Predicting Issue Types on GitHub

Sujan Dutta, Varun Tandon, Simran Lalwani

Abstract:

Software maintenance is preeminent to keep up with customer needs and hence for the success of software projects. Issue trackers are tools that address tickets or possible bugs in software. Software developers are expected to react to these issues with the least force and effort, ultimately keeping the costs for maintenance low.

Issue submitters report new issues in GitHub projects by simply providing a title and an optional description of the issue. GitHub also provides a customizable labeling system with easier management and prioritization of issues. Labels assigned to issues, aid in the classification and filtering of reports making issue handling efficient. But labels are not used much as manual labeling is time-consuming, error-prone, and labor-intensive.

For aiding software maintainers, we propose a model that uses the texts obtained from titles and descriptions in the issue reports to predict suitable labels. We also propose the use of domain-specific models for every type of Software Engineering project. Our experiments show that using a domain-specific model for a particular type of project outperforms the model trained on general data.

Problem Statement:

Popular projects on GitHub receive an overwhelming amount (tens of thousands) of weekly issue reports. So naturally, it becomes a huge problem to manage and prioritize the issue reports even with the help of issue trackers. Hence, it is necessary to have a tool that can automatically predict the type of the issue.

In this project, we treat the problem of issue labeling as a multiclass classification problem with the following classes (refer to Tabel. 1. for a snapshot of the data):

1. bug report
2. enhancement
3. questions

Moreover, we only consider the title and description of the reported issues so essentially this is an NLP problem of text classification.

Issues
__label__enhancement Switch Router to Browser Router If we can figure out how to switch to this router, then components will be linked to urls. https://reacttraining.com/react-router/web/api/BrowserRouter
__label__bug Missing dropout serialization
__label__question which types of python -V works this script on Windows 10. cordially. thank you. @Juniorn1003

Table. 1. Snapshot of raw data

Existing Studies:

Yu Zhou et al. [1] propose a multi-stage approach solution for predicting the bug report type. The first stage uses text mining algorithms on the summary part of the bug reports and classifies them into discrete levels. The second stage takes extracted features from stage one and some other structured features of the bug report and feeds them to a bayesian net to classify the type of bug.

Z. Liao et al. [2] analyzed the characteristics of issues reported on popular GitHub repositories to facilitate issue management and software management. They investigated the important degrees of behaviors to identify the common patterns in the issues.

In 2016, researchers at Facebook [3] introduced the fastText classifier that exhibits a similar accuracy as the deep learning models but takes a fraction of the required training time. The authors of the paper in the review used the fastText classifier to deploy a lightweight model.

Proposed Solution:

R. Kallis et al. in [4] proposed a lightweight model using fastText. We wanted to see if increasing the complexity by using a large language model like BERT would improve the performance. The dataset used to train the fastText model was created using GitHub Archive via Google BigQuery. The dataset consisted of a query run

over closed issues between 1st and 9th March of 2018, comprising the same types of issues, with a 49:41:10 ratio, termed as $D_{Unbalanced}$.

During our experiments, we wanted to test the effect of a domain-specific dataset on our model. To curate this dataset, we used the public dataset for the NLBSE 2022 [5] which contained over 900k issue reports (test + train). The domain we chose to work on is Text Editors. After compiling a list of some of the most popular Text Editor repositories on Github, we parsed through the dataset and selected the issues that had matching forked repositories. This dataset will be referred to as $D_{domainSpecific}$. Here, our assumption is if we use data from a particular domain, the model will learn domain-specific patterns that might be harder to learn from a general dataset.

Study Design:

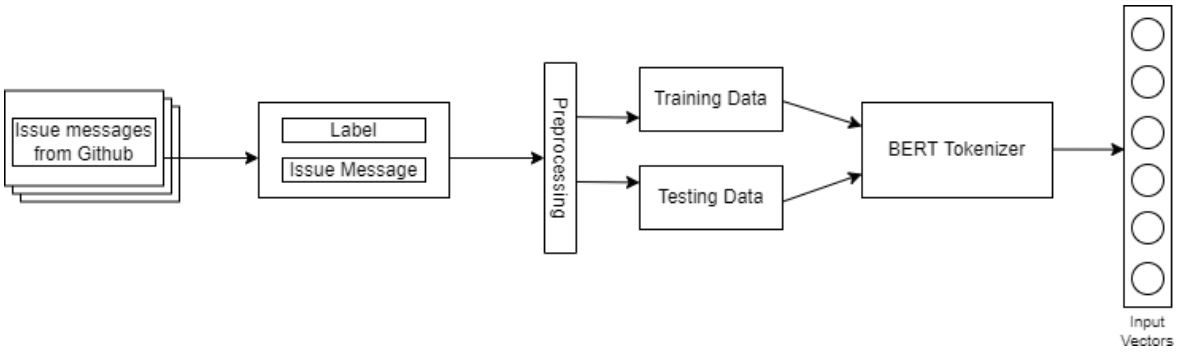

Fig. 1. Preprocessing of the Data

Here, we describe the steps followed in our preliminary study:

1. As mentioned earlier, we are using two datasets, both of which were scraped from the top projects on GitHub. Each issue message is parsed through and issues falling under the predefined categories (“bug”, “enhancement”, “question”) were compiled into a CSV file.

2. Each row of the CSV file contains a string (title and body of an issue) prefixed with “ label [label type]”. We split this label from the remainder of the string. The remaining text needed to be cleaned.

3. We followed a two-step process for cleaning and pre-processing the text. First, we removed (or masked) parts of the text that are uninformative and might lead to noise. It is to be noted that we masked some of the entities with a

generic word as we think they might carry important information about the issue type. Below is the list of entities that were discarded.

- Code lines (removed as we are focusing on only normal text)
- Images/GIFs (masked with the word 'IMAGE')
- Links (masked with the word 'LINK')
- Markdown comments (removed as they are not visible on GitHub)
- Versions like x.y.z (removed as uninformative)
- User tags (masked with the word 'USER')
- Punctuations (removed except '?' as this might help identify the questions)

After the first cleaning step, we removed every instance containing more than 50% of non-dictionary words. This helped us to deal with code-mixing (text containing more than one language) and noisy issue reports. Here is a snapshot of the cleaned data.

Uncleaned Data	Cleaned Data
_label_enhancement Switch Router to Browser Router If we can figure out how to switch to this router, then components will be linked to urls. https://reacttraining.com/react-router/web/api/BrowserRouter	Switch Router to Browser Router If we can figure out how to switch to this router, then components will be linked to urls. LINK

_label_bug dvc cache status: slow multiple files ``` $ dvc cache status (1/7): [##] 100% dog.168.jpg (2/7): [##] 100% dog.173.jpg (3/7): [##] 100% dog.174.jpg (4/7): [##] 100% dog.181.jpg (5/7): [##] 100% dog.19.jpg (6/7): [##] 100% dog.194.jpg (7/7): [##] 100% dog.198.jpg (8/7): [##] 100% dog.205.jpg (9/7): [##] 100% dog.238.jpg [SKIP 500 files overall] new file: .dvc/cache/fd47d523fa82bb9c282b897b55be0442 new file: .dvc/cache/0dd437f2f04fbf54355beab021494baa new file: .dvc/cache/730786e5c78eb2782124e998c802ae9c new file: .dvc/cache/0f69da47a6a31007964f838477825473 ``` It is very slow to check all the files. Any chance to improve the performance?	dvc cache status slow multiple files It is very slow to check all the files. Any chance to improve the performance?
_label_bug `parse_tax_data` should be able to generate default column names If the input to `class_key` is unnamed, I see: ``` > obj <- parse_tax_data(my_data, + class_cols = "org_class", + class_sep = ";", + class_regex = "(.+)_(.+)", + class_key = c("taxon_name", "info")) Show Traceback Rerun with Debug Error: Column 3 must be named ```	parse tax data should be able to generate default column names If the input to class key is unnamed I see

4. As our datasets are heavily imbalanced, we then undersampled the two majority classes (bug, enhancement). So, our final datasets are as follows:

    - $D_{Unbalanced}$: Containing 5000 bugs, 5000 enhancements, and 3000 questions.
    - $D_{domainSpecific}$: Containing 3000 bugs, 3000 enhancements, and 1700 questions.

5. This data is split into the train and test set, which is then tokenized using the BERT Tokenizer. The result of the tokenization is a vector of our input words.

6. This vector is embedded (by BERT embedder) and encoded (by BERT encoder) and finally passed through a softmax layer to classify into one of the three classes. In this phase, we provide a black-box of this approach that we plan to describe in detail in the future phases.

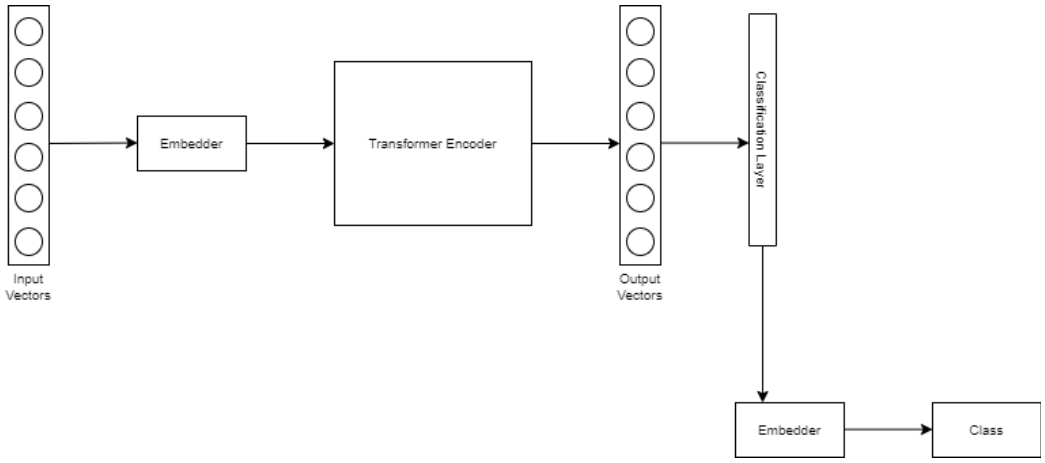

Fig. 3. A Black-box representation of our model

Model Description:

The BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based language model that captures the context in natural language to a great extent. It essentially has 3 layers: Embedding layer, Encoder, and Pooler.

The Embedding layer:

First of all, two tokens ([CLS] at the beginning and [SEP] at the end) are added with the input text. Then the sentence is tokenized into its words. Then this layer produces the individual embeddings from each word of the text input. Apart from the normal embedding it also generates a positional embedding that stores the sequential information of the words. Finally, the embedding layer sums the embeddings, normalizes the summed tensor, and sends it as inputs to the encoder.

The Encoder:

The encoder is at the heart of the BERT. This model is based on the Transformer model architecture. It works on a self-attention mechanism, which grasps the relationships between all words in a sentence, regardless of where they are positioned in the sentence. This method enables the language model to learn the contextual embeddings of the words. For example, consider these two sentences: Sentence 1: I'm gonna book a hotel.

Sentence 2: I'm gonna read a book.

The word 'book' has different contextual meanings in both sentences. While processing a particular word, the model looks at the context and decides how much importance it should give to a particular word by its context. This is done by the method of attention. Each word in the content is given an attention score and the vector embedding of the word is multiplied by the respective attention score. The model learns the attention score while training. This process enables the encoder to analyze the relevant meaning of the word 'book' in the context of the above sentences, based on all the words of the respective sentence. After that, the attention output is passed through multiple fully connected neural layers. Some skip connections are also implemented to preserve the states of previous hidden layers. Finally, the tensor normalization is performed and the output is passed on to the pooler.

The Pooler:

This layer takes only the contextual vector of the first token [CLS]. As it is the contextual vector, it has the information of the whole sentence. Finally, there are dense layers and softmax layers to make predictions.

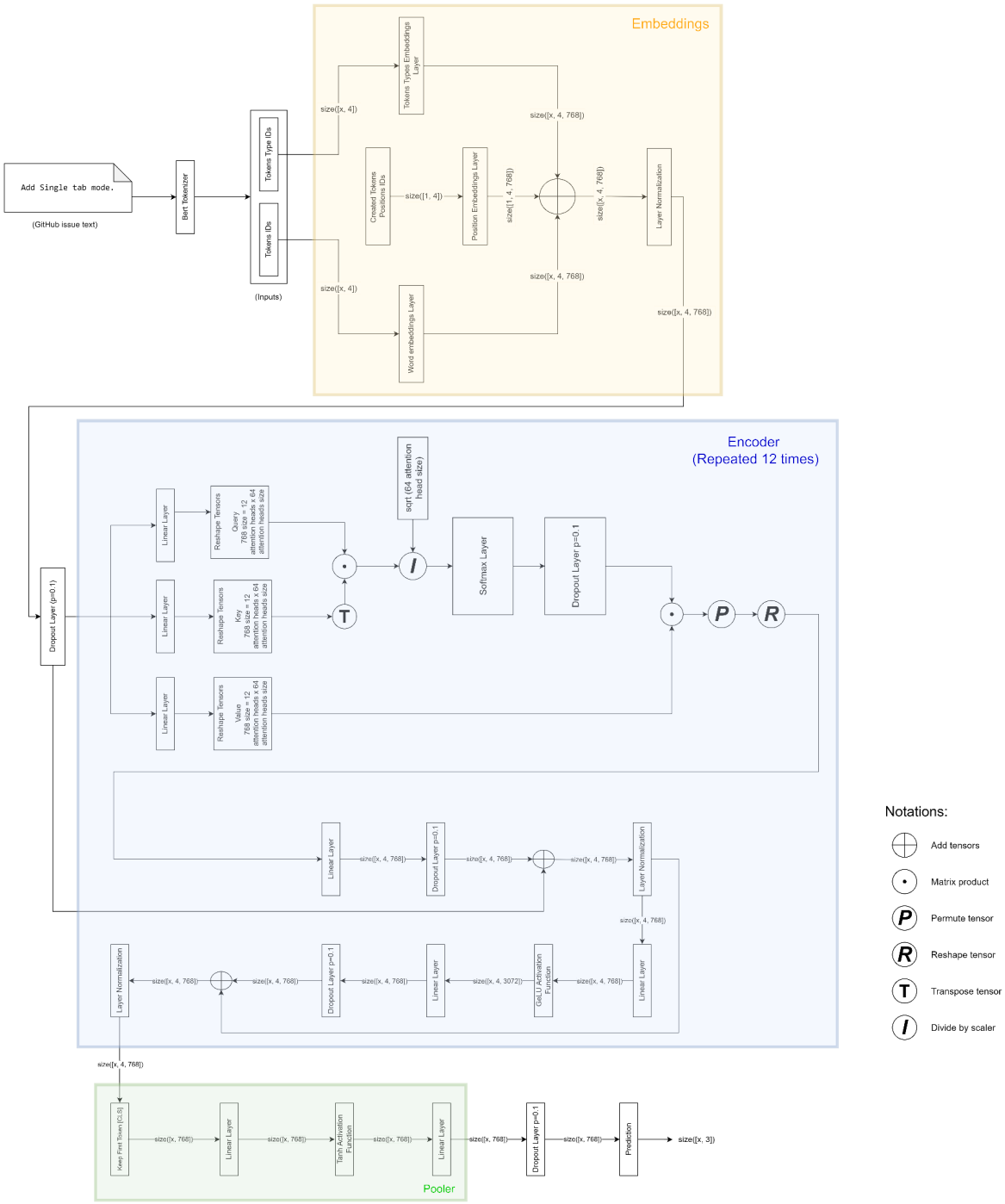

Fig. 4. A White-box representation of BERT

Experiments:

We try to answer the following questions research questions in this study:

- RQ1: How do the different classifiers perform at classifying the GitHub issues?
- RQ2: Does training the model on a domain-specific dataset improve performance?
- RQ3: Do the results improve with respect to the model complexity?

RQ1:

In our initial exploration, we compared the performance of two models on $D_{Unbalanced}$. The first model is fastText (proposed by [4]) and the second is BERT-base (proposed by us).

FastText results:

F1 Score: 68%

Accuracy: 70%

BERT results:

Macro F1: 78%

Accuracy: 78%

We found that our model performs better when compared to fastText, and predicts better true positives than the baseline.

RQ2:

The goal of this research question was to check if a "domain-specific" model would be more reliable than a generic model with respect to classifying issues. Our hypothesis was if we use data from a particular domain, the model will be able to learn some domain-specific patterns that might be impossible to learn from a generic dataset. As the conclusion of RQ1 suggests that BERT is a better model than FastText, we only trained BERT (with the same hyperparameters as the previous experiment) on $D_{domainSpecific}$ to investigate if there is any significant performance gain over the generic model. We found that domain-specific BERT performs significantly better, achieving an f1-score of 0.82. Thus we can conclude that domain-specific models perform better than models trained on a generic dataset.

RQ3:

In our first experiment on $D_{Unbalanced}$, we observed an increase of 14.7% in the F1 score for BERT over FastText. Another objective of this experiment was to check if a heavier model really has an advantage over the lightweight fastText and to check the tradeoff between the size and better performance. So, clearly, the results improve with respect to the complexity of the models.

Hyperparameter Tuning:

We perform a random search to find an optimal set of hyperparameters for BERT. It is to be noted that hyperparameter tuning for such big models is computationally very expensive hence, we only searched in the neighborhood of popularly used values of the hyperparameters. We found the below set of hyperparameters to work the best for our problem.

Hyperparameter	Value
weight decay	0.01
warmup steps	500
epochs	6

Performance Evaluation:

Here, we try to understand the improvements gained by BERT over fastText and possible incurred shortcomings by comparing our model's prediction with the ground truth.

Cleaned Issue Text	Ground Truth	BERT's Prediction	FastText's Prediction
1) PKSim Workshop Are there plans for a 2018 PKSIM workshop?	Question	Question	Enhancement
2) Sleuth Reactor. SpanSubscriber cannot be cast to reactor.core.Fuseable QueueSubscription We use Sleuth and Spring Reactor on our project. Official support hasn't been added yet however i can see that some changes were done on both sides. I have tried using Reactor Bismuth M3 SNAPSHOT and latest Sleuth BUILD SNAPSHOT which has now 'spring cloud sleuth reactor' module. See the stacktrace	Bug	Bug	Enhancement

Table. 2. Improvement over fastText

Table. 2. shows some examples where fastText produced the wrong output but BERT's prediction matches the ground truth. It seems like fastText misses some questions despite the presence of '?' for predicting questions. We think this happened probably because it is not robust enough to detect the informal framing of the questions (the question doesn't start with the be verb 'Are'). This also gets reflected in the low recall value (0.51) for the class question. (see Table. 3.)

	precision	recall	f1-score	support
BUG	0.68	0.76	0.72	997
ENHANCEMENT	0.71	0.75	0.73	981
QUESTION	0.70	0.51	0.59	622
accuracy			0.70	2600
macro avg	0.70	0.67	0.68	2600
weighted avg	0.70	0.70	0.69	2600

Table. 3. Classification report of fastText on $D_{Unbalanced}$

On the other hand, if we look at Table. 4., we notice that the recall for the question is more than the other classes. So, we conclude that the major improvement achieved by BERT is the ability to detect questions correctly. Also, we found that the f1-score for other classes improved by at least 5.5%.

	precision	recall	f1-score	support
BUG	0.79	0.80	0.80	997
ENHANCEMENT	0.83	0.72	0.77	981
QUESTION	0.72	0.84	0.77	622
accuracy			0.78	2600
macro avg	0.78	0.79	0.78	2600
weighted avg	0.79	0.78	0.78	2600

Table. 4. Classification report of BERT on $D_{Unbalanced}$

We also investigated the instances where our model predicted the wrong class. Table. 5. shows a few such examples. We notice that some of the examples are very hard to classify even for humans. For example, (3) lies at the boundary of bug and question class.

Cleaned Issue Text	BERT's prediction	Ground Truth
1) Remove unappropriate comment on TwingErrorLoader	Bug	Enhancement
2) Update README to use instead of manifold.lvh The.lvh domain is pretty Cast Iron specific. Let's target a localhost or domain instead.	Enhancement	Bug
3) How to load indicators and PSAR indicator. Good Day Friend Please i want to know how load your indicators onto the bot page i tried loading them like the other scripts but nothing comes up either in the functions tab or indicators tab. Also can you help develop a a Parabolic SAR indicator. I am using it manually to trade goes out and it looks promising. The strategy is when a Parabolic SAR dot appears at the end of a 1 minute candle trade is placed for 3 minutes and barrier is set to give a payout of 200. Present barrier for vol 100 is set at 14.5 Thank you.	Question	Bug

Table. 5. Misclassifications of BERT

Apart from the aforementioned evaluation metrics we also ran some statistical tests to estimate how significant the performance gain of BERT is over FastText. First, we compute the Matthews correlation coefficient (Table. 6.)

	BERT	FastText
MCC	0.67	0.53

Table. 6. Comparison of MCC values

We use McNemar's test to understand the difference between the two models. Table.

7. depicts the contingency table.

	FastText correct	FastText incorrect
BERT correct	1555	475
BERT incorrect	253	317

Table. 7. Contingency Table

Observing the results of the McNemar test (Table. 8.) we can say that BERT is significantly better at classifying the issues with an Odds Ratio over 4.

	p-value	FastText
BERT vs FastText	<0.001	4.12

Table. 8. McNemar's Test Results

Finally we evaluate the performance of domain-specific BERT.

	Accuracy	F1-score
Domain-specific BERT	0.83	0.82

Table. 9. Performance of domain-specific BERT

Threats to Validity:

- Threats to Conclusion validity:
    - The original dataset is highly imbalanced which could cause our model to be biased towards a certain class. To mitigate this issue, we have tried downsampling. We also use the F1 score as the metric to compare the models as it is more indicative in an unbalanced scenario.

- Threats to Internal validity:
    - The two models chosen were chosen to test whether a complex model can beat a simpler lightweight model. There can be other models we have not researched that may result in better results. We are using the BERT base, as BERT large could not be used due to computational limitations.

- Threats to External Validity:
    - The generated dataset is inclusive of the top 3k projects on GitHub, and thus the data is very diverse. However, this results in a wide variety of domain types the model has to learn about. To mitigate this issue, we curated a different, domain-specific dataset.

- Threats to Construct Validity:
    - The entire experiment is recreated for a college project. This is not at all indicative of a real-world scenario. An improvement could be made by converting our experiment into an online learning model.

Future Work:

In this work, we only focused on one domain (Text editors) to demonstrate that training BERT on domain-specific data helps it learn nuanced patterns. But this work can easily be extended to other software domains like media management apps, file management systems, web browsers, etc. Another interesting direction could be analyzing the issues labeled “good first issue” to investigate if there is any common pattern across different software projects for easily fixable issues.

Conclusion:

In this work, we proposed the use of BERT for classifying GitHub issues from their title and description. We showed that BERT performs better than the previously proposed FastText model with an F1-score gain of 14.7%. We ran statistical tests like McNemar's test to objectively prove that the performance gain in BERT is significant. We also showed that models trained on domain-specific data perform better than models trained on generic datasets.

References:

[1] Zhou, Yu et al. "Combining Text Mining and Data Mining for Bug Report Classification." 2014 IEEE International Conference on Software Maintenance and Evolution (2014): 311-320.

[2] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang and S. Liu, "Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques," in IEEE Access, vol. 6, pp. 24003-24015, 2018, doi: 10.1109/ACCESS.2018.2810295.

[3] Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).

[4] Kallis, Rafael, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. "Predicting issue types on GitHub." Science of Computer Programming 205 (2021): 102598.

[5] https://nlbse2022.github.io