Identifying Bug Types & Severity in Open-Source Code

Ishir Bhardwaj 2022223 | Manit Kaushik 2022277 | Pranav Gupta 2022364 | Raghav Wadhwa 2022385

1. Mid-Semester Progress Report

This report provides an update on the group project for the CSE363 Machine Learning Course, taught by Prof. Jainendra Shukla. Since the initial project proposal, we have made significant progress in dataset finalization, feature engineering, and model development.

2. Motivation

As engineering students, we frequently work on software projects where managing bugs is a crucial part of the development cycle. Manually categorizing bugs by severity and type can be time-consuming and prone to error, especially in large-scale projects. This project aims to automate the process using machine learning, improving the efficiency of bug triage, reducing manual intervention, and ensuring timely resolution of critical issues. The idea originated from the recurring need for faster bug handling in software development workflows.

3. Problem Statement

This project seeks to develop a machine learning-based system to predict both the severity and type of software bugs given in a bug report in a supervised learning context. Additionally, it aims to explore unsupervised learning methods to categorize bugs into specific types like memory, security, GUI etc. This will automate bug classification, helping teams prioritize and address issues more effectively in software projects.

4. Literature Review

Our project focuses on primarily two key aspects: identifying the type of bug and predicting its severity, encompassing both classification and regression tasks. For our literature review, we examined papers that addressed these two problems individually.

4.1. Not all bugs are the same: Understanding, characterizing, and classifying bug types

- **Goal**: Develop a taxonomy of bug types and create an automated model to classify bugs based on this taxonomy.
- **Dataset**: Manually classifying 1280 bug reports of 119 software projects belonging to ecosystems such as Mozilla, Apache and Eclipse.
- **Features**: Extracted textual descriptions from bug reports, including details such as error messages, file names, and system components. Then TF-IDF was used to identify relevant terms.
- **Method**: Logistic Regression classifier with TF-IDF features derived from bug report summaries.
- **Result**: Identified 9 bug types; model achieved 64% F-Measure and 74% AUC-ROC.

4.2. Machine Learning Approaches for Predicting the Severity Level of Software Bug Reports in Closed Source Projects

- **Goal**: Build prediction models to determine the class of severity (severe or non-severe) of reported bugs.
- **Dataset**: Bug reports extracted from the JIRA bug tracking system used by INTIX company, containing bug IDs and descriptions.
- **Features**: The bug report is transformed into a feature vector (Bag-of-Words) using Tokenization, Stop-word removal, and Stemming.
- **Method**: Naive Bayes, Naive Bayes Multinomial, Support Vector Machine (SVM), Decision Tree (J48), RandomForest, Logistic Model Trees (LMT), Decision Rules (JRip), and KNN.
- **Result**: LMT algorithms reported the best performance results with Accuracy = 86.31, AUC = 0.90, and F-measure = 0.91.

5. Dataset

Our project delves into the problem of identifying the type of bug and the severity of the bug reported. For our project, we have targeted datasets which contain bug reports of different open-source projects, softwares and bug-reporting systems.

5.1. Dataset for Predicting Bugs Severity

The dataset used for this part is the **Mozilla and Eclipse Defect Tracking Dataset**. These bug reports from Eclipse & Mozilla include a 'severity' field, which will be used as the target classes. There are 5 severity classes -**'Blocker', 'Critical', 'Major', 'Minor' & 'Trivial'**. Number of samples in the Mozilla dataset are 45461 & in the Eclipse dataset are 9664.

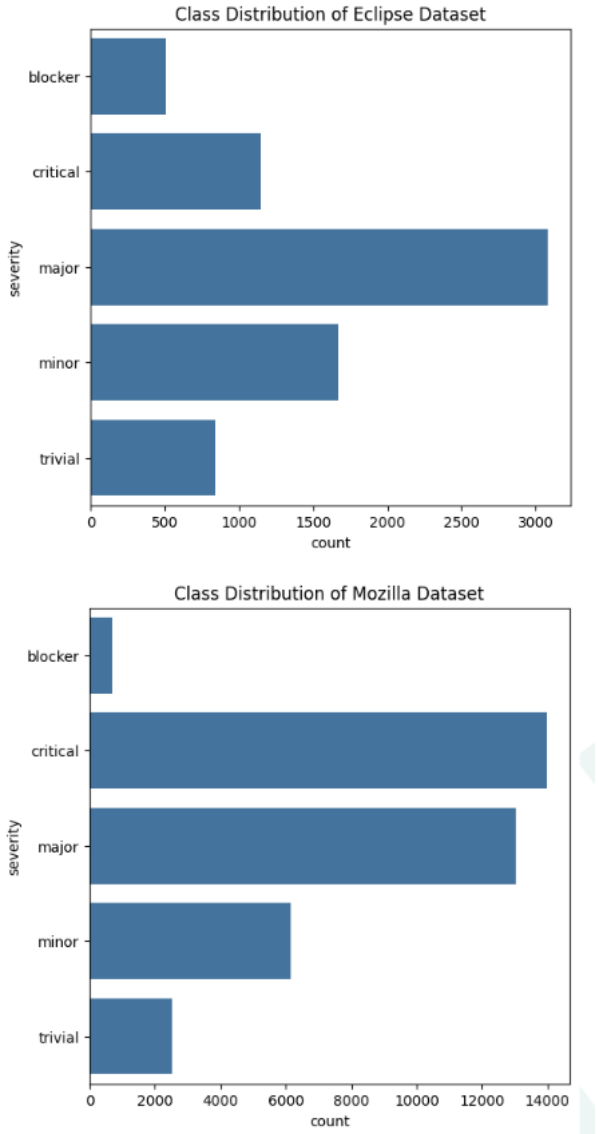

Figure 1. Sample Distribution of Bug Severity in Mozilla and Eclipse Datasets

5.2. Dataset for Supervised Bugs Type Prediction

For the task of predicting bug types, we collected issue reports generated by the **JIRA bug tracking system** from various open-source projects. These issue reports contain multiple categories for 'Bug Type.' For our project, we grouped these categories into broader classes and used these as our target labels. The dataset consists of 609,697 records with 18 features.

Table 1. Classification of Bug Types

Combined Class Name	**Subclasses**	**Number of Samples**
Defect	'Bug', 'Defect', 'Patch', 'Test', 'Clarification', 'Quality Risk', 'Support Patch', 'Backport', 'TCK Challenge', 'CTS Challenge'	382,411
Improvement	'Improvement', 'New Feature', 'Feature Request', 'Enhancement', 'Wish', 'Component Upgrade', 'Dependency upgrade', 'Remove Feature', 'Library Upgrade', 'Component Upgrade', 'Refactoring', 'Release'	216,161
Task	'Task', 'Sub-task', 'Story', 'Documentation', 'Question', 'Brainstorming', 'Technical task', 'Umbrella', 'Suitable Name Search', 'Tracker', 'Epic', 'Requirement', 'New JIRA Project', 'Temp', 'RTC', 'Pruning'	102,343

5.3. Dataset for Unsupervised Bugs Type Prediction

As the next step of our project, we will attempt to predict the bug field (e.g., memory, security, GUI) mentioned in the generated bug reports. Since the exact bug field is not available unless manually classified, we have adopted an unsupervised approach, with labeling applied only to the test portion of the dataset (done by LLM/manually). We have selected the **DeepTriage dataset** for this purpose, which has 383,104 bug reports from **Google Chromium**, 314,388 bug reports from **Mozilla Core**, and 162,307 bug reports from **Mozilla Firefox** each including the bug ID, title, and description.

6. Data Pre-processing

In this section, we outline the **Natural Language Processing (NLP)** preprocessing techniques applied to prepare the dataset for model training. This is crucial in converting raw text into a structured format. The following steps were used:

6.1. Tokenization:

This process involves breaking down text into smaller units, typically words or phrases, known as tokens. Tokenization helps in analyzing each word individually.

6.2. Stop Words Removal:

Common words such as "the", "and", "is" that occur frequently in the language but add little to no contextual meaning are removed. This helps in reducing noise and focusing the analysis on more meaningful words.

6.3. Lemmatization:

Words are converted to their base or root form, known as a lemma. For example, words like "running" or "ran" are reduced to "run." This normalization of words ensures that variations of the same word are treated as a single entity.

Example: "Watching the same user twice produces a SQL error" changes to "Watch user twice produce SQL error."

7. Methodology

7.1. Methodology for Predicting Bugs Severity

After preprocessing the data, **Named Entity Recognition (NER)** is employed to identify key entities. This step enriches the feature representation by allowing the model to focus on significant elements that may correlate with bug severity. Eg. for `[1.5][compiler] 3.4RC4 java compiler fails to correctly erase generic type information`, NER results in the following entities being extracted: `([’1.5][compiler’, ’CARDINAL’), (’3.4RC4’, ’CARDINAL’)`

The machine learning pipeline begins with **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization. TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It captures both unigrams (individual words) and bigrams (two-word combinations) to account for both single words and meaningful word pairs. The formula for TF-IDF is given by:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}; \text{IDF}(t) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

In the above, $f_{t,d}$ is the frequency of term t in document d, N is the total number of documents in the corpus, $|\{d \in D : t \in d\}|$ is the number of documents that contain the term t.

Following TF-IDF vectorization, **Latent Semantic Analysis (LSA)** is applied using **Truncated Singular Value Decomposition (SVD)** to reduce the feature space to 200 components. This dimensionality reduction minimizes noise and complexity in the dataset while preserving essential information. The refined feature vectors produced from these processes serve as input for different machine learning models.

7.2. Methodology for Supervised Bugs Type Prediction

To streamline the classification task, we combined related categories into broader **superclasses**, resulting in three main classes: **Defect**, **Improvement**, and **Task**. This aggregation simplifies the classification process while maintaining relevant distinctions between different bug types. However, the process of combining categories into superclasses introduced a significant **class imbalance** problem which can lead to biased models. To counteract this, we calculated and applied **class weights** during model training.

Moreover, incorporating **Term Frequency-Inverse Document Frequency (TF-IDF)** vectorization enables the model to capture meaningful text features from the bug descriptions. By representing the bug reports with both unigrams (individual words) and bigrams (pairs of words), the model gains deeper contextual understanding of the bug descriptions.

Following the vectorization step, we perform **feature selection** using the chi-squared (χ^2) statistic. This statistical method evaluates the independence of each feature in relation to the target class, allowing us to identify and retain only the most informative features, this reduces the feature space to 2000 from 3611 dimensions.

8. Results

Currently both of the created feature sets have only been tested with the **Multinomial Logistic Regression** model.

8.1. Results for Predicting Bugs Severity

Class	Precision	Recall	F1-Score
blocker	0.58	0.04	0.08
critical	0.35	0.14	0.20
major	0.50	0.82	0.62
minor	0.50	0.36	0.42
trivial	0.50	0.30	0.39
Accuracy	0.49		

Table 2. Performance Metrics for Eclipse Dataset

The overall accuracy for the Eclipse dataset is 0.49, reflecting that the model's performance is average and the overall accuracy for the Mozilla dataset is 0.60, a noticeable improvement over the Eclipse dataset because of more samples. Class imbalance in both datasets directly affects F1-scores. Handling the imbalance marginally improves F1-scores for minority classes but this reduces overall accuracy to 0.49 due to mis-classification of majority classes.

Class	Precision	Recall	F1-Score
blocker	0.60	0.18	0.28
critical	0.78	0.72	0.75
major	0.51	0.78	0.61
minor	0.45	0.20	0.27
trivial	0.45	0.09	0.15
Accuracy	0.60		

Table 3. Performance Metrics for Mozilla Dataset

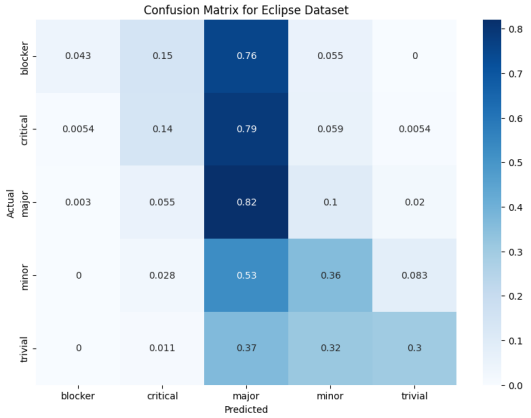

Figure 2. Sample Distribution of Bug Severity in Eclipse Dataset

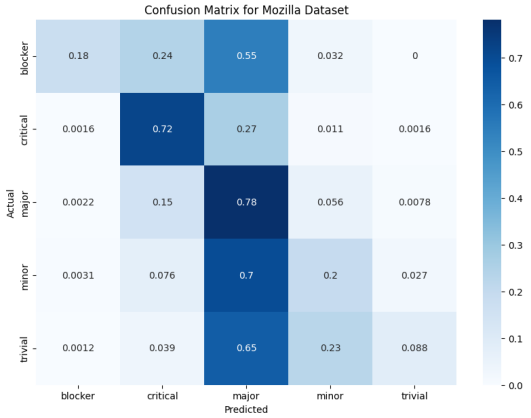

Figure 3. Sample Distribution of Bug Severity in Mozilla Dataset

The confusion matrices for the Eclipse and Mozilla datasets highlight the model's tendency to over-predict the major class, while struggling to accurately classify minority classes like blocker, minor, and trivial

8.2. Results for Supervised Bugs Type Prediction

Class	Precision	Recall	F1-Score
defect	0.86	0.74	0.79
improvement	0.64	0.59	0.61
task	0.32	0.62	0.42
Accuracy	0.68		

Table 4. Performance Metrics for JIRA Dataset

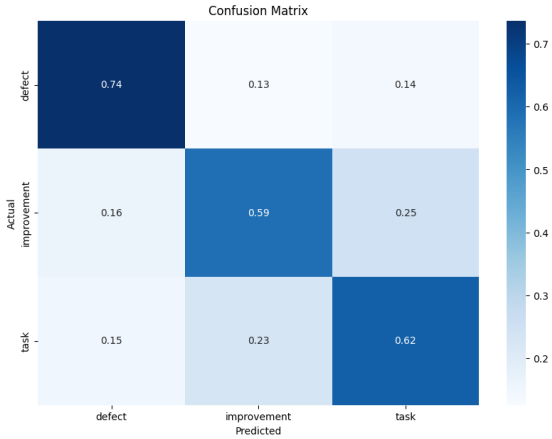

Figure 4. Sample Distribution of Bug Severity in JIRA Dataset

The model excels in predicting the "defect" class, achieving a high F1-score. In contrast, its performance for the "improvement" and "task" classes is moderate. The confusion matrix indicates strong recall for defects at 0.74 but reveals significant misclassification between "improvement" and "task" bugs.

9. Learnings

We learned to automate software bug classification by type and severity using machine learning. We also realized the significance of feature engineering and NLP techniques in enhancing bug triage efficiency.

10. Timeline

- **Week 1:** Finalize preprocessing and pipeline for JIRA severity prediction & evaluate all feature sets across ML paradigms.
- **Week 2:** Develop an unsupervised bug type feature set and create a labeled testing set.
- **Week 3:** Assess the labeled set with various ML algorithms and refine predictions.
- **Week 4:** Compile results, create a report, and prepare a summary presentation (PPT).

11. Contributions So Far

- **Ishir Bhardwaj**: Finalized dataset, Designed ML pipeline, tested using MLR, analyzed results.
- **Manit Kaushik**: Finalized dataset, Designed ML pipeline, tested using MLR, analyzed results.
- **Pranav Gupta**: Conducted literature review, preprocessing and cleaning the data, created presentation and report.
- **Raghav Wadhwa**: Conducted literature review, preprocessing and cleaning the data, created presentation and report.

12. References

[1] A. Baarah, A. Al-oqaily, Z. Salah, M. Sallam, & M. Al-qaisy. (2019), Machine Learning Approaches for Predicting the Severity Level of Software Bug Reports in Closed Source Projects, IJACSA.

[2] Tan, Y., Xu, S., Wang, Z., Zhang, T., Xu, Z., & Luo, X. (2020). Bug Severity Prediction Using Question-and-Answer Pairs from Stack Overflow. Journal of Systems and Software, 110567.

[3] Catolino, G., Palomba, F., Zaidman, A.,& Ferrucci, F. (2019). Not All Bugs Are the Same:Understanding, Characterizing, and Classifying Bug Types. Journal of Systems and Software.

[4] Senthil Mani, Anush Sankaran, Rahul Aralikatte, (IBM Research, India). DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging.

[5] Lamkanfi, A., Perez, J., & Demeyer, S. (2013). The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information. Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)

[6] Ahmed, H. A., Bawany, N. Z., & Shamsi, J. A. (n.d.). CaPBug: A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms.