GitHub issue classification using BERT-style models

Conference Paper · May 2022

DOI: 10.1145/3528588.3528663

2 authors:

Shikhar Bharadwaj
Google Inc.
14 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Tushar Kadam
Indian Institute of Science Bangalore
1 PUBLICATION **16** CITATIONS
SEE PROFILE

GitHub Issue Classification Using BERT-Style Models

Shikhar Bharadwaj*
shikharb@iisc.ac.in
Indian Institute of Science
Bengaluru, Karnataka, India

Tushar Kadam*
tusharpk@iisc.ac.in
Indian Institute of Science
Bengaluru, Karnataka, India

ABSTRACT

Recent innovations in natural language processing techniques have led to the development of various tools for assisting software developers. This paper provides a report of our proposed solution to the issue report classification task from the NL-Based Software Engineering workshop. We approach the task of classifying issues on GitHub repositories using BERT-style models [1, 2, 6, 8]. We propose a neural architecture for the problem that utilizes contextual embeddings for the text content in the GitHub issues. Besides, we design additional features for the classification task. We perform a thorough ablation analysis of the designed features and benchmark various BERT-style models for generating textual embeddings. Our proposed solution performs better than the competition organizer's method and achieves an F_1 score of 0.8653. Our code and trained models are available at https://github.com/Kadam-Tushar/Issue-Classifier.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*; • **Computing methodologies** → *Natural language processing*.

KEYWORDS

NLP, BERT, text classification

ACM Reference Format:
Shikhar Bharadwaj and Tushar Kadam. 2022. GitHub Issue Classification Using BERT-Style Models. In *The 1st Intl. Workshop on Natural Language-based Software Engineering (NLBSE'22), May 21, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3528588.3528663

1 INTRODUCTION

Issue report classification is a critical task for managing development on a GitHub repository. Developers open issues to ask questions, report bugs and request enhancements. In a collaborative development environment, such as GitHub, it becomes imperative for the project manager to tag these issues with appropriate labels for filtering and to prioritize them appropriately. The labelling task is arduous and should be automated. The goal of the issue report classification task in the NLBSE 2022 workshop is to develop a tool for automatically labelling the GitHub issues to ease this laborious task. Next, we describe the classification problem in detail.

Problem Description. Given data about a GitHub issue such as the text in the issue title and body, issue URL, issue author association, repository URL, and the issue creation date; the task is to design a classifier that can correctly label the issue as one of the following types - *Question, Enhancement* or *Bug*.

The following section describes our approach to solving the issue report classification task and the associated training and evaluation pipeline. Then we discuss the results and analyze the embeddings generated by our model in Section 3.

2 METHOD

We formulated the problem as a standard multi-class classification task and used the cross-entropy loss to train all our models. We performed Exploratory Data Analysis (EDA) on the provided dataset [3–5] to design distinguishing features for the classification task. The features used are *Early issue*, *Owner opened* and *Question title*.

- **Early Issue**: We parsed the issue URL to obtain the issue number. This number denotes the serial order of the issue. From the EDA, we observed that early issues (those with smaller issue numbers) had more enhancement requests. At the same time, the issues with large issue numbers had comparatively more questions and bugs reports. Therefore, we designed a binary feature obtained from the issue creation date. A value of 1 for this feature denotes that the issue is an early issue based on some threshold on the issue number. We used 98 as the threshold for generating this binary feature.
- **Owner Opened**: We found that project owners themselves authored most of the issues that asked for enhancement. Therefore we designed a binary feature to indicate if the issue is created by the project owner.
- **Question Title**: Since the data for the *question* class is less compared to the other two classes, we used simple natural language processing techniques to create a binary feature that denotes whether sentences in the issue title are questions.[1]

From the EDA, we also discovered that most of the issues had a body of length less than 512 words and a title of length less than 50 words. Therefore we employed pre-trained BERT-style models to capture context from the issue body and title without worrying about the text length constraints for such models.

2.1 Architecture

Our solution is based on BERT-style models. These are large language models pre-trained on a vast amount of textual data in a

*Both authors contributed equally to this research.

[1]We use the code from https://github.com/kartikn27/nlp-question-detection.

NLBSE 2022, May 8, 2022, Virtual Event
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9343-0/22/05…$15.00
https://doi.org/10.1145/3528588.3528663

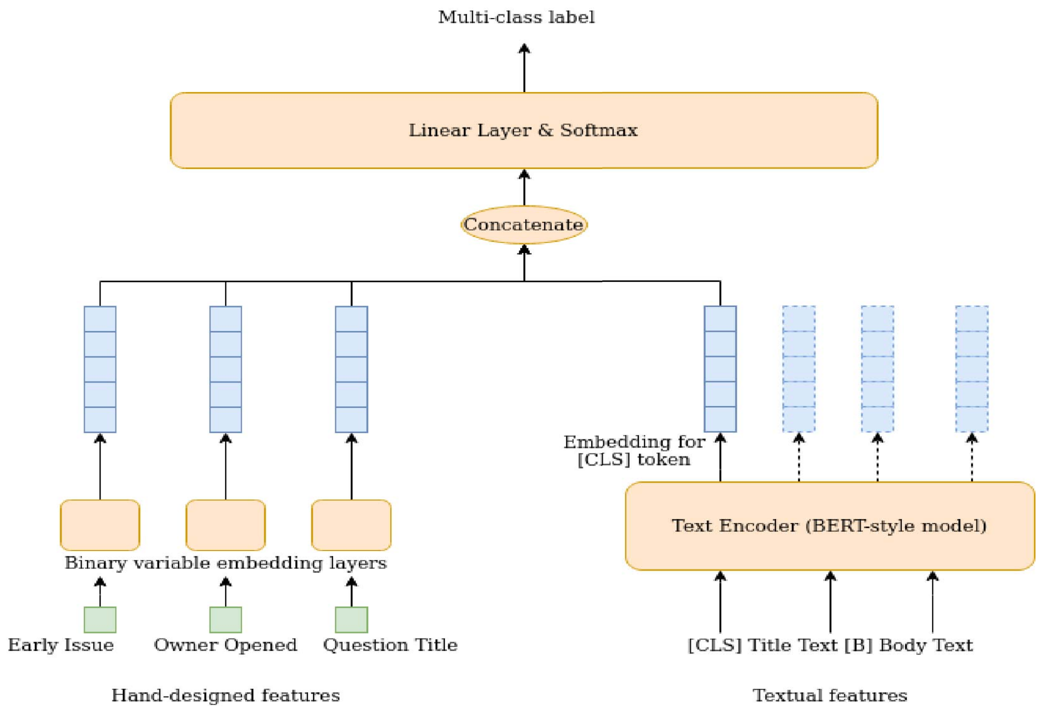


Figure 1: Proposed solution architecture.

self-supervised way using a language modelling objective. We use such pre-trained models in our solution. The complete solution architecture is shown in Figure 1. We performed experiments with various BERT-style models (listed in Table 2) to capture context from the textual content in the issue. The input to this BERT-style model is a text sequence composed of the issue title and the issue body appended to the `[CLS]` token, as shown in Figure 1. The embedding for `[CLS]` token is then passed through a dropout layer. We embed the hand-designed features (described above) and concatenate these embeddings to the embedding of the [CLS] token. This concatenated vector passes through a linear layer, and finally, a softmax function is applied to generate probabilities for each class. We use the standard cross-entropy loss for optimizing our neural network.

2.2 Data Preparation.

Data Split. The competition organizers provide data that is already split into training and testing folds. We sample 1000 examples from the training data to create a validation fold for tuning the hyper-parameters of our model. The sampling is stratified, assuring that the distribution of classes is maintained in the validation fold.[2]

Data Preprocessing. By examining a few examples from the dataset, we observe that the issue bodies contain non-natural language text like code snippets and URLs. We use regex based substitution for replacing such content with their semantic type as follows

- Code snippets are replaced by the token `[CODE]`.
- URLs are replaced by the token `[URL]`.
- @username is replaced by the token `[USER]`.
- > symbol is removed from the beginning of lines in the issue body.
- all numbers are replaced by the token `[NUMBER]`.
- inline code blocks are replaced by the token `[CODE_INLINE]`.
- newlines are replaced by spaces, and
- multiple spaces are replaced by a single space.

2.3 Hyperparameters

Table 1: Hyper-parameters for our best model.

Hyper-parameter	Value
Batch size	16
Learning rate	1e-05
Dropout	0.24
Issue number threshold	98
Feature embedding dimension	8
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	1e-08

We use the Adam optimizer for optimizing our model. We tune the learning rate on the validation set and found that a value of 1e-05 performs best on this dataset. We do not use a learning rate scheduler for any of our models. We set β_1 and β_2 to default values.

[2]The code for creating the validation set is at https://github.com/Kadam-Tushar/Issue-Classifier/blob/main/notebooks/data_split.ipynb.

Table 2: Results on the test set. Rows are named by the BERT-style model that is used as Text Encoder in the Figure 1.

	Bug			Enhancement			Question			
Model	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	**Global** F_1
FastText [3–5]	0.8314	0.8725	0.8515	0.8155	0.8464	0.8307	0.6521	0.3502	0.4557	0.8162
BERT [1]	0.8751	0.8922	0.8836	0.8666	0.8711	0.8688	0.6600	0.5702	0.6118	0.8554
XLNet [8]	0.8794	0.8850	0.8822	0.8533	0.8906	0.8716	0.7066	0.5349	0.6089	0.8568
CodeBERT [2]	**0.8830**	0.8941	0.8885	0.8666	**0.8912**	0.8787	0.6937	0.5512	0.6143	0.8629
RoBERTA [6]	0.8747	0.9069	0.8905	0.8780	0.8749	0.8765	0.6931	**0.5581**	**0.6183**	0.8633
RoBERTA w/o Issue Body	0.8380	0.8731	0.8552	0.8142	0.8619	0.8374	0.6825	0.3300	0.4449	0.8211
RoBERTA w/o Hand-designed features	0.8727	**0.9119**	**0.8919**	**0.8798**	0.8777	**0.8788**	**0.7149**	0.5391	0.6146	**0.8653**

We train for 4 epochs over the training fold with a training batch size of 16 and check the validation F_1 score after every 10,000 training steps on the validation fold. We report the result in Table 2 using the model that achieves the best validation F_1 score among all models. The issue number threshold and dropout parameter are obtained by tuning on the validation set. However, we do not perform any tuning for the feature embedding dimension. The hyper-parameters for our best model are listed in Table 1.

3 RESULTS

3.1 F_1 Scores

We measure the classification performance of our system using the micro-averaged F_1 score over all three classes. Besides, we also report Precision (P_c), Recall (R_c) and F_1 score ($F_{1,c}$) for each class $c \in \{bug, question, enhancement\}$. Precision and Recall for each class are defined as

$$P_c = \frac{TP_c}{TP_c + FP_c} \qquad R_c = \frac{TP_c}{TP_c + FN_c} \tag{1}$$

Here, TP_c, FP_c and FN_c represent the true-positives, false-positives and false-negatives over class c, respectively. The $F_{1,c}$ score is defined as the harmonic mean between P_c and R_c, for each class c

$$F_{1,c} = 2 \cdot \frac{P_c \cdot R_c}{P_c + R_c} \tag{2}$$

We also use the global Precision (P), Recall (R) and micro-averaged F_1 score. These metrics are defined as follows.

$$P = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c)} \quad R = \frac{\sum_c TP_c}{\sum_c (TP_c + FN_c)} \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3}$$

Since the global metrics are micro-averaged, we have $F_1 = P = R$. Therefore, we only show global F_1 score in the Table 2.

From the results reported in Table 2, we can observe that our best performing system outperforms the baseline score of 0.8162 and achieves **0.8653**. We also note that all the systems perform worst on the class *question*. This can be attributed to the limited number of samples of *question* in the dataset (~8.6%). Our experiments with multiple BERT-style models showed that RoBERTA performs the best. Although CodeBERT is trained on both code and natural language data, it does not perform better than RoBERTA. We hypothesize that this is because the natural language data in CodeBERT's pre-training set is paired with the corresponding code and not the same type as that found in GitHub issues and StackOverflow posts.

The ablation analysis shows that removing text from the GitHub issue body lowers the global F_1 score. Therefore, it is necessary to gather signal from the issue body for achieving the best performance. We also observe that RoBERTA is a powerful model, and it does not need hand-designed features to achieve the best performance.

3.2 Embedding Visualizations

We also show t-SNE [7] and PCA plots for the embeddings learnt by our model in Figure 2. These are the embeddings from the trained model just before the linear layer. It can be observed that the vectors for the *enhancement* class can be clearly separated from the other two classes, but the vectors for the *question* class are intermixed with the vectors for the *bug* class. Hence, we see that it is difficult for the model to learn valuable features for classifying *question* issues.

4 CONCLUSION AND FUTURE WORK

This report described our approach to solving the issue report classification task in the NLBSE 2022 tool competition. We design features by exploring the dataset and using BERT-style models to capture the issue text signal. Finally, we discussed our results and visualized the embeddings obtained by our model.

In future, we would like to experiment with various weighting techniques to tackle the data imbalance problem in this dataset. Perhaps, RoBERTA style training on domain-specific data might help increase the performance. Another promising direction to explore is to ensemble these models.

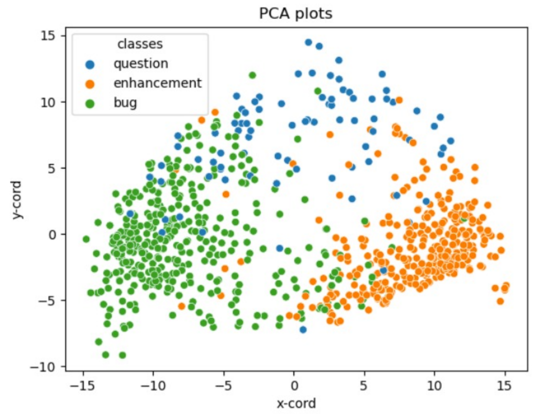
REFERENCES

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[2] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[3] Rafael Kallis, Oscar Chaparro, Andrea Di Sorbo, and Sebastiano Panichella. 2022. NLBSE'22 Tool Competition. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*.

[4] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 406–409.



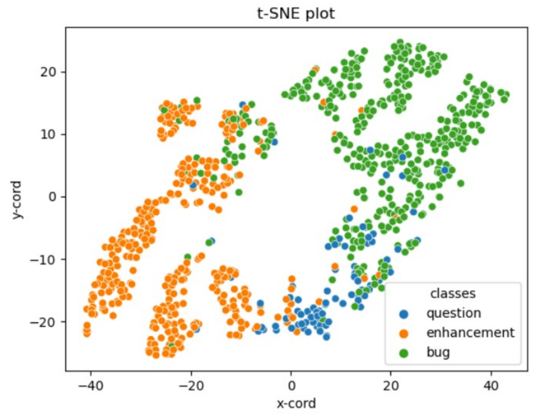


Figure 2: Visualizations of the learnt embeddings.

https://doi.org/10.1109/ICSME.2019.00070

[5] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2021. Predicting issue types on GitHub. *Science of Computer Programming* 205 (2021), 102598. https://doi.org/10.1016/j.scico.2020.102598

[6] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[7] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[8] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).