The Journal of Systems & Software 210 (2024) 111838

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Impact of data quality for automatic issue classification using pre-trained language models ☆

Giuseppe Colavito *, Filippo Lanubile, Nicole Novielli, Luigi Quaranta

*University of Bari, Italy*

ARTICLE INFO

*Keywords:*
GitHub
Issue trackers
Issue labeling
BERT
Model quality
Label correctness

ABSTRACT

Issue classification aims to recognize whether an issue reports a bug, a request for enhancement or support. In this paper we use pre-trained models for the automatic classification of issues and investigate how the quality of data affects the performance of classifiers. Despite the application of data quality filters, none of our attempts had a significant effect on model quality. As root cause we identify a threat to construct validity underlying the issue labeling.

*Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.*

## 1. Introduction

Issue trackers are used to manage requests for change, such as bug fixing or product improvement, and requests for support. An issue report usually includes an identifier, a description, the author, the status (e.g., open, assigned, closed), a thread of comments, and a label such as bug, enhancement or support. However, submitters frequently misclassify labels by confounding improvement requests as bugs, and vice versa (Antoniol et al., 2008). Herzig et al. (2013) report that 33.8% of all issue reports are incorrectly categorized as shown in an extensive investigation covering more than 7000 issues across 5 projects. Automatic classification of issues could be helpful in supporting effective issue management and prioritization, thus justifying the interest of the research community on this topic (Pandey et al., 2017).

Previous studies have proposed supervised approaches to address the task of automatically predicting the label that should be assigned to a new issue. Early studies leveraged traditional machine learning in combination with text-based features (Antoniol et al., 2008). Neural-network-based approaches to distributional semantics, also known as word embeddings (Levy and Goldberg, 2014; Mikolov et al., 2013), have received increasing attention and are now regarded as the state of the art for several natural language processing (NLP) tasks, including text categorization. Kallis et al. (2021, 2019) proposed Ticket Tagger, a machine learning classifier trained on GitHub data, which leverages the textual content of an issue title and body, whose vectorial representation is based on *fastText* (Joulin et al., 2017). Among recent advances, BERT (Bidirectional Encoder Representations from Transformers) has emerged as a robust approach for task-agnostic pre-training of language models (Devlin et al., 2019). It outperformed the state of the art in several NLP tasks, including issue classification.

In this paper, we report how we exploited pre-trained language models for automatic issue labeling. Hence, our first research question can be formulated as follows:

*RQ1: To what extent we can leverage pre-trained language models to enhance the state of the art in automatic issue labeling?*

To address our first research question, we performed an empirical study in the scope of the NLBSE'22 tool competition (Kallis et al., 2022). The goal of the challenge was to build a classifier for automatic issue report classification. The organizers provided a dataset including more than 800 K issue reports, extracted from GitHub open-source software projects and labeled by their authors as either *bug*, *enhancement*, or *question* (Kallis et al., 2021, 2019). The participants were invited to use the dataset to train and evaluate machine learning (ML) models for the automatic classification of issues. To solve the task, we proposed two models based on supervised learning that leverage the information available at the time of issue writing, that is the title and body of the issue and the issue-author association (e.g., collaborator, owner, etc.). We experimented with the fine-tuning of BERT (Devlin et al., 2019) and its variants ALBERT (Lan et al., 2020) and RoBERTa (Liu et al., 2019). To combine text and author information, we also trained a multilayer perceptron (MLP) classifier that leverages the BERT-based embedding of the issue with a one-hot encoding representation of the author-issue relation. Both ML models outperformed the baseline.

As a follow-up of the challenge – after analyzing misclassified cases – we focused on investigating the relationship between data and model

☆ Editor: Alexander Chatzigeorgiou.

* Corresponding author.
*E-mail addresses:* giuseppe.colavito@uniba.it (G. Colavito), filippo.lanubile@uniba.it (F. Lanubile), nicole.novielli@uniba.it (N. Novielli), luigi.quaranta@uniba.it (L. Quaranta).

https://doi.org/10.1016/j.jss.2023.111838
Received 16 December 2022; Received in revised form 6 April 2023; Accepted 4 September 2023
Available online 14 December 2023

quality as part of our study's second goal. The error analysis suggests that one of the main causes of issue misclassification is the presence of inconsistencies in the labeling rationale or the presence of issues tagged with more than one label, which might introduce noise in the model training. In fact, the issues in the dataset were collected using only a time-based criterion for inclusion. Conversely, previous work on GitHub mining suggests that a series of proxies could be used as indicators for data quality, such as the project star count (Biswas et al., 2019; Munaiah et al., 2016). Hence, we formulate the second research question as follows:

*RQ2: To what extent the performance of a model can be improved by improving the quality of the training data?*

Prior research already explored the influence of data quality on model quality by means of manual label verification (Wu et al., 2022). Nevertheless, manual annotation is a laborious and time-consuming task. This study investigates the efficacy of operationalizing data quality criteria in the form of filters that can be automatically applied on training data. To address our second research question, we define a number of data quality criteria based on previous research in this field. We operationalize them into a set of filters that we then apply on the former GitHub dataset to progressively filter out uncertain data. In addition, we include a new dataset of Jira issues (Montgomery et al., 2022) that already meets these data quality criteria.

The main contributions of this work are as follows:

- We propose and assess supervised classifiers for GitHub issue labeling that leverage pre-trained language models. The best model achieves a performance of F1 = .8591 using textual information only extracted from the issue body and title.
- We investigate the impact of data quality on our automatic issue classifiers. We found that neither the most popular nor the most mature projects generate better predictions of issue labeling. We speculate that the negative result in improving the issue classification is caused by conceptual inconsistencies in the labeling, which make any subsequent data cleanup action useless.
- We build and distribute a lab package to verify, replicate, and build upon the present study. The replication material is available on GitHub (Colavito et al., 2022b).

The remainder of this paper is structured as follows. In Section 2, we report the background on word embeddings and pre-trained language models (i.e., BERT and its variants) and we discuss the importance of ensuring data quality when building ML models. In Section 3, we present the datasets used in our experiments; then, in Section 4, we describe the methodology of our study. In Section 5, we address our first research question by reporting the results of the model performance evaluation conducted on the test set and comparing it with the baseline approach. As a further contribution of this study, we report the results of an error analysis carried out on the misclassified cases. In Section 6, we address our second research question by reporting the impact of data quality filters on the classification performance based on the GitHub and Jira datasets (see Sections 6.1 and 6.2, respectively). We discuss our findings in Section 7, where we also summarize recent related work on issue classification. The paper is concluded in Section 8.

## 2. Background and related work

### 2.1. Text embedding

Effectively modeling semantics of natural language has been a subject of study for computational linguistics since long. In line with the *meaning-is-use* assumption, – i.e., the semantics of words can be inferred by its contextual use – and thanks to the recent availability and accessibility of higher computational power resources, recent studies led to the development and release of robust pre-trained, task-agnostic language models that successfully achieve state-of-the-art performance in many natural language processing (NLP) applications. In particular, word embeddings (Levy and Goldberg, 2014; Mikolov et al., 2013), have been used to address several NLP tasks, including text categorization, achieving state-of-the-art performance.

Among others, BERT (Bidirectional Encoder Representations from Transformers) represents the most recent advancement of research in the NLP field. BERT was proposed by Devlin et al. (2019) for the pre-training of language models using deep bidirectional transformers. Since its introduction, BERT outperformed state-of-the-art approaches in several NLP tasks, thus representing a disruptive innovation in computational linguistic research. Differently from previous language models, which provide context-free embedding of words (see, for example Word2Vec (Mikolov et al., 2013)), BERT generates representations of words based on their context. BERT is task-agnostic and can be easily embedded in a text classifier thanks to transfer learning and fine-tuning of the parameters of the pre-trained model originally released by Google. One of the main advantages of using a BERT-based classifier is the possibility of leveraging transfer learning to adapt a pre-trained language model originally obtained by exploiting a huge corpus of unlabeled documents. Compared to model pre-training, the fine-tuning step is less expensive albeit able to outperform task-specific architectures for several NLP tasks (Devlin et al., 2019), while still enabling the training of robust task-specific classifiers.

Since its release, alternative versions of BERT-based language models have been proposed to address some of the limitations of the original model (Liu et al., 2019; Lan et al., 2020; Sanh et al., 2019). Sanh et al. (2019) released DistilBERT, a model trained with half of the BERT parameters to reduce the time needed to train the language model. Liu et al. (2019) replicated the original study by Devlin et al. and retrained the BERT model by introducing modifications to improve the accuracy. Specifically, they trained the Robustly-optimized BERT (RoBERTa) for a longer time, with more epochs and a bigger batch size, thus obtaining a more robust pre-trained language model. Differently, to build ALBERT (Lan et al., 2020), Lan et al. leveraged factorized embeddings to reduce overfitting during the fine-tuning of NLP models.

### 2.2. Automatic classification of issues

Issue tracking systems allow users to report the problems of a software product by entering a brief textual summary, typically composed of a title and an optional description. They can be standalone tools, such as Jira,[1] or tools integrated in code hosting platforms like GitHub.[2]

While this kind of software solution lowers the entry barrier and brings more novice external contributors, it complicates the work of maintainers, as several issues of various types and quality are typically submitted (Bissyandé et al., 2013; Panichella et al., 2014; Fan et al., 2017). Maintainers can utilize customized labeling to mark and organize issue reports. Labels can provide quick hints about issues, such as what kind of topic an issue is about, what development task the issue is related to, or what priority the issue has. Labels are then helpful for project management because they can act as both a classification and filtering mechanism (Cánovas Izquierdo et al., 2015; Liao et al., 2018). However, the labeling mechanism on GitHub is rarely used by contributors (Kallis et al., 2019; Bissyandé et al., 2013) and maintainers have to spend a lot of effort for manually labeling issues (Fan et al., 2017).

Previous studies presented several approaches to automatically categorize issues posted in tracking systems. Antoniol et al. (2008) show that machine learning models may be used to distinguish between bugs and other types of issues. Six alternative issue categories are introduced by Herzig et al. (2013): bug, feature request, improvement request, documentation request, and others. Zhou et al. (2014) merge structured

[1] https://www.atlassian.com/en/software/jira
[2] https://GitHub.com

and unstructured free-text data to train a classifier that can accurately determine if a bug report is indeed a bug or another type of issue.

More recently, researchers started using deep learning and, in particular, pre-trained language models, such as BERT and its variants (Wang et al., 2021; Izadi et al., 2022).

Lately, Kallis et al. (2021, 2019) proposed Ticket Tagger, a machine learning classifier that predicts the label to assign to issues trained on GitHub data. Specifically, Ticket Tagger leverages only the textual content of an issue title and body, whose vectorial representation is based on *fastText* (Joulin et al., 2017), an open-source tool released by Facebook AI research. Ticket Tagger was identified by the NLBSE tool competition organizers as the baseline system, and all participants were invited to compare the performance of the proposed systems with it.

### 2.3. Data quality

Data cleaning, i.e., the process of removing data quality problems, is an activity of uttermost importance in any ML workflow because performance can suffer considerably if models are trained on bad-quality data (Halevy et al., 2009; Kästner, 2021). However, data cleaning is among the most time-consuming chores in the data science practice (Sambasivan et al., 2021); according to a survey administered to 80 practitioners in the field, such task accounts for about 60% of the work accomplished by data scientists every day (Press, 2016).

To help practitioners timely detect data quality issues and fix them, researchers have started designing systems for the automatic detection and restoration of potential problems in data. For instance, Hynes et al. (2017) built the ‘Data Linter’, i.e., an open-source tool aimed at finding various data-related issues in ML pipelines. Similarly, Rekatsinas et al. (2017) developed HoloClean, a semi-automated data repairing framework; while Data Linter detects problems based on data-patterns and heuristics, HoloClean is powered by a weakly supervised ML approach based on statistical learning and inference. Both tools are optimized to work with structured datasets, although data cleaning is strongly advised also in the case of unstructured text data (Jain et al., 2020).

As regards the data quality of GitHub projects, notwithstanding the plethora of opportunities that GitHub provides for archival studies, some researchers reported a number of potential threats that their colleagues need to take into account when mining data from this platform (Kalliamvakou et al., 2014a; Gousios and Spinellis, 2017; AlMarzouq et al., 2020). In particular, besides enumerating the exciting promises of mining GitHub, Kalliamvakou et al. (2014a) provided evidence of 9 issues (perils) that might hinder the quality of data scraped from the website or gathered from its API: for instance, most of the publicly available projects are personal, they contain only a few commits, and are typically inactive; moreover, many repositories are not used for software development, since several users leverage GitHub as a free storage service or web hosting platform.

## 3. Datasets

We use two publicly available datasets of issues collected from GitHub and Jira projects.

### 3.1. GitHub dataset

The **GitHub** dataset is the gold standard dataset distributed by the NLBSE tool competition organizers (Kallis et al., 2022, 2021, 2019). The issues in the dataset were extracted from the GitHub Archive[3] using Google BigQuery.[4] The dataset consists of more than 800K GitHub issue reports extracted from open source software projects. Each issue receives a *label*, which represents the classification target. Possible class values are (i) *bug*, indicating that the issue contains a bug report, (ii) *enhancement*, indicating that the issue contains suggestion for improvements or requests for new features, and (iii) *question*, assigned to issues containing users' questions about the software usage. In Table 1, we present a sample of the GitHub dataset. The dataset is split into train (90%) and test set (10%), with the same label distribution (see Table 2). The label distribution is unbalanced, with the minority class of questions (9%) being underrepresented compared to bugs (50%) and enhancements (41%).

The organizers of the tool competition selected all the issues closed during the first semester of 2021 (from January 1st 2021 to May 31st 2021) that contained any of the labels *bug*, *enhancement*, and *question* at the issue closing time (Kallis et al., 2022). The dataset was distributed as a CSV file containing raw data, i.e., no preprocessing was applied to the text of the issues, which was shared in the original Markdown[5] format. For each issue, the dataset includes the issue URL, the creation date, the repository URL, the title and the body. Furthermore, the dataset includes an attribute describing the issue-author association, that is the role played by the author in the repository, with values in {*owner*, *contributor*, *member*, *collaborator*, *none*, *mannequin*}.

Labels in the dataset are assigned based on what observed in GitHub. In particular, labels in GitHub can be assigned by the user who opened the issue or by repository maintainers. In case of multiple labels, the organizers of the challenge used the most recent one as the gold label.

### 3.2. Jira dataset

The **Jira** dataset (Montgomery et al., 2022) is built from 16 public Jira repositories containing 1822 projects and 2.7 million issues. Each Jira repository contains issues for multiple projects, e.g. 657 projects for the Apache ecosystem. Issue labels in Jira are heterogeneous and vary across projects. The authors of the dataset performed a thematic analysis to derive a unified set of themes and codes, which is used to label the issues included in the dataset. Each original label in Jira is mapped to a code (e.g., *bug report*) associated to a theme (e.g., *maintenance*).

From the set of codes defined by Montgomery et al. (2022), we identify the ones whose semantics match the labels used in the GitHub dataset, namely *bug*, *enhancement*, and *question*. By doing so, we aim at enabling a fair comparison of the performance achieved by the classifier on the two datasets. We report the selected codes and their mapping to the classification labels in Table 3. In this study, we include only the issues that can be mapped as either *bug*, *enhancement*, or *question*. The resulting dataset, with label distribution and breakdown by project is reported in Table 4.

90% of the Jira dataset is used as training set for our experiments. The remaining 10% is kept out as test set. The split is stratified, in order to preserve the label distribution.

## 4. Methodology

In the following, we describe the design of the empirical study we performed to address our research questions.

To answer RQ1 (“*To what extent we can leverage pre-trained language models to enhance the state of the art in automatic issue labeling?*”) we implement a supervised approach by leveraging state-of-the-art pre-trained language models based on transformers. Specifically, we fine-tune BERT and its variants to address the issue classification task of the challenge and we assess the performance of the classifier on the GitHub dataset (see Section 4.4).

To address RQ2 (“*To what extent the performance of a model can be improved by improving the quality of the training data?*”) we replicate the fine-tuning and evaluation of BERT-based classifiers after the application of filters to improve the quality of the GitHub training data. In addition, we take into consideration the Jira dataset, which by

[3] https://www.gharchive.org/ (Last accessed: Dec. 2023)

[4] https://cloud.google.com/bigquery/ (Last accessed: Dec. 2023)

[5] https://daringfireball.net/projects/markdown/

**Table 1**
A sample of the GitHub dataset.

| Issue_url | Issue_label | Issue_created_at | Issue_author_association | Repository_url | Issue_title | Issue_body |
|---|---|---|---|---|---|---|
| api.GitHub.com/… | bug | 2021-01-02T18:07:30Z | NONE | api.GitHub.com/… | Welcome screen on every editor window is very tedious | I just discovered Gitlens and find the functionality useful, thank you to all who contribute... |
| api.GitHub.com/… | bug | 2020-12-31T18:19:31Z | OWNER | api.GitHub.com/… | "pcopy invite" and "pcopy paste abc:" does not check if clipboard exists | |
| api.GitHub.com/… | bug | 2021-01-03T04:33:36Z | OWNER | api.GitHub.com/… | UI: Modal overlay is half transparent, should not be | |
| api.GitHub.com/… | enhancement | 2020-12-25T00:46:00Z | OWNER | api.GitHub.com/… | Make the loading screen scale with browser window size | Currently the loading wheel is a fixed size in pixels, but it would be better to specify it in terms of percentage of the browser size. |
| api.GitHub.com/… | bug | 2021-01-02T21:36:57Z | OWNER | api.GitHub.com/… | Spectator - Investigate a way to strip weapons before they are spectating a player | To bring magneto stick floating |

**Table 2**
Label distribution in the GitHub dataset.

| | Overall | | Train set | | Test set | |
|---|---|---|---|---|---|---|
| Bug | 401,391 | (50%) | 361,103 | (50%) | 40,288 | (50%) |
| Enhancement | 332,577 | (41%) | 299,374 | (41%) | 33,203 | (41%) |
| Question | 69,449 | (9%) | 62,422 | (9%) | 7,027 | (9%) |
| Total | 803,417 | | 722,899 | | 80,518 | |

**Table 3**
The mapping applied from Jira codes to GitHub issue labels.

| Label | Codes |
|---|---|
| Bug | {'Bug Report'} |
| Enhancement | {'New Feature', 'Improvement Suggestion', 'Feature Request'} |
| Question | {'Support Request', 'Question'} |

**Table 4**
Label distribution in the subset of the Jira dataset used in our study.

| | | Bug | | Enhancement | | Question | |
|---|---|---|---|---|---|---|---|
| | | 1,522,538 | 70% | 628,308 | 29% | 8,787 | <1% |
| Breakdown by project | | | | | | | |
| Jira Name | Year | Bug | | Enhancement | | Question | |
| Apache | 2000 | 523,110 | 62% | 312,671 | 37% | 2,214 | <1% |
| Hyperledger | 2016 | 7,622 | 75% | 2,601 | 25% | 0 | – |
| IntelDAOS | 2016 | 3,616 | 100% | 0 | – | 0 | – |
| JFrog | 2006 | 8,236 | 62% | 4,993 | 38% | 34 | <1% |
| Jira | 2002 | 131,138 | 48% | 138,453 | 51% | 2.438 | 1% |
| JiraEcosystem | 2004 | 20,414 | 67% | 9,958 | 33% | 170 | <1% |
| MariaDB | 2009 | 22,800 | 95% | 1,151 | 5% | 0 | – |
| Mindville | 2015 | 860 | 40% | 1,274 | 60% | 0 | – |
| Mojang | 2012 | 420,819 | 100% | 0 | – | 0 | – |
| MongoDB | 2009 | 48,122 | 54% | 38,768 | 44% | 1,808 | 2% |
| Qt | 2005 | 106,804 | 87% | 15,943 | 13% | 0 | – |
| RedHat | 2001 | 160,937 | 71% | 66,596 | 29% | 408 | <1% |
| Sakai | 2004 | 33,216 | 85% | 5,985 | 15% | 0 | – |
| SecondLife | 2007 | 1,231 | 96% | 48 | 4% | 0 | – |
| Sonatype | 2008 | 6,495 | 61% | 2,480 | 23% | 1,597 | 15% |
| Spring | 2003 | 27,118 | 50% | 27,387 | 50% | 118 | <1% |

construction meets the quality criteria that inspired the design of our filters. We evaluate the performance of the BERT-based classifiers both on the filtered GitHub datasets and on the Jira dataset (see Section 4.5).

As a preliminary step, we preprocess both datasets as described in Section 4.1. The training of the issue classifiers, reported in Section 4.3, is performed by first fine-tuning the pretrained language models, as described in Section 4.2.

### 4.1. Pre-processing

As a first pre-processing step, we identify text patterns indicating non-textual items – such as images, links, or code snippets – and replace them with tokens (e.g., <img> for images). Then, we perform a further text normalization step using *ekphrasis Text Pre-Processor*,[6] which is able to identify URLs, email addresses, percentage or currency symbols, phone numbers, user mentions, times, dates, and numbers. We replace such items with *ad hoc* tokens; also, we use *ekphrasis* to unpack hashtags, contractions, and emojis.

Since the documents need to be fed into either BERT or one of its variants, we encode all the documents in the dataset using the model-specific tokenizer. To avoid exceeding the GPU memory capacity, we pad/truncate each document to 128 tokens, in line with previous work (Wang et al., 2021). We apply the same preprocessing steps to both datasets.

### 4.2. Model fine-tuning

We implement a supervised approach by leveraging state-of-the-art models based on transformers. Specifically – as depicted in Fig. 1 – for the GitHub dataset, we experiment with the fine-tuning of BERT-based models in two different settings. In the first setting (denoted as Classifier 1 in Fig. 1), we leverage the textual content of the issue (i.e., title and body) to fine-tune the language model and obtain the final classifier. In the second setting (denoted as Classifier 2 in Fig. 1), we combine textual data with the information provided by the author-association field and train a feed-forward network using PyTorch.[7]

[6] https://GitHub.com/cbaziotis/ekphrasis
[7] https://pytorch.org/

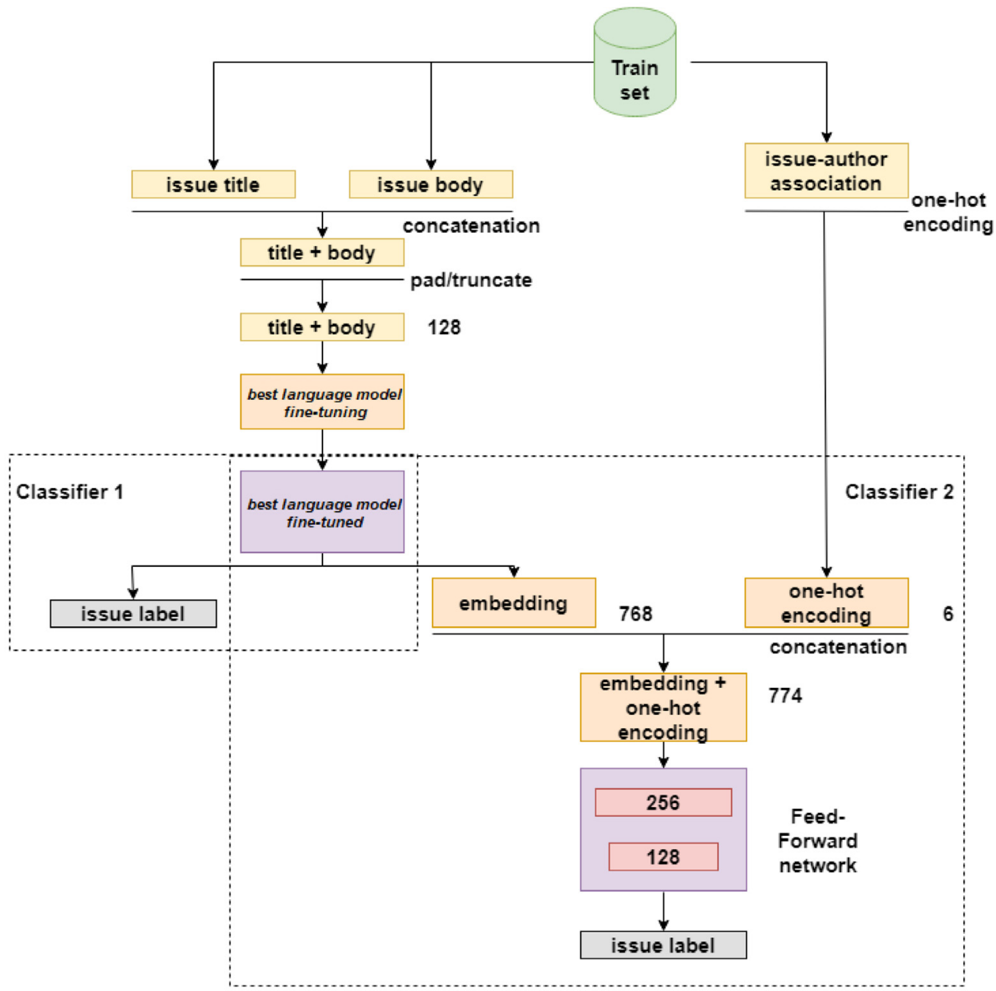


**Fig. 1.** The two classifiers implemented for issue labeling.

For the Jira dataset, we only implement the first approach, as we observe that it outperforms the second classifier trained on the GitHub dataset (Colavito et al., 2022a) (see Table 7).

As a preliminary step, we identify the best pre-trained language model to be used for the issue classification task. To this aim, we conduct some experiments on the GitHub dataset. In particular, we compare the performance of BERT (Devlin et al., 2019), ALBERT (Lan et al., 2020), and RoBERTa (Liu et al., 2019); as for BERT, we use both the base model and the large model. To select the best model, we fine-tune and evaluate each of them by leveraging the training set. Specifically, we split the training set to use 90% for training and 10% for validation. We use the training set to assess the performance of the model using different learning rates and number of epochs. In line with the recommendation provided by Devlin et al. (2019), we experimented with learning rates in [5e-5, 4e-5, 3e-5, 2e-5] and number of epochs in [1, 2, 3, 4]. We selected the final hyper-parameters to be used in this study based on the best micro-F1 observed on the validation set during the hyper-parameter tuning step. As a result of the hyper-parameter tuning, we decided to fine-tune each model using up to 4 epochs and learning rate = 2e-5. For fine-tuning all the models, we use the AdamW optimizer (Adam weight decay) with epsilon = 1e-8, which is the default value.

### 4.3. Training the issue classifiers

As a first step, we fine-tune the best language model using the full training set. To this aim, we replicate the same procedure adopted for model selection, i.e., we fine-tune the best language model using the issue title and body, which we pad/truncate to consistently represent documents with the same length (128 tokens). Then, we use the fine-tuned RoBERTa model to build the two classifiers. For Classifier 1, we simply rely on the textual information of the GitHub issues, i.e., on the concatenation of the title and body of each issue. For Classifier 2, we build a multilayer perceptron (MLP) classifier that leverages the combination of the textual information of the issues with the information regarding the issue-author association contained in the dataset. This decision was inspired by the issue-author association per class in the GitHub dataset. The distribution (see Table 5) suggests that the information regarding the issue-author association can provide useful insights for issue classification. For instance, questions and bugs appear to be primarily reported by non-collaborating users, while enhancements are mainly reported by repository owners. Thus, we decided to investigate to what extent the textual information alone is sufficient to perform accurate issue classification compared to the setting in which the issue-author association is also leveraged.

To this aim, we extract the text embeddings of each document, i.e., the concatenation of the title and body of the issues, using the last hidden layer before the classification layer of the fine-tuned model, obtaining a 768-dimension embedding. We then concatenate the text embedding with the one-hot-encoding representation of the issue-author association information (six dimensions overall, one for each possible value of the issue-author association attribute). The new vector is fed into a multi-layer perceptron with two hidden layers of size 256 and 128, respectively. In order to train the network, we use stratifed

**Table 5**
Issue-author association per class.

| Issue-author association/Class | Bug | Enhancement | Question |
|---|---|---|---|
| Collaborator | 12% | 13% | 4% |
| Contributor | 17% | 16% | 7% |
| Mannequin | – | – | – |
| Member | 12% | 14% | 3% |
| None | 43% | 21% | 81% |
| Owner | 16% | 34% | 5% |

sampling to split the training set into a training (90%) and a validation set (10%). The network is then trained with the following parameters: batch size = 32, learning rate = $1 \times 10^{-5}$, and the Adam optimizer. We set epochs = 100 and use an early stopping criterion with patience = 5. We use a callback function to save the model periodically, stop the training early if the validation loss stops improving, and select the model achieving the best (lower) validation loss. A callback function is a custom code that can be executed at specific stages of the training process, such as at the end of each epoch or batch. For the training, we use PyTorch *Negative Log-Likelihood Loss*[8] and set the weights of the loss function as inversely proportional to the class frequencies in the training data. For further details about this implementation, our Tool Competition code is available on GitHub (Colavito et al., 2022b).

### 4.4. Evaluating the performance of pre-trained language models (RQ1)

We provide the evaluation of the two classifiers on the test set in terms of precision, recall, and F1. Given the unbalanced distribution of the labels in the GitHub dataset, we report both the micro-F1 and macro-F1.

Precision and recall are two fundamental measures used for binary classification problems. Let us consider a binary classification problem with two classes, $c_1$ and $c_2$. Precision is the probability that, if a random document $d$ is classified as $c_1$, the decision is correct. Recall is the probability that, if a random document $d$ belongs to the class $c_1$, the classifier takes the exact decision. The mathematical expressions to calculate precision and recall are as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where TP is the number of true positives (correctly classified positive instances), FP is the number of false positives (negative instances incorrectly classified as positive), and FN is the number of false negatives (positive instances incorrectly classified as negative).

In addition to precision and recall, the F1 score is another popular measure for binary classification. It is the harmonic mean of precision and recall and provides a balanced measure of their trade-off. The $F1$ score ranges from 0 to 1, with 1 indicating perfect precision and recall. The mathematical expression for the F1 score is as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

In presence of multiple classes, the overall F1 score of a classifier can be calculated using two different methods: F1 micro- and macro-averaging. F1 micro takes into account the total number of true positives, false positives, and false negatives across all classes, while f1 macro computes the F1 score for each class independently and then takes their unweighted average. The mathematical expressions for f1 micro and f1 macro are as follows:

$$F1_{micro} = \frac{2 \times \sum_{i=1}^{C} TP_i}{2 \times \sum_{i=1}^{C} TP_i + \sum_{i=1}^{C} FP_i + \sum_{i=1}^{C} FN_i}$$

$$F1_{macro} = \frac{1}{C} \sum_{i=1}^{C} F1_i$$

where $C$ is the total number of classes and $F1_i$ is the $F1$ score for class $i$.

Indeed, micro-averaging is known to be influenced by the performance on the majority class; conversely, the ability of a classifier to correctly identify items belonging to classes with few training instances is correctly assessed by the macro-average (Sebastiani, 2002). For the Jira dataset, we only train one classifier, corresponding to the Classifier 1 architecture.

### 4.5. Evaluating the impact of improved data quality on training (RQ2)

In line with the goal of our second research question, we assess the performance of classifiers obtained using training datasets of improved quality. To this aim, we define a number of criteria to filter out noisy data from the GitHub dataset. In addition, we take into account the Jira dataset, which meets by construction the same set of data quality criteria operationalized by our filters.

When mining software repositories, it is important to appropriately define quality criteria for the inclusion/exclusion of each repository to be analyzed (Kalliamvakou et al., 2014b). However, the issues included in the GitHub dataset were collected by considering a time frame as the only inclusion criterion (Kallis et al., 2022). As a consequence, the dataset might be noisy and potentially include issues from toy projects. The quality criteria that we adopt in this study are based both on a manual inspection of the GitHub dataset as well as on previous research on this topic (Kalliamvakou et al., 2014b; Biswas et al., 2019; Munaiah et al., 2016).

As reported by Kalliamvakou et al. (2014b), the majority of the projects hosted on GitHub are either personal or inactive. Consistently with this finding, an inspection of the GitHub dataset revealed that the corpus contains several repositories including only one issue, a hint that the related projects might indeed be inactive or personal. In the light of this, we try to improve the quality of our training data by considering only high-quality projects that are likely to actively use the GitHub issue tracking system.

The project star count is usually regarded as a reliable indicator of the quality of a GitHub repository (Biswas et al., 2019; Munaiah et al., 2016). As such, we use the number of project stars as a proxy for data quality and experiment with training sets including issues from repositories with an increasing number of stars. Specifically, we filter training and test sets from the GitHub dataset using a progressive star threshold, i.e., {50, 100, 500, 1000, 1500} stars. Given the class imbalance, we also perform undersampling on the training set based on the support of the minority class (i.e., *question*). For each of the five settings in which we train the model using the filtered datasets, we also train a classifier on a random sampling of the training set. This is useful to assess the effectiveness of the filter in a setting in which the two models have been trained using a control dataset (no filter applied) with comparable size. The only difference is that, in the first case, the issues are the ones that match the quality criteria operationalized with the star-based filter, while in the second case they are randomly sampled. We then test both classifiers on the filtered test set.

As a further consideration emerging from the manual inspection of the misclassified cases from the GitHub dataset (Colavito et al., 2022a), we argue that the lack of consolidated issue labeling guidelines might be a cause for the lower quality of training data. Having consolidated contribution guidelines might help contributors label issues consistently over time. Such guidelines are more likely to be present in consolidated software projects: for this reason, we adopt project age as a second proxy for quality. To operationalize this quality criterion, we (i) remove projects with an age less than one year and (ii) we split the remaining projects into two ranges – i.e., $[1, 4]$ years and $]4, +\infty)$ years – as inspired by a previous work (Vasilescu et al., 2014). As in the case of

[8] https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html

**Table 6**
Pre-trained model selection: the best performance achieved on the validation set for all fine-tuned models.

| | ALBERT (*3 epochs*) | | | BERT-base (*2 epochs*) | | | BERT-large (*2 epochs*) | | | RoBERTa (*4 epochs*) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | Prec | Rec | F1 | Prec | Rec | F1 | Prec | Rec | F1 |
| Bug | .8695 | .8906 | .8799 | .8712 | .9069 | .8887 | .8694 | .9106 | .8895 | .8756 | .8985 | .8869 |
| Enhancement | .8615 | .874 | .8677 | .8709 | .8802 | .8756 | .8722 | .8763 | .8742 | .8743 | .8755 | .8749 |
| Question | .6734 | .532 | .5944 | .7142 | .5083 | .5939 | .7257 | .5104 | .5993 | .6667 | .5612 | .6094 |
| Micro | .8528 | .8528 | .8528 | .8614 | .8614 | .8614 | .8618 | .8618 | .8618 | .8599 | .8599 | .8599 |
| Macro | .8015 | .7655 | .7807 | .8188 | .7651 | .7860 | .8224 | .7658 | .7877 | .8055 | .7784 | .7904 |

the star-based filtering, we compare the performance of the classifier trained on filtered data with the one trained on a control dataset where the age filter is not applied.

Finally, another problem observed in the GitHub dataset is the presence of issues originally tagged with more than one label. Despite affecting a small portion of data (less than 3%), previous work suggests that this phenomenon might represent a source of noise and thus impair classifier performance (Wu et al., 2022). For this reason, we adopt a third filter specifically aimed at removing multi-labeled issues.

We apply all of the above-mentioned filters to the GitHub dataset only. As for the Jira dataset, it already meets the adopted quality criteria by design. Indeed, it is exclusively composed of popular OSS projects, each of which has been active for more than 4 years at the time of this writing. Concerning the reliability of the labels contained in this dataset, we consider it is ensured thanks the coding study performed by the dataset authors (Montgomery et al., 2022).

## 5. Leveraging pre-trained language models for automatic issue classification

In this section, we address our first research question: *To what extent we can leverage pre-trained language models to enhance the state of the art in automatic issue labeling?*

### 5.1. Model training

In the following, we report the results concerning the classifiers trained on the GitHub dataset. As described in Section 4.2, before training our classifiers, we selected the pre-trained language model to be used. Table 6 reports the results of the performance assessment on the validation set for all the models that we experimented with during the pre-trained model selection phase. Given the small differences observed for all models in the overall micro F1, we decided to pick as the best model the one achieving the best F1 on the minority class – i.e., the *question* class. Accordingly, we selected RoBERTa as the most promising language model to be used for further experiments.

In Table 7, we report the performance of the two classifiers trained on the GitHub dataset, comparing them with the approach based on *fastText* (Bojanowski et al., 2017), the state-of-the-art model at the time this study was performed.[9] This choice is in line with the recommendations of the organizers of the challenge to which our classifier trained on GitHub issues was originally submitted for evaluation (Colavito et al., 2022a). Both our classifiers outperform the FastText baseline and they achieve a performance comparable to the one reported by previous work on issue classification based on contextual embeddings (Izadi et al., 2022). In particular, Classifier 1 (RoBERTa fine-tuned) achieves the best micro F1 (.8591), while for Classifier 2 (MLP) – which also includes consideration of the author-issue association – we observe a lower micro F1 (.8295). Nonetheless, in the latter case, the recall for the minority class *question* is substantially improved – up to .7537 – as also reflected by the higher macro average recall (.7774 and .8092 for Classifier 1 and 2, respectively). Albeit the overall performance is substantially unvaried in terms of micro F1, the choice between the RoBERTa-based and the MLP-based model might not be trivial in practice, as RoBERTa optimizes the precision of the minority class while the MLP achieves a better recall.

[9] For the sake of completeness, we replicated the training of the text-based classifier using codeBERT (Feng et al., 2020), obtaining a performance comparable to the one achieved by the RoBERTa-based classifier. The results obtained with codeBERT are included in the replication package.

**Table 7**
Performance of the classifiers trained on the GitHub dataset, evaluated on the test set.

| Class | Classifier 1: RoBERTa *Title + Body* | | | Classifier 2: MLP *Author + Title + Body* | | | FastText Baseline *Title + Body* | | |
|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | Prec | Rec | F1 | Prec | Rec | F1 |
| Bug | .8750 | .8988 | .8867 | .8934 | .8346 | .8630 | .8314 | .8725 | .8515 |
| Enhanc. | .8713 | .8743 | .8728 | .8797 | .8394 | .8591 | .8155 | .8464 | .8307 |
| Question | .6760 | .5591 | .6120 | .4727 | .7537 | .5810 | .6521 | .3502 | .4557 |
| Micro avg | **.8591** | **.8591** | **.8591** | .8295 | .8295 | .8295 | .8162 | .8162 | .8162 |
| Macro avg | **.8074** | .7774 | **.7905** | .7486 | **.8092** | .7677 | .7663 | .6897 | .7126 |

**Table 8**
Confusion matrix on the test set for Classifier 1.

| Gold label | Classifier prediction | | |
|---|---|---|---|
| | Bug | Enhancement | Question |
| Bug | 36,210 (90%) | 3,106 (8%) | 972 (2%) |
| Enhancement | 3,261 (10%) | 29,031 (87%) | 911 (3%) |
| Question | 1,914 (27%) | 1,184 (17%) | 3,929 (56%) |

In Table 8, we report the confusion matrix for the RoBERTa classifier. We observe that the misclassification of *questions* as *bugs* is main cause of error (27% of test documents), immediately followed by the misclassification of *questions* as *enhancements* (17% of cases). As the third most frequent cause of error, we observe the misclassification of *enhancements* as *bugs* (10%). We conjecture that this can be also explained by the unbalanced distribution of labels in the dataset (see Table 2). For this reason, in subsequent experiments we performed an undersampling of the training set. Afterward, to get deeper insights on the difficulties inherent in our issue classification task, we performed an error analysis by manually inspecting the classification output of the RoBERTa fine-tuned model; the results are reported in the next section.

### 5.2. Error analysis

We manually examined a set of 370 misclassified issues, i.e., a statistically significant sample (with 95% confidence level) of the cases in which the classifier yielded a wrong prediction.

We observed that some issues labeled as *question* actually report inconsistent behavior or missing code, thus resembling the structure and content of *bug* reports (e.g., *"Fragrance not showing in Homekit - I cannot see the installed fragrance in HomeKit, however it is available in Homebridge".*). Often, questions contain an error message, which is also common for bugs. These cases are labeled as *question* in line with the information seeking goal of the author. However, a text-based classifier might not be able to disambiguate between bugs and questions in similar cases. A similar situation is observed for *questions* or *bugs* that also include suggestions for fixing the reported problem, which is possibly a cause for the misclassification of *questions* as *enhancement*. Finally, the dataset contains issues collected from different projects,

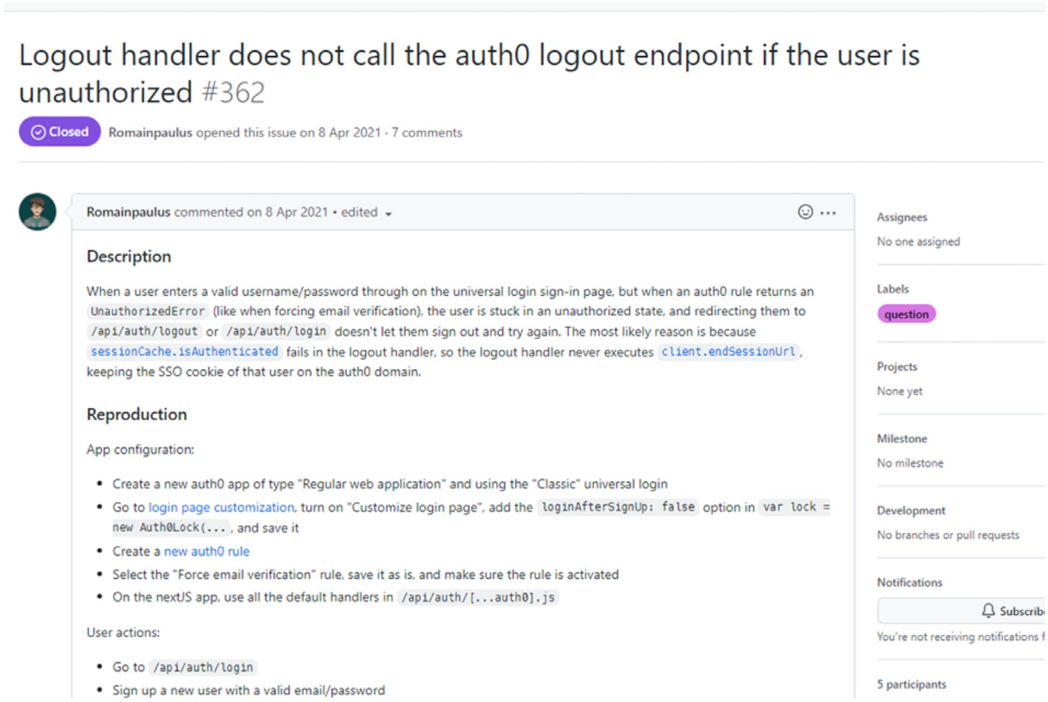


**Fig. 2.** An issue labeled as question from a maintainer of the repository.

thus reflecting possible inconsistencies in the labeling rationale, as well as a few examples of issues written in a language other than English.

In the following, we report and comment on some representative examples of issues that, for the aforementioned reasons, may be difficult to classify correctly and were indeed misclassified by our RoBERTa-based classifier.

Figs. 2 and 3 depict an issue labeled as *question*. The author of the issue did not originally assign any label to it. The issue content and structure are typical of bug reports, as the issue describes a problem and provides some instructions on how to reproduce the error. However, after one day from the issue submission, a maintainer starts handling the problem and adds a comment clarifying that *"This is the expected behavior of the code"* (see Fig. 3) - meaning that the code, used in that way, is actually supposed to throw an error. As a result of this analysis, the maintainer labels the issue as a *question*.

If the reported error was not the expected behavior of the code but rather the result of a bug, then the issue text would have been the same. However, the maintainer would have labeled it as a bug. This example demonstrates how the difference between *questions* and *bugs* might be subtle and not necessarily reflected in the textual content and in its organization.

The example in Fig. 4 reports an issue with two labels. The issue author, who is a project contributor, labels the issue as a *question*. Indeed, the text represents a question on repository usage. The author wants to know how to achieve a specific goal using the software contained in the repository. A project maintainer handles the issue, commenting: *"This is not possible at the moment"* and showing interest in integrating the feature in a future update. Eventually, the maintainer labels the issue as an enhancement, which is the final label included in the dataset. This example shows how team members may use labels differently: the issue is objectively a question, but the maintainer decided to use that question as a reminder or a starting point to enhance the repository by integrating the feature described therein. Accordingly, the maintainer labeled the issue as an enhancement. However, as in the previous example, the distinction between the author's intention to ask a *question* and the maintainer's intention to suggest an *enhancement* is not clearly reflected in the text, thus causing misclassification.

## 6. Impact of data quality for automatic issue classification

In this section, we address our second research question: *To what extent the performance of a model can be improved by improving the quality of the training data?*

### 6.1. Applying the quality filters on the GitHub dataset

In the following, we report on the results of the experiments conducted after applying the filtering criteria described in Section 4.5 on the GitHub dataset. Specifically, we report the performance of classifiers trained on the filtered training datasets, evaluated against the held-out test set. We address the problem of imbalanced training data by performing undersampling on the training set, based on the cardinality of the minority class.

**Applying the star filter to the GitHub dataset**. To filter out low-quality issues, we started by experimenting with a filter based on the number of project stars. In Table 9, we show the distribution of the datasets obtained by applying this filtering criterion with an increasing number of stars as a threshold. It should be noted that, if a project was removed as an effect of the star-based filtering, all of the issues belonging to that project were removed accordingly from both the training and the test set. Moreover, since we are removing issues from the test set, we cannot compare the performance computed for the resulting models with the ones obtained using the full dataset. In order to assess the effectiveness of the filter, we train the RoBERTa-based classifier on a random sampling of the dataset, which preserves the distribution of the corresponding filtered dataset. We then test the performance of the classifier on the filtered test set. In Table 10 and Table 11, we report the performance of the models trained on the filtered dataset and the randomly-sampled dataset, respectively. We also report the confusion matrices for the top performing models, corresponding to the 1500-star filter (see Table 12 and Table 13).

Comparing the results, we observe an improvement of the F1 macro for the classifiers trained on the filtered dataset. Specifically, as can also be seen from the confusion matrices, the performance improvement is mostly due to the increased precision of the *question* class, which is the most difficult to predict (see Section 4.4). The difference in the

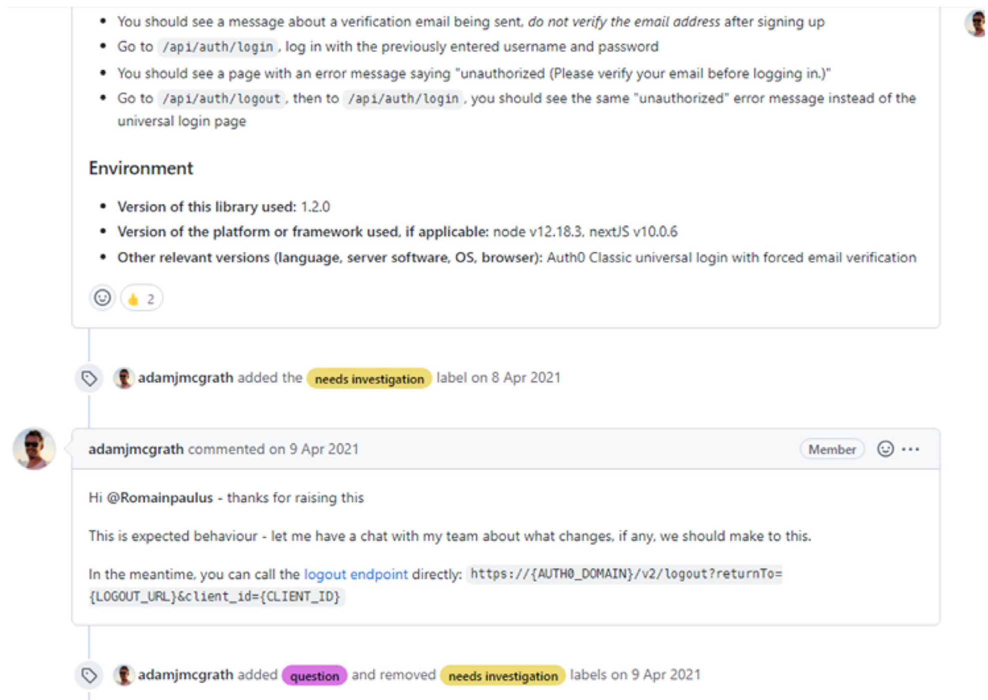


**Fig. 3.** The maintainer's answer to the issue depicted in Fig. 2.

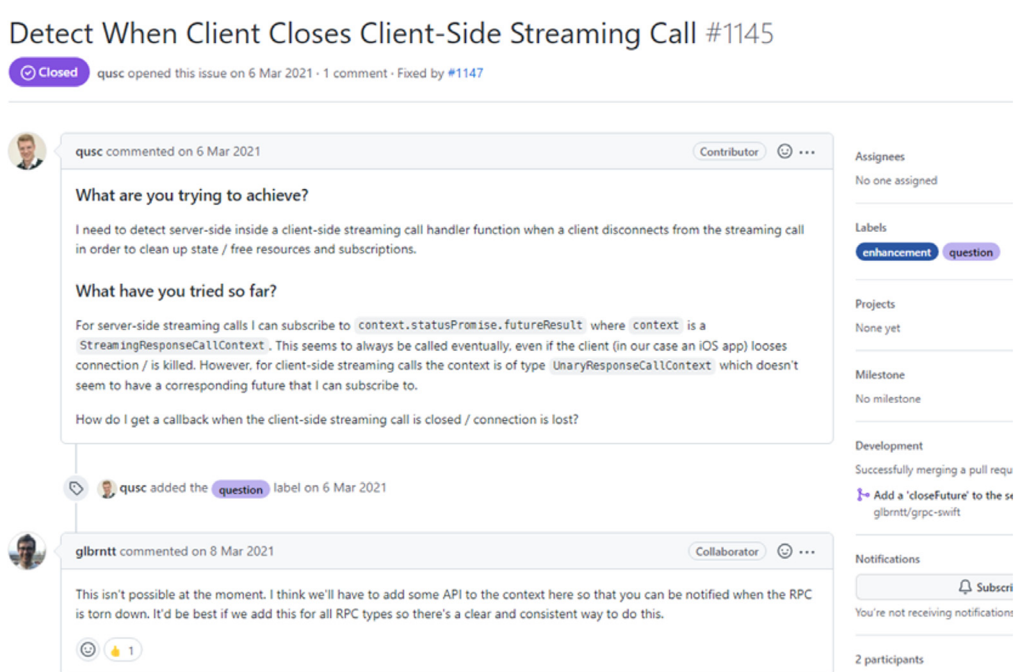


**Fig. 4.** An issue originally labeled as *question* by its author, which is eventually labeled as *enhancement* by a maintainer.

**Table 9**
Distribution of the dataset after filtering on the project star count.

| Star Filter | 50 Star | | 100 Star | | 500 Star | | 1000 Star | | 1500 Star | |
|---|---|---|---|---|---|---|---|---|---|---|
| | TRAIN | TEST | TRAIN | TEST | TRAIN | TEST | TRAIN | TEST | TRAIN | TEST |
| Bug | 202,085 (58%) | 23,124 (58%) | 178,913 (59%) | 20,507 (58%) | 120,721 (60%) | 13,715 (60%) | 97,797 (60%) | 11,079 (60%) | 84,819 (61%) | 9,551 (61%) |
| Enhancement | 101,453 (29%) | 11,537 (29%) | 83,657 (27%) | 9,588 (27%) | 49,108 (24%) | 5,568 (24%) | 36,431 (23%) | 4,128 (22%) | 30,398 (22%) | 3,449 (22%) |
| Question | 46,279 (13%) | 5,499 (13%) | 43,320 (14%) | 5,131 (15%) | 32,439 (16%) | 3,799 (16%) | 27,440 (17%) | 3,177 (17%) | 23,519 (17%) | 2,676 (17%) |
| Total | 349,817 | 40,160 | 305,890 | 35,226 | 202,268 | 23,082 | 161,668 | 18,384 | 138,736 | 15,676 |

**Table 10**
Performance of the models trained on the dataset filtered by the number of project stars and then undersampled with a non-minority strategy.

| Star filter | 50 Star | | | | 100 Star | | | | 500 Star | | | | 1000 Star | | | | 1500 Star | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp |
| Bug | .9160 | .8009 | .8546 | 23,124 | .9135 | .8188 | .8636 | 20,507 | .9176 | .8074 | .8590 | 13,715 | .9200 | .8125 | .8629 | 11,079 | .9234 | .8124 | .8643 | 9,551 |
| Enhancement | .8145 | .8139 | .8142 | 11,537 | .8033 | .8185 | .8108 | 9,588 | .7648 | .8059 | .7848 | 5,568 | .7578 | .8011 | .7789 | 4,128 | .7544 | .7979 | .7755 | 3,449 |
| Question | .5061 | .7743 | .6121 | 5,499 | .5517 | .7605 | .6395 | 5,131 | .5687 | .7705 | .6544 | 3,799 | .5825 | .7765 | .6657 | 3,177 | .5788 | .7840 | .6659 | 2,676 |
| Micro | .8010 | .8010 | .8010 | 40,160 | .8103 | .8103 | .8103 | 35,226 | .8010 | .8010 | .8010 | 23,082 | .8037 | .8037 | .8037 | 18,384 | .8044 | .8044 | .8044 | 15,676 |
| Macro | .7456 | .7964 | .7603 | 40,160 | .7561 | .7993 | .7713 | 35,226 | .7504 | .7946 | .7661 | 23,082 | .7534 | .7967 | .7691 | 18,384 | .7522 | .7981 | .7686 | 15,676 |

**Table 11**
Performance of the models trained on a randomly-sampled dataset having the same distribution as the corresponding dataset filtered by the number of project stars.

| Star filter | 50 Star | | | | 100 Star | | | | 500 Star | | | | 1000 Star | | | | 1500 Star | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp |
| Bug | .9075 | .7944 | .8472 | 23,124 | .9097 | .7890 | .8450 | 20,507 | .9101 | .7891 | .8453 | 13,715 | .9103 | .7906 | .8462 | 11,079 | .9120 | .7734 | .8370 | 9,551 |
| Enhancement | .8509 | .7481 | .7962 | 11,537 | .8511 | .7428 | .7933 | 9588 | .8282 | .7263 | .7739 | 5,568 | .8271 | .7081 | .7630 | 4,128 | .8267 | .6831 | .7481 | 3,449 |
| Question | .4600 | .8176 | .5888 | 5,499 | .4681 | .8277 | .5980 | 5,131 | .4971 | .8252 | .6204 | 3,799 | .5042 | .8297 | .6272 | 3,177 | .4812 | .8498 | .6144 | 2,676 |
| Micro | .7843 | .7843 | .7843 | 40,160 | .7820 | .7820 | .7820 | 35,226 | .7799 | .7799 | .7799 | 23,082 | .7788 | .7788 | .7788 | 18,384 | .7666 | .7666 | .7666 | 15,676 |
| Macro | .7395 | .7867 | .7441 | 40,160 | .7430 | .7865 | .7454 | 35,226 | .7451 | .7802 | .7465 | 23,082 | .7472 | .7761 | .7455 | 18,384 | .7399 | .7688 | .7332 | 15,676 |

**Table 12**
Confusion matrix of the model trained on the star-filtered dataset.

| Gold label | Star filter 1500 | | |
|---|---|---|---|
| | Prediction | | |
| | Bug | Enhancement | Question |
| Bug | 7,647 (80%) | 648 (7%) | 1,256 (13%) |
| Enhancement | 327 (9%) | 2,743 (80%) | 379 (11%) |
| Question | 285 (11%) | 260 (10%) | 2,131 (79%) |

**Table 13**
Confusion matrix of the model trained on the randomly-sampled dataset.

| Gold label | Random sampling | | |
|---|---|---|---|
| | Prediction | | |
| | Bug | Enhancement | Question |
| Bug | 7,321 (77%) | 353 (4%) | 1,877 20%) |
| Enhancement | 421 (12%) | 2,368 (69%) | 660 (19%) |
| Question | 257 (10%) | 141 (5%) | 2,278 (85%) |

outcome of the two classifiers is statistically significant, as proven with a McNemar test (Kirch, 2008; McNemar, 1947) performed to compare their output on the test set ($p < .05$).[10] However, the improvement is small and might not result in a more useful behavior of the classifier in practice.

**Applying the age filter to the GitHub dataset**. We set up the same set of experiments with the filter based on the age of a GitHub project. As done for the star filter, we compare the performance of resulting models with a random sampling of the original training set, preserving the distribution, and test the model on the same filtered test set. We obtain three datasets, with the distributions illustrated in Table 14. We report the results of our experiments in Tables 15 and 16. We also report the confusion matrices for both settings in Tables 17 and 18.

The results are comparable to what observed for the star-based filter. Once again, as can be seen from the confusion matrices, the performance improvement can be attributed to the increased precision of the *question* class. Also in this case, the difference in the classifiers outcome is statistically significant, as proven by the results of a McNemar test ($p < .05$). However, the improvement is even smaller than the one observed for the star-based filter, thus suggesting no tangible enhancement of the classifier in practice.

**Removing multi-label issues from the GitHub dataset**. As a third quality filter, we removed from the dataset the issues for which more than one label were provided. This information was obtained by querying the GitHub API. Other than removing the multi-labeled issues, we also removed those issues for which we could not retrieve this information using the GitHub API (i.e., all those issue that had been removed since the creation of the dataset). As a result of the application of this filtering criterion, 3% of the issues were removed from the original dataset and we obtained a new dataset containing only issues with a single label; the new distribution is shown in Table 19.

Then, we trained the issue classifier on the filtered dataset. The model performances are shown in Table 20 and the related confusion matrix in Table 21. Compared to the performance obtained with the unfiltered dataset (see the RoBERTa classifier performance reported in Table 7), we observe an improvement in the overall F1, both micro (from .8591 to .8697) and macro (from .7905 to .8065). This is also true for each class. In particular, for the most difficult class to predict, i.e. *question*, we observe an improvement of F1 (from .6120 to .6389), precision (from .6760 to .6814), and recall (from .5591 to .6014). While affecting only 3% of the issues, the application of this filtering criterion results in the biggest performance improvement. Still, the overall gain in performance can be considered negligible.

**Applying the combined filters to the GitHub dataset**. The decision to use filters separately was done on purpose to control for confounding factors. In other words, we want to test the impact on data quality for each of the filters that operationalize our data quality criteria. Nevertheless, for completeness, we experimented with the filtered dataset obtained by combining all the filters. Specifically, we combined the filters using the threshold that led to better performance improvement, i.e. Age>4, Stars>1500, and removal of multi-label issues.

We report the results on the dataset obtained using the combined filters in Table 22. As done for the individual filters, we compared the results obtained using a Randomly-sampled dataset with the same distribution observed for the filtered dataset (see Table 23). As already observed for the individual filters, the improvement in the overall F1 is negligible.

### 6.2. Experimenting with the Jira dataset

In the following, we report the results concerning our experiments with the Jira dataset (Montgomery et al., 2022). As already discussed in Section 4.5, this dataset guarantees the adopted quality criteria by design. As such, we do not apply any filtering in this case. The *question* class in the Jira dataset is heavily underrepresented, with *question*-labeled issues representing <1% of the dataset. We trained our RoBERTa model using 90% of the Jira dataset and tested it using the remaining 10%. We report the performance of the classifier in

[10] McNemar's test is a non-parametric test that can be used to compare classification algorithms (Salzberg, 1997).

**Table 14**
Distribution of the dataset after filtering by project age.

| Age filter | Age>4 | | | | 4>Age>1 | | | | Age>1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TRAIN | | TEST | | TRAIN | | TEST | | TRAIN | | TEST | |
| Bug | 126,935 | (58%) | 14,489 | (58%) | 208,936 | (48%) | 23,870 | (48%) | 335,871 | (51%) | 38,359 | (50%) |
| Enhancement | 62,499 | (28%) | 7,199 | (28%) | 204,102 | (47%) | 23,382 | (46%) | 266,601 | (41%) | 30,581 | (41%) |
| Question | 30,945 | (14%) | 3,640 | (14%) | 24,767 | (5%) | 3,143 | (6%) | 55,712 | (8%) | 6,783 | (9%) |
| Total | 220,379 | | 25,328 | | 437,805 | | 50,395 | | 658,184 | | 75,723 | |

**Table 15**
Performance of the models trained on the dataset filtered by project age and than undersampled with a non-minority strategy (tested on the filtered test set).

| Age filter | Age>4 | | | | 4>Age>1 | | | | Age>1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp |
| Bug | .9043 | .8039 | .8512 | 14,489 | .8938 | .7946 | .8413 | 23,870 | .9005 | .7992 | .8468 | 38,359 |
| Enhancement | .7969 | .7908 | .7938 | 7,199 | .8797 | .8428 | .8609 | 23,382 | .8690 | .8358 | .8521 | 30,581 |
| Question | .5352 | .7799 | .6347 | 3,640 | .3477 | .7493 | .4750 | 3,143 | .4313 | .7799 | .5554 | 6,783 |
| Micro | .7967 | .7967 | .7967 | 28,471 | .8141 | .8141 | .8141 | 50,395 | .8123 | .8123 | .8123 | 75,723 |
| Macro | .7454 | .7915 | .7599 | 28,471 | .7071 | .7956 | .7257 | 50,395 | .7336 | .8050 | .7515 | 75,723 |

**Table 16**
Performance of the models trained on a randomly-sampled dataset having the same distribution as the corresponding dataset filtered by project age (tested on the filtered test set).

| Age filter | Age>4 | | | | 4>Age>1 | | | | Age>1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | supp | Prec | Rec | F1 | Supp | Prec | Rec | F1 | Supp |
| Bug | .8983 | .7943 | .8431 | 14,489 | .8919 | .7865 | .8359 | 23,870 | .8975 | .7941 | .8427 | 38,359 |
| Enhancement | .8304 | .7414 | .7834 | 7,199 | .8745 | .8541 | .8642 | 23,382 | .8720 | .8316 | .8513 | 30,581 |
| Question | .4834 | .8088 | .6051 | 3,640 | .3525 | .7299 | .4754 | 3,143 | .4226 | .7861 | .5497 | 6,783 |
| Micro | .7813 | .7813 | .7813 | 28,471 | .8143 | .8143 | .8143 | 50,395 | .8086 | .8086 | .8086 | 75,723 |
| Macro | .7374 | .7815 | .7439 | 28,471 | .7063 | .7902 | .7252 | 50,395 | .7307 | .8039 | .7479 | 75,723 |

**Table 17**
Confusion matrix of the model trained on the age-filtered dataset.

| Gold label | Age Filter>4 | | |
|---|---|---|---|
| | Prediction | | |
| | Bug | Enhancement | Question |
| Bug | 11,648 (80%) | 1,108 (8%) | 1,733 (12%) |
| Enhancement | 774 (11%) | 5,693 (79%) | 732 (10%) |
| Question | 459 (13%) | 343 (9%) | 2,838 (78%) |

**Table 18**
Confusion matrix of the model trained on the randomly-sampled dataset.

| Gold label | Random Sampling | | |
|---|---|---|---|
| | Prediction | | |
| | Bug | Enhancement | Question |
| Bug | 11,508 (79%) | 804 (6%) | 2,177 (15%) |
| Enhancement | 893 (12%) | 5,337 (74%) | 893 (13%) |
| Question | 410 (11%) | 286 (8%) | 2,944 (81%) |

**Table 19**
Distribution of the GitHub dataset after removing the multi-class examples and the unavailable ones.

| | Remove Multi-Class | |
|---|---|---|
| | Train | Test |
| Bug | 281,732 (49%) | 32,106 (49%) |
| Enhancement | 249,429 (43%) | 28,065 (43%) |
| Question | 49,974 (9%) | 5,780 (9%) |
| Total | 581,135 | 65,951 |

**Table 20**
Results of training and testing after removing issues with more than one label from the GitHub dataset.

| | Remove Multi-Class | | | |
|---|---|---|---|---|
| | Prec | Rec | F1 | Support |
| Bug | .8832 | .9010 | .8920 | 32,106 |
| Enhancement | .8882 | .8893 | .8887 | 28,065 |
| Question | .6814 | .6014 | .6389 | 5,780 |
| Micro | .8697 | .8697 | .8697 | 65,951 |
| Macro | .8176 | .7972 | .8065 | 65,951 |

**Table 21**
Confusion matrix of the predictions from the classifier trained without multi-class issues.

| Gold label | Prediction | | |
|---|---|---|---|
| | Bug | Enhancement | Question |
| Bug | 28,926 (90%) | 2,294 (7%) | 886 (3%) |
| Enhancement | 2,369 (8%) | 24,957 (89%) | 739 (3%) |
| Question | 1,457 (25%) | 847 (15%) | 3,476 (60%) |

**Table 22**
Performance of the model trained on the filtered dataset using the most restrictive filters.

| | Prec | Rec | F1 | Support |
|---|---|---|---|---|
| Bug | 0.9032 | 0.7751 | 0.8342 | 2,094 |
| Enhancement | 0.7763 | 0.8290 | 0.8018 | 1,164 |
| Question | 0.5587 | 0.7617 | 0.6446 | 600 |
| Micro | 0.7893 | 0.7893 | 0.7893 | 3,858 |
| Macro | 0.7461 | 0.7886 | 0.7602 | 3,858 |

Table 24. To address the problem of imbalanced training data, we also experimented with undersampling. However, the related attempts did not result in improved performance; therefore, we do not report these results here.

The performance of the model trained on the Jira dataset is comparable to the one obtained with the GitHub dataset, except for the *question* class. The smaller number of questions has a significant impact on the model performance, resulting in a lower F1, precision, and recall for all classes except for bug.

As a further analysis, we performed a follow-up experiment training individual models for each of the four projects containing at least 1,000 issues labeled as *question*. The projects included in this machine-learning experiment are Apache, Jira, MongoDB and Sonatype. The results are reported in Table 25. Overall, the performance obtained

**Table 23**
Performance of the model trained on the random sampled dataset, having the same distribution as the dataset filtered using most restrictive filters.

| | Prec | Rec | F1 | Support |
|---|---|---|---|---|
| Bug | 0.8935 | 0.7937 | 0.8407 | 2,094 |
| Enhancement | 0.8496 | 0.7569 | 0.8005 | 1,164 |
| Question | 0.5140 | 0.8233 | 0.6329 | 600 |
| Microavg | 0.7872 | 0.7872 | 0.7872 | 3,858 |
| Macroavg | 0.7524 | 0.7913 | 0.7580 | 3,858 |

**Table 24**
Performance of the system trained on the Jira dataset, evaluated on 10% of the dataset (with the same distribution of the overall dataset).

| Class | RoBERTa Title + Body | | | |
|---|---|---|---|---|
| | Prec | Rec | F1 | Support |
| Bug | .9378 | .9413 | .9395 | 152,252 |
| Enhancement | .8540 | .8569 | .8554 | 62,831 |
| Question | .6402 | .3766 | .4742 | 879 |
| Micro | .9136 | .9136 | .9136 | 215,962 |
| Macro | .8116 | .7240 | .7564 | 215,962 |

by training on the individual projects is lower than the one achieved using the full dataset; the only exception was observed for the Sonatype project: in this case, we obtained an F1 of .8915 for the *question* class, while for *bug* and *enhancement* the F1 is still lower than the ones obtained for the full dataset.

## 7. Discussion

The use of BERT-based models in software engineering is not new. Specifically, BERT was used to automatically classify the sentiment of technical texts such as Stack Overflow posts or GitHub comments (Biswas et al., 2020; Batra et al., 2021). As far as GitHub issue tagging is concerned, Wang et al. (2021) compared the performance of a pre-trained contextual language representation obtained with BERT with the performance achieved by traditional deep-learning models leveraging GLoVe (Pennington et al., 2014) for the initialization of word embeddings. They found that BERT outperforms other deep learning language models when large training data is used. Conversely, Convolutional Neural Networks perform better than BERT in presence of small-size training data. In their study, Wang et al. (2021) experiment with the BERT model originally developed by Google (Devlin et al., 2019; Liu et al., 2019) to recommend a label for GitHub issues, i.e. their models recommend *k* possible tags for any issue. As such, the performance of their recommender is measured using F1-score@k, which impairs direct comparison with the performance obtained in our study where a classification task is addressed. As a further difference, Wang et al. trained their model separately for each project. Furthermore, in our study we advance the state of the art by also experimenting with AlBERTo and RoBERTa, as well as with the BERT *large* version. RoBERTa was also leveraged by Izadi et al. (2022) for predicting both the type and priority of an issue. Specifically, they model issue type prediction as a classification task and fine-tuned RoBERTa on a dataset of 817,743 GitHub issues from over 60 K repositories labeled as *bug report* (362 K), *enhancement* (342 K), and *support/documentation* (112 K). Similarly to what we do in our study, they rely on the textual information contained in each issue title and body and achieve an overall accuracy of 82%, with F1 equal to .85, .84, and .67 for bug, enhancement and documentation/support, respectively.

However, regardless of algorithmic choices, the quality of ML-based systems can suffer considerably if models are trained on bad-quality data (Halevy et al., 2009; Kästner, 2021). The results of the analysis of the misclassified cases reported in Section 5.2 suggest that a shared understanding of the issue labeling criteria is essential to ensure consistency of the labels in the training data.

Inspired by the findings of this qualitative analysis, we designed and operationalized data quality criteria to filter out issues based on the presence of multiple labels, the project star count, and the project age. We evaluated the impact of applying such data quality filters on the model performance. Unfortunately, this did not result in an improved performance. The main cause of misclassification in all settings remains the confusion between *question* and other labels. We also experimented with project-specific training of issue classifier, without observing a significant improvement in the performance. The only project for which we observe a good performance for all classes is Sonatype. By inspecting the project website, we found that the creation of issues is guided by a wizard that ensures consistency in the labeling.[11]

Our findings suggest that filtering projects included in the training set to improve data quality do not necessarily result in a substantial improvement of model performance. These results are apparently in contrast with recent findings by Wu et al. (2022), who demonstrate how the performance of models can be substantially improved by enhancing the quality of training data. However, we point out that the strategy followed by these authors to improve the quality of their datasets is not directly comparable with ours. Indeed, they fixed bug labeling issues by means of a costly manual annotation process, involving trained professionals and a rigorous annotation protocol. On the other hand, we tried to clean our dataset – thus limiting uncertain labels – through automated filtering procedures based on the operationalization of generic data quality criteria. Moreover, we note that – despite some clear similarities – the classification tasks addressed in the two studies are different. In particular, Wu et al. aim to separate security bug reports from other kinds of bug reports. For this task, they claim that manual effort is still required because current automated approaches do not handle it well. Our negative results suggest that this might be the case also for the issue labeling task: in future work, we set out to confirm this hypothesis by exploring the impact of manual issue label correction on model performance.

## 8. Conclusion

In this paper, we exploit pre-trained language models for automatic issue classification. In particular, we experimented with BERT and its variants and found that RoBERTa-based classifiers achieve state-of-the-art performance in automatic issue labeling. Then, we also investigate the impact of data quality on the classifier performance using filters that operationalize generic data quality criteria. None of the attempts to improve the quality of data had a significant effect on the model performance. We identify the weak definition of the *question* label as the main threat to construct validity affecting the overall data quality.

This study confirms that the use of noisy data has a detrimental impact on model performance. Indeed, while the effects of random errors in data can be tamed by collecting more data, the existence of systematic and conceptual flaws in data cannot be overcome statistically and necessarily entail defects in the resulting ML models.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This research was co-funded by the NRRP Initiative, Mission 4, Component 2, Investment 1.3 - Partnerships extended to universities, research centers, companies and research D.D. MUR n. 341,

[11] https://support.sonatype.com/hc/en-us/requests/new

**Table 25**
Performances of Jira project-specific training and testing.

| Jira project | Apache | | | | Jira | | | | MongoDB | | | | Sonatype | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | F1 | Support | Prec | Rec | F1 | Support | Prec | Rec | F1 | Support | Prec | Rec | F1 | supp |
| Bug | .8919 | .7602 | .8208 | 52,310 | .9312 | .8173 | .8705 | 13,114 | .8553 | .6978 | .7686 | 4,812 | .9560 | .8354 | .8916 | 650 |
| Enhancement | .7685 | .8000 | .7839 | 31,267 | .9322 | .9001 | .9159 | 13,845 | .7964 | .8161 | .8061 | 3,877 | .7087 | .8831 | .7864 | 248 |
| question | .0300 | .9009 | .0581 | 222 | .0951 | .9057 | .1721 | 244 | .1370 | .7348 | .2309 | 181 | .8398 | .9500 | .8915 | 160 |
| micro | .7754 | .7754 | .7754 | 83,799 | .8602 | .8602 | .8602 | 27,203 | .7503 | .7503 | .7503 | 8,870 | .8639 | .8639 | .8639 | 1,058 |
| macro | .5635 | .8204 | .5543 | 83,799 | .6528 | .8744 | .6528 | 27,203 | .5962 | .7496 | .6019 | 8,870 | .8348 | .8895 | .8565 | 1,058 |

15/03/2022 – Next Generation EU ("FAIR - Future Artificial Intelligence Research", code PE00000013, CUP: H97G22000210007), ("SERICS - SEcurity and Rights In the CyberSpace", code PE0000014, CUP: H93C22000620001'), the Complem. National Plan PNC-I.1 - Research initiatives for innovative technologies and pathways in the health and welfare sector, D.D. 931 of 06/06/2022 ("DARE - DigitAl lifelong pRevEntion initiative", code PNC0000002, CUP: B53C22006420001), and by the PRIN 2022 call of Italian Ministry for Universities and Research (MUR), D.D. n. 104 of 02/02/2022 ("QualAI: Continuous Quality Improvement of AI-based Systems", CUP: H53D23003510006).

## References

AlMarzouq, M., AlZaidan, A., AlDallal, J., 2020. Mining GitHub for research and education: challenges and opportunities. Int. J. Web Inf. Syst. 16 (4), 451–473. http://dx.doi.org/10.1108/IJWIS-03-2020-0016, URL https://www.emerald.com/insight/content/doi/10.1108/IJWIS-03-2020-0016/full/html.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.-G., 2008. Is it a bug or an enhancement? A text-based approach to classify change requests. In: Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. CASCON '08, ACM, New York, NY, USA, http://dx.doi.org/10.1145/1463788.1463819.

Batra, H., Punn, N.S., Sonbhadra, S.K., Agarwal, S., 2021. BERT-based sentiment analysis: A software engineering perspective. Database Expert Syst. Appl. 138–148. http://dx.doi.org/10.1007/978-3-030-86472-9_13.

Bissyandé, T.F., Lo, D., Jiang, L., Réveillère, L., Klein, J., Traon, Y.L., 2013. Got issues? who cares about it? a large scale investigation of issue trackers from github. In: 2013 IEEE 24th International Symposium on Software Reliability Engineering. ISSRE, pp. 188–197. http://dx.doi.org/10.1109/ISSRE.2013.6698918.

Biswas, S., Islam, M.J., Huang, Y., Rajan, H., 2019. Boa meets python: A boa dataset of data science software in python language. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, pp. 577–581. http://dx.doi.org/10.1109/MSR.2019.00086.

Biswas, E., Karabulut, M.E., Pollock, L., Vijay-Shanker, K., 2020. Achieving reliable sentiment analysis in the software engineering domain using BERT. In: 2020 IEEE Int'L. Conf. on Software Maintenance and Evolution. ICSME, pp. 162–173. http://dx.doi.org/10.1109/ICSME46990.2020.00025.

Bojanowski, P., Grave, E., Joulin, A., Mikolov, T., 2017. Enriching word vectors with subword information. In: Lee, L., Johnson, M., Toutanova, K. (Eds.), Trans. Assoc. Comput. Linguist. 5, 135–146. http://dx.doi.org/10.1162/tacl_a_00051, URL https://aclanthology.org/Q17-1010.

Cánovas Izquierdo, J.L., Cosentino, V., Rolandi, B., Bergel, A., Cabot, J., 2015. Gila: Github label analyzer. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering. SANER, pp. 479–483. http://dx.doi.org/10.1109/SANER.2015.7081860.

Colavito, G., Lanubile, F., Novielli, N., 2022a. Issue report classification using pre-trained language models. In: 2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering. NLBSE, IEEE Computer Society, Los Alamitos, CA, USA, pp. 29–32. http://dx.doi.org/10.1145/3528588.3528659, URL https://doi.ieeecomputersociety.org/10.1145/3528588.3528659.

Colavito, G., Lanubile, F., Novielli, N., 2022b. Issue-report-classification-using-RoBERTa. URL https://github.com/collab-uniba/Issue-Report-Classification-Using-RoBERTa.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL, Minneapolis, Minnesota, pp. 4171–4186. http://dx.doi.org/10.18653/v1/N19-1423, URL https://aclanthology.org/N19-1423.

Fan, Q., Yu, Y., Yin, G., Wang, T., Wang, H., 2017. Where is the road for issue reports classification based on text mining? In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, pp. 121–130. http://dx.doi.org/10.1109/ESEM.2017.19.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., 2020. CodeBERT: A pre-trained model for programming and natural languages. In: Cohn, T., He, Y., Liu, Y. (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020. Association for Computational Linguistics, pp. 1536–1547. http://dx.doi.org/10.18653/v1/2020.findings-emnlp.139, Online. URL https://aclanthology.org/2020.findings-emnlp.139.

Gousios, G., Spinellis, D., 2017. Mining software engineering data from GitHub. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion. ICSE-C, IEEE, Buenos Aires, Argentina, pp. 501–502. http://dx.doi.org/10.1109/ICSE-C.2017.164, URL http://ieeexplore.ieee.org/document/7965403/.

Halevy, A., Norvig, P., Pereira, F., 2009. The unreasonable effectiveness of data. IEEE Intell. Syst. 24 (2), 8–12. http://dx.doi.org/10.1109/MIS.2009.36, URL http://ieeexplore.ieee.org/document/4804817/.

Herzig, K., Just, S., Zeller, A., 2013. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: 2013 35th International Conference on Software Engineering. ICSE, pp. 392–401. http://dx.doi.org/10.1109/ICSE.2013.6606585.

Hynes, N., Sculley, D., Terry, M., 2017. The data linter: Lightweight automated sanity checking for ML data sets. URL http://learningsys.org/nips17/assets/papers/paper_19.pdf.

Izadi, M., Akbari, K., Heydarnoori, A., 2022. Predicting the objective and priority of issue reports in software repositories. Empir. Softw. Eng. 27, http://dx.doi.org/10.1007/s10664-021-10085-3, URL https://link.springer.com/article/10.1007/s10664-021-10085-3.

Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Sharma Mittal, R., Munigala, V., 2020. Overview and importance of data quality for machine learning tasks. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, Virtual Event CA USA, pp. 3561–3562. http://dx.doi.org/10.1145/3394486.3406477, URL https://dl.acm.org/doi/10.1145/3394486.3406477.

Joulin, A., Grave, E., Bojanowski, P., Mikolov, T., 2017. Bag of tricks for efficient text classification. In: Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics. ACL, Valencia, Spain, pp. 427–431, URL https://aclanthology.org/E17-2068.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D., 2014a. The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014. ACM Press, Hyderabad, India, pp. 92–101. http://dx.doi.org/10.1145/2597073.2597074, URL http://dl.acm.org/citation.cfm?doid=2597073.2597074.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D., 2014b. The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories. In: MSR 2014, Association for Computing Machinery, New York, NY, USA, pp. 92–101. http://dx.doi.org/10.1145/2597073.2597074.

Kallis, R., Chaparro, O., Di Sorbo, A., Panichella, S., 2022. NLBSE'22 tool competition. In: Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering. NLBSE'22.

Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S., 2019. Ticket tagger: Machine learning driven issue classification. In: 2019 IEEE Int'L. Conf on Software Maintenance and Evolution. ICSME, pp. 406–409. http://dx.doi.org/10.1109/ICSME.2019.00070.

Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S., 2021. Predicting issue types on GitHub. Sci. Comput. Programm. 205, 102598. http://dx.doi.org/10.1016/j.scico.2020.102598, URL https://www.sciencedirect.com/science/article/pii/S0167642320302069.

Kästner, C., 2021. Data quality for building production ML systems. Medium. URL https://ckaestne.medium.com/data-quality-for-building-production-ml-systems-2e0cc7e6113f.

Kirch, W. (Ed.), 2008. McNemar's test. In: Encyclopedia of Public Health. Springer Netherlands, Dordrecht, p. 886. http://dx.doi.org/10.1007/978-1-4020-5614-7_2075.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R., 2020. ALBERT: A lite BERT for self-supervised learning of language representations. arXiv:1909.11942.

Levy, O., Goldberg, Y., 2014. Neural word embedding as implicit matrix factorization. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems. Vol. 27. Curran Associates, Inc, URL https://proceedings.neurips.cc/paper/2014/file/feab05aa91085b7a8012516bc3533958-Paper.pdf.

Liao, Z., He, D., Chen, Z., Fan, X., Zhang, Y., Liu, S., 2018. Exploring the characteristics of issue-related behaviors in GitHub using visualization techniques. IEEE Access 6, 24003–24015. http://dx.doi.org/10.1109/ACCESS.2018.2810295, Acceptance from VoR OA article however no CC licence on article (see p1 of VoR). Applied 'no exception' as article doesn't meet our definition for Gold exception. ET 14/1/20 ET.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692.

McNemar, Q., 1947. Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika 12 (2), 153–157. http://dx.doi.org/10.1007/BF02295996.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems. Vol. 2. NIPS '13, Curran Associates Inc, Red Hook, NY, USA, pp. 3111–3119.

Montgomery, L., Lüders, C., Maalej, W., 2022. An alternative issue tracking dataset of public jira repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories. MSR '22, Association for Computing Machinery, New York, NY, USA, pp. 73–77. http://dx.doi.org/10.1145/3524842.3528486.

Munaiah, N., Kroh, S., Cabrey, C., Nagappan, M., 2016. Curating GitHub for engineered software projects. http://dx.doi.org/10.7287/PEERJ.PREPRINTS.2617.

Pandey, N., Sanyal, D., Hudait, A., Sen, A., 2017. Automated classification of software issue reports using machine learning techniques: an empirical study. Innov. Syst. Softw. Eng. 13, http://dx.doi.org/10.1007/s11334-017-0294-1.

Panichella, S., Bavota, G., Penta, M.D., Canfora, G., Antoniol, G., 2014. How developers' collaborations identified from different sources tell us about code changes. In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 251–260. http://dx.doi.org/10.1109/ICSME.2014.47.

Pennington, J., Socher, R., Manning, C.D., 2014. GloVe: Global vectors for word representation. In: Empirical Methods in Natural Language Processing. EMNLP, pp. 1532–1543.

Press, G., 2016. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says. Forbes. URL https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/.

Rekatsinas, T., Chu, X., Ilyas, I.F., Ré, C., 2017. HoloClean: Holistic data repairs with probabilistic inference. Proc. VLDB Endow. 10 (11), 1190–1201. http://dx.doi.org/10.14778/3137628.3137631.

Salzberg, S.L., 1997. On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min. Knowl. Discov. 1 (3), 317–328. http://dx.doi.org/10.1023/A:1009752403260.

Sambasivan, N., Kapania, S., Highfill, H., Akrong, D., Paritosh, P., Aroyo, L.M., 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. ACM, Yokohama Japan, pp. 1–15. http://dx.doi.org/10.1145/3411764.3445518, URL https://dl.acm.org/doi/10.1145/3411764.3445518.

Sanh, V., Debut, L., Chaumond, J., Wolf, T., 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. CoRR. arXiv:1910.01108.

Sebastiani, F., 2002. Machine learning in automated text categorization. ACM Comput. Surv. 34 (1), 1–47. http://dx.doi.org/10.1145/505282.505283.

Vasilescu, B., van Schuylenburg, S., Wulms, J., Serebrenik, A., van den Brand, M.G., 2014. Continuous integration in a social-coding world: Empirical evidence from GitHub. In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 401–405. http://dx.doi.org/10.1109/ICSME.2014.62.

Wang, J., Zhang, X., Chen, L., 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? Knowl.-Based Syst. 232, 107476. http://dx.doi.org/10.1016/j.knosys.2021.107476, URL https://www.sciencedirect.com/science/article/pii/S0950705121007383.

Wu, X., Zheng, W., Xia, X., Lo, D., 2022. Data quality matters: A case study on data label correctness for security bug report prediction. IEEE Trans. Softw. Eng. 48 (07), 2541–2556. http://dx.doi.org/10.1109/TSE.2021.3063727.

Zhou, Y., Tong, Y., Gu, R., Gall, H.C., 2014. Combining text mining and data mining for bug report classification. In: 2014 IEEE International Conference on Software Maintenance and Evolution. pp. 311–320.