priority of the bug, estimate the typical time required for fixing the bug, and identify the most appropriate developers for fixing the bug. Approach A automates all of these tasks to provide the input needed to support the decision-making process when performing triaging against other events.

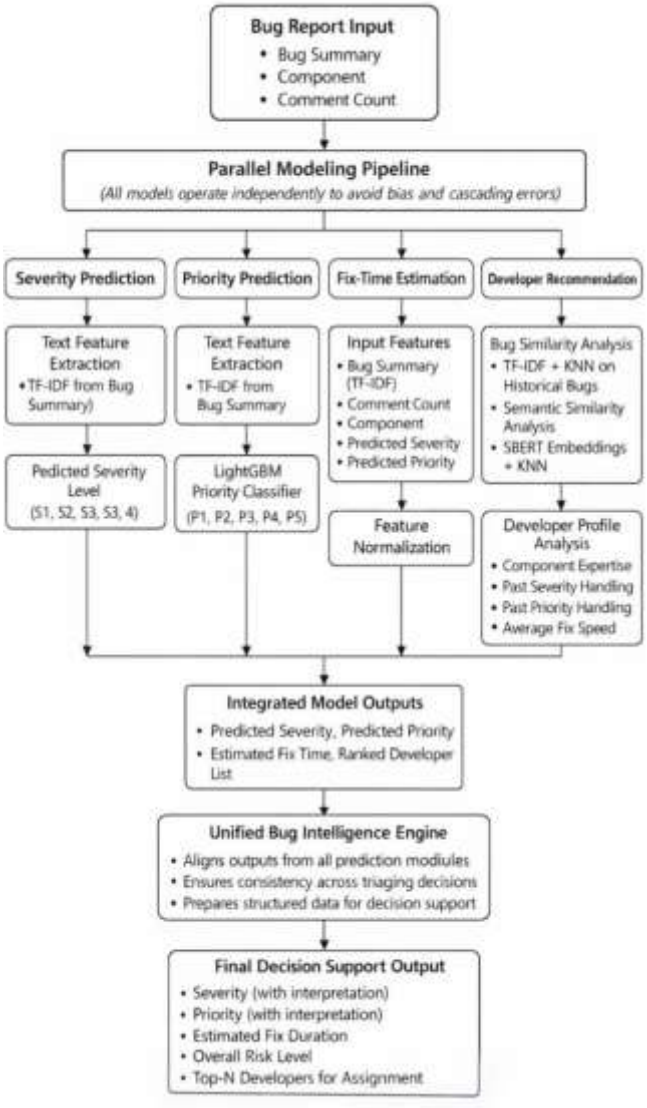

Fig. 2. Overall architecture of the proposed hybrid bug intelligence framework.

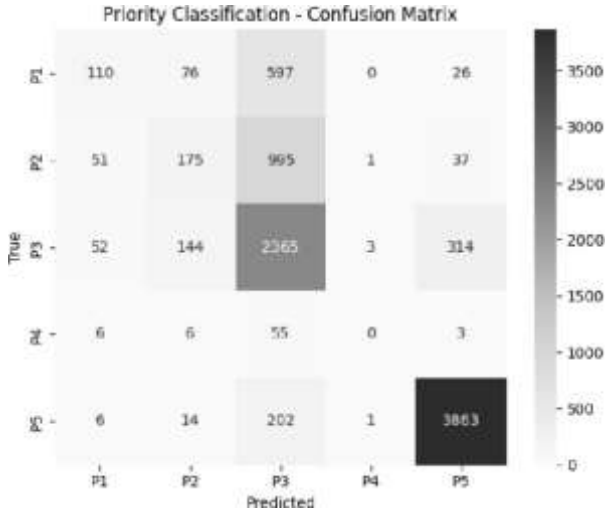

Fig. 3. Confusion matrix for priority classification using the LightGBM model.

B. *Priority Predictions*

Describing the bug summary in TF-IDF form provides a numerical representation of the estimated bug's priority using a LightGBM classifier to predict the bug priority between P1-P5 (5 levels). The LightGBM classifier results in an approximate accuracy rate of 72%, with better performance on mid- to lower-level priority classifications, and lower accuracy at the P1 or P2 classification level due to the inherent ambiguity of assigning urgency based on actual bug behaviour.

C. *Fix-Time Estimation*

Using bug summary text Characteristics in conjunction with structured information, such as severity, priority, number of comments, and component, a regression model based on LightGBM has been created to predict the amount of days it would take to fix each bug. The actual number of days it takes to fix each bug shows a clear overall correlation with the prediction of number of days to fix each bug; however, when delays are very long, prediction is very difficult due to external factors.

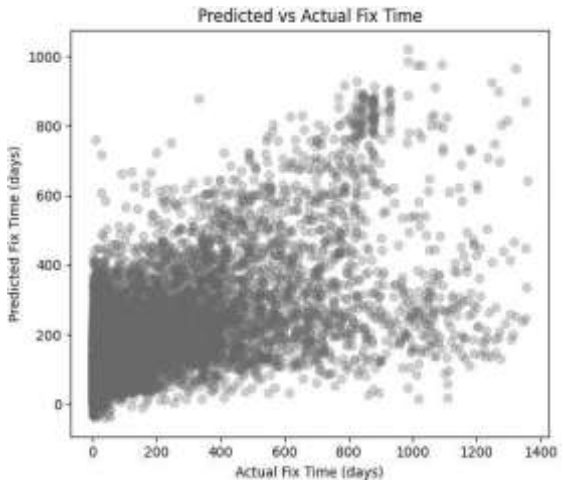

Fig. 4. Predicted versus actual fix-time values for the regression model.

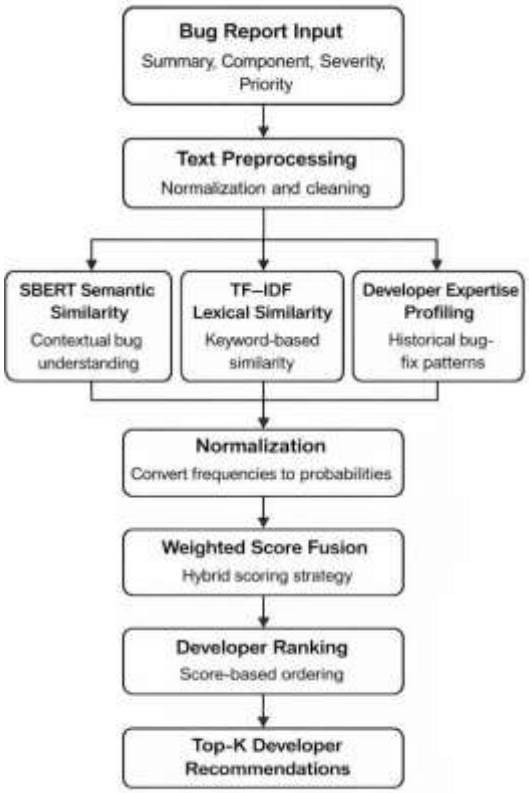

Fig. 5.

D. *Developer Recommendation*

Assigning the 'right' developer is managed through a hybrid approach. [6] The system combines developer's past performance from the assigned and un-assigned bugs, textual similarity between the assigned bug and the un- assigned bug, and semantic similarity between the assigned and un-assigned bugs using sentence embeddings to rank them based on a composite score from the above three measures. The recommendation system's Top-1 accuracy is approximately 50% and Top-3 accuracy is nearly 65%, which is remarkable when the large number of developers is taken into account along with the various real-world limitations.

E. *Performance Summary*

The overall best performance metric was made when estimating severity level with accuracy of approximately 82%, with the majority of the errors being between adjacent severity levels; and when estimating priority, there is enough reliable data to estimate urgency and not actual priority levels. The fix time estimates are based on general patterns in fixing bugs and the developer recommendation, in general, provides helpful recommendations to help triage bugs. The framework provides useful support in the assessment of decisions making in the triaging of bugs at an enterprise level.

1) Bug Severity Prediction:

Estimated at 82% Accuracy using TF-IDF + Linear SVM. The model is also very good at finding Critical and Low Severity Bugs.

2) Bug Priority Prediction:

Estimated at 72% Accuracy using Light GBM Classifier. This Model is also very strong at predicting Priority 3 and 5 Bugs. The model also preserves Urgent Bug Order.

3) Bug Fix Time Estimation:

MAE of 139 Days using the Light GBM Regressor. This Model was able to pick up very Strong Trends in Bug Resolution.

4) Developer Recommendation:

Using TF-IDF, SBERT, and Profiling Models together make this an Effective Hybrid Recommendation. Top 1 ~50% and Top 3 ~65%.

VI. APPROACH B- CLASSICAL MACHINE LEARNING FOR SEVERITY & PRIORITY

A. Overview

Approach B is focused solely on predicting the severity and priority of bugs using well-established machine learning algorithms. Unlike Approach A's multi-task methodology, Approach B provides clarity, speed, and straightforward deployment with existing systems. The process begins by cleaning the bug summaries and then utilizing TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction to generate separate predictions for both severity and priority.

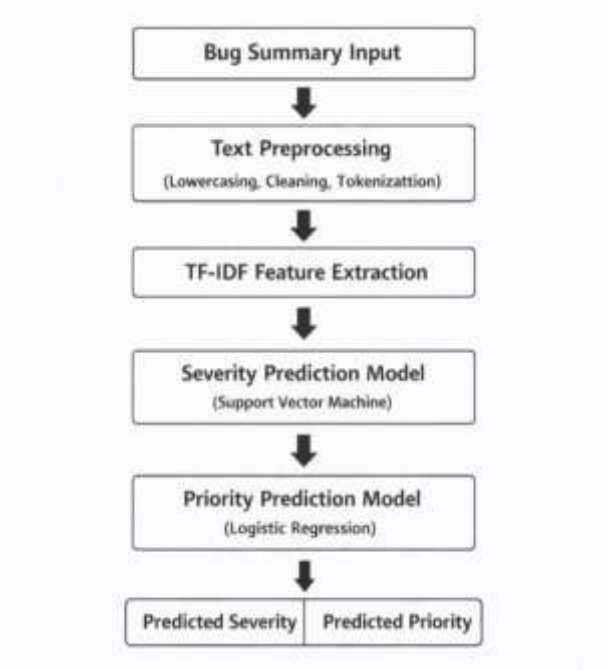

Fig. 6. Workflow of Approach B.

B. Text Processing and Features

The bug summaries are processed by doing the following: converting text to lower case; eliminating null values; removing superfluous special characters; and eliminating extra spaces. TF-IDF (unigrams and bigrams) is used to convert text into numerical feature vectors. [10] This method was selected because it provides a lightweight approach that is easy to interpret and works well for textual information about bugs.

C. Predicting Severity

The severity prediction will utilize SVM specifically. [3] To minimize confusion with closely-related severity levels, the severity levels are collected into a smaller set of groups (e.g., major, minor). In terms of separating major and minor severity levels, the method used to separate the two will work very well. The majority of misclassifications that occur will occur between two neighbouring severity levels. [5]

D. Predicting Priority

For predicting priority, we applied logistic regression to make predictions. [9] Priority levels were grouped into high, medium, and low in order to simplify the classification process. The model provided strong signals for whether a bug summary indicates an urgent or a non-urgent bug.

E. Implementation of the System

To implement the system in practice, we created a simple web interface using Streamlit that enables users to input a bug summary and to return both severity and priority predictions within seconds. The system can operate using minimal hardware and infrastructure requirements.

F. Discussion Limitations

Results indicate that using classical machine learning techniques and simple text features for bug classification remain effective. Although our approach did not account for advanced bug classification tasks such as estimating the time to fix or recommending developers, the simplicity, transparency, and success of our approach supports real-world use. Future improvements may include using longer bug descriptions, adding comments as features, and developing the system to include recommendations to developers.

VII. PERFORMANCE SUMMARY

From a system perspective, Approach A performs better than all other approaches in all areas of the proposed framework. For bug severity prediction, the use of TF-IDF feature extraction in combination with a linear support vector machine yields an approximate accuracy of 82%, with an excellent ability to discriminate between: Critical versus Low Severity Bugs. For bug priority prediction, the LightGBM classifier provides approximately 72% accuracy with excellent performance on mid and low priority classes (specifically P3 and P5), while maintaining the relative urgency order between priority levels. The use of a LightGBM regression model for estimating fix time, with a mean absolute error of approximately 139 days, illustrates that the model appropriately captures general trends in the duration of bug resolution despite the variability among

individual cases. For developer recommendation, the hybrid of TF-IDF-based lexical similarity, SBERT-based Semantic Similarity, and Historical Developer Profiling provides a Top-1 accuracy of approximately 50%, and a Top-3 accuracy of approximately 65%, demonstrating that the combination of: textual similarity, and expertise modeling, provides effective and practical recommendations for developer assignment.

VIII. Discourse

The present manuscript provides a comparative analysis of approaches towards performing automated bug triage and highlights the complexities of each methodology from a practical standpoint. [1] [2] Furthermore, both systems were evaluated utilising the same dataset in order to facilitate drawing conclusions regarding their comparative efficacy; specifically in relation to how much complexity each solution offers and ease of use for practitioners. From the results obtained from analysing the implementation of Approach A's multi-task framework, it is clear that a significant portion of the triaging process can be automated within a unified framework that is capable of automating many aspects of bug triage (e.g., predicting priority, estimating time to resolve, recommending developers). [4] [8] Therefore, Approach A would be advantageous for larger scale software projects as manual triaging would be impractical due to the number of bugs being reported. Conversely, Approach B provides a solution that adheres closely to the principle of "simplicity and transparency." As it uses classical machine learning techniques and bug summary data only, it delivers accurate predictions regarding both severity and priority at a low computational expense. Thus, practitioners seeking an easy to deploy, clearly defined model behaviour solution would benefit from using Approach B.

Overall, the conclusions that can be drawn from the results presented in this manuscript are that the suitability of either of these approaches is dictated by (1) the size of the software project, (2) the resource constraints of the project team, and (3) the operational needs of the project team. Consequently, the two methodologies should not be viewed as competing approaches but rather as complementary alternative solutions.

IX. Caveats to Validity

The present study contains limitations. Specifically, the dataset is based upon people-assigned labels which could have implicit subjectivity associated with the assignment of labels to the data. With regard to Fix Time, external factors (i.e., delayed or increased workloads) will also impact the Fix Time chosen to represent the true workload put forth in order to resolve defects. Finally, Bugzilla is strictly the primary bug-tracking system used in this evaluation, so the findings here may not be applicable or generalizable to other bug-tracking systems.

X. Conclusion

The completion of this analysis has led us to a conclusion regarding comparisons of two separate approaches (Machine Learning) within an automated bug triage system using a large set of real-world data. Overall, our findings indicate both approaches are capable of effective automated bug triage; [3] [9] however, depending upon individual project requirements & characteristics, either approach could be more appropriate to use than the other.

The hybrid approach is more likely to lead to a broader level of automation for larger projects, whereas the classical method provides a highly efficient, lightweight & more easily deployable solution for smaller projects.

In the future additional research can be completed by utilizing bug comments, testing both approaches on multiple platforms and continuing research on model adaptation for defects as software projects evolve. [5] [7]

References

[1] Y. Zhang, X. Li, and Y. Zhou, “SevPredict: Exploring large language models for bug severity prediction,” in *Proc. IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, 2024.

[2] H. Wang, Z. Chen, and Y. Liu, “Automatic bug triage using pre-trained language models,” in *Proc. ACM/IEEE Int. Conf. Software Engineering (ICSE)*, 2024.

[3] Y. Sui, Y. Wu, and H. Zhang, “Multi-task learning for bug severity and priority prediction,” *Journal of Systems and Software*, 2024.

[4] M. Ahmed, M. Rahman, and A. E. Hassan, “Leveraging machine learning for predictive bug analysis,” in *Proc. IEEE Int. Conf. Software Analysis, Evolution and Reengineering (SANER)*, 2024.

[5] J. Liu, B. Xu, and S. Kim, “An empirical study on bug report classification using transformer models,” in *Proc. IEEE Int. Conf. Software Maintenance and Evolution (ICSME)*, 2024.

[6] R. Patel, D. Shah, and K. Mehta, “Improving bug triage with data reduction and deep learning techniques,” in *Proc. Int. Conf. Evaluation and Assessment in Software Engineering (EASE)*, 2025.

[7] Q. Zhou, S. Tang, and T. Chen, “Ticket-level bug prediction using temporal features,” in *Proc. ACM/IEEE Int. Conf. Software Engineering (ICSE)*, 2025.

[8] T. T. Nguyen, N. H. Pham, and A. T. Nguyen, “Large language models for automated bug report analysis,” in *Proc. ACM Symp. Foundations of Software Engineering (FSE)*, 2025.

[9] S. Kumar, P. Verma, and R. Singh, “Predicting bug severity and priority using ensemble learning,” *IEEE Access*, 2025.

[10] F. Alqahtani, A. Alshamrani, and N. Almutairi, “Automated bug triage and severity prediction using hybrid deep learning models,” *Journal of Systems and Software*, 2026.