CatIss: An Intelligent Tool for Categorizing Issues Reports using Transformers

Maliheh Izadi
m.izadi@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

Users use Issue Tracking Systems to keep track and manage issue reports in their repositories. An issue is a rich source of software information that contains different reports including a problem, a request for new features, or merely a question about the software product. As the number of these issues increases, it becomes harder to manage them manually. Thus, automatic approaches are proposed to help facilitate the management of issue reports. This paper describes ***CatIss***, an automatic **Cat**egorizer of **Iss**ue reports which is built upon the Transformer-based pre-trained RoBERTa model. CatIss classifies issue reports into three main categories of *Bug report*, *Enhancement/feature request*, and *Question*. First, the datasets provided for the NLBSE tool competition are cleaned and preprocessed. Then, the pre-trained RoBERTa model is fine-tuned on the preprocessed dataset. Evaluating CatIss on about 80 thousand issue reports from GitHub, indicates that it performs very well surpassing the competition baseline, TicketTagger, and achieving 87.2% F1-score (micro average). Additionally, as CatIss is trained on a wide set of repositories, it is a generic prediction model, hence applicable for any unseen software project or projects with little historical data. Scripts for cleaning the datasets, training CatIss and evaluating the model are publicly available. [1]

KEYWORDS

Issue report Management, Classification, Repositories, Transformers, Machine Learning, Natural Language Processing

ACM Reference Format:
Maliheh Izadi. 2022. CatIss: An Intelligent Tool for Categorizing Issues Reports using Transformers. 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/10.1145/3528588.3528662

1 INTRODUCTION

Issue reports are used for communication, decision making, and collecting users' feedback in software repositories. Any GitHub user can contribute to the progress of a project using issue reports. Users open issue reports for different reasons including reporting a bug, requesting a new feature, asking for improvement, asking questions, and asking for support.

Issues in software repositories must have a *title*, a *description*, and a *state* (open/closed). They can also have additional data such as *labels*, *assignees*, *milestone*, *timestamps*, *author association*, *comments*, and more. Issue description usually includes useful information about the reported problem and even code snippets to elaborate on the reported problem. Moreover, each issue can have several labels such as `bug report` to denote the goal behind opening the issue. Labels, as a sort of metadata, describe the goal and content of an issue. They are mainly used for categorizing, managing, searching, and retrieving issues. Thus, assigning labels to issues facilitates task assignment, maintenance, and management of a software project. Cabot et al. [1] analyzed about three million non-forked GitHub repositories to investigate the label usage and its impact on resolving issues. They showed only about 3% of these repositories had labeled issues. Furthermore, in the repositories which incorporated issue labeling, only about 58% of issues were labeled. The authors showed addressing an issue and the engagement rate both have a high correlation with the number of labeled issues in a repository [1]. Recently, Liao et al. [7] investigated the effect of labeling issues on issue management and found labeled issues were addressed immediately, while unlabeled issues could remain open for a long time.

As the number of users and reported issues increases, efficient and timely management of issue reports becomes harder. Team members should address these issues as soon as possible to keep their audience engaged and improve their software product. Thus, automatic approaches for managing issues are proposed. However, their accuracy can be improved by using contextual information better.

In this tool paper, using data-driven approaches, The author fine-tunes a Transformer-based pre-trained RoBERTa-based model to predict the category of issue reports. These categories are the three frequent reasons for opening an issue, namely *Bug* reports, *Enhancement* requests, and *Questions*. CatIss achieves 87.2% F1-score as the micro average score for all three categories of issues. Furthermore, CatIss obtains 89.6%, 87.9%, and 69.1% of per class F1-score for bugs, enhancements, and questions, respectively. As CatIss is built using a Transformer architecture, it is able to better utilize the contextual information in issue reports, hence providing more accurate predictions. CatIss surpasses TicketTagger [5, 6], the baseline approach in all categories and for all metrics. More importantly, based on the Recall score of classes, CatIss significantly outperforms TicketTagger by 89% for the `Question` category. Compared to Bug and Enhancement, "Question" is the least-represented class in the datasets, and consequently suffered from weak accuracy scores in previous studies. However, CatIss is able to obtain much higher

[1] https://github.com/MalihehIzadi/catiss

NLBSE'22, May 21, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9343-0/22/05.
https://doi.org/10.1145/3528588.3528662

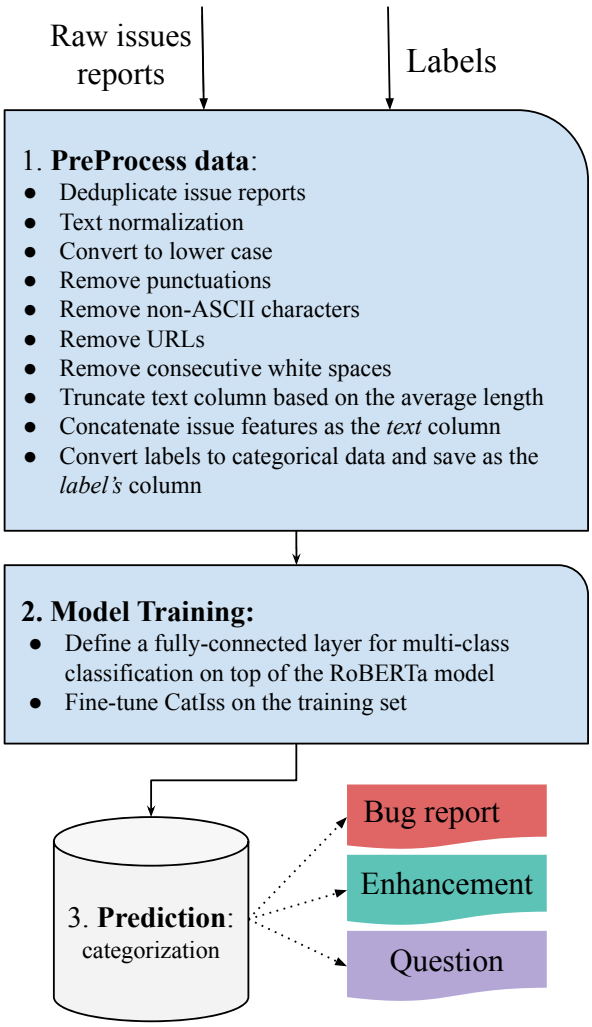

Figure 1: CatIss workflow

results. CatIss is open source and can be accessed on GitHub. [2] The Jupyter notebooks describes data processing, model training, and evaluation. Note that this work is based on our previous study on the management of issue reports through categorization and prioritization [2].

2 TOOL DESCRIPTION

The author first cleans and pre-processes the dataset provided for the NLBSE tool competition [4]. Then, she proceeds to fine-tune the pre-trained Transformer-based RoBERTa model [8] on the pre-processed data, and finally, evaluates it on the competition test set. In the following, CatIss will be described in more detail. Figure 1 presents a concise summary of the proposed approach.

[2]https://github.com/MalihehIzadi/catiss

2.1 Preprocessing Data

Data processing techniques used to clean the provided training set are as follows. Using the "describe" function of the Pandas dataframes, it is evident that the training set consists of duplicate issue reports. Thus, the author first removes the duplicate issues using their issue URL as their unique ID. This process removes 26220 from the 722899 available issues in the training set. Note that deduplication is only performed on the training set to avoid modifying the samples' distribution in the test set.

Next, the author perform normalization on the text of issue titles and bodies. Normalization refers to the technique of replacing the content of a concept in the text with the name of that concept. For instance, function names are extracted using regular expressions and replaced with a fixed token $\langle FUNCTION \rangle$. Text normalization is conducted with two objectives; (1) to keep the existence of a function in text, and (2) to reduce the vocabulary size for the machine learning models [3].

Then, punctuation marks, and non-ASCII characters are removed from both title and body of issues. Next, the text in titles and bodies is converted to lower case. The base URL of repositories (https://api.github.com/repos/) is also removed from the `repository_url` column as it does not provide additional information for the model.

After removing white spaces from the title and body columns, the five remaining columns are concatenated together to construct the input for the model (excluding `issue_url` column). These five columns are: (1) `issue_create_at`: the time that an issue was created, (2) `issue_author_association`: the issue author role, (3) `repository_url`: the repository 'owner + repository' names, (4) `issue_title`, and finally (5) `issue_body`.

In the end, the `text` columns in both train and test datasets are truncated based on the average and median number of tokens in issue titles and bodies of the *train* set. Note that one must not rely on the test set's statistical information as the model will not see the test set before training. The average numbers of tokens for issue titles and bodies after cleaning are 6 and 49 tokens. Also, the median number of tokens for the latter is about 100 token. Hence, the cutoff point is to 200 tokens in the text column. Note that the issue classes in the `label` column (bug, enhancement, and question) are also converted to their categorical codes (0, 1, 2) and then fed to the model.

2.2 Model Training

For the past few years, Transformers have significantly impacted the Natural Language Processing (NLP) domain. Now, Pre-trained models such as Bidirectional Encoder Representations from Transformers (BERT) can be fine-tuned using additional outputs layers to create state-of-the-art models for a wide range of NLP tasks, without major task-specific architecture modifications. Based on our recent study on issue report management [2], CatIss adapts a Robustly-optimized BERT approach (RoBERTa) and fine-tunes it [8]. RoBERTa, developed by Facebook [8], includes additional pre-training improvements using only unlabeled text from the web, with minimal fine-tuning and no data augmentation. The authors modified the Masked Language Modeling (MLM) task in the BERT using dynamic masking. The authors also eliminated the Next Sentence Prediction (NSP) task since Facebook's analysis indicated

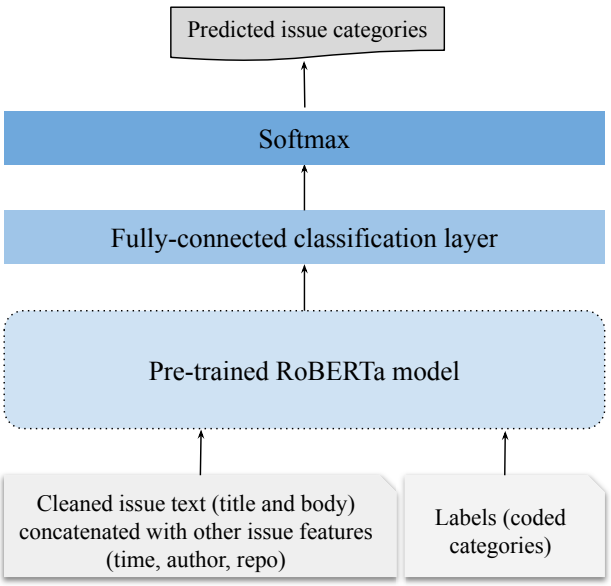

Figure 2: Fine-tuning RoBERTa for multi-class classification on issue reports

that it actually hurts the model's performance. Finally, RoBERTa was trained using larger mini-batch sizes compared to BERT. The author defines a multi-class classification layer on top of this model as the downstream task and fine-tunes it for four epochs on the pre-processed training set. Figure 2 depicts the architecture of CatIss and the fine-tuning process.

3 EXPERIMENT SETUP

In this section, first datasets and their characteristics are reviewed, then the evaluation metrics are defined. And finally, implementation details are presented.

3.1 Datasets

The training and test sets provided by the NLBSE workshop [4] are used for training and evaluating the model. Training and test sets contain 722899 and 80518 issue reports, respectively. The datasets contain various types of information on issue reports including time of creation, repository URL, issue URL, author association role, label, title, and body. Issue reports are collected from 3093 repositories hosted on GitHub. There are six types of author association rules available in the dataset, namely OWNER, MAINTAINER, CONTRIBUTOR, COLLABORATOR, MEMBER, and MANNEQUIN. Issues belong to one category of the three labels: bug, enhancement, and question. About $8K$ of issues do not have a body text, that is, they are lacking any description.

3.2 Metrics

CatIss is evaluated using standard measures of evaluating classifiers, namely *precision*, *recall*, and *F1-score*. Equations 1, 2, and 3 compute the above measures for each class of issues, namely bug, enhancement and question. Micro averages of the above measures are also reported. TP, FP, TN, and FN indicates number of True Positives, False Positives, True Negatives, and False negatives, respectively.

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

$$R_c = \frac{TP_c}{TP_c + FN_c} \tag{2}$$

$$F_{1,c} = \frac{2 \cdot P_c \cdot R_c}{P_c + R_c} \tag{3}$$

$$P = \frac{\sum TP_c}{\sum (TP_c + FP_c)} \tag{4}$$

$$R = \frac{\sum TP_c}{\sum (TP_c + FN_c)} \tag{5}$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \tag{6}$$

3.3 Implementation

The author uses HuggingFace Transformers [3], PyTorch [4], and SimpleTransformers [5] libraries to train CatIss. She fine-tunes the pre-trained RoBERTa model accessible through the HuggingFace API [6]. The author also sets the learning rate to $3e-5$, the number of epochs to 4, the maximum input length to 200 and the training and evaluation batch sizes to 100. Experiments are conducted on a machine equipped with Ubuntu 16.04, 64-bit as the operating system, two GeForce RTX 2080 GPU cards, AMD Ryzen Threadripper 1920X CPU with 12 core processors, and 64G RAM. Training lasts for four hours and 20 minutes.

4 RESULTS

Table 1 presents CatIss's results on classifying issues of the test set compared to the provided baseline, TicketTagger [5] and an additional baseline, Logistic Regression (LR) as a classical classifier. As depicted, CatIss is outperforming the baselines, and achieving an F1-score of 0.872 (micro average). Note that the more populated categories (*Bug*, and *Enhancement*), obtain higher classification results for all metrics compared to the less-represented class *Question*. This is probably due to two reasons; (1) the fewer number of samples for this class, (2) the diversity of information and vocabulary in this category. Nonetheless, CatIss significantly outperforms TicketTagger for the *Question* category (by 89% based on the Recall). Compared to Bug and Enhancement, "Question" is the least-represented class in the datasets, and consequently has suffered from weak accuracy scores in the previous studies. However, CatIss, utilizing contextual information using its transformer architecture and the knowledge transferred from the pre-trained model, is able to obtain higher scores for this class, specifically for Recall and F1-measure. It is worth mentioning that the author also performed a data ablation study. The results of this experiment indicated that including and concatenating the five sources of information reviewed in Section 2.1 provide better results.

[3]https://huggingface.co
[4]https://pytorch.org/
[5]https://github.com/ThilinaRajapakse/simpletransformers
[6]https://huggingface.co/api/models/roberta-base

Table 1: Classification Results

Metric	CatIss			TicketTagger [5]			Logistic Regression		
	P	R	F1	P	R	F1	P	R	F1
Bug	**0.894**	**0.897**	**0.896**	0.831	0.872	0.851	0.841	0.867	0.854
Enhancement	**0.874**	**0.885**	**0.879**	0.815	0.846	0.831	0.822	0.850	0.835
Question	**0.720**	**0.664**	**0.691**	0.652	0.350	0.455	0.655	0.432	0.521
Micro Avg	**0.872**	**0.872**	**0.872**	0.816	0.816	0.816	0.822	0.822	0.822

5 RELATED WORK

Kallis et al. [5, 6] has proposed *TicketTagger*, a tool based on FastText for classifying issues to three categories of `Bug`, `Enhancement` and `Question`. These categories are among the default labels of the GitHub issue system. They trained their model on the text (title and description) of $30K$ issue reports from about $12K$ GitHub repositories. Their evaluation reports 82%, 76%, and 78% of precision/recall scores for three classes of Bug, Enhancement, and Question, respectively. Recently, BEE was proposed by Song and Chaparro [9], which uses the pre-trained model of TicketTagger to label issues. Then it proceeds to identify the structure of bug descriptions from predicted reports that are predicted to be a bug in the issue-objective prediction phase. In our recent study [2], we propose two models to first extract issues' objectives and then prioritize them using the predicted objectives and other issue features. CatIss is based on this study.

6 CONCLUSIONS

CatIss is an automatic categorizer of issue reports that is built upon the Transformer-based pre-trained RoBERTa model. CatIss categorizes issue reports into three main categories of *Bug*, *Enhancement*, and *Question*. CatIss is trained on the provided training set for the NLBSE tool competition and achieves 87.2% F1-score (micro average) on the test set. Specifically, CatIss significantly outperforms TicketTagger, the main baseline by 89% based on the Recall score of the Question class. Scripts for cleaning the datasets, training CatIss and evaluating the model are publicly available. [7]

REFERENCES

[1] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. 2015. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 550–554.

[2] Maliheh Izadi, Kiana Akbari, and Abbas Heydarnoori. 2022. Predicting the objective and priority of issue reports in software repositories. *Empirical Software Engineering* 27, 2 (2022), 1–37.

[3] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. 2021. Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering* 26, 5 (2021), 1–33.

[4] 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)*.

[5] 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)*. IEEE, Cleveland, OH, USA, 406–409. https://doi.org/10.1109/ICSME.2019.00070

[6] 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

[7] Zhifang Liao, Dayu He, Zhijie Chen, Xiaoping Fan, Yan Zhang, and Shengzong Liu. 2018. Exploring the characteristics of issue-related behaviors in github using visualization techniques. *IEEE Access* 6 (2018), 24003–24015.

[8] 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).

[9] Yang Song and Oscar Chaparro. 2020. BEE: a tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1551–1555.

[7] https://github.com/MalihehIzadi/catiss