NLBSE'22 Tool Competition

Rafael Kallis
Rafael Kallis Consulting
Switzerland

Oscar Chaparro
College of William & Mary
USA

Andrea Di Sorbo
University of Sannio
Italy

Sebastiano Panichella
Zurich University of Applied Sciences
Switzerland

ABSTRACT

We report on the organization and results of the first edition of the Tool Competition from the International Workshop on Natural Language-based Software Engineering (NLBSE'22). This year, five teams submitted multiple classification models to automatically classify issue reports as *bugs*, *enhancements*, or *questions*. Most of them are based on BERT (Bidirectional Encoder Representations from Transformers) and were fine-tuned and evaluated on a benchmark dataset of 800k issue reports. The goal of the competition was to improve the classification performance of a baseline model based on fastText. This report provides details of the competition, including its rules, the teams and contestant models, and the ranking of models based on their average classification performance across the issue types.

ACM Reference Format:
Rafael Kallis, Oscar Chaparro, Andrea Di Sorbo, and Sebastiano Panichella. 2022. NLBSE'22 Tool Competition. 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/10.1145/3528588.3528664

1 INTRODUCTION

This year, we organized the first edition of the NLBSE'22 tool competition on automated issue report classification. The goal of the competition was to bring together practitioners and researchers into developing more accurate classification models for automatically identifying the type of a given issue report. We focused on issue report classification for two reasons: (1) it is an important task for developers in the context of the issue management and prioritization process, and (2) extensive research has been dedicated to addressing this problem using natural language processing and machine learning techniques.

Five teams [1–5] participated in the competition. Each team proposed one or more classification models (or classifiers) trained and evaluated on the dataset we provided for the competition [6], which contains more than 800k issue reports labeled as *bugs*, *enhancements*, or *questions*, extracted from the repositories of open-source projects. Given this dataset and the classification results of a baseline model (based on fastText [7–9]), the participants had to design their classifiers to outperform the baseline in detecting the correct type of an issue (*bug*, *enhancement*, or *question*).

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

ACM ISBN 978-1-4503-9343-0/22/05...$15.00
https://doi.org/10.1145/3528588.3528664

2 BENCHMARK DATASET

We provided a dataset of 803,417 issue reports extracted from 127,595 open source projects hosted on GitHub. The issues were extracted from The GitHub Archive [10] using Google BigQuery [11]. We extracted *closed* issues during the first semester of 2021 (from January 1st to May 31st) that contained any of the labels *bug*, *enhancement*, and *question* at the issue closing time.

We extracted the following data attributes for each issue: its title or summary, the issue body, the issue URL, the repository URL, its creation/submission timestamp, and the issue author association (*e.g.,* owner, contributor, or member). Additionally, each issue is labeled with one class that indicates its type, namely, *bug*, *enhancement*, or *question*. The dataset was given in CSV format without applying any preprocessing on the issues.

We partitioned the dataset into a training set (≈90%) and a test set (≈10%). The distribution of (722,899) issues in the training set is: 361,239 (50%) *bugs*; 299,287 (41.4%) *enhancements*; and 62,373 (8.6%) *questions*. The distribution of (80,518) issues in the test set is: 40,152 (49.9%) *bugs*; 33,290 (41.3%) *enhancements*; and 7,076 (8.8%) *questions*.

3 COMPETITION RULES

The participants had to train and tune their classification models using the training set, and evaluate the models using the test set. The test set was used to determine the official classification results and the ranking of the contestant models.

The participants were free to select and transform the data from the training set as they pleased with the restriction that no new information sources were utilized by the models. In other words, any inputs or features used to create the classifiers had to be derived from the provided issues and their attributes. Participants were allowed to preprocess, (over/under-)sample, select a subset of the attributes, and perform feature-engineering on the training set. The participants were also allowed to split the training set into a model-tuning validation set.

The participants were free to apply any preprocessing or feature engineering on the test set except sampling, rebalancing, undersampling or oversampling techniques.

The proposed models were evaluated based on their classification performance on the test set. The classifiers had to assign a single label to an issue: *bug*, *enhancement*, or *question*. The classification performance of a model is measured by the micro-average F1-score over all three classes. Micro-averaging was chosen as the cross-class aggregation method due to the class imbalance present in the data. While the F1-score was used for ranking the models and determining the winner of the competition, we also asked the participants to report the following metrics: precision and recall for each class. Note that micro-average precision and recall are the same as micro-average F1-score.

We provided a Colab notebook [6] for the competition with the specific competition instructions and rules, including the results of a baseline model based on fastText [7–9] (see Table 1). More importantly, this notebook aimed to facilitate participation in the competition as it was ready to be adapted, used, and executed.

While the competition aimed to receive classifiers trained from scratch using the provided dataset, we decided to relax this constraint in order to maximize participation. This means that pre-trained and fine-tuned models were allowed to participate in the competition.

Table 1: Issue classification results for *bugs, enhancements*, and *questions*. The models are ranked by Avg.: the micro average precision/recall/F1-score over the three issue types.

Classification model	Metric	Bug	Enh.	Que.	Avg.
CatIss (RoBERTa) by *Izadi* [3]	Precision	0.894	0.874	0.720	0.872
	Recall	0.897	0.885	0.664	
	F1-score	0.896	0.879	0.691	
RoBERTa by *Bharadwaj & Kadam* [1]	Precision	0.872	0.879	0.714	0.865
	Recall	0.911	0.877	0.539	
	F1-score	0.891	0.878	0.614	
CodeBERT by *Bharadwaj & Kadam* [1]	Precision	0.883	0.866	0.693	0.862
	Recall	0.894	0.891	0.551	
	F1-score	0.888	0.878	0.614	
RoBERTa by *Colavito et al.* [2]	Precision	0.875	0.871	0.767	0.859
	Recall	0.898	0.874	0.559	
	F1-score	0.886	0.872	0.612	
BERT by *Siddiq & Santos* [4]	Precision	0.883	0.859	0.678	0.858
	Recall	0.888	0.888	0.546	
	F1-score	0.885	0.873	0.605	
seBERT (BERT) by *Trautsch & Herbold* [5]	Precision	0.866	0.864	0.731	0.857
	Recall	0.906	0.877	0.487	
	F1-score	0.886	0.871	0.584	
XLNet by *Bharadwaj & Kadam* [1]	Precision	0.879	0.853	0.706	0.856
	Recall	0.885	0.890	0.534	
	F1-score	0.882	0.871	0.608	
BERT by *Bharadwaj & Kadam* [1]	Precision	0.875	0.866	0.660	0.855
	Recall	0.892	0.871	0.570	
	F1-score	0.883	0.868	0.611	
MLP by *Colavito et al.* [2]	Precision	0.893	0.879	0.472	0.829
	Recall	0.834	0.839	0.753	
	F1-score	0.863	0.859	0.581	
Logistic Regression by *Izadi* [3]	Precision	0.841	0.822	0.655	0.822
	Recall	0.867	0.850	0.432	
	F1-score	0.854	0.835	0.521	
Baseline (fastText) by *Kallis et al.* [8, 9]	Precision	0.811	0.844	0.669	0.818
	Recall	0.904	0.815	0.336	
	F1-score	0.855	0.830	0.447	

4 SUBMITTED CLASSIFICATION MODELS

Five teams submitted one or more classifiers to participate in the competition, as listed in Table 1. Almost all of the classifiers are based on BERT (Bidirectional Encoder Representations from Transformers) [12] and use different attributes of the issues. These models were pre-trained and fine-tuned using a preprocessed version of the training issues. A variety of preprocessing techniques were used to prepare the data for training and evaluation.

Izadi [3] proposed CatIss, a fine-tuned pretrained RoBERTa model [13] that uses as input the issue text (title and body) concatenated with the issue timestamp, author, and repository (specifically, the owner and repository name). The processing of the issues included removal of exact duplicate issues (performed on the training set only), text normalization to replace content with a predefined tag (*e.g.,* <FUNCTION> for function names), special character removal, and lower casing. Izadi also reported a Logistic Regression model as an additional baseline model.

Bharadwaj and Kadam [1] proposed multiple classifiers based on BERT (vanilla BERT [12], CodeBERT [14], and RoBERTa [13]) and XLNet [15] to encode the issue text (title and body) as embeddings. These embeddings are combined with embeddings obtained from additional issue features, namely whether or not the issue was submitted early in the project history (defined based on a threshold), the project owner, and whether or not the issue title describes a question. The combined embeddings are the input to classification layers. The BERT-based models used by the authors were also fine-tuned. The main preprocessing applied include regex-based substitution of code snippets, URLs, usernames, and numbers with predefined tags.

Colavito *et al.* [2] proposed multiple classifiers based on BERT, which were fine-tuned for the issue classification task. These classifiers include vanilla BERT [12], ALBERT [16], and RoBERTa [13]. According to experiments made by the authors on a validation set, RoBERTa achieved the best performance when only the issue body and title were input to the model (experiments included issue author information). Colavito *et al.* also included a Multilayer Perceptron (MLP) model that used both textual and author information. As before, the authors preprocessed the issues by replacing textual elements such as images, URLs, email addresses, numbers, and usernames with predefined tags.

Siddiq and Santos [4] proposed a BERT-based classifier, fine-tuned using the issue title and body. Preprocessing included removal of repeating white space characters and replacement of tabs and line breaks with spaces.

Trautsch and Herbold [5] fine-tuned seBERT [17], a model for the software engineering domain that is pretrained using posts from Stack Overflow and issues/commit messages from the repositories of open source projects. The model was fine-tuned using the issue text (title and body) after preprocessing (*e.g.,* replacement of line breaks with spaces and removal of repeating white space characters).

5 CLASSIFIER EVALUATION AND RESULTS

Based on the replication package provided by each team, we replicated the results reported in their papers [1–5]. Specifically, we executed the code that each team provided, using cloud instances equipped with a A100 GPU. Training and fine-tuning lasted up to 14 hours and GPU memory usage peaked at 16 GB for some replication packages.

The classification performance obtained by the proposed classifiers on the test set is shown in Table 1. First of all, we observe that all the proposed classifiers outperform the baseline model on average (across all the three issue types). While the classical models (MLP and Logistic Regression) achieve higher performance, the improvement is small (0.004 and 0.011). The improvements seem to come from the higher performance achieved for *questions*. The remaining classifiers, based on BERT, achieve a comparable average classification performance (0.855 - 0.872) among them. CatIss is the one with the highest overall performance (0.054 higher than fastText). Compared to fastText and the other models, CatIss achieves the highest average F1-score over the three issue types. Given that these models are based on BERT, we conjecture that the main reason for the CatIss' superior performance is due to the additional data processing that was applied, most notable the de-duplication of the training issues. Additionally, it is unclear if these models would have a notable effect in a real-life issue classification scenario, given the moderate improvements with respect to fastText and the classical models. We also note that the BERT-based models might be resource intensive while the other models have a lower computational overhead.

Based on the classification results, we rank the five contestant teams as follows:

a) Izadi [3] takes the first place in the competition with their CatIss approach (0.872 micro avg. F1-score).
b) Bharadwaj and Kadam [1] occupy the second place with their RoBERTa and CodeBERT approach (0.865 and 0.862 micro avg. F1-score, respectively).
c) The remaining teams (Colavito *et al.* [2], Siddiq and Santos [4], and Trautsch and Herbold [5]) are placed in the third position of the competition as their best classifiers achieved virtually the same performance (0.855 - 0.859 micro avg. F1-score).

6 CONCLUSIONS AND FINAL REMARKS

The NLBSE'22 Tool Competition attracted five teams that proposed a diverse set of classification models to automatically classify issues as *bugs*, *enhancements*, or *questions*. Most of these classifiers utilized BERT, a state-of-the-art language model based on the Transformer architecture, leveraging various information sources from the issues. While most of these classifiers achieved comparable average classification performance, CatIss by Izadi [3] performed best by a considerable margin. Additional pre-processing to the issues appear to be the main factor for achieving such performance. We expect that future editions of the competition would lead to more accurate models as well as their application to additional software engineering tasks that require the analysis and processing of textual artifacts. We plan to extend the competition with techniques previously used for user review analysis [18–21], for classifying code comments [22], or for the analysis of bug reports [23–25].

ACKNOWLEDGMENTS

We thank all the participants of the competition for their support in launching the first edition of the NLBSE'22 Tool Competition. We gratefully acknowledge the Horizon 2020 (EU Commission) support for the projects *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS. Chaparro was supported in part by grant CCF-1955853 from the NSF.

REFERENCES

[1] Shikhar Bharadwaj and Tushar Kadam. Github issue classification using bert-style models. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, page (to appear), 2022.
[2] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. Issue report classification using pre-trained language models. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, page (to appear), 2022.
[3] Maliheh Izadi. Catiss: An intelligent tool for categorizing issues reports using transformers. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, page (to appear), 2022.
[4] Mohammed Latif Siddiq and Joanna C.S. Santos. Bert-based github issue report classification. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, page (to appear), 2022.
[5] Alexander Trautsch and Steffen Herbold. Predicting issue types with sebert. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, page (to appear), 2022.
[6] Rafael Kallis, Oscar Chaparro, Andrea Di Sorbo, and Sebastiano Panichella. Nlbse'22 tool competition. In *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, 2022.
[7] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
[8] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. Ticket tagger: Machine learning driven issue classification. In *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME'19)*, pages 406–409, 2019.
[9] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. Predicting issue types on github. *Science of Computer Programming*, 205:102598, 2021.
[10] Ilya Grigorik. The github archive. https://www.gharchive.org/, 2022.
[11] Cloud Google. Bigquery - google cloud platform. https://cloud.google.com/bigquery/, 2022.
[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
[14] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.
[15] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.
[16] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.
[17] Julian von der Mosel, Alexander Trautsch, and Steffen Herbold. On the validity of pre-trained transformers for natural language processing in the software engineering domain. *arXiv preprint arXiv:2109.04738*, 2021.
[18] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *International Conference on Software Maintenance and Evolution*, pages 281–290. IEEE, 2015.
[19] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *International Symposium on Foundations of Software Engineering*, pages 499–510. ACM, 2016.
[20] Andrea Di Sorbo, Giovanni Grano, Corrado Aaron Visaggio, and Sebastiano Panichella. Investigating the criticality of user-reported issues through their

relations with app rating. *J. Softw. Evol. Process.*, 33(3), 2021.

[21] Sebastiano Panichella. Summarization techniques for code, change, testing, and user feedback (invited paper). In Cyrille Artho and Rudolf Ramler, editors, *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*, pages 1–5. IEEE, 2018.

[22] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz. How to identify class comment types? A multi-language approach for class comment classification. *J. Syst. Softw.*, 181:111047, 2021.

[23] Yang Song and Oscar Chaparro. 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 (ESEC/FSE'20)*, pages 1551–1555, 2020.

[24] Oscar Chaparro, Carlos Bernal-Cárdenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. Assessing the quality of the steps to reproduce in bug reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*, pages 86–96, 2019.

[25] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*, pages 396–407, 2017.