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)
https://doi.org/10.1109/NLBSE59153.2023.00007

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights. We will remove access to the work immediately and investigate your claim.
Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.
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  
osarch@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-
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.

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], astatic 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.

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 F1-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 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 [30]. Note that micro-average Precision and Recall scores are the same as macro-average F1-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 F1-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 F1-score of 0.852.
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 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 1,060 comments.

We applied various pre-processing steps to the comments. We split the comments into sentences based on the NEON tool\(^3\), 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\(^3\) 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\(^3\), [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/hlbose2023/code-comment-classification

\(^3\)https://tinyurl.com/45ccyv6m
the FastText model outperforms the baselines as much as possible, measured by F1-score. With this formula, the participants were encouraged to model across all the 19 categories, and

\[ \text{avg. } F_1 \times 0.75 + (\% \text{ OC}) \times 0.25 \]

where \( \text{score}(m) \) 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 \( \% \text{ OC} \) 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 `#`.

### TABLE II

<table>
<thead>
<tr>
<th>Language</th>
<th>Categories</th>
<th>Training</th>
<th>Testing</th>
<th>Training + Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Positive</td>
<td>Negative</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Positive</td>
<td>Negative</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Positive</td>
<td>Negative</td>
<td>Total</td>
</tr>
<tr>
<td>Java</td>
<td>Expand</td>
<td>505</td>
<td>1,426</td>
<td>1,931</td>
</tr>
<tr>
<td></td>
<td>Ownership</td>
<td>90</td>
<td>1,839</td>
<td>1,929</td>
</tr>
<tr>
<td></td>
<td>Deprecation</td>
<td>100</td>
<td>1,831</td>
<td>1,931</td>
</tr>
<tr>
<td></td>
<td>Rational</td>
<td>223</td>
<td>1,707</td>
<td>1,931</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>328</td>
<td>1,600</td>
<td>1,928</td>
</tr>
<tr>
<td></td>
<td>Pointer</td>
<td>289</td>
<td>1,640</td>
<td>1,929</td>
</tr>
<tr>
<td></td>
<td>Usage</td>
<td>728</td>
<td>2,103</td>
<td>1,931</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,263</td>
<td>11,246</td>
<td>13,509</td>
</tr>
<tr>
<td></td>
<td>Responsibilities</td>
<td>267</td>
<td>1,139</td>
<td>1,406</td>
</tr>
<tr>
<td></td>
<td>Key messages</td>
<td>242</td>
<td>1,165</td>
<td>1,407</td>
</tr>
<tr>
<td></td>
<td>Key impl. points</td>
<td>184</td>
<td>1,222</td>
<td>1,406</td>
</tr>
<tr>
<td></td>
<td>Collaborators</td>
<td>99</td>
<td>1,307</td>
<td>1,406</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>596</td>
<td>812</td>
<td>748</td>
</tr>
<tr>
<td></td>
<td>Class references</td>
<td>60</td>
<td>1,348</td>
<td>1,408</td>
</tr>
<tr>
<td></td>
<td>Intent</td>
<td>173</td>
<td>1,236</td>
<td>1,409</td>
</tr>
<tr>
<td>Pharo</td>
<td></td>
<td>1,624</td>
<td>8,229</td>
<td>9,850</td>
</tr>
<tr>
<td></td>
<td>Expand</td>
<td>402</td>
<td>1,637</td>
<td>2,039</td>
</tr>
<tr>
<td></td>
<td>Parameters</td>
<td>633</td>
<td>1,404</td>
<td>2,037</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>361</td>
<td>1,678</td>
<td>2,039</td>
</tr>
<tr>
<td></td>
<td>Dev. notes</td>
<td>247</td>
<td>1,236</td>
<td>1,483</td>
</tr>
<tr>
<td></td>
<td>Usage</td>
<td>637</td>
<td>1,401</td>
<td>2,038</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,280</td>
<td>7,912</td>
<td>10,192</td>
</tr>
</tbody>
</table>

**Usage 637 1,401 2,038 163 354 517 800 1,755 2,555
Dev. notes 247 1,792 2,039 65 451 516 312 2,243 2,555
Summary 361 1,678 2,039 93 423 516 454 2,101 2,555
Parameters 633 1,404 2,037 161 357 518 794 1,761 2,555
Expand 402 1,637 2,039 102 414 516 504 2,051 2,555
Key messages 242 1,165 1,409 93 423 516 415 2,003 2,555
Collaborators 99 1,307 1,406 37 311 359 232 1,534 1,765
Example 596 812 748 152 205 357 748 1,017 1,765
Class references 60 1,348 1,408 45 311 356 454 2,101 2,555
Intent 173 1,236 1,409 45 311 356 218 1,547 1,765
Key impl. points 184 1,222 1,406 37 311 356 232 1,534 1,765
Key messages 242 1,165 1,407 37 311 356 232 1,534 1,765
Collaborators 99 1,307 1,406 37 311 356 232 1,534 1,765
Example 596 812 748 152 205 357 748 1,017 1,765
Class references 60 1,348 1,408 45 311 356 232 1,534 1,765
Intent 173 1,236 1,409 45 311 356 218 1,547 1,765
Key impl. points 184 1,222 1,406 37 311 356 232 1,534 1,765

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 F1-score on the testing set. Although the participants were expected to report these three metrics, we used the F1-score to measure the overall performance of the models. The 19 F1-scores of the proposed classifiers were compared against the 19 F1-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:

\[ \text{score}(m) = (\text{avg. } F_1) \times 0.75 + (\% \text{ OC}) \times 0.25 \]
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 SVM, 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 F1-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% F1-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 F1-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 F1-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.

---

### TABLE III

<table>
<thead>
<tr>
<th>Participants</th>
<th>Classification Model</th>
<th>Average Precision</th>
<th>Average Recall</th>
<th>Average F1-score</th>
<th>Outperformed Categories</th>
<th>Ranking Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al-Kaswan et al.</td>
<td>STACC</td>
<td>0.710</td>
<td>0.794</td>
<td>0.744</td>
<td>19/19</td>
<td>0.808</td>
</tr>
<tr>
<td>Li et al. [16]</td>
<td>CodeT5</td>
<td>0.728</td>
<td>0.606</td>
<td>0.657</td>
<td>19/19</td>
<td>0.743</td>
</tr>
<tr>
<td>Indika et al. [15]</td>
<td>Logistic Regression</td>
<td>0.540</td>
<td>0.560</td>
<td>0.547</td>
<td>19/19</td>
<td>0.660</td>
</tr>
<tr>
<td></td>
<td>Linear SVC</td>
<td>0.542</td>
<td>0.558</td>
<td>0.547</td>
<td>18/19</td>
<td>0.647</td>
</tr>
<tr>
<td></td>
<td>Random Forest</td>
<td>0.661</td>
<td>0.479</td>
<td>0.537</td>
<td>17/19</td>
<td>0.626</td>
</tr>
<tr>
<td></td>
<td>Decision Tree</td>
<td>0.495</td>
<td>0.506</td>
<td>0.493</td>
<td>18/19</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>Multinomial Naïve Bayes</td>
<td>0.484</td>
<td>0.589</td>
<td>0.493</td>
<td>16/19</td>
<td>0.604</td>
</tr>
<tr>
<td></td>
<td>Multi-Layer Perceptron</td>
<td>0.564</td>
<td>0.507</td>
<td>0.523</td>
<td>16/19</td>
<td>0.603</td>
</tr>
<tr>
<td></td>
<td>Bernoulli Naïve Bayes</td>
<td>0.478</td>
<td>0.585</td>
<td>0.523</td>
<td>16/19</td>
<td>0.602</td>
</tr>
<tr>
<td></td>
<td>K-Nearest Neighbors</td>
<td>0.526</td>
<td>0.490</td>
<td>0.503</td>
<td>16/19</td>
<td>0.588</td>
</tr>
<tr>
<td>Rani et al. [3]</td>
<td>Random Forest</td>
<td>0.439</td>
<td>0.245</td>
<td>0.309</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

---

```
Indika et al.’s Logistic Regression model is the highest-ranked model, achieving an average F1-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 F1-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 F1-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 transformer-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 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
