GitHub Bug Prediction

Eshaan Gupta[1*]
University of Manitoba
Winnipeg, Canada
guptae@myumanitoba.ca

Aman P Singh[1*]
University of Manitoba
Winnipeg, Canada
Singhap2@myumanitoba.ca

ABSTRACT

GitHub is an open-source platform widely used for collaborating on software development projects. A vast number of GitHub issues are posted on a regular basis, and these are divided into three main categories: bugs, features, and questions. While these GitHub issues are valuable in the overall development lifecycle, it remains tedious to look through and categorize the bugs from them manually. In this work, we propose using NLP-based machine learning models to streamline the process of identifying bugs in GitHub Issues for the purpose of better collaboration within the software community. We aim to significantly speed up the bug-fixing process by training models on the data that is available on many existing GitHub issues. As a proof of concept, we provide the design and implementation of a Bug classification model which automatically learns the common patterns of a GitHub bug issue based on existing instances of GitHub Bugs and further leverages them for valuable suggestions to the developers. Using the results of our research, we aim to provide valuable insight into the potential of NLP-based models for bug prediction. We believe our work can serve as an important stepping stone for future projects aiming to classify Software bugs.

Keywords

NLP; Lemmatization; Stemming; Machine Learning; Naive Bayes, Boost.

1. INTRODUCTION

The software development lifecycle involves various stages such as coding, testing, and debugging. Identifying and fixing bugs is a crucial part of this process that requires significant time and effort. Software developers come across many bugs in their daily projects, and it is very important that these bugs are identified and resolved in time. GitHub is a large open-source platform used for software development and hosts a plethora of software projects, making it an ideal platform to study bug prediction.

Identifying and resolving bugs manually can be a dull and time-consuming task, especially when working with a large codebase with plenty of development Issues. To address this challenge, we are proposing an NLP-based solution to automate the process of identifying bugs in GitHub issues. In recent years, natural language processing (NLP) techniques have been widely used in software engineering to analyze unstructured data such as source code, commit messages, and issue reports [3]. It provides numerous benefits, including improved efficiency, accuracy, and speed of bug detection, leading to the production of higher-quality software [1].

In this research paper, our proposed solution utilizes a dataset of 150000 instances of GitHub issues with three categories: bug, feature, and question. There are several studies focusing on using NLP-based machine learning models for bug prediction in software. The current research mainly focuses on the classification of bug reports and uses various machine-learning algorithms to achieve accurate results [3] while highlighting the need for efficient bug prediction models to improve the quality of software development [4].

In comparison, our proposed research study aims to identify bugs in GitHub issues using NLP-based machine-learning models while making a comparison between different models and techniques. We specifically focused on the use of a dataset based on the text of GitHub issues that contains three attributes: title, body, and label. Our proposed solution involves the use of data preprocessing techniques such as tokenization, stemming, and stop word removal to reduce the noise and complexity of the data. Additionally, we will perform exploratory data analysis on our dataset to look for trends such as the most common words used when the label is bug, feature, or question. We will use machine learning models like XgBoost, Random Forest, and Naive Bayes Classifier for the classification task and evaluate the performance of our model using standard metrics such as accuracy, precision, and recall.

Another major difference between our proposed research and the previous research is that we focus on identifying bugs in GitHub Issues rather than bug reports themselves. Furthermore, we plan to measure the effect of using techniques like hyperparameter tuning and cross-validation to improve our model performance and achieve higher accuracy. Overall, our research aims to ease the process of bug prediction in software development by using NLP-based machine-learning models for identifying bugs in GitHub issues. We believe our work can provide insights into the potential of NLP-based machine learning models for bug prediction and identify areas for future work to improve the efficiency and effectiveness of software development.

The subsequent sections of this paper are arranged in the following manner. The motivation behind this work is introduced in Section II. Section III outlines and elaborates on the techniques used. The evaluation results are presented in Section IV, while limitations are discussed in Section V, and related work is discussed in Section VI. Finally, Section VII concludes the paper.

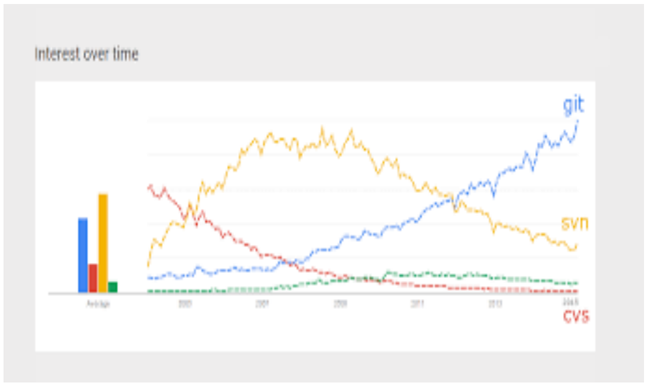


Figure --

2. OBSERVATION & MOTIVATION

In this section, we will discuss the open-source platform under study GitHub and provide some key observations on GitHub issues and define the motivation of our study.

A. *Platform under study*

In our study, we use the open-source platform GitHub to obtain the data for our research. The platform hosts many software repositories and is used by millions of users in their routine development activities. At the start of this study, we found that there were 100,615,019 open (unanswered) issues, while 257,941,048 closed (answered) issues. This reflects the importance of streamlining the process of these large number of GitHub issues. As our research mainly focuses on the textual analysis of GitHub issues, there is no apparent need to check for generalizability over other platforms. Also, we do not expect any apparent effect of having issues related to different programming languages.

B. *Frequency of GitHub Issues*

According to the GitHub guidelines, developers can create an issue from a repository, an item in a task list, a note in a project, a comment in an issue or pull request, a specific line of code, or a URL query. As discussed earlier there were 100,615,019 open issues at the start of this study and a recent article [5] suggests that this number keeps growing because of the growing interest of software developers in the platform. This shows that there is an apparent need for new methodologies to streamline the process of resolving these Issues.

C. *Kind of GitHub Issues*

GitHub provides many inbuilt tags to help classify the Issues. These tags include bug, help wanted, revision needed, enhancement, feature, and question. While these tags may be helpful in general, there is much room for errors. These tags are very broad and do not represent separate categories, for example, "help wanted" and "revision needed" are practically the same in the context of these Issues. Therefore, our study aims to classify these issues into 3 different categories bug, feature, and question.

D. *Time to resolve GitHub Issue*

GitHub has a large and active community of software developers who try to resolve any issues posted on the platform. Despite strong community engagement, sometimes it takes a while for an issue to be resolved. While our research may not have any impact on improving the developer's engagement, it could improve the classification of bugs thus decreasing the time to resolve them.

E. *Exploratory data analysis*

After doing exploratory data analysis on the text of the GitHub issue in our dataset we were able to find out which words were most common in each category of GitHub Issues. Figures 4,5,6 represent the frequency of each word in each category.

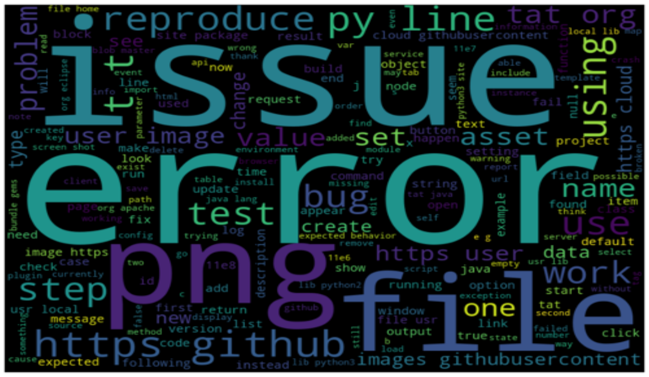


Figure 1. Words related to Bug.

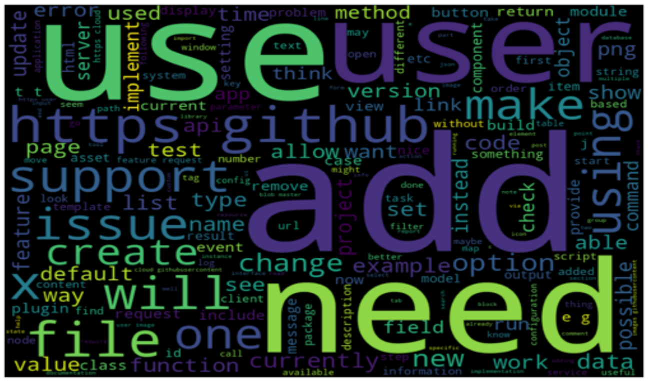


Figure 1. Words related to Bug.

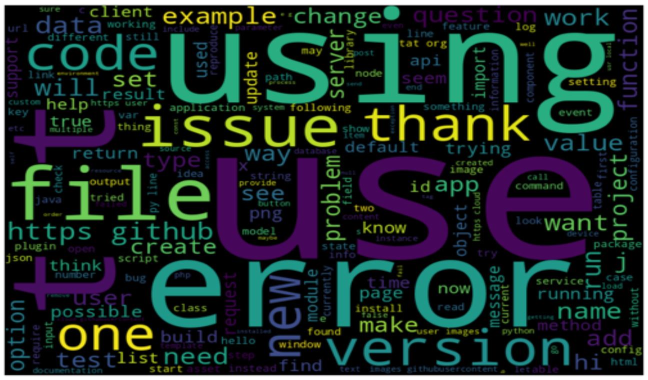


Figure 1. Words related to Bug.

3. MOTIVATION

Predicting bugs in software development projects on GitHub can bring about numerous benefits to the development process, making it more efficient and ultimately leading to a better user experience. Using advanced techniques such as Natural Language Processing (NLP), developers can analyze vast amounts of unstructured data, such as source code, commit messages, and issue reports, to identify potential bugs early in the development cycle. By identifying these bugs early on, developers can prioritize their workload, allocate resources more effectively, and address issues before they become bigger problems.

This not only results in cost savings but also allows for better time management and a higher-quality software product. Furthermore, bug prediction on GitHub can help developers to categorize issues more effectively. By identifying keywords and phrases in GitHub issues, NLP models can categorize them into bugs, features, and questions, which can allow for better resource allocation and prioritization. Multi-label classification of GitHub issues is also possible with NLP models, which can be extremely useful for large-scale software projects with numerous issues that need to be addressed.

In addition to the practical benefits of bug prediction, the research conducted into NLP models can provide valuable insights into the capabilities of machine learning in software development. Identifying which NLP-based machine learning models perform best on a specific dataset can be useful in identifying which models should be used in future software development projects. Furthermore, the development of such models can contribute to the advancement of the field of machine learning, not just in software development but in a range of other industries.

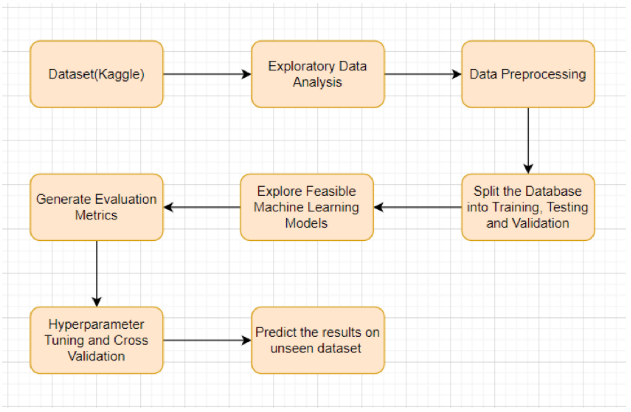
Ultimately, the goal of our research is to successfully answer the fundamental questions:

1. Can natural language processing techniques be used to accurately predict bugs in GitHub issues?
2. Can NLP models identify keywords and phrases in the GitHub issues to categorize them into bug, feature, and question?
3. Can NLP models handle multi-label classification of GitHub issues?
4. Which NLP-based Machine learning models perform the best classification on our data?
5. How does our model perform on raw data taken from the GitHub platform?

By doing so, we hope to contribute to the development of more efficient and effective software development processes and a better user experience. In conclusion, the use of NLP models to predict bugs in software development projects on GitHub has the potential to bring about numerous benefits, including better resource allocation, improved time management, higher-quality software, and cost savings. Furthermore, the development of such models can contribute to the advancement of the field of machine learning, making our research not just valuable for software development but for a range of industries. By answering fundamental questions related to the use of NLP in bug prediction on GitHub, we aim to contribute to a more streamlined and efficient development process, ultimately leading to a better user experience.

4. METHODOLOGY

In this section, we will provide a description of our dataset and discuss in-depth the methodology, tools, and technology used in our research. The complete workflow of our research methodology is described in figure {}.

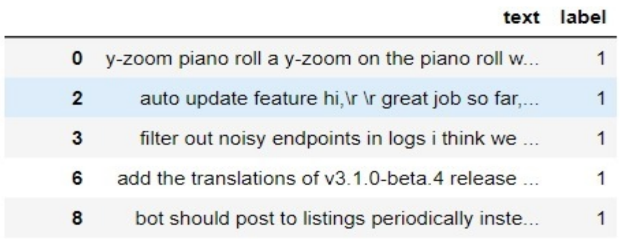


Dataset(Kaggle)
Exploratory Data Analysis
Data Preprocessing
Split the Database into Training, Testing and Validation
Explore Feasible Machine Learning Models
Generate Evaluation Metrics
Hyperparameter Tuning and Cross Validation
Predict the results on unseen dataset

Figure 1. Methodology chart.

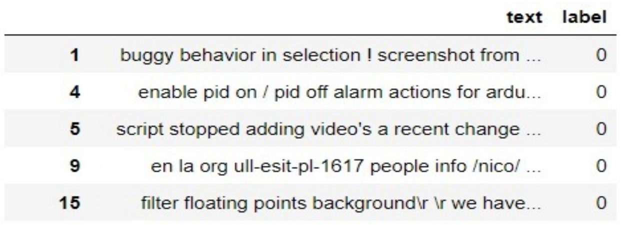
4.1 Dataset Description

Our proposed solution involves the use of a dataset based on the text of GitHub issues; it contains three attributes: title, body, and label, as shown in Figure 1. The target attribute is Label, with three different categories: 0 denotes a bug, 1 denotes a feature, and 2 denotes a question. The said dataset has been retrieved from [3] and it contains 150000 instances of GitHub issues representing all three categories.



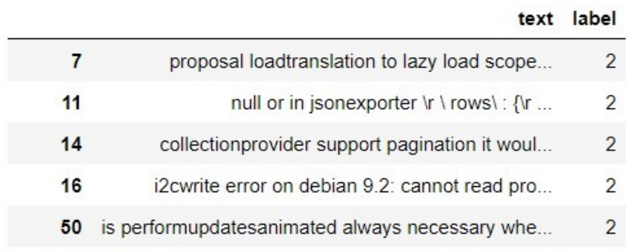
	text	label
0	y-zoom piano roll a y-zoom on the piano roll w...	1
2	auto update feature hi,\r \r great job so far,...	1
3	filter out noisy endpoints in logs i think we ...	1
6	add the translations of v3.1.0-beta.4 release ...	1
8	bot should post to listings periodically inste...	1

Figure 1. Dataset structure for feature label.



	text	label
1	buggy behavior in selection ! screenshot from ...	0
4	enable pid on / pid off alarm actions for ardu...	0
5	script stopped adding video's a recent change ...	0
9	en la org ull-esit-pl-1617 people info /nico/ ...	0
15	filter floating points background\r \r we have...	0

Figure 1. Dataset structure for bug label.



	text	label
7	proposal loadtranslation to lazy load scope...	2
11	null or in jsonexporter \r \ rows\ : {\r ...	2
14	collectionprovider support pagination it woul...	2
16	i2cwrite error on debian 9.2: cannot read pro...	2
50	is performupdatesanimated always necessary whe...	2

Figure 1. Dataset structure for question label.

Although there is a suitable number of instances from each category of the target variable accounting for better training of our machine learning models, the text values in our data contain an excessive amount of redundant stop words, links, punctuation, etc. as they might be grammatically important but do not provide any value for the purpose of training out models. We have used the NLTK library to address this issue as it "comes bundled with the stop words corpus, a list of 2400 stop words across 11 different languages" [2]. Furthermore, we plan to do exploratory data analysis on our dataset to look for trends such as the most common words used when the label is bug, feature, or question.

4.2 Exploratory Data Analysis

A. Label counts

The first step for our exploratory data analysis was to check the value count for each of the label categories of GitHub issues. There are 69106 values of Features, 66827 values of Bugs, and 14067 values of Questions in our data as shown in figure {} Label counts. This step is important to check for any data imbalance.

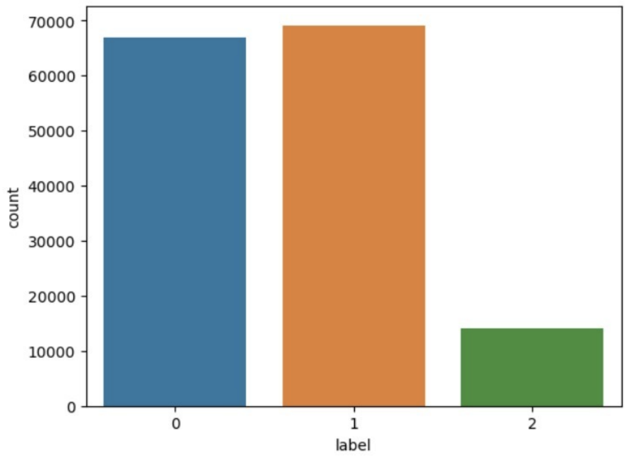


Figure 1. Label Counts.

B. Redundant words

In the second step, we checked for redundant words and punctuation in our data. Such words include and are not limited to "\r, \n, =" etc. as shown in Image {}. It is important to recognize these words as they play a negligible role in textual semantic analysis and might affect the performance of our models.

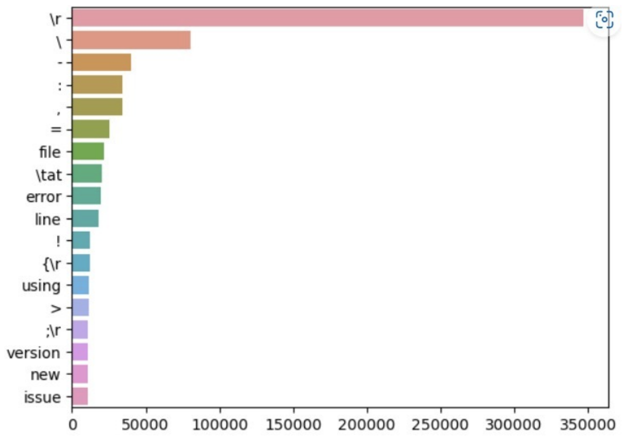
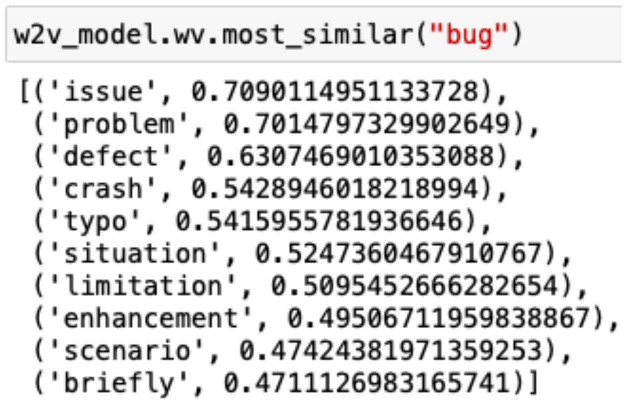


Figure 1. Redundant words.

C. Semantic analysis

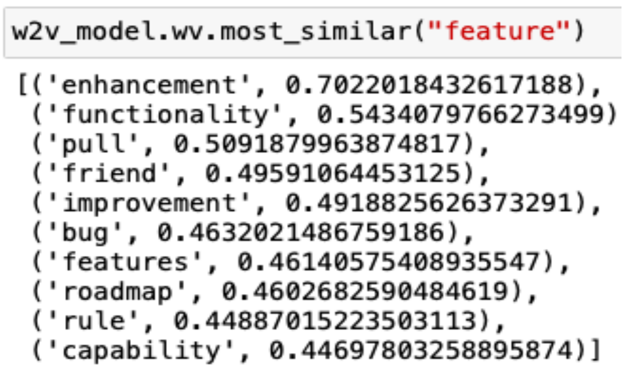
In the third step, we checked for the most relevant words for each category of GitHub Issues. For this, we used a word2vec model for the semantic analysis of our text. As a result, we found that "issue" and "problem" are the most related words to the Bug, while "enhancement" and "questions" were the most related words to Feature and Question respectively. The results of this textual analysis are shown in figure {}{}{}.



```
w2v_model.wv.most_similar("bug")

[('issue', 0.7090114951133728),
 ('problem', 0.7014797329902649),
 ('defect', 0.6307469010353088),
 ('crash', 0.5428946018218994),
 ('typo', 0.5415955781936646),
 ('situation', 0.5247360467910767),
 ('limitation', 0.5095452666282654),
 ('enhancement', 0.49506711959838867),
 ('scenario', 0.47424381971359253),
 ('briefly', 0.4711126983165741)]
```

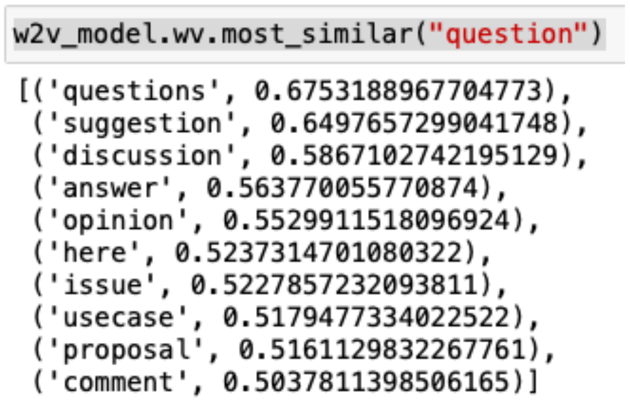
Figure 1. Words related to Bug.



```
w2v_model.wv.most_similar("feature")

[('enhancement', 0.7022018432617188),
 ('functionality', 0.5434079766273499)
 ('pull', 0.5091879963874817),
 ('friend', 0.49591064453125),
 ('improvement', 0.4918825626373291),
 ('bug', 0.4632021486759186),
 ('features', 0.46140575408935547),
 ('roadmap', 0.4602682590484619),
 ('rule', 0.44887015223503113),
 ('capability', 0.44697803258895874)]
```

Figure 1. Words related to Feature.


```
w2v_model.wv.most_similar("question")
```

```
[('questions', 0.6753188967704773),
 ('suggestion', 0.6497657299041748),
 ('discussion', 0.5867102742195129),
 ('answer', 0.563770055770874),
 ('opinion', 0.5529911518096924),
 ('here', 0.5237314701080322),
 ('issue', 0.5227857232093811),
 ('usecase', 0.5179477334022522),
 ('proposal', 0.5161129832267761),
 ('comment', 0.5037811398506165)]
```

Figure 1. Words related to Question.

4.3 Data Pre-processing

Before using our dataset to train the machine-learning models we used some data pre-processing techniques to refine our data. First, we checked for Nan and duplicate values in our data, but surprisingly there were no Nan and duplicate values in our data. Furthermore, we combined the title and text of our dataset to form a single variable. Then we used generic functions from the regex library to make our text lowercase, remove text inside square brackets, remove links, remove punctuation, and remove words containing numbers. After this, we also used the STOPWORDS library from nltk to remove the stop words from the text in our dataset.

After the initial pre-processing, we used two kinds of vectorizers namely tf-idf and count vectorizer to convert text into numerical representations that can be analyzed and processed by machine learning algorithms. Count Vectorizer counts the occurrences of each word in the text and creates a vector matrix of the counts. TF-IDF considers the frequency of words in the text as well as their relative importance. The final dimensions of the dataset after both vectorizers are represented in table {}. This is the final step before we trained our machine-learning models on the resulting dataset from both vectorizers.

Vectorizer	Rows	Columns
Tf-idf vectorizer	150000	7929037
Count vectorizer	150000	7929037

Figure 1. Redundant words.

4.4 Machine learning models

After data preprocessing, the dataset was divided into training and testing sets. Then, we trained the machine learning models like XgBoost, Random Forest, and Naive Bayes Classifier. These models are used in various tasks such as text classification, sentiment analysis, and spam detection, therefore we have chosen these algorithms for the purpose of this research. These algorithms often perform well in practice, and are computationally efficient, making them a popular choice for many NLP-based applications.

We have used the sklearn library to import prebuilt Random Forest and Naive Bayes Classifier models, while the XgBoost library for its inbuilt XGBClassifier. Further, we trained these models on the training dataset and used the trained model to identify bugs, features, and questions in the testing dataset. The results of our trained model will be evaluated based on the evaluation metrics described in Section 4.5. In addition, we plan to use techniques like hyperparameter tuning, cross-validation, etc. to improve our model performance and achieve higher accuracy.

Through the above learning process, we can obtain a predictive model to perform accurate bug predictions given the text of a GitHub issue. This predictive model can be further extended to an overall bug prediction model that might be able to predict bugs in any kind of textual data. The above learning workflow is generic and works similarly to many other machine learning applications in software development.

4.5 Evaluation metrics

Since this is a classification task, we will evaluate the performance of our model using standard metrics such as accuracy, precision, and recall. We will compute the result using various vectorizers like count vectorizer, tf-idf vectorizer, and word2vec, each of them training our models in a different way. Further, we will analyze how our machine-learning models perform on these vectorizers. Additionally, we will perform classifications on an unseen evaluation dataset to see how our model performs on real-world data.

5. EVALUATION

In this section, we will discuss the effectiveness of NLP-based machine-learning models in classifying GitHub issues and predicting if an issue is a bug. In particular, we aim to answer these research questions:

RQ1: Can natural language processing techniques be used to accurately predict bugs in GitHub issues?

RQ2: Can NLP models identify keywords and phrases in the GitHub issues to categorize them into bug, feature, and question?

RQ3: Can NLP models handle multi-label classification of GitHub issues?

RQ4: Which NLP-based Machine learning models perform the best classification on our data?

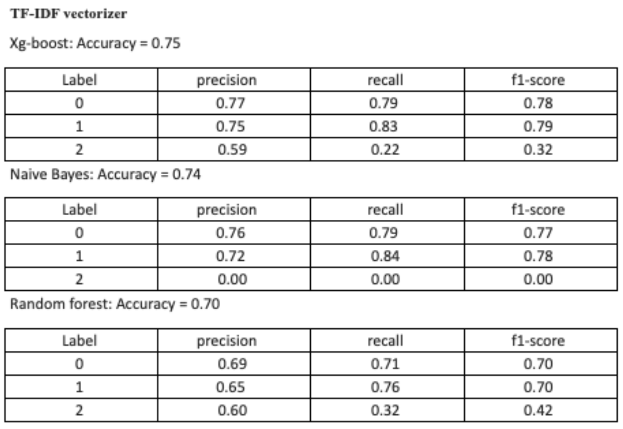
RQ5: How does our model perform on raw data taken from the GitHub platform?

After performing the initial text preprocessing and selecting the machine learning models, we performed classification on our dataset. By default, we use precision, recall, and f1-score as the evaluation metrics for our NLP-based machine learning model because of their ease of interpretation in classification tasks. The f1-score is computed as follows:

$$\text{F1-score} = \frac{TP}{TP + \frac{1}{2}(FP+FN)}$$

where TP, FP, TN, and FN denote true positives, false positives, true negatives, and false negatives, respectively. The results are computed based on the dataset created by both the tf-idf vectorizer and count vectorizer separately. Table {} and table {}

show the resulting evaluation metrics for both tf-idf and count vectorizer for different machine learning models under study.



TF-IDF vectorizer

Xg-boost: Accuracy = 0.75

Label	precision	recall	f1-score
0	0.77	0.79	0.78
1	0.75	0.83	0.79
2	0.59	0.22	0.32

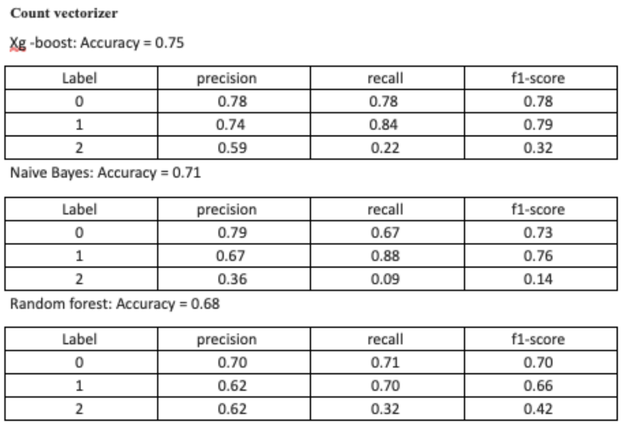
Naive Bayes: Accuracy = 0.74

Label	precision	recall	f1-score
0	0.76	0.79	0.77
1	0.72	0.84	0.78
2	0.00	0.00	0.00

Random forest: Accuracy = 0.70

Label	precision	recall	f1-score
0	0.69	0.71	0.70
1	0.65	0.76	0.70
2	0.60	0.32	0.42

Figure 1. Redundant words.



Count vectorizer

Xg -boost: Accuracy = 0.75

Label	precision	recall	f1-score
0	0.78	0.78	0.78
1	0.74	0.84	0.79
2	0.59	0.22	0.32

Naive Bayes: Accuracy = 0.71

Label	precision	recall	f1-score
0	0.79	0.67	0.73
1	0.67	0.88	0.76
2	0.36	0.09	0.14

Random forest: Accuracy = 0.68

Label	precision	recall	f1-score
0	0.70	0.71	0.70
1	0.62	0.70	0.66
2	0.62	0.32	0.42

Figure 1. Redundant words.

A. Results of RQ1:

We classified the GitHub issues into 3 categories using our NLP-based machine-learning models. Since the data is the text values of the GitHub issues, the results are expected to mimic the real-world data. Based on the results provided in table {} and table {} we believe natural language processing (NLP) techniques can be used to accurately predict bugs in GitHub issues.

As we can observe, the machine learning models based on the Tf-idf vectorizer and the machine learning models based on the count vectorizer both provide accurate predictions on average. The lowest accuracy amongst models based on both these vectorizers was seen to be of the Random Forest classifier, having an accuracy of 68%. While the highest accuracy was seen to be 75% of the Xgb-Classifier, which is the same for both tf-idf and count vectorizer.

Overall, NLP-based machine learning models can be a valuable tool for predicting bugs in GitHub issues. Developers can improve the overall quality and efficiency of the development process using these models to classify the bugs in GitHub issues.

B. Results of RQ2:

As discussed in section 4.3 sub-C of this paper, we used a word2vec model to do a semantic analysis of the text in our dataset. In the results, we found that it can be a very efficient tool for finding the relevant words for each category of GitHub issues.

Figure {}, figure {}, and figure {} show the most common words in all the categories of GitHub issues. we found that "issue" and "problem" are the most related words to the Bug, "enhancement" and "functionality" were the most related words to Feature, while "questions" and "suggestions" were the most related words to the Question.

Overall, the results show that NLP models can effectively identify keywords and phrases in the GitHub issues to categorize them into bugs, features, and questions.

C. Results of RQ3:

Our dataset is separated into 3 categories namely Bug, Feature, and Question. Each of the categories contains roughly the same number of values. For the purpose of multi-label classification, we had to convert the labels into numerical values so that it is easier to train our models, 0 represents a Bug, 1 represents a Feature, and 2 represents a Question.

In conclusion, we saw that all the machine learning models under consideration performed well in classifying the GitHub Issues, although it is important to note Naive Bayes model performed poorly while classifying the Question. This is because we have a smaller number of Question instances in our dataset.

D. Results of RQ4:

Determining which NLP model will perform the best for multi-label classification of GitHub issues depends on various factors, including the size and complexity of the dataset, the amount of available training data, the specific labels being used, and the computational resources available for training and inference.

Among the chosen models Xg-boost classifier had the highest accuracy of 75% in classifying the GitHub issue. The accuracy remained the same for both the TF-IDF vectorizer and the count vectorizer. Precision and recall for TF- IDF vectorizer-based Xg-boost model is 0.77, 0.79 respectively, while for the count vectorizer, the Xg-boost classifier had both precision and recall of 0.78.

Other than these models we also trained a transformer-based model BERT which is proven to achieve state-of-the-art performance in NLP-based classification tasks. The BERT model had the highest accuracy among all our models. The results of classification are displayed in figures {} {} {}.

E. Results of RQ5:

After training all the machine-learning models and testing them on our dataset, we finally took some raw data values of GitHub issues to test the performance of our models. We tested our models on 5 random GitHub issues. The BERT and Xg-boost classifier model was able to correctly classify all 5 of them, while Naïve Bayes and Random Forest model classified 3 of them correctly. Overall, we believe that the BERT and Xg-boost models can be generalized on raw data values, and both models can accurately classify the Bugs from GitHub issues.

Based on these results, it is apparent that our research has the potential to improve GitHub bug prediction significantly and thus improve the overall software development lifecycle. Amongst the traditional models, the Xg-Boost classifier was able to achieve the highest accuracy and performed well in other performance metrics, while the transformer-based BERT model had the overall best performance.

6. LIMITATIONS

Our research was able to produce some valuable insights into using machine learning to classify bugs on the GitHub platform. But it is important to consider the limitations and areas where we lacked some details in our research. The few limitations of the research that we were able to identify were:

A. *Limited dataset size:*

The size of the dataset used for the study was only 150000, which may be a limitation as machine learning algorithms typically require large amounts of data to be trained effectively. This dataset is not representative of all the Issues on the platform. This may also limit the generalizability of the results to other datasets.

B. *Data preprocessing limitations:*

While we performed some preprocessing on the data, we were not able to use all relevant preprocessing techniques that could have been helpful in improving the performance of our machine-learning models. For example, we used stemming and lemmatization which are common techniques for reducing words to their base form but other techniques like normalization and synonym replacement could also help improve the overall performance. Additionally, the removal of stop words may not always be appropriate depending on the task at hand.

C. *Limited choice of models*:

While for simplification of the research we mostly used traditional models like XgBoost, Random Forest, and Naive Bayes Classifier, there are many other machine-learning models that could have been used for text classification tasks, and the choice of models may limit the overall precision of our study.

D. *Evaluation metrics limitations*:

While we used common evaluation metrics such as accuracy, precision, and recall, there are many other metrics that could be used to evaluate the performance of the model, and the choice of metrics may affect the interpretation of the results. As an example, ROC and AUC could have been more valid for evaluating NLP-based classification models.

E. *Hyperparameter tuning*:

While we discussed the use of Hyperparameter tuning in the traditional models, the complexity of the transformer-based BERT model did not allow us to use any relevant tuning. It is an important aspect of machine learning and can significantly affect the performance of the model.

F. *Potential biases*:

We believe our data is taken over a certain time period and might be prone to bias with respect to the advancements in technology. Potential biases in the dataset or in the model could limit the generalizability of the results or lead to incorrect conclusions.

Overall, the study provides some valuable insights into the use of NLP-based machine learning models for bug prediction, but we believe there is a lot more room for improvement.

7. RELATED WORK

Our research has been inspired by previous work in the field of bug prediction using machine-learning models. There have been several research works in GitHub bug prediction using machine learning. Some of them are:

1. "Predicting Bugs in GitHub: A Large-Scale Study with Machine Learning" by Yu et al.

   The purpose of this research is to predict GitHub repository failures using machine learning techniques. The author has collected data from his over 10,000 open-source projects on GitHub, including project metadata, code, problem reports, and bug reports. Extract various features from the data, such as the number of shows, number of participants, number of topics, and number of comments to use as input for the machine learning model.

   In this study, the authors used four machine learning algorithms, including Naive Bayes, Random Forest, Support Vector Machines, and Neural Networks, to predict the occurrence of failures. The authors evaluate model performance using various metrics such as accuracy, precision, recall, and F1 score.

   This research showed that machine learning algorithms can accurately predict the occurrence of bugs in GitHub projects. The authors reported his 78.6% overall accuracy, which outperformed the original model by 16.7%. He also tested and found that Random Forest and Support Vector Machine performed the best among the four algorithms.

   The authors also performed a feature importance analysis and found that the number of code changes, number of code contributors, and number of issues had the greatest impact on bug prediction. They also found that file size, number of comments, and number of collaborators were not all that important in predicting failure.

   Overall, this research found that machine learning algorithms can be used effectively to predict bugs in GitHub projects. The results help developers find potential bugs early in the development process, thereby improving software quality.

2. "Predicting Bug-Prone Java Files in GitHub" by Meng et al. This study focuses on predicting bug-prone Java files in GitHub repositories.

   The authors collected a dataset consisting of Java files from 25 GitHub projects, which were classified as either bug-prone or not bug-prone based on the number of bug reports associated with each file.

They used a total of 16 software metrics, such as the number of lines of code, number of methods, number of comments, and code complexity to represent each file in the data set. The authors then used four machine learning algorithms, including Random Forest, Vector Machine support, Naive Bayes, and Logistic Regression, to build predictive models of the error susceptibility of Java files on GitHub.

They evaluated the performance of these models using three different measures: recall, accuracy, and F-measure. The results show that the random forest algorithm performs best with an F-measure of 0.716. This means the model can correctly identify 71.6% of error-prone files while minimizing false positives.

The study also found that the most important metrics for predicting error-prone files were code complexity and code revenue. Overall, research has shown that machine learning techniques can be used to predict error-prone Java files on GitHub, and these predictions can help developers prioritize their efforts. their error correction. The study also highlights the importance of software metrics such as code complexity and code obfuscation in predicting software failures.

3. "Bug Prediction in GitHub using Machine Learning Techniques" by Akter et al. This study aims to predict bugs in GitHub repositories using machine learning techniques.

To evaluate the effectiveness of the machine learning models, the authors used a dataset of 11 open-source projects on GitHub. Decision Tree (DT), Random Forest (RF) and Support Vector Machine (SVM) were the four machine learning algorithms used in the study.

Furthermore, Precision, accuracy, recall and F1 scores were some of the evaluation metrics used by the authors to evaluate the performance of these algorithms. The results of the study showed that the Random Forest model was the best-performing model of all in the evaluation metrics. The Random Forest model achieved 87.13 percent accuracy, 0.87 precision, 0.87 recall, and 0.86 F1 score. The Decision tree (DT) model also showed promising results with an accuracy of 84.63 percent, a precision of 0.85, a recall score of 0.84, and an F1 score of 0.83.

The study also looked at key features that can be used to predict errors in GitHub repositories. The authors found that the number of code changes, the number of participants and the number of solved problems were the most important factors in predicting software bugs.

The study highlights how machine-learning techniques can be used to predict software bugs in GitHub repositories. The results suggest that random forest and decision tree algorithms can be useful for error prediction and can be used as software quality control tools.

Overall, these studies demonstrate the potential of using machine learning techniques for bug prediction in GitHub repositories. They also highlight the importance of having large and diverse datasets for training and evaluating machine learning models.

8. FUTURE WORK

Our proposed solution has the main goal of automating the process of identifying bugs in GitHub issues, which will help inspire further work in this area. One potential example for future work is to extend the model to how severe the bug is and to rank them based on their predicted severity. While there are many other works that can be inspired by our research, we also suggest exploring different machine-learning algorithms and other techniques to improve our model's performance.

9. CONCLUSION

In conclusion, our research explored the effectiveness of NLP-based machine learning models in classifying GitHub issues and predicting bugs. Through data pre-processing techniques such as removing redundant words and using vectorizers, we were able to train and evaluate machine learning models such as XgBoost, Random Forest, and Naive Bayes Classifier on our dataset. We evaluated the performance of our models using standard metrics such as accuracy, precision, and recall and found that NLP techniques can be used to accurately predict bugs in GitHub issues.

The results showed that the machine learning models based on both the Tf-idf vectorizer and the count vectorizer provide accurate predictions on average. The XgBoost Classifier performed the best with an accuracy of 75% on both vectorizers, while the Random Forest classifier performed the worst with an accuracy of 68%. These results suggest that the use of NLP-based machine learning models can be a useful tool in predicting bugs in GitHub issues.

The research also aimed to answer other research questions such as whether NLP models can identify keywords and phrases in GitHub issues to categorize them into bug, feature, and question, and whether they can handle multi-label classification. The results showed that NLP models are capable of handling multi-label classification and can identify keywords and phrases to categorize GitHub issues accurately.

Moreover, this research has potential implications for software development, where predicting bugs can save time and resources. The predictive model developed in this research can be further extended to predict bugs in any kind of textual data. Future research can focus on improving the accuracy of the model through hyperparameter tuning, cross-validation, and exploring other NLP techniques.

Overall, this research highlights the potential of NLP-based machine learning models in predicting bugs in GitHub issues. With the increasing importance of software development, such techniques can be valuable tools for improving the quality and efficiency of software development processes.

10. ACKNOWLEDGEMENT

We would like to thank the department of Computer Science, University of Manitoba and professor Shaowei Wang for helping us throughout our research process.