Identifying Bug Types & Severity in Open-Source Code

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

1. Abstract

This report presents the outcomes of the CSE363 Machine Learning group project, guided by Prof. Jainendra Shukla. The project develops a machine learning system to predict bug severity and classify issue types, automating bug management to address challenges in large projects and enhance workflow efficiency through supervised and unsupervised learning methods. Github repository for this project: Link

2. Introduction

The problem addressed in this project is the manual and time-consuming process of bug severity prediction and issue classification in software development. By using machine learning, the goal is to predict both the severity and issue type of software issues given in a bug report in a supervised learning context. Additionally, it aims to explore unsupervised learning methods to categorize the bug domains like memory, security, GUI etc.

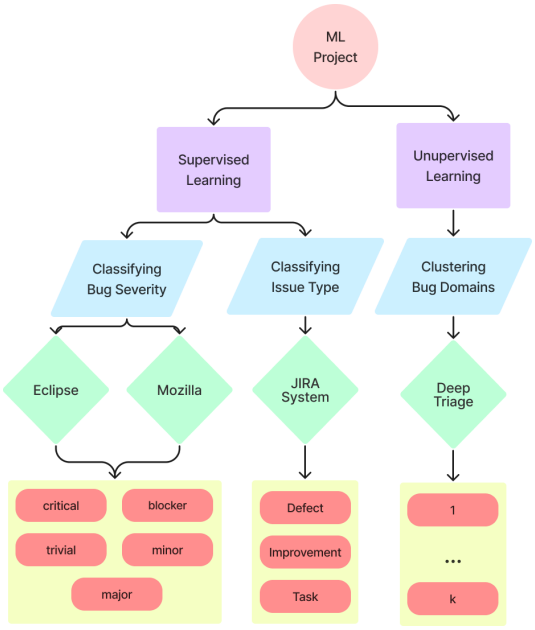

Figure 1. Project Flowchart

3. Literature Review

3.1. 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.

3.2. 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.

4. Supervised Learning

This section has 2 tasks namely, Classifying Bugs Severity & Classifying Issue Type.

4.1. Dataset & EDA for Classifying Bugs Severity

The bug reports from **Eclipse & Mozilla** include a severity field, which are 5 types - **blocker, critical, major, minor & trivial**. Each bug report also has ID and a description of the issue.

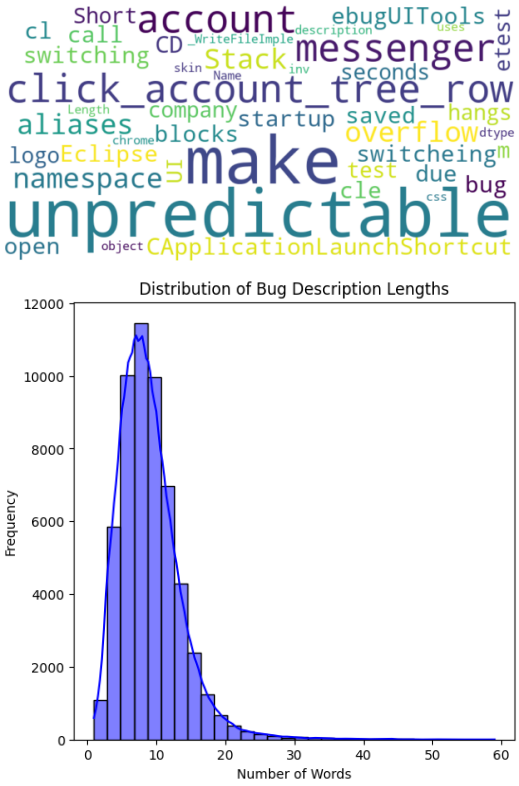

Figure 2. EDA Graphs for Bug Severity Dataset

4.2. Dataset & EDA for Classifying Issue Type

The JIRA bug tracking system has issue reports contain multiple categories for 'Bug Type'. For our project, we grouped these categories into broader issues i.e. **Defect, Improvement & Task**.

4.3. Data Pre-processing for Supervised Learning

The NLP preprocessing steps applied to both datasets included **tokenization** to split text into individual words or tokens, **lowercasing** to ensure uniformity, and the **removal of stop words** like "the" and "is" to reduce noise. Additionally, **removal of non-alphabetic characters** and **lemmatization** was performed to reduce words to their base forms, such as converting "running" to "run".

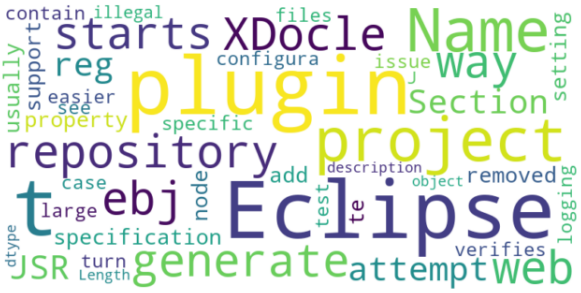

Figure 3. EDA Graphs for Issue Type Dataset

Example: "Stack overflow with namespace aliases" changes to "Stack overflow namespace alias."

4.4. Methodology for Predicting Bugs Severity

After preprocessing, features are extracted using **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization, which measures a word's importance within a document relative to the corpus. We also included unigrams and bigrams in TF-IDF vectorization to capture single words and meaningful word pairs.

After TF-IDF vectorization, **Latent Semantic Analysis (LSA)** is performed using **Truncated Singular Value Decomposition (SVD)** to reduce the feature space to 1000 components, minimizing noise while preserving essential information. The refined features are used as input for various machine learning models, including a multinomial **Logistic Regression** with the lbfgs solver and 1000 maximum iterations, a **Decision Tree** and **Random Forest**, both with a maximum depth of 100, and a **Multilayer Perceptron** with two hidden layers (100 and 50 units), 1000 maximum iterations, tanh activation, and an sgd solver. Additionally, an **Ensemble Voting** classifier combines predictions from all models through majority voting.

4.5. Analysis & Conclusion for Predicting Bugs Severity

The results indicate that Logistic Regression and Ensemble Voting both achieved the highest accuracy of 0.60. Decision Tree performed the worst with an accuracy of 0.44, while Random Forest and Multilayer Perceptron (MLP) performed similarly with accuracies of 0.57 and 0.59, respectively. In terms of the weighted F1-score, MLP showed strong performance with a score of 0.58, outperforming the other models. The macro F1-scores were generally lower, with the ensemble model achieving a score of 0.44, reflecting the overall balance in model performance.

The performance of class prediction for major and minor classes is relatively strong, while the prediction for the blocker class remains below average across all models. The Multilayer Perceptron (MLP) model achieves a notably higher F1-score across all classes when compared to

Table 1. Performance Metrics for Models

Model	F1-Score		Accuracy
	Macro	Weighted	
Logistic Regression	0.44	0.57	0.60
Decision Tree	0.34	0.44	0.44
Random Forest	0.39	0.53	0.57
Multilayer Perceptron	0.48	0.58	0.59
Ensemble Voting	0.44	0.56	0.60

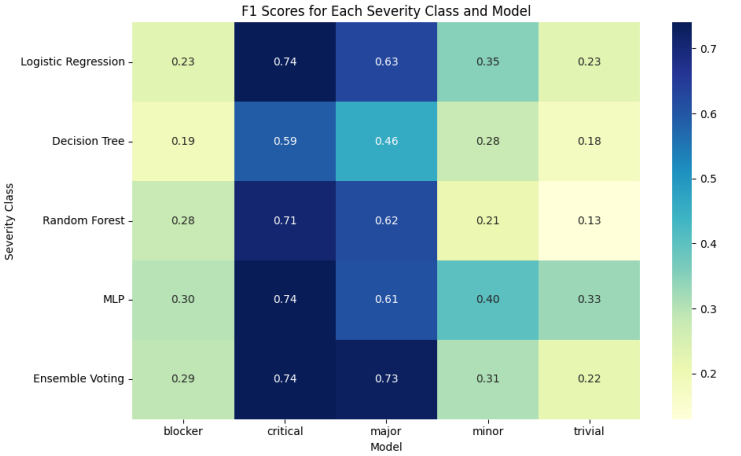

Figure 4. Per Class Heatmap for Severity Classification

the other models, demonstrating a more consistent performance.

4.6. Analysis & Conclusion for Classifying Issue Type

The overall performance metrics, including accuracy, macro F1-score, and weighted F1-score, highlight the strong performance of Logistic Regression and MLP, both achieving high macro and weighted F1-scores of 0.60 and 0.71 and accuracies of 0.72 and 0.73, respectively. Majority Voting showed moderate performance with a weighted F1-score of 0.68, while Decision Tree and Random Forest performed lower, particularly the Decision Tree, which struggled with class balance.

Table 2. Performance Metrics for Models

Model	F1-Score		Accuracy
	Weighted	Macro	
Logistic Regression	0.72	0.60	0.71
Decision Tree	0.58	0.48	0.58
Random Forest	0.68	0.50	0.53
Multilayer Perceptron	0.73	0.60	0.71
Ensemble Voting	0.71	0.56	0.68

Performance of all models is decently well for the defect

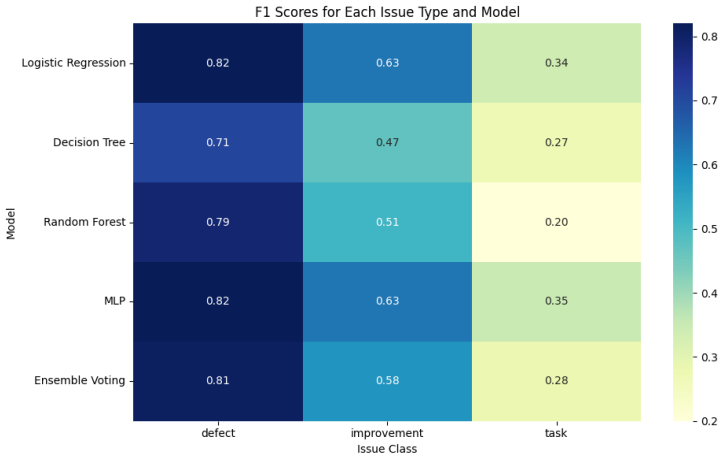

Figure 5. Per Class Heatmap for Issue Type Classification

class but below average for the task class. The improvement class lies in the middle, with moderate F1-scores, suggesting partial success in distinguishing these instances but leaving room for better class-specific predictions.

5. Unsupervised Learning

In this section, we try to cluster the bug reports into bug domains using unsupervised learning techniques.

5.1. Dataset & Data Pre-processing

For this task, we used the **DeepTriage** dataset which contans bug reports from **Google Chromium, Mozilla Core, & Mozilla Firefox**, each including the bug ID, title, and description. After removing duplicates and null values, the total samples came out to be **116371**.

Description of bug reports are used as the unlabelled dataset. The pre-processing steps used were same as before i.e. **Tokenization, Lowercasing, Stopwords Removal, Non-Alphabetic Character Removal & Lemmatization**.

5.2. Methodology for Clustering

After pre-processing the bug descriptions, 5000 features per sample are created using **TF-IDF Vectorization**. The created feature space is then applied to the following unsupervised paradigms:

1. **K-Means Clustering**: The optimal number of clusters, k, was determined using the Elbow and Knee methods. A word cloud was then generated for each cluster to visually represent the most frequent terms within them.

2. **Gaussian Mixture Model (GMM)**: Clusters were created using the GMM approach, with the number of clusters (k) obtained from the previous step. Each data

point was assigned to the cluster with the highest posterior probability. The clustering results were visualized in both 2D and 3D using Principal Component Analysis (PCA). For computational efficiency, 25% of the dataset was sampled.

5.3. Analysis & Conclusion for K Means Clustering

In the Within-Cluster Sum of Squares (WCSS) vs. k graph, the 'elbow' point occurs at k=7, which indicates the optimal number of clusters. This is the point where the rate of decrease in WCSS slows significantly.

Labels for the clusters are manually assigned based on insights derived from their respective word clouds. For instance, a cluster had terms like **'linux', 'useragent', 'intel', 'macintosh', 'mac', 'window', and 'gecko'**, which point to compatibility or configuration issues across various operating systems and hardware environments and was labeled as **OS Related**. Another cluster included words such as **'crash', 'build', 'run', 'test', 'logging', 'failed', 'valgrind', 'null', 'missing', and 'exception'**, reflecting errors encountered during the build and compilation processes and was given **Compilation Errors** as the label. These clusters are visualized below:

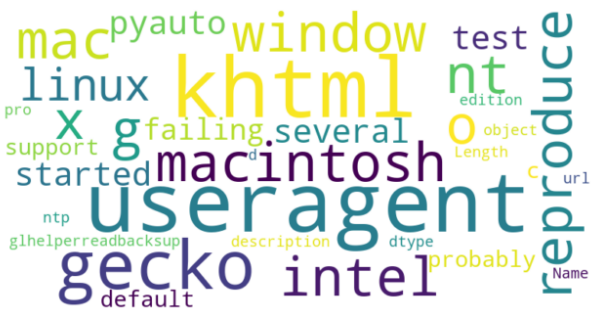

Figure 6. Cluster 1: OS Related

Figure 7. Cluster 2: Compilation Error

5.4. Analysis & Conclusion for Gaussian Mixture Model

The Gaussian Mixture Model (GMM) clustering reveals an imbalanced distribution across the 7 clusters, with two dominant groups containing the majority of the data points, while the remaining clusters are more sparse. This indicates that a central pattern defines most of the data, while the other clusters represent less frequent, niche occurrences.

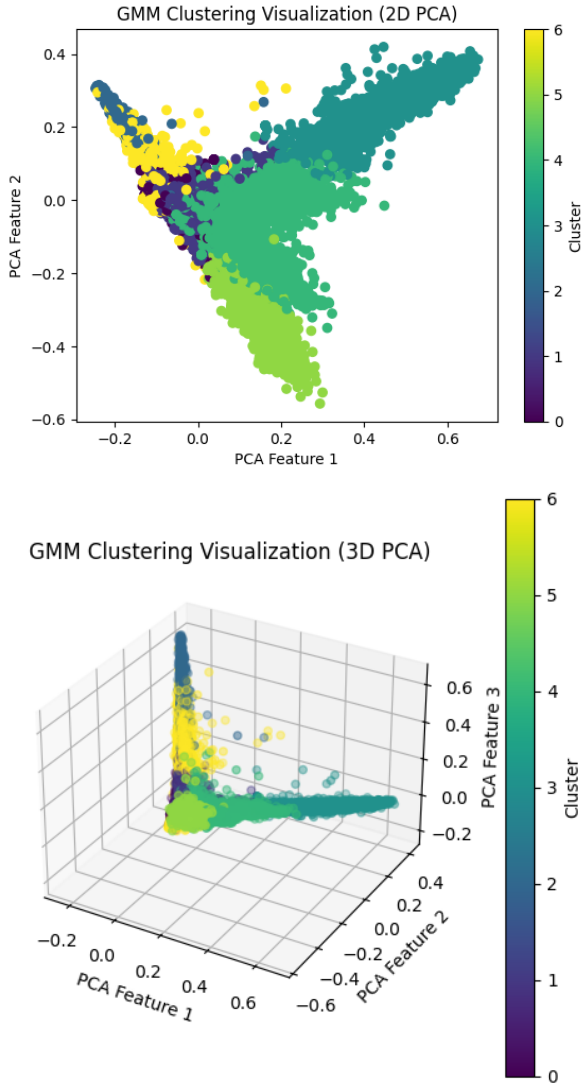

Figure 8. 2D & 3D Visualization

This imbalance in cluster distribution is mirrored in the visualizations, where the dominant groups stands out clearly, but overlapping regions hint at the complexities within the smaller clusters. Both the 2D and 3D visualizations demonstrate successful clustering, with data points generally well-separated. However, some overlap between clusters suggests underlying similarities or noise. This overlap may decrease in higher dimensions, where cluster separation could become more pronounced. As the current dimensionality may not fully capture the data's complexity, further analysis in higher dimensions could enhance the clarity of cluster distinctions.

6. Learnings & Contributions

The team learned data preprocessing techniques (e.g., tokenization, TF-IDF), applied supervised algorithms (e.g., Logistic Regression), and evaluated models using metrics like accuracy and F1-score. Challenges included data imbalance, model generalization, and preprocessing unstructured text data, requiring fine-tuning and optimization for bug severity and issue classification. Contributions of each member:

- **Ishir Bhardwaj**: Unsupervised Learning & making project report
- **Manit Kaushik**: Supervised Learning & making project presentation
- **Pranav Gupta**: Unsupervised Learning & making project report
- **Raghav Wadhwa**: Supervised Learning & making project presentation

7. 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.