Bug Classification: A Text Classification Approach for Bug Classification

Adithya Seesanabilu Nagaraj
Purdue University FortWayne
seesa01@pfw.edu

Sinduja Kuna
Purdue University FortWayne
kunas01@pfw.edu

Abstract

The rapid growth of software products in the industry has led to an exponential surge in bug reports from diverse sources. Managing and categorizing these reports is a daunting task due to their vast volume and heterogeneity. This paper proposes a novel approach to bug classification, leveraging Natural Language Processing (NLP) techniques to identify similar bugs and categorize them automatically. Our system aims to facilitate efficient bug management and resolution by comparing and contrasting existing approaches. In this paper we introduced a fine-tuned transformer based model which performed well resulting in accuracy of 78% for bug classification.

1 Introduction

Software bugs are an inevitable aspect of the software development life cycle, and their efficient management is paramount for ensuring product quality and user satisfaction. The exponential growth in software products has resulted in an overwhelming surge of bug reports from diverse sources, including users, testers, and automated tools. Manual bug triaging and classification are labor-intensive and error-prone, prompting the need for automated solutions. This research proposes a novel approach for bug classification using advanced natural language processing (NLP) techniques. Our system aims to identify similar bugs and categorize them automatically, contributing to more efficient bug management and resolution.

The motivation behind our research stems from the pressing need for efficient bug management systems in the software industry. The ever-increasing complexity of software systems and the proliferation of bug reports pose significant challenges to software development teams. By automating the process of bug classification, we aim to streamline bug management workflows, reduce manual effort, and expedite bug resolution.

The primary objective of our research is to develop a system that can automatically identify similar bugs from a large corpus of bug reports and categorize them into predefined classes. Specifically, we aim to address the following challenges:

- Handling the unstructured nature of bug reports written in natural language.
- Dealing with the variability and noise present in bug reports from different sources.
- Designing an efficient text classification model capable of accurately categorizing bug reports into relevant classes.

This paper presents various approaches we have implemented for bug classification, including the employment of Word2Vec and TF-IDF vectorizer embeddings coupled with Naive Bayes, Random Forest, Logistic Regression and Multi-Layer Perceptron (MLP). While these models yielded promising results, the transformer-based BERT model stood out with its superior performance compared to all others.

2 Related Work

In the study entitled ”CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms,” Ahmed et al. (2021) it was observed that class imbalance presents a challenge in effectively training models and accurately categorizing bugs. The investigation explored the utilization of machine learning methodologies such as naive Bayes and decision tree classifiers as potential solutions. The research paper underscores the consideration of these ML techniques within the context of bug classification. Additionally, it highlights the importance of addressing class imbalance to enhance the efficacy of bug classification systems.

The concept of employing summaries or descriptions in lieu of titles for model training was derived from the journal titled ”Deep Learning-based Software Bug Classification” Meher et al. (2024). This study served as the basis for adopting alternative textual representations to enhance the efficacy of bug classification models. The exploration of this approach underscored its potential to improve the accuracy and robustness of bug classification systems based on deep learning methodologies.

3 Dataset Details

Table 1 shows the dataset details that will be referred to in this paper, along with the corresponding URL. Dataset 1 is a GitHub page which is having the data in either .tor file or .json file format. Dataset belongs to Eclipse and Mozilla bugs data and we considered Mozilla data for our paper as it is giving better results. Dataset 2 consists of bug reports from various software projects such as Eclipse Platform, JDT, Firefox, Mozilla Core, and Thunderbird. In this paper, particular emphasis was placed on bug reports originating from the Mozilla Core project. Dataset 3 is also sourced from Bugzilla; however, it pertains to bugs related to the Mozilla core and includes descriptions for each bug.

Dataset #	URL	Number of Rows	Unique Components
Dataset 1	msr2013-bug_dataset	394878	268
Dataset 2	Bughub/Bugzill	10000	132
Dataset 3	Mozilla_Core	205069	138

Table 1: Dataset Details

4 Methodology

The input text to the model underwent various preprocessing steps to remove noise and enhance its quality. The following data preprocessing techniques were implemented:

- Removing HTML Tags
- Removing URLs
- Removing punctuation
- Removing stop words
- Stemming (PorterStemmer)
- Spelling correction (TextBlob)

We explored two vectorization methods, Word2vec and TF-IDF vectorizers, and also experimented with pre-trained word2vec vectorizers such as Google News[1] and GloVe[2] embeddings. Our objective was to assess their effectiveness in handling bug classifications.

We included Naive Bayes, Logistic Regression, and Random Forest algorithms in our model portfolio, as these algorithms are well-known for their efficiency in text classification tasks. Models using MultinomialNB, Random Forest, and Logistic Regression coupled with the TF-IDF vectorizer yielded better results. We also considered other vectorization techniques such as Count Vectorizers, but they did not achieve satisfactory accuracy in any combination.

Building upon the Naive Bayes implementation, we transitioned to a more advanced bug classification system using Word2Vec embeddings and Multi-layer Perceptron (MLP) classifiers. The Word2Vec embeddings, obtained from pre-trained models like Google News and GloVe datasets, facilitated the conversion of bug report descriptions into dense vectors. These vectors captured semantic relationships between words, thereby enhancing the feature representation of bug reports.

We shifted our focus to transformer-based models, leveraging their ability to learn from large corpora of data. Specifically, we fine-tuned the smallest BERT-based model, DistilBERT Sanh et al. (2019), on our dataset. We trained the model in batches of data for two epochs.

[1]https://code.google.com/archive/p/word2vec/
[2]https://nlp.stanford.edu/projects/glove/

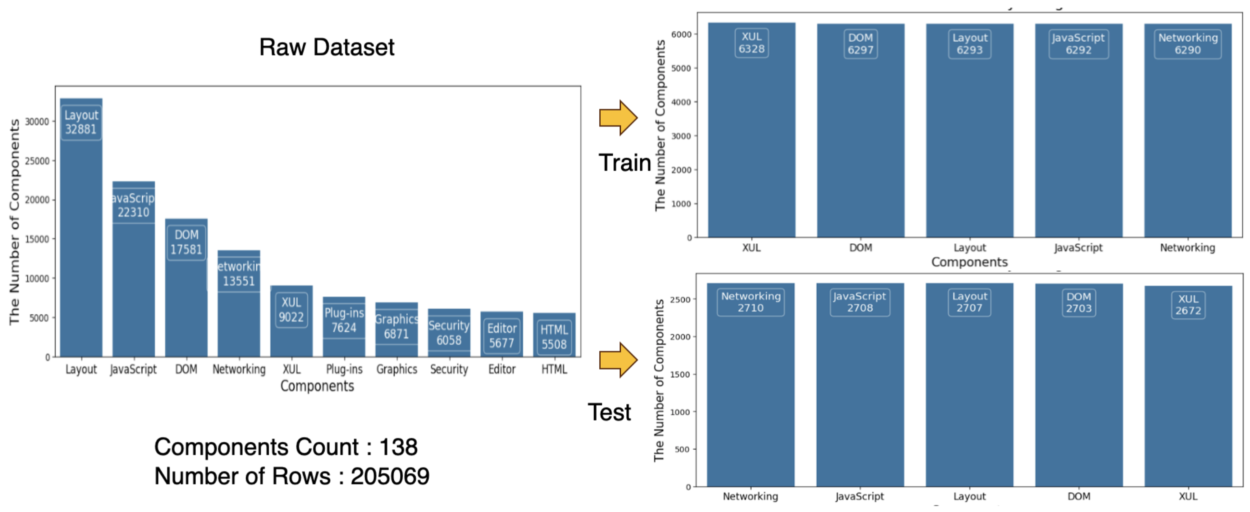

Figure 1: Dataset Balancing to Test and Train Data

4.1 Hyperparametrs

In the Hyperparameters section of the research paper, the configuration for the BERT-based model utilizing the DistilBERT architecture involved setting a learning rate of 2e-5 and conducting training for 2 epochs. For the Multi-Layer Perceptron (MLP) model comprising three layers, each layer consisted of 100 neurons, and the maximum iterations tested were set to 200. Additionally, in the case of the Random Forest Classifier, an ensemble learning algorithm, various hyperparameters were explored, including the number of estimators, maximum depth, minimum samples split, and minimum samples leaf.

5 Experiments

We evaluated the effectiveness of our bug classification models using three datasets: Dataset 1, comprising approximately 400,000 rows from a GitHub repository. Dataset 2, containing 10,000 rows from BugHub; however, we encountered limitations that restricted access to the full dataset. Dataset 3, consisting of 205000 rows from BugHub. Given Dataset 3's sufficient training data, we chose to work with it, despite its high imbalance. To improve accuracy, we refined and balanced the dataset. A 70%-30% training-testing split was used for all experiments.

To address the issue of a substantial volume of unique components within the Dataset, we employed a strategy of merging similar components based on their names. All subcategories of components were consolidated into their main component, resulting in a reduction of unique components from 138 to 72. Due to limited data availability, we focused on training the top 5 most prevalent components.Consequently,we opted to balance the dataset by selecting 8000 rows for each component as shown in Figure 1

We experimented with various vectorizers, including TF-IDF, count vectorizers, and Word2Vec, alongside pre-trained embeddings such as Google News and GloVe. Additionally, we endeavored to enhance accuracy by training the pre-trained embeddings. However, this effort yielded marginal improvements, as many words lost meaning during pre-processing. Despite this, TF-IDF consistently outperformed other methods in terms of accuracy across all models.

The dataset initially comprised over 4000 rows with titles containing 8 to 9 words on average, extending up to 30 words. This quantity of data was deemed insufficient for robust model training in component prediction tasks. Consequently, we transitioned to utilizing bug descriptions, which provided a more substantial content. With approximately 1600 rows containing descriptions averaging 25 to 30 words and some extending to 300 words or more, this dataset proved adequate for model

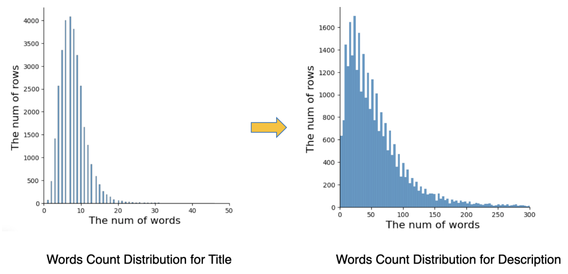

Figure 2: Word Count Distribution in Title and Description

training and significantly contributed to the overall accuracy. This transition is potrayed in the Figure 2.

In reviewing the dataset description, we observed a significant amount of noise. Many descriptions consist of script snippets or error messages without any accompanying explanations. To address this issue, we applied text pre-processing techniques to refine the data within the descriptions. This preprocessing step contributed to enhancing the accuracy of the models.

6 Results

The entirety of the results has been consolidated within Table 2. We employed various machine learning models and techniques, including Word2vec and GloVe pre-trained embeddings with Multi-Layer Perceptron (MLP) and Random Forest, achieving accuracies of 49% and 65%, respectively; TF-IDF vectorization with Random Forest, Multinomial Naive Bayes (NB), and Logistic Regression, yielding accuracies of 72%, 75%, and 76%, respectively; and a transformer-based DistillBERT model, achieving the highest accuracy of 78%.

Model	Max Accuracy (%)
Word2Vec and MLP (Google News)	48.9
Word2Vec and Random Forest	65.19
Tfidf - Random Forest	72.19
Tfidf - Naive Bayes	75.31
Tfidf - Logistic Regression	76.16
DistilBERT	**78.33**

Table 2: Accuracy result

7 Analysis Conducted

We primarily focused on the Tfidf and word2vec vectorizers. The models combined with Tfidf demonstrated superior performance compared to those coupled with word2vec. For instance, when employing Random Forest with both Tfidf and word2vec vectorizers, the Tfidf model achieved an accuracy of 72%, while the word2vec model yielded a result of 65%. This clearly indicates that semantic relations between words are not essential for classification; rather, the frequency of words in the document is crucial. Word2vec may be more suitable for tasks such as text generation, where capturing relationships between words is critical

In our exploration of advanced NLP topics, we investigated transformer-based models for classification, specifically leveraging the BERT model's encoder architecture. We selected DistilBERT, the smallest and most efficient model in the BERT family, for its ease of training. This model was pre-trained on a large corpus of data and fine-tuned according to our dataset. In the first epoch, we achieved an accuracy of 68%, which improved to 78% in the second epoch. This yielded the highest accuracy compared to all the other models.

8 Future Work

Our next phase of work will look into:

1. Further exploration could involve extending the training duration of the DistilBERT model across additional epochs to potentially enhance its accuracy.
2. Future investigations could explore the integration of advanced neural network architectures such as CNNs, RNNs, and LSTMs, which hold the potential for improved prediction outcomes compared to the current MLP implementation.
3. Currently the dataset is balanced to top 8000 rows for each components. Future iterations could explore a refined approach to dataset selection, focusing on identifying and including rows with relevant content rather than solely relying on top records. While this may require manual curation, it holds promise for enhancing prediction accuracy.

9 Challenges Encountered

To address the computational demands of training the BERT model, we employed a strategy of training in batches of 50 rows and storing the resulting .h5 file. Subsequent iterations utilized the saved .h5 file to continue training with additional chunks of data. This approach allowed us to overcome memory limitations, completing training within 51GB of CPU RAM in approximately 9.5 hours.

The dataset contained significant noise, necessitating extensive pre-processing efforts to retain word meaning. Despite our diligence in pre-processing, we observed a loss of semantic coherence in the resulting text, which raised concerns about potential accuracy degradation in the model while we are using word2vec vectorizers.

Realization that the bug classification model may be organization-specific, requiring a tailored training approach for different organizational structures and bug report formats

10 Conclusion

In conclusion, our research journey into bug classification has been marked by significant challenges and enlightening discoveries. Despite encountering issues such as dataset noise and computational limitations, our commitment to automating bug management practices remained steadfast. Through rigorous experimentation with various methodologies, we explored different vectorizers like TF-IDF and Word2Vec and coupled with models like MultinomialNB, Logistic Regression, MLP, Random Forest assess their performance in bug classification.Additionally, we incorporated transformer-based model like DistillBERT, achieved maximum accuracy, surpassing all other models. This research experience has underscored the importance of data preprocessing and meticulous model selection in enhancing classification accuracy, providing invaluable insights into the field of Natural Language Processing (NLP). Future research directions include refining our approaches to provide software development teams with more efficient, tailored bug management solutions that address their specific needs and challenges.

References

Hafiza Anisa Ahmed, Narmeen Zakaria Bawany, and Jawwad Ahmed Shamsi. 2021. Capbug-a framework for automatic bug categorization and prioritization using nlp and machine learning algorithms. *IEEE Access*, 9:50496–50512.

Jyoti Prakash Meher, Sourav Biswas, and Rajib Mall. 2024. Deep learning-based software bug classification. *Information and Software Technology*, 166:107350.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Appendix

The source code is available in this Git repository and can be run directly in Google Colab. https://github.com/adithya1012/nlp_bug_classification