CISC499 Project: SO-directed issues

Runze Lin [1] **and Runfeng Qian**[2]

[1]**Stu# : 20143725**
[2]**Stu# : 20168074**

ABSTRACT

The issue tracker on GitHub is designed to track ideas, enhancements, tasks, and bugs for software project maintainers to address. In contrast, Stack Overflow is a Q&A forum intended for specific programming questions from programmers of all levels of expertise. Despite the clear differences between these two platforms, a common issue arises when questions more suited for Stack Overflow, such as those asking how to implement a function using a specific library, are posted on GitHub's issue tracker. Developers often redirect these support questions to Stack Overflow as they do not report bugs or propose features. These Stack Overflow-directed issues consume developers' time and leave the original question unanswered.

The study investigates the characteristics and trends of Stack Overflow-oriented questions on GitHub's issue tracker and developed several deep learning models to classify questions posted in the future automatically. The best model got a test accuracy of up to 76.2%. the results of this study emphasize the importance of properly categorizing questions on online platforms. By utilizing deep learning models to classify questions automatically, developers can more efficiently address issues and ensure that questions are directed to the appropriate platform. This, in turn, can lead to quicker and more effective resolutions to programming-related issues.

Keywords: GitHub, Stack Overflow, Text Classification, Deep learning

1 INTRODUCTION

The rapid growth of software development and open-source projects has led to the emergence of platforms such as GitHub and Stack Overflow, which serve distinct but complementary purposes in supporting developers. GitHub is a version control and collaboration platform primarily focused on managing software projects, where its built-in issue tracker allows developers to keep track of ideas, enhancements, tasks, and bugs that need to be addressed. In contrast, Stack Overflow is a popular Q&A forum designed to facilitate knowledge sharing among programmers, catering to a wide range of expertise levels and providing specific programming/usage solutions. However, the boundaries between these two platforms can sometimes become blurred, leading to the posting of questions more appropriate for Stack Overflow on GitHub's issue tracker. This phenomenon is the result of a variety of factors, such as misinformation, lack of education, inner-project culture, and misbehavior.

The misplacement of Stack Overflow-related questions on GitHub's issue tracker poses several challenges for software project maintainers. First, it consumes valuable time and resources as developers need to redirect these support questions back to the appropriate platform. Second, it leaves the original question unanswered and may deter the asker from seeking help on the correct platform. Finally, it could clutter the issue tracker, making it difficult to manage and prioritize genuine issues that require the developers' attention.

To better understand the nature of this problem, this study investigates the characteristics and trends of Stack Overflow-oriented questions posted on GitHub's issue tracker. The objectives of this research are to (1) identify the common features and key phrases of such questions, (2) apply an unsupervised clustering algorithm to label the dataset automatically, and (3) explore potential solutions to streamline the issue-triaging process with deep learning NLP models. In pursuit of these objectives, we have employed several deep learning models from bi-directional LSTM to different variations of Bert to automatically

classify questions as Stack Overflow-related or GitHub-appropriate, with the best model achieving a test accuracy of up to 82.6%.

The findings of this research are expected to contribute to the development of practical tools and strategies that can assist software project maintainers in efficiently managing their issue trackers. Furthermore, this study aims to provide insights into the broader challenges associated with question misplacement across different platforms and the implications for knowledge sharing in the software development community.

2 BACKGROUND

a simple examination of the issue tracker in large-scale GitHub projects, using a straightforward keyword search for 'Stack Overflow,' reveals a plethora of results. Upon closer inspection, it remains unclear what percentage of these results correspond to SO-directed issues. However, a review of a random sample of these issues indicates that numerous instances of this phenomenon are present. In response to such support-related questions, developers frequently employ generic responses directing users to Stack Overflow. Consequently, developers divert their attention away from resolving pressing software issues, while users must also invest additional effort to obtain answers to their questions. To avoid the human capital cost of this process entirely, this paper aims to enhance our understanding of Stack Overflow-centric issues within GitHub's issue tracker and strives towards developing an effective solution.

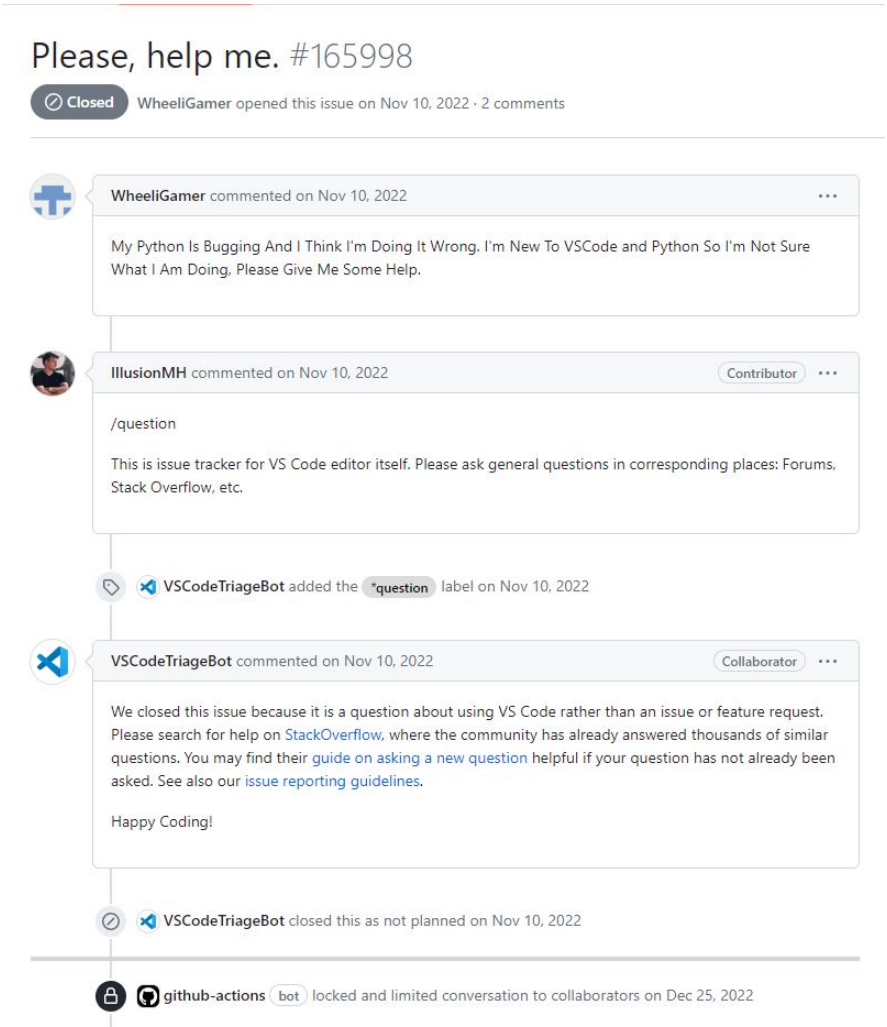


Figure 1. Motivation Example of a Stack-Overflow Oriented Issue within Visual Studio Code's Issues Tracker.

To illustrate this issue, consider the example depicted in Figure 1, taken from Microsoft's Visual Studio Code project on GitHub. In this instance, a user has submitted an issue to Microsoft's VScode project due to their lack of familiarity using Visual Studio Code and Python. Subsequently, a Microsoft software engineer reviews the user's query and labels it as a '*question'. The VScode triage-bot then directs the user to Stack Overflow. Although large projects like Visual Studio Code have a reasonably effective automated triage system, typically based on naïve bayes or support vector machines (SVM)(Xuan et al., 2010), to manage such issues, many medium and small-scale projects lack efficient methods for doing so. In these smaller projects, a rudimentary and often unreliable key phrase recognition-based approach may be employed (Moin and Neumann, 2012), or worse, developers must manually classify, monitor, and respond to each issue akin to the vscode-triage-bot. By thoroughly examining this problem, this paper endeavors to enhance the experiences of both users and developers and contribute to reducing the prevalence of Stack Overflow-style questions being erroneously reported as issues on GitHub.

This study adopts a variety of machine learning algorithms, ranging from clustering techniques to deep learning natural language processing models, to address the problem of Stack Overflow-oriented questions on GitHub's issue tracker.

In the data labeling process, SentenceTransformers(Reimers and Gurevych, 2019) was used to calculate the similarity between comments. SentenceTransformers is a Python framework for cutting-edge text, picture, and phrase embeddings. It can calculate embeddings for more than 100 languages and is based on transformer networks such as BERT and RoBERTa. These embeddings may be applied to tasks like mining paraphrases, semantic textual similarity, and semantic search. The framework also makes it simple to fine-tune unique models for certain use-cases. The publication Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks introduces sentence transforms.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)(Ester et al., 1996), used in the dataset automatic labeling process, is a popular clustering algorithm that separates high-density clusters from low-density clusters in a spatial dataset. It is robust to outliers and can identify arbitrarily shaped clusters. However, it is sensitive to the selection of its two key parameters, min_sample and eps, and struggles with datasets having similar density clusters. Additionally, it struggles with high dimensionality data.

We choose BERT and LSTM(Hochreiter and Schmidhuber, 1997) as our deep-learning natural language processing models.
The BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model that can be fine-tuned for various natural languages processing tasks(Devlin et al., 2018), such as text classification, question answering, sentiment analysis, and more. BERT leverages the power of transformers, which are neural network architectures that can encode both the left and right context of a word simultaneously, unlike traditional sequential models. BERT also uses a large corpus of unlabeled text data to learn general linguistic features that can be transferred to downstream tasks.

BERT is chosen as our primary model (Ezen-Can, 2020)as it can capture complex and long-range dependencies in natural language, which are often crucial for understanding the meaning and intent of a text. BERT also reduces the need for task-specific architectures and data preprocessing, as it can handle various types of inputs and outputs with minimal modifications. Furthermore, BERT has achieved state-of-the-art results on several benchmark datasets, demonstrating its effectiveness and versatility. However, BERT also has some limitations and challenges. One of them is the computational cost and memory requirement of training and deploying BERT models, which are very large and resource intensive. Another challenge is the explain ability and interpretability of BERT models, which are often regarded as black boxes that produce outputs without revealing how they arrived at them. Moreover, BERT may not be able to capture some domain-specific or low-frequency features that are important for certain tasks or domains, requiring further adaptation or augmentation.

On the other hand, Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is capable of learning long-term dependencies. It is explicitly designed to avoid the long-term

dependency problem and remember information for longer periods of time. However, compared to BERT for text classification tasks, experiment results have shown that bidirectional LSTM models can achieve significantly higher results than a BERT model for small datasets and can be trained in much less time than tuning pre-trained counterparts. Meanwhile, BERT has been shown to have better performance on larger datasets. After experimenting with both models, we found that BERT showed better results for our datasets and dropped LSTM for our final model.

3 METHODOLOGY

3.1 Data Collection

GitHub projects were selected based on scale, which was measured according to the quantity of stars assigned to projects by GitHub users. This process led to the selection of the following projects: Flutter, Spring Boot, Tensorflow, React Native, and Vscode. To develop a data set, all of the closed issues from each of these projects were mined. the methodology employed for mining issues and comment data from the GitHub repository employs the GitHub GraphQL API. The pipeline consists of several key steps, including data fetching, processing, and storage.

3.1.1 Data Fetching

The GitHub GraphQL API is leveraged to fetch the required data for issues and their corresponding comments in a target repository. A custom GraphQL query is designed to extract issue-related information such as URL, title, labels, body, state, creation and closing times, author details, and comments. To avoid reaching the rate limits imposed by GitHub, multiple API tokens are utilized in a rotation strategy. The miner iterates through a list of issue IDs, fetching the respective data using the constructed GraphQL query.

3.1.2 Data Processing

Once the JSON data is fetched from the GitHub API, it undergoes a series of processing steps. With the following attributes extracted: project name, issue ID, state, creation time, closing time, author name, author ID, issue title, labels, total labels, total comments issue body and comment body. All Issue attributes are then combined into a comma-separated string, with newline and carriage return characters removed from issue and comment bodies to ensure proper formatting.

3.1.3 Data Storage

The processed data is then stored in two separate CSV files. One file contains the issue data with each row representing an issue and its attributes, while the other file contains the comment data with each row representing a comment and its attributes. This storage format facilitates further analysis and manipulation of the data using various data analysis tools and libraries.

Later in the project, we made use of some stack overflow datasets, as we found that they could help us train classifying models to a higher accuracy. We obtained this dataset as a pre-mined dataset from a former study done by graduate student Cruse.

3.2 Data labeling

As mentioned in the background section, the simplest way of finding SO-directed issues is to do a simple keyword search. However, just because stack overflow is mentioned in the comment does not mean it is a SO-directed issue. For example, the sentence ”you should ask it on stack overflow” is SO-related, while ”This is a stack overflow error” is not. Similarly, “so” can be mistakenly written as “SO” like in this example “SO, otherwise, I guess then it just might be my system”. This creates a lot of incorrect labels, which if fed into the supervised classification model will drastically reduce validation accuracy and test accuracy. In response, we propose using an unsupervised classification model to get further rid of the incorrect labels.

The final labeling algorithm we designed consists of these steps: First, replace all stack overflow URLs with the string “URL”, since all stack overflow URLs contain the word “stackoverflow” which might hinder the clustering performance. Second, fuzzy search is used to filter out all comments containing SO-key words. Third, we extract a window from all the comments containing SO keywords, using a sliding window size of 5 to capture 5 tokens before and after the word ”Stack Overflow”. For example,

from the comment ”Our models are evaluated extensively and achieve better ask on Stack Overflow, let’s see what happens in the new version”, we extract the phrase ”and achieve better ask on Stack Overflow, let’s see what happens in”. Four, we compute the similarity(Yasmin et al., 2022) score matrix of all the extracted phrases using SentenceTransformers. Then we apply unsupervised clustering using DBSCAN, an algorithm that can group similar phrases together. By doing so, we can obtain a set of positive phrases that are related to Stack Overflow. After manually checking a sample of 200 phrases (20 positive and 20 negative labels from each project), we noted that our unsupervised model has an accuracy of 98% for the positive labels.

In the end, the issues with positive unsupervised clustering labels became our positive data set. Meanwhile, all issues with strictly no SO keywords became the negative dataset.

Through trial and error, we have determined many hyperparameters for the automatic labeling process. The following paragraph explains how we selected these hyperparameters.

To perform the fuzzy search algorithm, several keywords relating to the topic of ’stack overflow’ are used, such as: ’ SO ’, ’so question’, ’so issue’, and ’so forum’. possible spelling errors are accounted for by allowing a maximum Levenshtein distance of a set amount. This Levenshtein distance is determined by experimenting with different values from 1 to 4 and evaluating the results manually. We found that the algorithm achieved the best performance with a maximum Levenshtein distance of 2. To optimize the performance of DBSCAN, we conducted a series of experiments with different values of the window size and the algorithm’s hyperparameters. Based on the results, we selected the optimal configuration as follows: window size = 5, eps=0.5, min_samples=5, metric=’precomputed’.

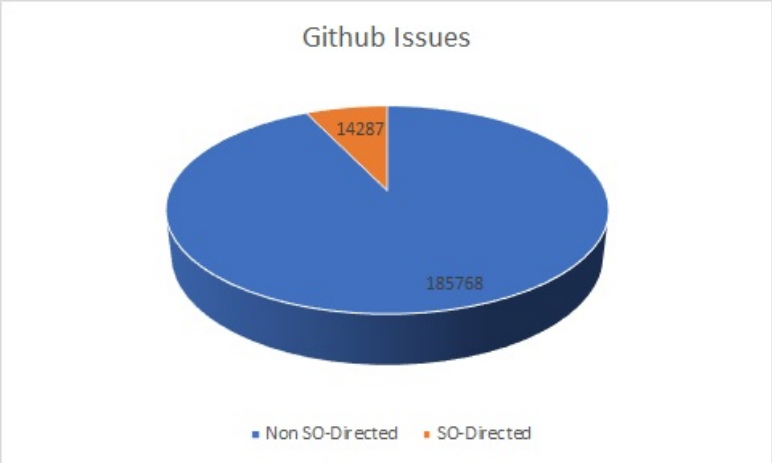


Figure 2. Total number of SO and non-SO directed issues.

Table 1. Table showing the number of SO and non-SO directed issues of each project

Project	Non-SO-Directed	SO-Directed	Total Issue
Flutter	293348	6075	299423
Facebook	85617	5182	90799
Spring-Boot	54337	2998	57335
Tensorflow	171651	4939	176590
VSCode	242016	1704	243720

3.3 Data Preprocessing

In the previous section, we applied data labeling to GitHub issues and obtained a positive set and a negative set. After merging all project datasets, we noted that the positive set comprised 14287 distinct

issues, while the negative set comprised 185768 distinct issues. We merged these sets into a data frame with three columns: the question title, question body, and label. We then randomly sampled 2500 data points from each set and used them as our negative and positive testing datasets. All our testing data points originated from GitHub.

For the positive training and validation dataset, we randomly sampled 7500 data points from the remaining positive set and combined them with another 7500 data points that we collected from the Stack Overflow forum. After reviewing some of the data, We found that GitHub SO-directed questions were essentially similar to SO questions, except that they might exhibit some features of GitHub issues. Therefore, we mixed these two sources to facilitate the model's learning of the different characteristics between the GitHub SO-directed questions and the non-SO-directed questions. However, our test set only contained issues from GitHub, since the classifier was designed to operate on GitHub. For the negative training and validation dataset, we randomly sampled 15000 data points from the remaining negative set.

In order to prepare the data for the model, we performed data cleaning on both the positive and negative datasets. The data we collected from the website contained many symbols and words that were not meaningful or recognizable for the model, such as HTML tags, URLs, parentheses, and punctuation marks. These symbols and words could introduce noise and bias in the data and reduce its usefulness for sentiment analysis. Therefore, we applied some data-cleaning techniques to preprocess the data before feeding it to the model. We used some Python libraries, such as 'nltk', to remove HTML tags, URLs, parentheses, and punctuation marks from the data. We also used the Python library 'wordninja' to split the words that were concatenated without spaces. For example, the word 'loveit' was split into 'love' and 'it'. This way, we ensured that each word in the data was meaningful and recognizable for the model. By applying these data-cleaning techniques, we improved the quality and readability of the data and prepared it for further analysis.

3.4 Model training

An LSTM model was implemented as a baseline to compare with our main BERT model; using TensorFlow, the model was designed with two inputs representing a question's title and a question's body. Please see Figure 10. While the network treats both of these inputs identically, they are processed separately prior to converging. The data is first transitioned through a Keras embedding layer so that the model can learn an embedding for the words within the training data. A bidirectional LSTM is then used to read the sequences in both directions and learn meaning from the data. Finally, the data is flattened, transitioned into a dense layer, concatenated, and directed into a single dense unit, where sigmoid activation represents a given issue's assigned class.

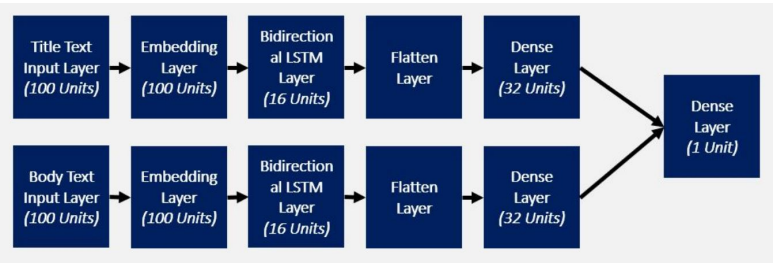


Figure 3. LSTM Issue-Classification Mode.

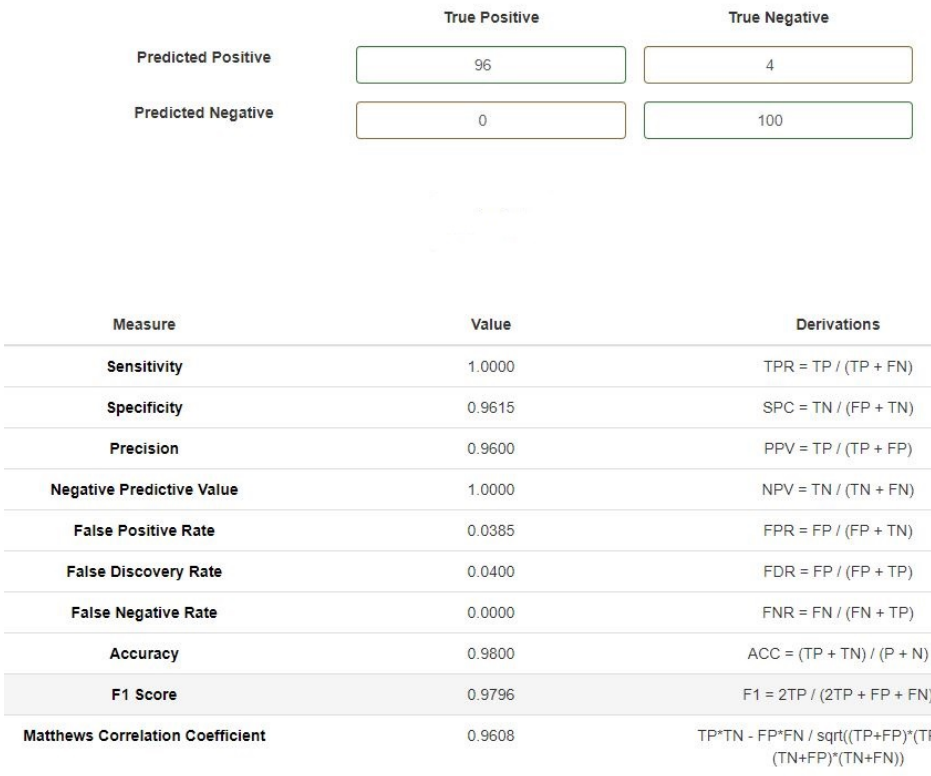
By utilizing the described deep-learning approach, the model successfully learned to classify Stack Overflow-oriented GitHub issues with 69% accuracy. Although inferior to BERT, the LSTM model runs much faster and requires way less computing power.

Besides LSTM, we also applied a fine-tuning technique to a pre-trained BERT model (bert-base-cased). We used 10% of the dataset as our validation set to monitor the model performance. The fine-tuning process consisted of adding a dropout layer and a linear layer with a ReLU activation function to the pre-trained BERT model. The input size of the linear layer was 768, and the output size was 2, corresponding

to the two classes of the text classification task. The dropout rate of the dropout layer was 0.5 to prevent overfitting. We used cross entropy as the loss function and Adam as the optimizer. We set the learning rate to 1e-6 and trained the model for 6 epochs. We stopped the training if we observed overfitting on the validation set.

4 EXPERIMENTS AND RESULTS

4.1 labeling result



	True Positive	True Negative
Predicted Positive	96	4
Predicted Negative	0	100

Measure	Value	Derivations
Sensitivity	1.0000	TPR = TP / (TP + FN)
Specificity	0.9615	SPC = TN / (FP + TN)
Precision	0.9600	PPV = TP / (TP + FP)
Negative Predictive Value	1.0000	NPV = TN / (TN + FN)
False Positive Rate	0.0385	FPR = FP / (FP + TN)
False Discovery Rate	0.0400	FDR = FP / (FP + TP)
False Negative Rate	0.0000	FNR = FN / (FN + TP)
Accuracy	0.9800	ACC = (TP + TN) / (P + N)
F1 Score	0.9796	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.9608	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Figure 4. LSTM Issue-Classification Model.

After manually checking a sample of 200 phrases (20 positive and 20 negative labels from each project), we noted the results down and plotted a confusion matrix. As you can see from the figure above the labels created after unsupervised clustering achieved an accuracy of 98 percent, with sensitivity, specificity, and precision well over 90 percent.

4.2 Supervised Classification Result

During our initial experimentation, we observed that the BERT model outperformed the bidirectional LSTM model by achieving 5 percent higher test accuracy (0.725 vs 0.690) on the same dataset. However, it is important to note that both accuracies are relatively low. Due to the superior performance of the BERT model, we decided to proceed with it, despite the fact that it demands more resources and takes considerably longer to train. Specifically, the BERT model required 4 hours to train on Google Colab's fast GPU runtime, whereas the LSTM model trained in just 10 minutes using our local machine's CPU.

For our final BERT classification model, we explored an innovative approach by incorporating Stack Overflow questions directly into the positive dataset. This strategy resulted in a modest increase in test accuracy by less than 5 percent. In the final model, we achieved a test accuracy of up to 0.762, as illustrated in the training graph presented below. It is noteworthy that during the training of the final neural network, we observed overfitting beyond the 4th epoch, with the highest validation accuracy reaching

0.819. This observation informed our decision to halt the training process at the 6th epoch to prevent further overfitting and ensure robust performance on unseen data.

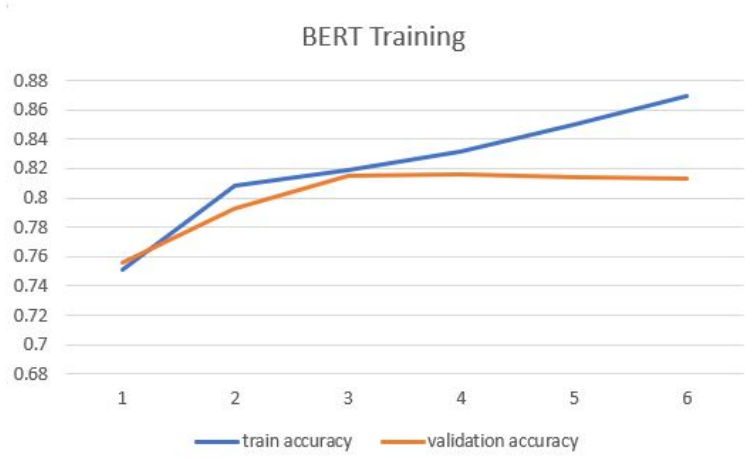


Figure 5. Training graph of the finalized BERT model.

4.3 Discussion

In conclusion, although the accuracy achieved by our NLP classification model is not sufficient for deployment as a real-world, fully automated classifier for identifying Stack Overflow-related questions within GitHub issue trackers, it demonstrates a reasonable level of performance to serve as an advisory tool. We believe that this model can offer valuable assistance to users during the process of posting issues on GitHub.

By implementing our model as an advisory mechanism, users can receive suggestions on whether their questions might be better suited for Stack Overflow, potentially reducing the volume of misplaced questions and improving the overall efficiency of both platforms. This integration can help users make informed decisions about where to post their questions, allowing them to receive appropriate support and guidance from the respective communities.

4.4 Threats to validity

One potential explanation for the limited accuracy of our NLP classification model is the presence of noise in the crawled dataset. Although we were able to remove HTML tags, URLs, and miscellaneous symbols from the data, the nature of GitHub issues often includes numerous code snippets. Consequently, the questions contain a significant amount of non-natural language and model-unrecognizable words and semantics, which might impede the model's learning capabilities.

In the context of an NLP classification task, the presence of these non-natural language elements can create additional challenges for the model, as it attempts to derive meaningful patterns from the text. The noise introduced by the code snippets may obscure the true linguistic structures and relationships within the questions, making it more difficult for the model to accurately identify and classify the issues.

To address this issue and potentially improve the model's accuracy, future research could explore the following strategies:

1. Enhancing the pre-processing steps to filter out or isolate code snippets more effectively, enabling the model to focus on the natural language components of the questions.
2. Developing specialized tokenization or embedding techniques that can better handle mixed natural language and code snippets, allowing the model to understand the context of both types of content.

3. Investigating alternative model architectures or training strategies that are specifically designed to work with noisy, mixed-content data, such as code-augmented language models or models that incorporate attention mechanisms to focus on relevant portions of the text.

By considering these approaches, we may be able to massively improve the performance of our NLP classification model and achieve a higher level of accuracy in classifying GitHub issues.

Apart from the challenges previously discussed concerning noise in the crawled dataset, another possible factor contributing to the low accuracy of our classifier model is the inherent semantic similarity between Stack Overflow-directed and GitHub issues. Differentiating these two types of issues can be challenging, as programming and usage questions more suitable for Stack Overflow might closely resemble ideas and bugs posted on the GitHub issue tracker when analyzed by an NLP model. Both types of questions address programming challenges, troubleshooting, and problem-solving within the realm of software development. Consequently, an NLP model may have difficulty discerning between the two forms of content, as they share commonalities in keywords, terminology, and programming languages. The linguistic patterns and code snippets frequently encountered on both platforms bear a striking resemblance, making it challenging for NLP models to accurately differentiate between Stack Overflow questions and GitHub issues without a deeper contextual understanding of the project or topic.

To classify questions precisely as either Stack Overflow-directed or GitHub-focused, the model would need to possess a deeper understanding of the specific GitHub project in question, similar to the knowledge that a project contributor or employee would have. This level of understanding would enable the model to better discern subtle differences in context, scope, and relevance between the two types of issues, thus improving its classification accuracy.

However, achieving this level of understanding presents an impossible challenge for a general-purpose NLP model, as it requires the incorporation of domain-specific knowledge and contextual information related to each GitHub project. Ultimately, a precise classification model can be most effectively implemented at the project level, with additional metadata and contextual information about the GitHub project also as a part of the input.

Lastly, the current study focuses on large-scale projects, which may introduce potential biases in understanding user and developer behaviors compared to smaller-scale projects. For instance, developers working on smaller or newer projects might be more receptive to Stack Overflow-oriented questions on GitHub issues, given the absence of a well-established support community on Stack Overflow. Moreover, the response time for questions on Stack Overflow and GitHub issues may vary across projects.

Furthermore, it has been observed that developers occasionally redirect users to Stack Overflow while still providing an answer to the user's question on GitHub issues. This phenomenon suggests that, in some cases, addressing Stack Overflow-oriented questions on GitHub issues could be beneficial for users despite causing inconvenience to project developers. Consequently, the generalizability of the findings from this study to a broader range of projects should be considered with caution, as the observed trends and behaviors may not be universally applicable.

5 CONTRIBUTIONS

We collaborated on various stages of the research project. First, we jointly performed data crawling to collect relevant texts from online sources. Second, we applied keyword searching and finding techniques to identify the key terms and topics in the texts. Third, Runfeng extracted phrase windows from the texts based on the keywords, while Runze implemented DBSCAN, an unsupervised clustering algorithm, to group the texts into different clusters. Fourth, we conducted data cleaning to remove noise and errors from the texts, using multiple methods such as spelling correction, punctuation removal, and stopword filtering. Finally, we trained two different models to analyze the texts: Runfeng used LSTM, a recurrent neural network architecture, and Runze fine-tuned BERT, a pre-trained language model.

6 LIST OF DELIVERIES

The code for our project is publicly available in the following GitHub repository: https://github.com/Runze-Lin/SO-directed-issue

The project consists of several steps: data collecting, data labeling, data preprocessing, LSTM training, and BERT finetuning. Each step is implemented in a separate Python or Jupyter Notebook file, as described below.

- The file collectPRinfo.py performs data crawling from GitHub pull requests and stores the results in txt files.
- The file USC_labeling.ipynb performs data labeling using a heuristic based on the pull request status and comments. It takes the txt files from the previous step as input and outputs a CSV file with labels.
- The file training_dataset_generation.ipynb performs data preprocessing on the labeled CSV file and the original txt files. It splits the data into training, validation, and testing sets, and applies some data-cleaning techniques such as tokenization and punctuation removal.
- The file LSTM_dataset_gen.ipynb performs additional preprocessing on the training set for the LSTM model.
- The file LSTM_model.ipynb implements and trains an LSTM model on the preprocessed training set. It evaluates the model on the validation and testing sets and reports the performance metrics.
- The file BERT_colab_final_version.ipynb implements and finetunes a BERT model on the original training set (from training_dataset_generation.ipynb). It evaluates the model on the validation and testing sets and reports the performance metrics.

Due to the size limitation of GitHub, we have uploaded the crawled data to Google Drive: https://drive.google.com/drive/folders/1TEemMON2rPt9iJ1zhVlNODNy2CxMkLHm?usp=sharing. We have also uploaded the training dataset file to Google Drive: https://drive.google.com/drive/folders/1PYgi42qvHJDgDwEr-SxJmhvPPXwl81nI?usp=sharing. This file contains the labeled data.

7 CONCLUSION

Our study has successfully explored the prevalence of Stack Overflow-oriented questions being posted on GitHub's issue tracker and has made significant strides in understanding the characteristics and trends associated with this phenomenon. Through the development of deep learning NLP models, the research achieved a test accuracy of up to 76.2% in classifying Stack Overflow-related questions, demonstrating a promising direction for future advancements.

The findings of this research are expected to contribute to the development of practical tools and strategies that can assist software project maintainers in efficiently managing their issue trackers. By implementing the proposed model as an advisory tool, software maintainers can streamline the issue-triaging process, reducing the volume of misplaced questions and thereby improving the overall efficiency of both GitHub and Stack Overflow platforms.

Future research directions include refining the classifier's performance by addressing the challenges posed by noise in the dataset, exploring alternative model architectures or training strategies, and incorporating domain-specific knowledge and contextual information.

In conclusion, this research has made substantial progress in tackling the issue of question misplacement across different platforms, providing valuable insights and practical solutions that can benefit both software project maintainers and developers in managing their issue trackers more efficiently. The positive results achieved in this study demonstrate the potential of deep learning models to revolutionize issue-triaging processes and contribute to a more effective and collaborative software development ecosystem.

REFERENCES

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

Ezen-Can, A. (2020). A comparison of lstm and bert for small corpus. *arXiv preprint arXiv:2009.05451*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Moin, A. and Neumann, G. (2012). Assisting bug triage in large open source projects using approximate string matching. In *Proc. 7th Int. Conf. on Software Engineering Advances (ICSEA 2012). Lissabon, Portugal*.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Xuan, J., Jiang, H., Ren, Z., Yan, J., and Luo, Z. (2010). Automatic bug triage using semi-supervised text classification. In *SEKE*, pages 209–214.

Yasmin, J., Sheikhaei, M. S., and Tian, Y. (2022). A first look at duplicate and near-duplicate self-admitted technical debt comments. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 614–618.