The NLBSE'23 Tool Competition

Kallis, Rafael; Izadi, Maliheh; Pascarella, Luca; Chaparro, Oscar; Rani, Pooja

DOI
10.1109/NLBSE59153.2023.00007

Publication date
2023

Document Version
Final published version

Published in
Proceedings of the 2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering, NLBSE 2023

Citation (APA)
Kallis, R., Izadi, M., Pascarella, L., Chaparro, O., & Rani, P. (2023). The NLBSE'23 Tool Competition. In L. O'Conner (Ed.), *Proceedings of the 2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering, NLBSE 2023* (pp. 1-8). IEEE. https://doi.org/10.1109/NLBSE59153.2023.00007

The NLBSE'23 Tool Competition

Rafael Kallis
Rafael Kallis Consulting
Zurich, Switzerland
rk@rafaelkallis.com

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

Luca Pascarella
ETH Zurich
Zurich, Switzerland
lpascarella@ethz.ch

Oscar Chaparro
College of William & Mary
Williamsburg, USA
oscarch@wm.edu

Pooja Rani
University of Zurich
Zurich, Switzerland
rani@ifi.uzh.ch

***Abstract*—We report on the organization and results of the second edition of the tool competition from the International Workshop on Natural Language-based Software Engineering (NLBSE'23). As in the prior edition, we organized the competition on automated issue report classification, with a larger dataset. This year, we featured an extra competition on automated code comment classification. In this tool competition edition, five teams submitted multiple classification models to automatically classify issue reports and code comments. The submitted models were fine-tuned and evaluated on a benchmark dataset of 1.4 million issue reports or 6.7 thousand code comments, respectively. The goal of the competition was to improve the classification performance of the baseline models that we provided. This paper reports details of the competition, including the rules, the teams and contestant models, and the ranking of models based on their average classification performance across issue report and code comment types.**

***Index Terms*—Tool-Competition, Labeling, Benchmark, Issue Reports, Code Comments.**

I. Introduction

The first competition was held in 2022 [1], [2] and this one continues the series with the second edition of the Natural Language-based Software Engineering (NLBSE'23) tool competition on automated issue report classification. In addition, we also featured a competition on code comment classification for the first time, given the importance of identifying relevant information from comments to support developers in software development and maintenance tasks [3], [4]. Both competitions aimed to bring practitioners and researchers together into developing more accurate classification models for automatically identifying the type of given issue report or code comment.

We focused on issue report classification for two reasons: (i) it is an important task for developers in the context of the issue management and prioritization process [5], and (ii) extensive research has been dedicated to addressing this problem using natural language processing (NLP) and machine learning (ML) techniques [6], [7]. Similarly, several works have shown the importance of source code comments in software development and maintenance [8]–[10]. For example, well-written code comments actively enhance code readability by documenting code changes. Nonetheless, not all code comments fit this role. Indeed, code comments are used to accomplish different tasks, such as code documentation, license declaration, report work in progress, *etc.* In other words, code comments contain various kinds of information that can support developers in different program comprehension and maintenance tasks [11]. To satisfy different needs, the information is written using a mix of code and natural language sentences; consequently, researchers have leveraged various NLP and ML-based techniques to identify the types of information in these sentences.

Five teams [12]–[16] participated in the two competitions. Each team proposed classification models trained and evaluated on one of the two datasets we provided [17]. The first dataset contains 1.4 million issue reports extracted from the repositories of open-source projects and each issue is labeled with one type. This is an expanded version of the dataset of the NLBSE'22 tool competition [1], [2]. The second dataset is a subset of Rani *et al.*'s dataset [3] and contains the ground truth categories, *i.e.,* information types, of 6,738 comment sentences from 1,066 class comments of 20 projects written in three programming languages: Java, Pharo, and Python [3].

The baseline models provided for issue classification were based on two approaches; FastText [18] and RoBERTa [19]. FastText is used by Kallis *et al.*'s Ticket Tagger [20], [21] and RoBERTa is used by Izadi's CatIss [7], [22]. The baseline models provided for code comment classification were Rani *et al.*'s Random Forests [3], which performed best in their evaluation.

Given these datasets and the classification results of the baseline models, the participants of this year's competitions were expected to design their classifiers to outperform the baselines in identifying the correct type(s) of issue reports or code comments.

II. Issue Report Classification

In this section, we report the structure and measures for the tool competition on issue report classification. The competition followed a similar structure to the previous edition [2]. We received feedback from last edition's participants [22]–[26] concerning the dataset that was used.

In this year's competition, we incorporated their feedback to improve the quality of the dataset. Specifically, we ex-

979-8-3503-0178-6/23/$31.00 ©2023 IEEE
DOI 10.1109/NLBSE59153.2023.00007

panded the dataset from 800 thousand to 1.4 million issue reports, added a new category labeled *documentation*, included common synonyms of labels, and finally removed multi-label issues and non-English issues from the dataset. We also introduced an additional RoBERTa [19] baseline model based on last edition's winner CatIss by Izadi [22].

The remainder of this section is organized as follows. We first describe the dataset, then list the competition rules, then summarize this year's submissions, and finally, we present the evaluation and results of the submissions. We published a GitHub repository[1] to guide and inform potential participants about the competition.

A. Benchmark Dataset

We provided a dataset of 1,418,201 issue reports extracted from the population of open-source projects hosted on GitHub. The issues were extracted from GH Archive [27] using Google BigQuery. We extracted closed issues during the first, second and third quarter of 2022, *i.e.,* January 1st to September 30st, that contained any of the labels *bug*, *features*, *question*, and *documentation* at the issue closing time. These are the most frequently used labels on GitHub [7], [28].

We extracted the following data attributes for each issue: its *ID*, the issue *title* or summary, the issue *body*, 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*, *feature*, *question*, or *documentation*. Issues that are labelled with synonyms of the above labels, as reported by Izadi *et al.* [7], are mapped to the original labels and included in the dataset. To reduce possible inconsistencies in the labeling rationale as discussed by Colavito *et al.* [24], we exclude issue reports with multiple labels and remove any non-English issue reports based on the FastText language identification model lid.176.bin [18], [29]. The dataset was given in CSV format without applying any further pre-processing on the issues.

We partitioned the dataset into a training set and a test set. The distribution of 1,275,881 (90%) issues in the training set is: 670,951 (52.6%) *bugs*; 472,216 (37%) *features*; 76,048 (6%) *questions*; and 56,666 (4.4%) *documentations*. The distribution of 142,320 (10%) issues in the test set is: 74,781 (52.5%) *bugs*; 52,797 (37.1%) *features*; 8,490 (6%) *questions*; and 6,252 (4.4%) *documentations*.

We published a Jupyter notebook that performs the above steps in our tool competition's repository on GitHub.

B. Baselines and Competition Rules

We published two classification models as baselines for the competition. The first baseline uses FastText [18], a static word embedding used in the tool Ticket Tagger by Kallis *et al.* [17], [20], [21] that was also used as baseline in last year's competition. Our second baseline uses RoBERTa [19], the backbone transformer in the CatIss tool by Izadi [7], [22], who was the winner of the last edition of the competition.

[1]https://github.com/nlbse2023/issue-report-classification

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 any variables from the training set. Pre-trained models were permitted but can only be finetuned on the training set. Any inputs or features used to create or finetune the classifier, had to be derived from the provided training set. Participants were allowed to pre-process, sample, apply over/under-sampling, select a subset of the attributes, perform feature engineering, filter records, split the training set into a model-tuning validation set.

The participants were free to apply any pre-processing 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*, *feature*, *question*, or *documentation*. The classification performance of a model is measured by the micro-average F_1-score over all four classes. Micro-averaging was chosen as the cross-class aggregation method due to the class imbalance present in the data. While the F_1-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 [30]. Note that micro-average Precision and Recall scores are the same as micro-average F_1-score.

The competition's GitHub repository contained specific instructions and rules, including replication package and results of the baseline models based on FastText and RoBERTa. More importantly, the repository contained notebooks aimed to facilitate participation in the competition as they were ready to be adapted, used, and executed.

C. Submitted Classification Models

Three teams submitted one or more classifiers to participate in the competition, from which two papers were accepted. As listed in Table I, none of the participants were able to outperform the RoBERTa baseline based on the competition's assessment metric, *i.e.,* the micro-average F_1-score over all classes. However, Laiq's SGD-based classifier [12] performed better than the FastText baseline. Participants used a variety of machine learning approaches to address this challenge. Next, we provide an overview of the two accepted approaches.

Laiq [12] used the SGD classifier [31], [32], which is an efficient technique for solving convex loss functions with the hinge loss function and implemented a linear SVM model. The author tuned parameters such as alpha, penalty, and max iterations to enhance the classification performance. The pre-processing steps included merging the title and body fields of each issue report, removal of special characters, HTML tags, punctuation, numbers, consecutive white spaces, and stop words, stemming of words, and conversion of labels to numbers. Laiq [12] then applied TF-IDF to the cleaned data to generate sparse matrices for both the training and testing datasets. This work achieves a micro-average F_1-score of 0.852.

TABLE I
ISSUE CLASSIFICATION RESULTS OVER THE FOUR ISSUE TYPES SORTED BY MICRO-AVERAGE F_1-SCORE.

Classification Model	Metric	Bug	Feature	Question	Documentation	Average F_1-score
RoBERTa by *Kallis and Izadi* [17], [22]	Precision	0.911	**0.895**	0.730	0.759	**0.890**
	Recall	**0.939**	**0.896**	0.568	**0.697**	
	F_1-score	**0.924**	**0.895**	**0.639**	**0.727**	
SGD by *Laiq* [12]	Precision	0.870	0.840	**0.770**	**0.780**	0.852
	Recall	0.920	0.860	0.380	0.570	
	F_1-score	0.900	0.850	0.510	0.660	
FastText by *Kallis et al.* [17], [20], [21]	Precision	0.877	0.841	0.670	0.736	0.851
	Recall	0.917	0.862	0.455	0.501	
	F_1-score	0.896	0.851	0.542	0.596	
SetFit by *Colavito et al.* [13]	Precision	**0.915**	0.804	0.351	0.442	0.784
	Recall	0.789	0.825	**0.645**	0.571	
	F_1-score	0.847	0.814	0.455	0.498	

Colavito *et al.*'s [13] work is based on SetFit [33] and SBERT [34]. The authors investigated the impact of label consistency on the performance of supervised issue classification models by manually improving label correctness on a subset of the train and test data. To mitigate the effects of label noise, the authors randomly sampled 400 instances from the dataset and manually labeled them. This smaller, hand-labeled dataset served as a gold standard for training and evaluating their few-shot learner. The annotation procedure involved three annotators independently labeling each issue, reaching a consensus in cases of disagreement, and discarding cases where the author's intention could not be interpreted. The authors aim to understand if a SetFit few-shot learner can generalize the hand-labeled examples to the entire dataset using transfer learning. The authors conducted multiple experiments with different configurations and achieved a F_1-score of 0.83 over all classes. Finally, the authors also evaluated their model on the challenge test set, and obtained an F_1-score of 0.784.

D. Classifier Evaluation and Results

Based on the replication package provided by each team, we replicated the results reported in their papers [12], [13]. We executed the code using a workstation equipped with a RTX 3060 GPU. Training and fine-tuning of the RoBERTa baseline lasted 17.5 hours, and GPU memory usage peaked at 12 GB.

The classification performance obtained by the proposed classifiers on the test set is shown in Table I. The proposed approaches were able to outperform the baselines per individual classes or per evaluation metrics such as Precision and Recall. For instance, Laiq's SGD method achieves the best result for the Precision of two classes; *Question* and *Documentation* [12]. Covalito's approach also obtains the best results for the *Bug* class based on Precision measure and for the *Question* class based on the Recall score [13]. However, we observe that none of the proposed classifiers outperform the RoBERTa baseline approach according to the micro-average F_1-score, which is the competition's main assessment metric which is the harmonic mean of Precision and Recall scores. Hence, the first interesting observation is the importance of evaluation metrics and the differences in the performance of classifiers per issue report class. Next, all classifiers perform the best for the *Bug* and the *Feature* classes while struggling for the other two classes; *Question* and *Documentation*. This can be due to the unbalanced nature of the dataset. Therefore, approaches which can address this challenge or work well with less data can improve this aspect. The third observation is the fact that a traditional classifier such as the SGD-based one proposed by Laiq [12] has been able to perform very well, even outperforming the SetFit approach proposed by Colavito *et al.* [13]. This is interesting as Transformer-based models can be very resource-intensive, while traditional classifiers mostly have lower computational overhead. However, in this case, note that the dataset used by Colavito *et al.* [13] is much smaller than what is used to train the SGD classifier. Hence, one cannot fairly compare the results obtained on the smaller dataset with another approach trained on the larger one. Additionally, Colavito *et al.* [13] showed that the few-shot learning method outperforms a RoBERTa-based classifier that is trained on the same limited dataset. This brings us to the notion of noisy labels identified by Colavito *et al.* [13] in the dataset collected from GitHub. The authors conducted an error analysis and found that the presence of noisy labels was a cause for misclassification in their previous work [24]. The authors identified two reasons for the noise; (i) variability in labeling rationale among different projects and (ii) difficulty in distinguishing between *Bugs* and *Questions* related issue reports.

In this round of the competition, the contestants have put in their best effort to outperform the RoBERTa baseline on the given dataset and the selected evaluation metric, *i.e.,* the micro-average F_1-score). However, we have found that neither of the participants has been able to outperform the baseline. Therefore, we have decided not to announce a winner for this round of the competition. Despite this, we recognize and appreciate the hard work, creativity, and dedication put in by both teams. Each work has its own unique merits that deserve recognition.

III. CODE COMMENT CLASSIFICATION

The code comment classification competition consisted of building and testing a set of binary classifiers to classify code comment sentences as belonging to one or more categories. These categories represent the types of information that a sentence is conveying, in comments of code classes. We provided (i) a dataset of code comment sentences and (ii) baseline classifiers based on the Random Forest model The competition called for participants that proposed classifiers with the goal of outperforming the baseline classifiers. We provided a GitHub repository[2] and a Colab notebook[3] to guide and inform potential participants about the competition.

A. Benchmark Dataset

The competition included a dataset composed of 1,060 manually-labeled class comments and 6,738 comment sentences from 20 open-source projects written in three programming languages: Java, Python, and Pharo. This dataset is a subset of the one provided by Rani *et al.* [3]. The dataset contains class comments of various open-source, popular, and heterogeneous projects that vary in terms of contributors, size, and development ecosystem. The Java projects are Apache Spark, Guava, Guice, Eclipse, Vaadin, and Apache Hadoop. The Python projects are Pandas, IPython, PyTorch, Mailpile, Request, PipeEnv, and Django. The Pharo projects are Pillar, Petit, PolyMath, Seaside, GToolkit, Roassal, and Moose.

A sample of comments extracted from the aforementioned projects has been manually analyzed to identify the information that each comment sentence conveys. Based on the analysis, more than 19 types of information are found in the comment sentences across the three programming languages. For the competition, we focused on the most frequent categories, *i.e.,* with 50+ coded sentences per category, for a total of 19 code comment categories. Specifically, we selected seven Java categories: *summary*, *pointer*, *deprecation*, *rational*, *ownership*, *usage*, and *expand*; five Python categories: *summary*, *parameters*, *usage*, *development notes*, and *expand*; and seven Pharo categories: *key messages*, *intent*, *class references*, *example*, *key implementation*, *responsibilities*, and *collaborators*. The definitions of these categories can be found in the original paper by Rani *et al.* [3]. The 19 categories are found in 376 comments for Java, 340 for Pharo, and 344 for Python, for a total of unique 1,060 comments.

We applied various pre-processing steps to the comments. We split the comments into sentences based on the NEON tool [35], changed the sentences to lowercase, transformed multiple line endings into one ending, and removed special characters, *e.g.,* `@#&%.,!?\n`. These symbols were removed to ensure uniformity across languages, as they are used in each language differently. We also removed periods in numbers or special abbreviations, such as "*e.g.*", "*i.e.*", and numbers to minimize incorrect splitting of the comments into sentences.

Each comment sentence can belong to one or more categories, to a maximum of 5 to 7 categories, depending on the language. Each category represents the type of information that the sentence is conveying. While one sentence can belong to multiple categories, the competition focused on binary classification for each category, rather than multi-class classification. In other words, participants were meant to build multiple binary classifiers, each focusing on one category to determine if a sentence does or does not belong to such category. Therefore, for each category, we built the sets of positive and negative sentences used for binary classification, *i.e.,* belonging and not belonging to a category, based on the ground-truth categories of the 6,738 unique comment sentences in our dataset. The distribution of positive and negative sentences across categories is reported in Table II.

We randomly partitioned the comment sentence dataset into training (80%) and testing (20%) sets, both containing a similar proportion of positive and negative sentences as the entire set of sentences for a category. The dataset was provided in CSV files where the attribute *ID* represents the unique sentence ID, *class* represents the class name referring to the source code file where the sentence comes from, *sentence* represents the text of the sentence, *partition* denotes the dataset it belongs to, *i.e., one* for training and *zero* for testing, *category* denotes the ground-truth category the sentence belongs to. The distribution of these sentences in both training and test sets is reported in Table II.

B. Baselines and Competition Rules

We trained and tested 19 binary Random Forests, one for each category, as the competition baseline classifiers, using the training and test sets. We used the Weka toolkit [36] to train and test the models using the parameters determined in Rani *et al.*'s work [3], since they lead to the best classification performance according to their evaluation. The baseline models learn from two types of features for a comment sentence:

1) *NLP features*: these are binary features that indicate whether or not the sentence matches grammar patterns detected by the NEON tool [35], [37].
2) *Textual features*: these are continuous features based on TF-IDF scoring. Each feature represents the importance/weight of a word in the sentence considering the word's term frequency (TF) and inverse document frequency (IDF) in a corpus of sentences.

More information about the features can be found in Rani *et al.*'s work [3]. Potential participants were allowed to use these features in their models, as they were made available in our GitHub repository. Additionally, since the training dataset is unbalanced, we used Weka's ClassBalancer filter to calibrate the instance weights that Weka uses during training to account for data imbalance. The classification performances of the baselines are shown in Table III.

The participants were expected to train their classification models using the provided training dataset and evaluate them on the testing dataset. However, we restricted the use of any external sources beyond the class comment sentences and associated source code of the class. Note that the dataset provides the mapping of a class name to its class comment sentences and to its project so that the participants could identify the source code of the class from the project and thus can leverage it to fine-tune their models. The projects' source code was released in our GitHub repository.

[2]https://github.com/nlbse2023/code-comment-classification
[3]https://tinyurl.com/45ccyv6m

TABLE II
DISTRIBUTION OF POSITIVE/NEGATIVE COMMENT SENTENCES PER CATEGORY, LANGUAGE, AND DATASET (TRAINING AND TESTING).

Language	Categories	Training			Testing			Training + Testing		
		Positive	*Negative*	*Total*	*Positive*	*Negative*	*Total*	*Positive*	*Negative*	*Total*
Java	Expand	505	1,426	1,931	127	360	487	632	1,786	2,418
	Ownership	90	1,839	1,929	25	464	489	115	2,303	2,418
	Deprecation	100	1,831	1,931	27	460	487	127	2,291	2,418
	Rational	223	1,707	1,931	57	431	488	280	2,138	2,418
	Summary	328	1,600	1,928	87	403	490	415	2,003	2,418
	Pointer	289	1,640	1,929	75	414	489	364	2,054	2,418
	Usage	728	1,203	1,931	184	303	487	912	1,506	2,418
		2,263	11,246	13,509	582	2,835	3,337	2,845	14,081	16,926
Pharo	Responsibilities	267	1,139	1,406	69	290	359	336	1,429	1,765
	Key messages	242	1,165	1,407	63	295	358	305	1,460	1,765
	Key impl. points	184	1,222	1,406	48	311	359	232	1,533	1,765
	Collaborators	99	1,307	1,406	28	331	359	127	1,638	1,765
	Example	596	812	748	152	205	357	748	1,017	1,765
	Class references	60	1,348	1,408	17	340	357	77	1,688	1,765
	Intent	173	1,236	1,409	45	311	356	218	1,547	1,765
		1,621	8,229	9'850	422	2,083	2'505	2'043	10'312	12,355
Python	Expand	402	1,637	2,039	102	414	516	504	2,051	2,555
	Parameters	633	1,404	2,037	161	357	518	794	1,761	2,555
	Summary	361	1,678	2,039	93	423	516	454	2,101	2,555
	Dev. notes	247	1,792	2,039	65	451	516	312	2,243	2,555
	Usage	637	1,401	2,038	163	354	517	800	1,755	2,555
		2,280	7,912	10,192	584	1,999	2,583	2,864	9,911	12,775

Despite the restriction on external sources, the participants were permitted to use pre-trained models as long as they were fine-tuned on the given training set. Also, they were allowed to perform pre-processing, sampling, over/under-sampling, and feature selection and engineering on the training dataset, but they were prohibited from performing the same steps on the testing set except for pre-processing and feature engineering.

Since the competition focused on binary classification for a given category, *i.e.,* a sentence does or does not belong to a category, we evaluated the classification performance of each classifier using Precision, Recall, and F_1-score on the testing set. Although the participants were expected to report these three metrics, we used the F_1-score to measure the overall performance of the models. The 19 F_1-scores of the proposed classifiers were compared against the 19 F_1-scores achieved by the baseline classifiers to rank the participants and determine a winner. We only allowed the classifiers to implement a single model, *e.g.,* BERT or SVM, for all categories, rather than implementing distinct models for different categories.

The winner of the competition was the model with the highest score as determined by the following formula:

$$score(m) = (avg.\ F_1)\ \times\ 0.75\ +\ (\%\ OC)\ \times\ 0.25$$

where *score(m)* represents the score of the model m, *avg.* F_1 is the average of the F_1-scores achieved by the proposed model across all the 19 categories, and $\%\ OC$ represents the proportion of the 19 categories for which the proposed model outperforms the baseline Random Forest model by F_1-score. With this formula, the participants were encouraged to outperform the baselines as much as possible, measured by the F_1-score, for as many categories as possible.

C. Submitted Classification Models

Three teams participated in the competition by submitting one or more classification models. Two teams proposed fine-tuned transformer-based models [14], [16] and the remaining team proposed canonical machine learning models [15].

Al-Kaswan *et al.* [14] proposed SentenceTransformer-Assisted Comment Classifiers (STACC), a set of SentenceTransformers-based binary code comment classifiers. These are lightweight classifiers trained and tested on the provided dataset. The authors used SetFit [33], an efficient and prompt-free framework for few-shot fine-tuning of Sentence Transformers. For the fine-tuning, the authors relied on the Optuna backend with SetFit to find the best hyperparameters using data from the Java *deprecation* category. The performance of the model with the best hyperparameters was obtained on the data from all the categories in the test set. Finally, the authors experimented with appending the code file name to the corresponding comments, separated by the '|' symbol, to improve the performance of the models. The authors made all their fine-tuned models available on the HuggingFace Hub and integrated the models into an interactive HuggingFace Space that is accessible online and via a free API [14].

Li *et al.* [16] relied on transfer learning from pre-trained language models for code-related tasks. In particular, they proposed to fine-tune CodeT5 [38], a Transformer-based model pre-trained on source code from a variety of open-source projects. In addition, the authors conducted a pre-processing step to normalize the competition dataset, *e.g.,* lowercase transformation and removal of special characters such as #%.?. They also leveraged the NLP features provided

TABLE III
RESULTS OF THE CODE COMMENT CLASSIFICATION COMPETITION. THE MODELS ARE RANKED USING THE SCORE GIVEN IN SECTION III-B.

Participants	Classification Model	Average Precision	Average Recall	Average F_1-score	Outperformed Categories	Ranking Score
Al-Kaswan *et al.* [14]	STACC	0.710	**0.794**	**0.744**	19/19	**0.808**
Li *et al.* [16]	CodeT5	**0.728**	0.606	0.657	19/19	0.743
Indika *et al.* [15]	Logistic Regression	0.540	0.560	0.547	19/19	0.660
	Linear SVC	0.542	0.558	0.547	18/19	0.647
	Random Forest	0.661	0.479	0.537	17/19	0.626
	Decision Tree	0.495	0.506	0.493	18/19	0.607
	Multinomial Naïve Bayes	0.484	0.589	0.493	16/19	0.604
	Multi-Layer Perceptron	0.564	0.507	0.523	16/19	0.603
	Bernoulli Naïve Bayes	0.478	0.585	0.523	16/19	0.602
	K-Nearest Neighbors	0.526	0.490	0.503	16/19	0.588
Rani *et al.* [3]	Random Forest	0.439	0.245	0.309	-	-

by Rani *et al.* [3], *e.g.,* phrases such as "see example" or "results" for the Python category *usage*). These category-specific features were wrapped between special tokens in the form of <*s*>*features*<*s*> to make the model pay more attention to key features that can help with classification. A pre-trained tokenizer based on the Byte-Pair Encoding (BPE) is used to tokenize the code comments and fixed hyperparameters were used to fine-tune the model.

Indika *et al.* [15] experimented with eight canonical machine learning models for code comment classification, namely Logistic Regression, Linear SVC, Random Forest, Decision Tree, Multinomial Naïve Bayes, Multi-Layer Perceptron, Bernoulli Naïve Bayes, and K-nearest Neighbors. The best model hyperparameters, based on grid search, were found using 10-fold cross-validation on the training set. The models with the best hyperparameters were executed on the test set to measure their classification performance. Before training or inference, the authors included a pre-processing step to adapt English sentences of a typical code comment to a standard format: they applied a set of text transformations to remove white spaces, expand English contraction, remove non-alphanumeric characters, and more. Random oversampling was applied to mitigate the issue of imbalanced positive and negative comment sentences.

D. Classifier Evaluation and Results

We executed the code provided in the replication packages of the three teams to replicate the results reported in the corresponding papers [14]–[16]. For all three submissions, we were able to replicate the results reported in the corresponding papers.

For replicating Al-Kaswan *et al.* [14] the authors provided Google Colab notebooks ready to use. However, due to the complexity of the model, running the notebooks on Colab demanded extensive computational resources that were only available with a paid subscription. Therefore, we relied on a local workstation, equipped with three Tesla T4 GPUs, each with 15GB of memory, to replicate the results. Following the advice of the authors, we reduced the number of trials to two (2) in the model selection pipeline, which executed successfully. Additionally, the model training pipeline executed successfully for 17 of 19 models, raising an exception that prevented the training of the remaining two ones. After informing the authors about the issue, they provided a new notebook that executed their fine-tuned models hosted on HuggingFace. We executed the notebook and were able to replicate the paper results. Al-Kaswan *et al.*'s STACC classifiers achieved an average F_1-score of 0.744 and outperformed the baseline model on all categories. Based on the results, the ranking score for the competition is 0.808. The best STACC results coincide with the best results of Li *et al.* [16]: *Ownership* (Java) shows 100% F_1-score. The two teams also share the class *Development notes* as the worst performing case due to the same challenges previously described.

Besides the entire pipeline for training their models, Li *et al.* [16] provided all fine-tuned models ready to use for inference as well as the dataset split in train, validation, and test sets. In any case, we used a workstation equipped with a Tesla V100S GPU with 32GB of memory to re-execute their entire pipeline. This allowed us to verify that there were no issues in the written report. Although split into batches, the entire process requested approximately four days of computation. As reported in the paper [16], the proposed CodeT5 model achieves an average F_1-score of 0.657 and outperformed the baseline model on all categories (*cf.* Table III). Based on the results, the ranking score for the competition is 0.743. The model achieves the best performance with the class *Ownership* for the programming language Java with an F_1-score of 100%. In contrast, the model struggles with the *Development notes* class (Python), as our model does, probably due to a less formalism adopted by developers in writing this kind of comments.

To replicate the eight models of Indika *et al.* [15], we used the same workstation with the three Tesla T4 GPUs. We executed the pre-trained best classifiers provided by the authors, which match the results of their paper. In contrast to the other two teams, by providing eight different models, Indika *et al.* [15] showed that the best-performing model does not necessarily outperform the baseline for all categories. For example, although the proposed Decision Tree achieves only 0.607 with our ranking score, it shows the best performance on the challenging case of Python's *Development notes* category.

Indika *et al.*'s Logistic Regression model is the highest-ranked model, achieving an average F_1-score of 0.547 and outperforming the baselines on all categories. Based on the results, the ranking score for the competition is 0.660, which is considerably lower than the models from the other two teams.

Table III reveals that all the proposed models significantly outperform the baseline Random Forests by Rani *et al.* [3]. However, the transformer-based models STACC [14] and CodeT5 [16] are significantly superior to the canonical models proposed by Indika *et al.* [15], in terms of average Precision, Recall, and F_1-score. CodeT5's Precision is slightly higher than that of STACC, but STACC's Recall is substantially higher, thus explaining the STACC's higher average F_1-score. It is interesting that Indika *et al.*'s Random Forests achieve slightly higher Precision than the baseline Random Forests, yet their Recall is substantially higher. These results are likely explained by the fact that Indika *et al.* performed a hyper-parameter search while the baseline Random Forests re-used prior hyper-parameters which may not necessarily be optimal for the competition dataset.

Table III shows that the highest score is achieved by Al-Kaswan *et al.*'s models [14], making Al-Kaswan *et al.* the winners of the competition. The competition ranking is as follows:

1) Al-Kaswan *et al.* [14] is the winner of the competition with their transformer-based STACC models;
2) Li *et al.* [16] take the second place of the competition with their transformed-based CodeT5 models; and
3) Indika *et al.* [15] take the third place with their logistic regression classifiers.

IV. Conclusions and Final Remarks

The NLBSE'23 Tool Competition attracted five teams that proposed a diverse set of classification models to automatically classify issue reports or code comments.

Our issue report classification baseline model remained uncontested despite the hard work from the participants [12], [13]. We believe more models were trained and tested but not submitted to the competition due to the difficulty in outperforming our baseline. The baseline classifier [17], [22] utilized RoBERTa [19], a state-of-the-art language model based on the Transformer architecture, leveraging various information sources from the issues. Careful pre-processing of the issue reports appear to be the main factor for achieving such performance. For the next edition, we plan to improve the issue report classification dataset by reducing the variability in labeling rationale as it affects the reliability and effectiveness of models, which may lead to inaccurate results [13], [39].

Three teams participated in the code comment classification competitions, all outperforming the baseline model based on Random Forest [3]. The competition showed a significant superiority of language-model-based approaches, *i.e.,* STACC [14] and CodeT5 [16], over canonical machine learning models, *e.g.,* SVMs, Logistic Regression, or Decision Trees [3], [15]. This means that general-purpose textual data (used by STACC [14]) and open-source code (used by CodeT5 [16]) used for pre-training the models is beneficial for code comment classification. Fine-tuning may have have had an effect on these models, yet it is unclear by how much. In contrast, lexical features and grammatical patterns used by the canonical machine learning models are insufficient to achieve adequate classification performance. While the teams performed data pre-processing, it is unclear how much it helped for improving the performance. Finally, we plan to include in future editions of the code comment competition a larger and more balanced dataset, deep-learning-based baselines, and possibly, a code comment multi-label classification task.

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 (non)code-related textual artifacts. We also plan to extend the competition with techniques previously used for user review analysis [40]–[43], categorizing safety-related issues [44], or fine-grained analysis of bug reports [45]–[49].

Acknowledgments

We thank all the participants of the competition for their support in launching the second edition of the NLBSE'23 Tool Competition. We gratefully acknowledge the Horizon 2020 (EU Commission) support for the project *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] A. Di Sorbo and S. Panichella, "Summary of the 1st Natural Language-based Software Engineering Workshop (NLBSE 2022)," *ACM SIGSOFT Softw. Eng. Notes*, vol. 48, no. 1, pp. 101–104, 2023.

[2] R. Kallis, O. Chaparro, A. Di Sorbo, and S. Panichella, "NLBSE'22 Tool Competition," in *Proceedings of The 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*, 2022.

[3] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, and O. Nierstrasz, "How to identify class comment types? A multi-language approach for class comment classification," *Journal of Systems and Software*, vol. 181, p. 111047, 2021.

[4] P. Rani, S. Panichella, M. Leuenberger, M. Ghafari, and O. Nierstrasz, "What do class comments tell us? An investigation of comment evolution and practices in Pharo Smalltalk," *Empirical software engineering*, vol. 26, no. 6, p. 112, 2021, publisher: Springer.

[5] S. Panichella, G. Canfora, and A. Di Sorbo, "Won't We Fix this Issue? Qualitative characterization and automated identification of wontfix issues on GitHub," *Information and Software Technology*, vol. 139, p. 106665, 2021.

[6] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5333–5369, 2020.

[7] M. Izadi, K. Akbari, and A. Heydarnoori, "Predicting the objective and priority of issue reports in software repositories," *Empirical Software Engineering*, vol. 27, no. 2, p. 50, 2022, publisher: Springer.

[8] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, 2005, pp. 68–75.

[9] L. Pascarella and A. Bacchelli, "Classifying code comments in Java open-source software systems," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 227–237.

[10] A. Aghamohammadi, M. Izadi, and A. Heydarnoori, "Generating summaries for methods of event-driven programs: An Android case study," *Journal of Systems and Software*, vol. 170, p. 110800, 2020, publisher: Elsevier.

[11] P. Rani, A. Blasi, N. Stulova, S. Panichella, A. Gorla, and O. Nierstrasz, "A decade of code comment quality assessment: A systematic literature review," *Journal of Systems and Software*, vol. 195, p. 111515, 2023.

[12] M. Laiq, "An Intelligent Tool for Classifying Issue Reports," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023, p. (to appear).

[13] G. Colavito, F. Lanubile, and N. Novielli, "Few-Shot Learning for Issue Report Classification," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023, p. (to appear).

[14] A. Al-Kaswan, M. Izadi, and A. van Deursen, "STACC: Code Comment Classification using Sentence Transformers," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023, p. (to appear).

[15] A. Indika, P. Y. Washington, and A. Peruma, "Performance Comparison of Binary Machine Learning Classifiers in Identifying Code Comment Types: An Exploratory Study," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023, p. (to appear).

[16] Y. Li, H. Wang, H. Zhang, and S. H. Tan, "Classifying Code Comments via Pre-trained Programming Language Model," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023, p. (to appear).

[17] R. Kallis, M. Izadi, L. Pascarella, O. Chaparro, and P. Rani, "The NLBSE'23 Tool Competition," in *Proceedings of The 2nd International Workshop on Natural Language-based Software Engineering (NLBSE'23)*, 2023.

[18] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," Aug. 2016, arXiv:1607.01759 [cs]. [Online]. Available: http://arxiv.org/abs/1607.01759

[19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," Jul. 2019, arXiv:1907.11692 [cs].

[20] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket Tagger: Machine Learning Driven Issue Classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2019, pp. 406–409, iSSN: 2576-3148.

[21] ——, "Predicting issue types on GitHub," *Science of Computer Programming*, vol. 205, p. 102598, May 2021.

[22] M. Izadi, "CatIss: An Intelligent Tool for Categorizing Issues Reports using Transformers," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, May 2022, pp. 44–47.

[23] S. Bharadwaj and T. Kadam, "GitHub Issue Classification Using BERT-Style Models," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, May 2022, pp. 40–43.

[24] G. Colavito, F. Lanubile, and N. Novielli, "Issue report classification using pre-trained language models," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 2022, pp. 29–32.

[25] M. L. Siddiq and J. C. S. Santos, "BERT-Based GitHub Issue Report Classification," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, May 2022, pp. 33–36.

[26] A. Trautsch and S. Herbold, "Predicting Issue Types with seBERT," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, May 2022, pp. 37–39.

[27] I. Grigorik, "GH Archive," 2012. [Online]. Available: https://www.gharchive.org/

[28] J. Cabot, J. L. C. Izquierdo, V. Cosentino, and B. Rolandi, "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, 2015, pp. 550–554.

[29] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.zip: Compressing text classification models," Dec. 2016, arXiv:1612.03651 [cs]. [Online]. Available: http://arxiv.org/abs/1612.03651

[30] M. Izadi and M. N. Ahmadabadi, "On the evaluation of NLP-based models for software engineering," in *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 2022, pp. 48–50.

[31] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 116.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] L. Tunstall, N. Reimers, U. E. S. Jo, L. Bates, D. Korat, M. Wasserblat, and O. Pereg, "Efficient Few-Shot Learning Without Prompts," *arXiv preprint arXiv:2209.11055*, 2022.

[34] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[35] A. Di Sorbo, C. A. Visaggio, M. Di Penta, G. Canfora, and S. Panichella, "An NLP-based Tool for Software Artifacts Analysis," in *IEEE International Conference on Software Maintenance and Evolution, ICSME, Luxembourg*. IEEE, 2021, pp. 569–573.

[36] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009, publisher: ACM New York, NY, USA.

[37] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "Exploiting Natural Language Structures in Software Informal Documentation," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1587–1604, 2021.

[38] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," *arXiv preprint arXiv:2109.00859*, 2021.

[39] X. Wu, W. Zheng, X. Xia, and D. Lo, "Data quality matters: A case study on data label correctness for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2541–2556, 2021, publisher: IEEE.

[40] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *International Conference on Software Maintenance and Evolution*. IEEE, 2015, pp. 281–290.

[41] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 499–510.

[42] A. Di Sorbo, G. Grano, C. A. Visaggio, and S. Panichella, "Investigating the criticality of user-reported issues through their relations with app rating," *J. Softw. Evol. Process.*, vol. 33, no. 3, 2021.

[43] S. Panichella, "Summarization techniques for code, change, testing, and user feedback (Invited paper)," in *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*, C. Artho and R. Ramler, Eds. IEEE, 2018, pp. 1–5.

[44] A. Di Sorbo, F. Zampetti, C. A. Visaggio, M. Di Penta, and S. Panichella, "Automated Identification and Qualitative Characterization of Safety Concerns Reported in UAV Software Platforms," *ACM Trans. Softw. Eng. Methodol.*, Sep. 2022, place: New York, NY, USA Publisher: Association for Computing Machinery.

[45] Y. Song, J. Mahmud, N. De Silva, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk, "BURT: A Chatbot for Interactive Bug Reporting," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, 2023, p. (to appear).

[46] Y. Song, J. Mahmud, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk, "Toward interactive bug reporting for (Android app) end-users," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'22)*, 2022, pp. 344–356.

[47] Y. Song and O. 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)*, 2020, pp. 1551–1555.

[48] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. 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)*, 2019, pp. 86–96.

[49] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*, 2017, pp. 396–407.