Do CONTRIBUTING Files Provide Information about OSS Newcomers' Onboarding Barriers?

Felipe Fronchetti
Virginia Commonwealth University
Richmond, USA
fronchettl@vcu.edu

David C. Shepherd
Lousiana State University
Baton Rouge, USA
dshepherd@lsu.edu

Igor Wiese
Universidade Tecnologica Federal do Parana
Campo Mourao, Brazil
igor@utfpr.edu.br

Christoph Treude
The University of Melbourne
Melbourne, Australia
christoph.treude@unimelb.edu.au

Marco Aurélio Gerosa
Northern Arizona University
Flagstaff, USA
marco.gerosa@nau.edu

Igor Steinmacher
Northern Arizona University
Flagstaff, USA
igor.steinmacher@nau.edu

ABSTRACT

Effectively onboarding newcomers is essential for the success of open source projects. These projects often provide onboarding guidelines in their 'CONTRIBUTING' files (e.g., CONTRIBUTING.md on GitHub). These files explain, for example, how to find open tasks, implement solutions, and submit code for review. However, these files often do not follow a standard structure, can be too large, and miss barriers commonly found by newcomers. In this paper, we propose an automated approach to parse these CONTRIBUTING files and assess how they address onboarding barriers. We manually classified a sample of files according to a model of onboarding barriers from the literature, trained a machine learning classifier that automatically predicts the categories of each paragraph (precision: 0.655, recall: 0.662), and surveyed developers to investigate their perspective of the predictions' adequacy (75% of the predictions were considered adequate). We found that CONTRIBUTING files typically do not cover the barriers newcomers face (52% of the analyzed projects missed at least 3 out of the 6 barriers faced by newcomers; 84% missed at least 2). Our analysis also revealed that information about choosing a task and talking with the community, two of the most recurrent barriers newcomers face, are neglected in more than 75% of the projects. We made available our classifier as an online service that analyzes the content of a given CONTRIBUTING file. Our approach may help community builders identify missing information in the project ecosystem they maintain and newcomers can understand what to expect in CONTRIBUTING files.

CCS CONCEPTS

• **Software and its engineering → Open source model**; • **Human-centered computing → Collaborative and social computing systems and tools**.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3616288

KEYWORDS

novices, onboarding, FLOSS, open source, software engineering

ACM Reference Format:
Felipe Fronchetti, David C. Shepherd, Igor Wiese, Christoph Treude, Marco Aurélio Gerosa, and Igor Steinmacher. 2023. Do CONTRIBUTING Files Provide Information about OSS Newcomers' Onboarding Barriers?. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616288

1 INTRODUCTION

Newcomers to Open Source Software (OSS) projects encounter several barriers to making their first contribution [73]. For example, an overly complex codebase or a workspace that is challenging to build saps newcomers' motivation to contribute [76]. Research shows that these barriers discourage newcomers, who often give up before completing a single contribution [48].

To onboard, newcomers usually consult the project's documentation or contact the project team [22, 34, 41]. Yet project members are busy making their own contributions, can only help a limited number of newcomers at a time, and may not be able to manage synchronous communication due to time zone differences [23, 35]. For onboarding newcomers, appropriate documentation is more efficient and scalable [71, 74].

Unfortunately, most OSS projects' existing documentation is either low quality or non-existent [1, 41, 76]. Some studies point to problems such as documentation files that are incorrect, incomplete, and outdated [1, 48]. Other studies identified further documentation barriers for newcomers, including unclear, and scattered documentation, with information overload from unimportant information sharply contrasting with missing necessary information [73]. These and other documentation deficiencies impact all contributors but have more impact on newcomers since they need to orient themselves in a new environment [34].

Previous work [74, 78] showed that newcomers found themselves more oriented and understood the process better when the right information was provided in an organized way. However, little work has specifically focused on enhancing newcomers' documentation and identifying what information is missing from the contribution guidelines [74]. With the goal of improving this situation, we automatically analyze and classify the distinct information types contained in existing CONTRIBUTING documentation aimed at newcomers. In this study, we answer the following research questions:

RQ1. How accurately can we automatically classify the content of CONTRIBUTING files in GitHub projects?

RQ2. To what extent do OSS projects' CONTRIBUTING files cover content related to newcomers' contribution barriers?

To answer these questions, we created an oracle by manually annotating the CONTRIBUTING files from 500 software projects according to the newcomers' barriers model proposed by Steinmacher et al. [73]. Then, we trained a machine learning classification model that identifies Steinmacher et al.'s six categories of barriers (precision: 0.655, recall: 0.662). The results were further validated through a survey with experienced developers, where the developers agreed that ≈75% of the categories predicted were adequate. Finally, we used our model across 2,274 publicly available projects to better understand to what extent contribution files cover the information about the barriers faced by newcomers. We found that CONTRIBUTING files are woefully inadequate for supporting newcomers. Most contain four or fewer of the six expected categories of onboarding barriers, and thousands of projects (≈65%) do not have a contribution file.

To facilitate researchers and practitioners to build upon our work by accessing its categorization model, we developed an online service that automatically analyzes the content of a given CONTRIBUTING file (http://contributing.streamlit.app/). Our tool offers potential benefits for project maintainers and community managers by allowing them to evaluate and enhance their CONTRIBUTING files based on feedback from our classifier. This becomes especially significant in ecosystems comprising multiple projects, where community managers oversee various distinct projects. Achieving consistency across projects is crucial to reduce cognitive load and promote smooth transitions between projects within a software ecosystem. Adhering to the maintenance of CONTRIBUTING files is a recognized best practice to assist newcomers in onboarding open source software projects [71].

2 RELATED WORK

In this section, we highlight related studies about documentation in OSS, the automatic categorization of software engineering artifacts, and barriers newcomers face in OSS projects.

Documentation issues in OSS repositories. Documentation plays a crucial role in software projects, and deficiencies in documentation files can hinder their utility for developers [39, 49, 68]. Lethbridge et al. [42] identify that documentation files contain excessive information, are hard to maintain, and make it challenging to locate helpful information. Such considerations are also present in the context of OSS communities [1, 18, 77]. According to Dias et al. [15], from the perspective of OSS developers and maintainers, OSS contributors need to ensure the quality and consistency of documentation files. Our study helps to process existing documentation files and classify content relevant for newcomers, helping maintainers identify missing information in their contributing guidelines and newcomers locate relevant information.

Automatic classification of software engineering artifacts. Several studies have automated the categorization of artifacts in software engineering [27, 43, 62]. For example, Prana et al. [61] broke down the headers of README files in OSS repositories into eight categories of information. Based on the manual annotation of 4,226 README file sections, the authors implemented a classification model that automatically identifies the context of a section in a README file. They argue that labeling sections makes the knowledge discovery process easier for visitors. We followed a similar method and share their idea that categories may help navigate the information space, especially for outsiders. For a different type of documentation file, Robillard and Chhetri [65] categorized text fragments from API documentation based on their relevance for programmers. The authors proposed a coding guide and an automated technique to classify text fragments into three levels of relevance for programmers. The variety of studies exploring the automatic categorization of information in software-related artifacts is wide (e.g., [45, 60, 87]), but our study is among the first to automatically categorize information in contribution guidelines to address newcomers' contribution barriers.

Newcomers in OSS communities: Supporting and engaging newcomers increases the likelihood of newcomers completing their contributions, which is essential for the long-term viability of OSS projects [25, 57, 78, 81]. Without adequate retention, project development progress slows, jeopardizing the existence of such communities [80]. To study this issue, researchers identified different obstacles newcomers face in the onboarding process, focusing on the period between their initial contact with the OSS community and their first contribution [1, 76–78].

Steinmacher et al. [73] propose a taxonomy of 58 barriers newcomers face when joining OSS projects. Documentation issues appear as a central source of problems for newcomers, including already mentioned challenges such as information overload, scattered and outdated documentation, and lack of necessary project information. Some researchers investigated how existing approaches support newcomers onboarding. More specifically, they focus on understanding labels to guide newcomers to choose their tasks [80, 81], exploring the role of Q&A websites in helping the onboarding [84], and code visualization [57]. Other studies discuss how documentation can help and cause problems and how it may impact the newcomers' experience [47, 55]. We believe our results inform OSS projects toward better supporting newcomers with the information they need when joining a project.

It is clear from the literature that documentation is critical for onboarding newcomers in OSS. Despite the efforts in categorizing artifacts related to project documentation, no body of knowledge exists about the appropriateness of contribution guidelines for onboarding newcomers. In this paper, we address this by analyzing the content of CONTRIBUTING files from OSS repositories in terms of barriers newcomers face.

3 RESEARCH METHOD OVERVIEW

To answer our research questions, we manually analyzed CONTRIBUTING files from 500 projects and built a classifier to label information known to be relevant for newcomers. According to GitHub [31] guidelines, CONTRIBUTING files are where one should

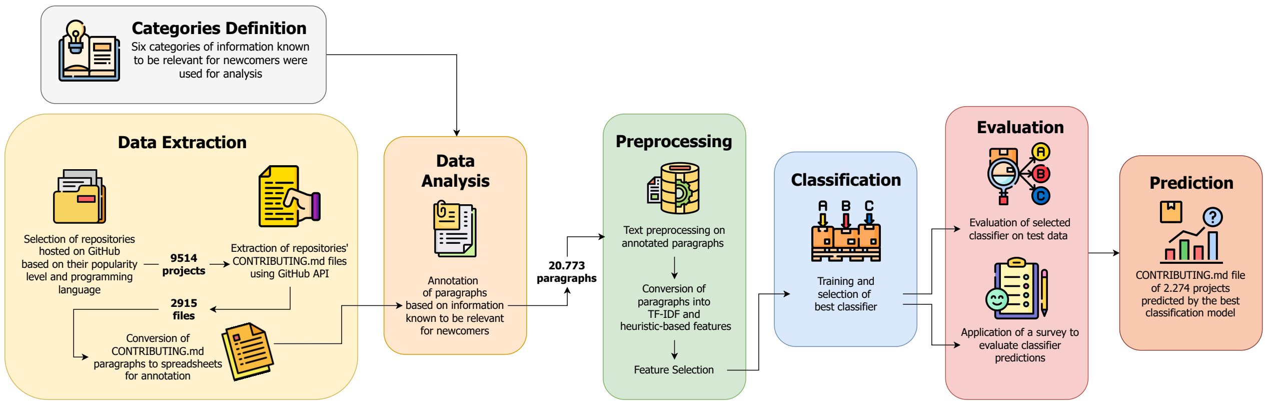
Categories Definition
Six categories of information known to be relevant for newcomers were used for analysis
Data Extraction
Selection of repositories hosted on GitHub based on their popularity level and programming language
9514 projects
Extraction of repositories' CONTRIBUTING.md files using GitHub API
2915 files
Conversion of CONTRIBUTING.md paragraphs to spreadsheets for annotation
Data Analysis
Annotation of paragraphs based on information known to be relevant for newcomers
20.773 paragraphs
Preprocessing
Text preprocessing on annotated paragraphs
Conversion of paragraphs into TF-IDF and heuristic-based features
Feature Selection
Classification
Training and selection of best classifier
Evaluation
Evaluation of selected classifier on test data
Application of a survey to evaluate classifier predictions
Prediction
CONTRIBUTING.md file of 2.274 projects predicted by the best classification model

Figure 1: Research method followed in this research, from building the corpus to assessing the classifier

"create guidelines to communicate how people should contribute to your project." Additionally, the Open Source guide [53] reinforces that *"a CONTRIBUTING file tells your audience how to participate in your project... [and] is an opportunity to communicate your expectations for contributions."* Therefore, newcomers expect to find relevant information to avoid common onboarding barriers [73].

The research method was conducted in six steps, as presented in Figure 1: (1) We extracted the CONTRIBUTING files from 2,913 OSS projects hosted on GitHub. (2) The paragraphs of a random sample of files were manually annotated. (3) The annotated paragraphs were pre-processed and then converted into statistical features (i.e., term frequency-inverse document frequency) and heuristic-based features (in which a rule-based approach was performed). (4) We trained five different classification models with these features and compared their performances. (5) We surveyed developers to assess the quality of the classifications. (6) Finally, we used our model to classify the content of 2,274 CONTRIBUTING files and to understand to what extent they cover the onboarding barriers.

All scripts, models, data, and results are available in our replication package [20]. In the following, we present more details of the method and results of each step.

4 BUILDING THE CORPUS

To train our models we collected and manually categorized the content of CONTRIBUTING files from a set of OSS projects.

4.1 Categories Definition

We manually labeled each paragraph of the 500 CONTRIBUTING files according to the way Steinmacher et al. organized the categories of barriers on the FLOSScoach portal [78]. The portal was created based on a barriers model built based on a systematic literature review, interviews with multiple stakeholders, and surveys within OSS communities, providing a comprehensive aggregation of the barriers newcomers face when joining OSS projects. In addition to the comprehensiveness of the model, we chose to follow these categories since the work by Steinmacher et al. [74, 78] showed that organizing the information in these categories lowered the barriers related to orientation and contribution process. These are the categories we used:

CF - Contribution flow: Derived from the "Newcomer Orientation" barrier category mapped to the contribution flow shown under "How to Start" in FLOSScoach, this category defines the steps that a newcomer needs to follow to contribute to the project. This category appears as, for example, an ordered list of steps to follow or as a set of paragraphs describing the current project workflow.
CT - Choose a task: Also derived from the "Newcomer Orientation" barrier, it is mapped from the "Choose a Task" menu item in FLOSScoach. This category explains how newcomers can find a task (or issue) to contribute to the project. It may also contain descriptions of different types of tasks appropriate for newcomers.
TC - Talk to the community: Related to the "Communication Issues" barriers, this category refers to information about how a newcomer can get in touch with community members and how to find a mentor. This category includes, for example, links to communication channels, communication etiquette, community guidelines, and tutorials on how to start a conversation.
BW - Build local workspace: Mapped from the "Local Environment Setup Hurdles," this category determines the steps a newcomer needs to follow to build the local workspace. It may include instructions such as bash commands and changes in computer settings.
DC - Deal with the code: Derived from "Code/Architecture Hurdles," it describes how newcomers should deal with the source code. This category may contain code conventions, descriptions of the source code, and guidelines on how to write code for the project.
SC - Submit the changes: Directly mapped from "Change Request Hurdles," this category represents information about how newcomers should submit a contribution to the project.

4.2 Data Collection

4.2.1 Project Selection. We selected the most popular OSS repositories hosted on GitHub when we started the data collection (Aug 2020), written in at least one of the top 10 programming languages used in the platform. We selected projects based on their popularity and programming language to avoid repositories that were toy projects or unrelated to software development. The selection of projects by popularity was based on the study of Borges et al. [8], which discusses stars as a unit to measure the popularity of OSS projects on GitHub, and shows that, in their population, "three out of four developers consider the number of stars before using or

Table 1: Number of projects removed per language and their respective reasons for exclusion. The "n" value represents the total number of projects collected for a language. CONTRIBUTING files may have been excluded for more than one reason.

Removed because CONTRIBUTING...	JavaScript (n=824)	Python (n=929)	Java (n=942)	PHP (n=941)	C# (n=990)	C++ (n=942)	TypeScript (n=990)	Shell (n=990)	C (n=947)	Ruby (n=1019)
was missing	381 (46%)	527 (57%)	692 (73%)	593 (63%)	651 (66%)	604 (64%)	474 (48%)	785 (79%)	702 (74%)	646 (63%)
size <0.5kB	41 (5%)	37 (4%)	22 (2%)	49 (5%)	54 (5%)	35 (4%)	42 (4%)	23 (2%)	28 (3%)	33 (3%)
was not in English	3 (< 1%)	12 (1%)	4 (< 1%)	4 (< 1%)	3 (< 1%)	2 (< 1%)	3 (< 1%)	4 (< 1%)	2 (< 1%)	1 (< 1%)
was not in Markdown	2 (< 1%)	91 (10%)	5 (1%)	4 (< 1%)	1 (< 1%)	14 (1%)	4 (< 1%)	6 (1%)	26 (3%)	11 (1%)
Projects removed	**425 (52%)**	**661 (71%)**	**721 (76%)**	**647 (68%)**	**709 (71%)**	**651 (69%)**	**521 (52%)**	**818 (82%)**	**755 (79%)**	**691 (67%)**

contributing to a GitHub project". In addition to it, this is a fairly common way to sample projects on GitHub [33, 59, 61, 63, 75].

To identify the top 10 most-used languages, we used the ranking provided by GitHub Octoverse [29], which showed, at that time: JavaScript, Python, Java, PHP, C#, C++, TypeScript, Shell, C, and Ruby. We aimed to get the first 1,000 projects per language ranked by stars. However, the GitHub API provides only a few pages containing the top projects and we could not collect 1,000 projects for some languages. We collected a total of 9,514 repositories.

To ensure that all the selected repositories had a valid CONTRIBUTING file, we defined a set of filters to remove projects in our dataset. We removed from our sample the projects that had a CONTRIBUTING file:

i. missing—we focused only on projects that followed the guidelines from GitHub to keep in this specific file information about how to contribute;
ii. smaller than 0.5kB—to filter out those files that redirect to guidelines not hosted on GitHub, or empty files;
iii. written in a language other than English;
iv. not in Markdown format—which was the most prevalent format in our sample (3,295 out of 3,459 projects that had a CONTRIBUTING file were in Markdown−95.2%).

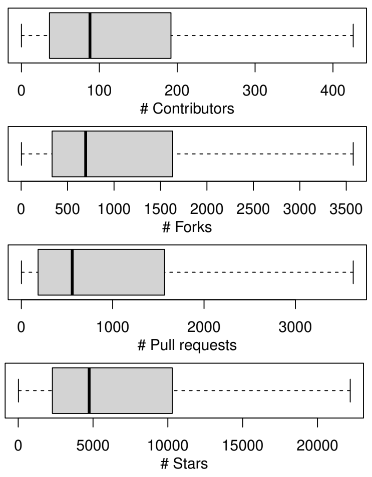

Figure 2: Distribution of contributors, forks, pull requests, and stars per project considered as valid.

Table 1 shows the number of projects per programming language removed from our dataset. The final set of repositories comprised 2,915 projects. After applying the filters, we kept a diverse number of projects in terms of the number of contributors, forks, pull requests, and stars (see Figure 2). The programming languages with the highest number of repositories included in the analysis were TypeScript, JavaScript, and Ruby.

4.2.2 Documentation Formatting. To prepare the projects for the qualitative analysis, we converted the contribution files into spreadsheets. Each spreadsheet maps to all paragraphs of one contributing file in our sample. The first column of each row of the spreadsheet contained in plaintext format one paragraph of the documentation file for the respective project. We followed the definition of a paragraph provided by the specification of GitHub Flavored Markdown [30], which specifies it as "*a sequence of non-blank lines that cannot be interpreted as other kinds of blocks forms.*" To facilitate the work of the annotators, we created headers for six columns, each representing one of the six categories we aimed to identify during the qualitative analysis.

4.3 Data Annotation

After transforming the CONTRIBUTING files into spreadsheets, we conducted the annotation process. We annotated a total of 500 spreadsheets (from 500 projects). In the first step, two annotators labeled 30 spreadsheets of a random subset of projects and discussed how the categories should be assigned to each paragraph. To measure the agreement between the annotators, they independently labeled the spreadsheets divided into three consecutive stages—consisting of 10 spreadsheets per stage. The annotation consisted of analyzing and labeling each paragraph according to the categories presented in Section 4.1. At the end of each stage, the reviewers compared their labels and discussed their differences to align their understanding of each category. We use Cohen's kappa coefficient to measure the agreement between the annotators [13]. After the first stage, the annotators reached an agreement of 73% and discussed the potential meaning of categories. For the other two stages, the agreement was 85% and 79%, respectively. The overall agreement between the annotators was 79%, which was considered sufficient given the multi-class nature of the data.

4.3.1 Documentation Annotation. After reaching a substantial agreement, the reviewers proceeded to analyze the remaining files, which were split between them. A total of 500 spreadsheets were annotated during the qualitative analysis, resulting in 20,733 paragraphs analyzed. We had to dismiss 66 files that did not present any information about the six categories of barriers, which were replaced by

66 other files from our dataset. After the replacement, we ended up with 19,961 paragraphs.

4.4 Corpus Characterization

In Figure 3, we present the distribution of paragraphs analyzed per file. The average number of paragraphs for our set of 500 projects was 41, and the median was 29. Two projects had only 2 paragraphs (minimum), and one had 422 paragraphs in a single file (maximum).

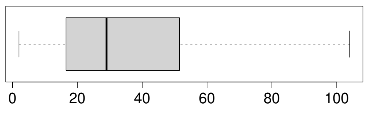

Figure 3: Distribution of paragraphs per file.

Table 2 shows the distribution of categories in our sample. More than 6,000 paragraphs were categorized as "Submit the changes", and more than 2,000 as "Deal with the code" and "Contribution flow." On the other hand, "Choose a task" and "Talk to the community" appear in 116 and 183 paragraphs respectively. Still, 7,461 paragraphs could not be categorized under any category. We analyzed these paragraphs and found different types of content that did not belong to any category. The most recurring cases were: "thank you" messages; license statements or the complete license; links to other sites; instructions on how to open an issue; information in different languages; GitHub badges; and lists of contributors.

Table 2: Characterization of the dataset considering the six categories of barriers [74]

Category	# Paragraphs	# Projects	Avg. per file
Choose a Task	116 (0.58%)	116 (23%)	1
Talk to the Community	183 (0.92%)	183 (37%)	1
Build Local Workspace	1,444 (7.23%)	139 (28%)	10.4
Contribution Flow	2,088 (10.64%)	319 (64%)	6.5
Deal with the Code	2,495 (12.5%)	280 (56%)	8.9
Submit the Changes	6,174 (30.93%)	396 (79%)	15.6
No category	7,461 (37.38%)	483 (97%)	15.4

5 BUILDING AND EVALUATING THE CLASSIFIER

We trained machine learning models to classify information according to the six categories defined in Section 4.1. The 500 annotated spreadsheets were used to extract features for classification (Section 5.1). The data was prepared using text pre-processing techniques. The features created were divided into statistical features (i.e., extracted using statistical methods) and heuristic features (i.e., extracted through identifying linguistic patterns).

The features were then applied to supervised learning algorithms to find the best classification model for this problem (Section 5.2). The annotated dataset was split into two random subsets. A training set (80% of the dataset) was used to compare the different classifiers, and a test set (20% of the dataset) was reserved for testing the classification algorithm with the highest evaluation score. The algorithms were evaluated based on their classification scores, and a final model was trained using the best-performing classification algorithm (Section 5.5). Figure 4 provides an overview of the classification process, which is detailed in the following sections.

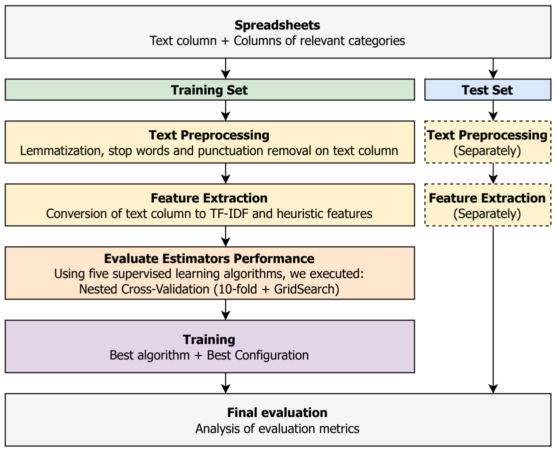
Spreadsheets
Text column + Columns of relevant categories
Training Set
Test Set
Text Preprocessing
Lemmatization, stop words and punctuation removal on text column
Text Preprocessing (Separately)
Feature Extraction
Conversion of text column to TF-IDF and heuristic features
Feature Extraction (Separately)
Evaluate Estimators Performance
Using five supervised learning algorithms, we executed:
Nested Cross-Validation (10-fold + GridSearch)
Training
Best algorithm + Best Configuration
Final evaluation
Analysis of evaluation metrics

Figure 4: The classification process.

5.1 Feature Extraction

In the feature extraction process, the annotated paragraphs (Section 4.3) were converted into numerical data. We divided the feature extraction process into four stages: text pre-processing, the definition of statistical features, the definition of heuristic features, and feature selection.

5.1.1 Text Pre-processing. Before creating any features for the classifier, three pre-processing techniques used in text classification were applied to the paragraphs: lemmatization, stop words, and punctuation removal. In the lemmatization process, the affixes of words in each paragraph were removed, turning the words back to their root form [37]. Words such as *submits*, *submitted*, and *submitting*, for example, were returned to their root form *submit*. To reduce the number of ineffective words in the paragraphs' classification, we also removed stop words, excluding words commonly found in the English vocabulary (e.g., conjunctions and pronouns) [86]. For the same purpose, punctuation was also removed from the text. For both lemmatization and stop words removal, we used the implementations provided by the NLTK library [44].

5.1.2 Statistical Features. We converted the annotated paragraphs into TF-IDF features using the *TfIdfVectorizer* method of the scikit-learn library [6, 58]. In this approach, we represented words as n-grams of size one and two [5]. The acronym TF-IDF is a reference for the multiplication of two statistical measures used in text classification, term frequency (TF) and inverse document frequency (IDF) [38, 82]. For term frequency, we measured how often words occur in a paragraph (number of occurrences of each word per paragraph, divided by the total words in that paragraph). For the inverse document frequency, we counted how often words occur compared to the entire set of paragraphs. The multiplication of both measures gives us statistical features that show the relative importance of each word.

5.1.3 Heuristic Features. The set of statistical features was combined with heuristics found through qualitative analysis to enrich the characteristics used in classification. We adopted a strategy used by previous work [56, 61], in which features are generated by analyzing linguistic patterns in the annotated paragraphs. During the manual analysis, the annotators selected words that could characterize specific categories and be used as patterns for a classification based on heuristics. For example, the word "*commit*" was commonly found in paragraphs annotated as *Submit the changes* (see Table 3 for examples of other categories).

Table 3: Examples of heuristic features per category.

Category	Related examples
Contribution Flow	Clone, push, merge, pull request, contribution
Choose a task	Issue, issue tracker, label, fork
Talk to the community	Mailing list, contact, email, conduct, slack
Build local workspace	Tool, package, update, dependencies
Deal with the code	Code snippet, library, debug, coding convention, method, variable
Submit the changes	Commit, diff, review, test, fetch

Using the rule-based matching approach of the Spacy library [72], we assigned an equal set of heuristic features to each paragraph in the training process. Each feature represented a pattern; paragraphs were assigned the value of 1 when they contained the respective words and 0 otherwise.

5.1.4 Feature Selection. To avoid using features that could be considered irrelevant to our classification, we removed the ones with the lowest scores. The *SelectPercentile* [69] method of the scikit-learn library was used with Chi-square as the score function. Features that fell below the 15^{th} percentile were removed. We manually tested a set of percentiles (5th, 10th, 15th, 20th) based on the default value of Scikit-learn [69], which is the 10th percentile. We chose the 15^{th} percentile as it performed best, as it is commonly done in the literature [4, 14, 70].

5.2 Finding the Best Classifier

A set of classifiers was trained to find the best learning algorithm to solve our classification problem. To train the classifiers, we used two multi-class training strategies: one-vs-rest (OvR) and one-vs-one (OvO) [7]. In the OvR strategy, a binary classifier was trained for each category. The assignment of a category for a paragraph was then made by identifying the binary classifier that best represented the respective paragraph (i.e., the one with the best scores). In the OvO strategy, the samples of each category were grouped in pairs, and the comparison was made in a binary classifier for two categories at a time. To identify what category should be assigned for a paragraph, the predominance of a category among all the pairs was considered as the decision method.

The following classification algorithms were trained during this step: RandomForestClassifier, KNeighborsClassifier, LinearSVC, MultinomialNB, LogisticRegression. The selection was based on similar studies using text classification in Software Engineering [56, 61]. As a baseline, we trained two dummy classifiers, one using the most frequent class label observed in the training set and one providing completely random predictions. As highlighted in Table 2, we noticed that the number of instances per category was unbalanced in our data set, so we used the SMOTE oversampling technique to achieve a better balance between the classes. The SMOTE algorithm was implemented using the imbalanced-learn library [40], a module designed for unbalanced datasets that are recommended by the scikit-learn community. Still, we used chatGPT (GPT 3.5 model) [54] using a few-shot learning approach [10] to compare our results with the performance of this LLM. For the few-shot learning, we randomly selected 12 instances of paragraphs in our training set for each category. Then, we prompted chatGPT to classify 200 instances randomly selected from our test set.

5.3 Evaluation Metrics

To measure the overall performance of the classifiers, we used a combination of three evaluation metrics for data classification: precision, recall, and F1 score. Precision, also known as confidence, provides the proportion of positive samples that were correctly predicted, in contrast to all the samples predicted as positive [28, 85]. Recall gives the fraction of positive samples correctly predicted by the classifier, and the F1 score provides the harmonic mean between precision and recall values [12, 36]. This is a multi-class problem, and the resulting values are the weighted average of all classes.

5.4 Cross-validation

We tested the performance of our classifiers using a nested ten-fold cross-validation strategy [17, 52]. This algorithm divides the dataset of features and labels into ten parts. The ten parts are combined into ten different training and validation subsets, also known as folds. For each fold i divided into k parts ($k = 10$), the k_i part is used as the validation set, and the remaining parts are used as training for each classification algorithm in our list. The average of the weighted F1 scores of the k different classifiers gave us an overall performance for each learning algorithm.

To increase the chance of selecting the best parameters for each algorithm, we applied the GridSearch method to the cross-validation internal loop. The values tested were based on the default values in the scikit-learn library. We selected the best configuration (i.e., classifier, parameters, and training strategy) to train a final classification model.

5.5 Classifier Training

With the best learning algorithm selected, we trained a classifier using the complete dataset used in the previous step (training set in Fig. 4) and the test set (Fig. 4) was used for testing. This was done to show the reliability of both the model and the results [56, 61, 69]. This enabled us to use our complete annotated training set (80% of the sample) and test on the 20% of the original annotated dataset that was not previously used.

5.6 Classifier Human Evaluation

To further evaluate our classification model, we surveyed 46 individuals using Amazon Mechanical Turk [2]. We invited only individuals with prior programming experience by specifying on the Amazon platform "Employment Industry - Software & IT Services" as a selection criterion. The survey was divided into training, evaluation, and demographics. In the training section, we introduced the survey and described the six categories of information used to classify the content. To guarantee that we would only consider participants who paid attention to our questionnaire, we asked them to match each category with their corresponding definitions on a subsequent page. An attention check question was also included among the questions of our questionnaire. We dismissed the answers from 29 participants who selected the wrong definition for more than one category (28) or did not mark the correct answer on the attention check (6)—5 of them selected the wrong answers in both. We ultimately considered 17 valid answers for analysis. Although the number of remaining participants is small due to the substantial number of discarded answers, this is considered a common limitation of crowdsourcing platforms, as suggested in recent studies [46, 64, 79]). The literature also mentions that cleaning the answers is necessary to guarantee data quality and consistency [64].

In the evaluation section, we asked participants to judge the quality of the predictions. We used paragraphs from CONTRIBUTING files of 75 randomly selected projects that were not part of the training or test set—and thus were unknown by the classifier. We randomly selected ten paragraphs classified into each category from that set of OSS projects to use in the questionnaire. For each participant, we randomly assigned 18 paragraphs (3 per category). We gave the same number of sections for each category (instead of a complete file) per participant to ensure the number of paragraphs assessed per category was balanced. Moreover, we use an approach in which the participants recognize if an item belongs to a category (aided recall) instead of asking the participants to label them (unaided recall) because items requiring recognition are easier than items that use unaided recall [9]. So, we provided annotated paragraphs and asked whether the category was correct. We asked each participant to rate each prediction using a 4-point Likert scale (extremely adequate to extremely inadequate). We employed a 4-point Likert scale to compel respondents to take a definitive stance, preventing the use of a neutral option [11].

In the demographics section, we asked participants to provide information about their experience with OSS projects. This section included two multiple-choice questions about years of experience in programming and maintenance of OSS projects and two questions about their role as participants in OSS (coder or non-coder) and their contributions to documentation in OSS projects (Yes/No). In Table 4, we present the overall experience of our participants in programming and OSS projects. As expected, all of our participants had some experience in programming, with the majority of them (64%) having between 3 and 15 years of programming experience. In terms of experience as maintainers in OSS projects, 82% of our participants had at least some experience, and 41% of them had between 3 and 15 years of experience as software maintainers. All 14 participants with experience in OSS defined themselves as coders, and 10 worked with documentation in their projects.

Table 4: Experience of survey participants in programming and OSS project maintenance.

Experience / Type	Programming	OSS
No experience	0	3
Less than or equal to 3 years	4	7
Greater than 3 years and less than 15 years	11	7
Greater than or equal to 15 years	2	0

Table 5: F1 scores for classifiers tested in the ten-fold cross-validation process.

	With SMOTE		Without SMOTE	
	OvR	OvO	OvR	OvO
RF	0.636	0.625	0.620	0.609
kNN	0.563	0.566	0.516	0.530
SVC	0.630	0.634	0.652	0.646
LR	0.612	0.606	0.617	0.602
NB	0.579	0.580	0.636	0.633
Dummy (Freq.)	0.001	0.009	0.001	0.190
Dummy (Rand.)	0.001	0.010	0.001	0.010
chatGPT (macro F1)		0.272		

6 RQ1. HOW ACCURATELY CAN WE AUTOMATICALLY CLASSIFY THE CONTENT OF CONTRIBUTING FILES?

This section details the evaluation of the machine learning models.

6.1 Comparing Different Classifiers

To identify the best classifier for our problem, we compared the outputs of five machine learning algorithms and two dummy algorithms in a ten-fold cross-validation process, in addition to chatGPT with a few-shot learning approach [10]. Table 5 presents the F1 scores for each classifier. The best F1 score of 0.652 is from the LinearSVC classifier, without oversampling and using the OvR strategy. The second-best score is from the same classifier configuration but uses the OvO multi-class strategy. The performance of chatGPT using a few-shot approach reached an overall macro precision of 0.250, recall of 0.322, and F1 equal to 0.272. Ignoring the dummy classifiers and chatGPT, the classification model with the worst scores of 0.516 and 0.530 was kNN. Such results follow similar performance found by Prana et al. [61], who categorized the content of README files.

Because of its scores and similar performance in other studies, LinearSVC was chosen as the final machine learning algorithm. Based on the outputs of the GridSearch algorithm, we found that the best hyper-parameters for LinearSVC were 1,000 iterations (max_iter = 1000), regularization equal to one (C = 1), and tolerance equal to 0.001 (tol = 0.001). The LinearSVC algorithm was trained again with this final configuration without oversampling and using the OvR strategy, as this combination provided the best F1 score in our comparison of classifiers. Table 6 presents the training data and its performance per class in relation to the test set.

In Table 6, we can see that the performance varies per category. The information about Deal with the code (DC) and Build local workspace (BW) barriers is fairly well predicted (F1 0.711 and 0.716, respectively). On the other hand, Choose a task (CT) and Contribution flow (CF) had the lowest scores of 0.379 and 0.345, respectively. Some external factors may have influenced such performances. The number of instances per class, for example, might justify the low score of Choose a Task (CT), which on average had only 1 paragraph per project analyzed (see Section 4.4). The fact that the Contribution flow contained more generic information than other content-specific categories, such as Build local workspace, might also explain the difference in performance.

For the sake of comparison, we include the results of the chatGPT few-shot learning approach per class in Table 7. As can be observed, the Linear SVC model outperforms chatGPT in this context in almost all metrics and categories. The exception are the recall for the Talk to Community and Contribution Flow categories. Interestingly, chatGPT could not correctly identify any paragraph belonging to the Choose a Task Category—although it received 5 instances, and (incorrectly) predicted 3 instances. Considering the overall metrics, the linearSVC model is more than 2x better than chatGPT in terms of recall, precision, and F1.

Table 6: Performance of the final model (LinearSVC).

Category	F1	Precision	Recall
Build local workspace	0.716	0.674	0.764
Deal with the code	0.711	0.682	0.743
Talk to the community	0.648	0.657	0.639
Submit the changes	0.617	0.717	0.541
Choose a task	0.379	0.687	0.261
Contribution flow	0.345	0.519	0.258
No categories identified	0.592	0.596	0.588
Overall	0.651	0.655	0.662

Table 7: Performance of chatGPT with few-shot learning.

Category	F1	Precision	Recall
Build local workspace	0.438	0.389	0.500
Deal with the code	0.154	0.200	0.125
Talk to the community	0.522	0.400	0.750*
Submit the changes	0.114	0.160	0.089
Choose a task	0.000	0.000	0.000
Contribution flow	0.237	0.184	0.333*
No categories identified	0.436	0.420	0.453
Overall	0.322	0.250	0.272

Confusion between categories In Figure 5, we present the confusion matrix produced by the final classification model. Using a confusion matrix, we can assess the similarity between the different categories of information and verify what labels contain false positives. The main diagonal represents the true positives for each class, and the upper and lower triangular submatrices represent the misclassifications.

In line with the previous results, contribution flow (CF) and Choose a task (CT) are the categories of information with the highest amount of misclassifications, with only 26% true positives. Contribution flow (CF) had more false positives assigned to deal with the code (DC) than its own category. Such results may confirm the assumption that because contribution flow (CF) contains a wide range of information, and choose a task (CT) has just a few samples used for training, they performed poorly.

All other categories had more than 50% true positives. Build local workspace (BW) and Deal with the code (DC) had the lowest number of false positives ($< 25\%$). Talk to the community (TC) also presented good performance, with less than 36% incorrect predictions. This may be because such categories contain more specific content and a good number of samples per class.

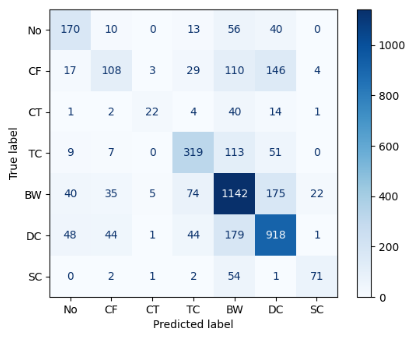

Figure 5: Confusion matrix for LinearSVC.
Legend: BW (Build local workspace), DC (Deal with the code), TC (Talk with the community), SC (Submit the changes), CT (Choose a task), CF (Contribution flow).

6.2 Observations from the Survey

In Figure 6, we present the participants' evaluation of the predictions made by our final model. For all the categories, at least 30% of the predictions were considered extremely adequate for their paragraphs, and at least 69% of the predicted categories were considered at least somewhat adequate. The best-evaluated category was Build local workspace (BW), with 47% of participants considering its predictions extremely adequate.

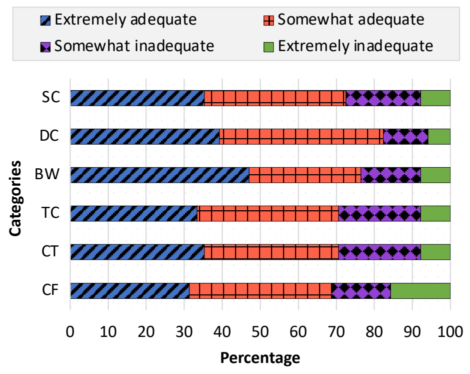

Figure 6: Survey: Participants' evaluation of predictions made by the final classification model.
Legend: BW (Build local workspace), DC (Deal with the code), TC (Talk with the community), SC (Submit the changes), CT (Choose a task), CF (Contribution flow).

When we aggregate extremely adequate and somewhat adequate, Deal with the code (DC) leads the adequacy board with 82% of predictions considered adequate. Contribution flow (CF) has the lowest estimates, with 31% of its predictions estimated as somewhat or extremely inadequate. The second to last place is held by Choose a task (CT) and Talk to the community (TC), with 29% of their predictions considered somewhat inadequate or less. Such results follow similar outcomes found in the evaluation scores of Table 6, confirming the nature of our predictions.

To further understand the disagreement between the classifier output and the crowd, we manually analyzed the 12 paragraphs in which 50% or more of the respondents disagreed with the predictions. In summary, we found that the prediction was incorrect in 9 cases. In the 3 other cases, the prediction was correct (2 related to Choose a Task and one related to Build the Workspace).

Answer to RQ1: After comparing five supervised learning algorithms, we were able to classify the content of CONTRIBUTING files achieving an F-measure of 0.651, with precision = 0.655 and recall = 0.622. Although different categories of information differed in performance, in general 69% of the classifications were considered appropriate by external reviewers.

7 RQ2. TO WHAT EXTENT DO CONTRIBUTING FILES COVER CONTENT RELATED TO CONTRIBUTION BARRIERS?

We used the classification model to predict a set of the remaining 2,274 CONTRIBUTING files from our dataset that we had not used in the previous steps. From the 9,514 files, we removed 6,599 because they did not meet the filtering criteria presented in Section 4.2.

Figure 7 shows the distribution of projects and the average of paragraphs per category in the CONTRIBUTING files in which each category appeared at least once. A total of 2,265 (99.6%) projects had at least one paragraph that did not belong to any category, with an average of 15 unidentified paragraphs per file.

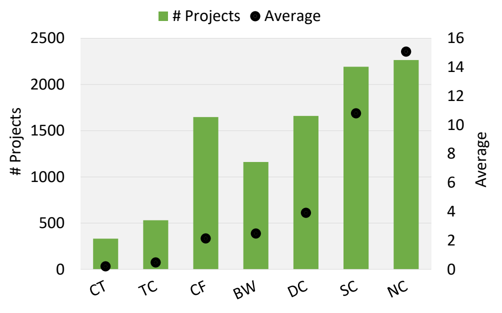

Figure 7: Average number of paragraphs per category in the CONTRIBUTING files predicted.
Legend: BW (Build local workspace, DC (Deal with the code), TC (Talk with the community), SC (Submit the changes), CT (Choose a task), CF (Contribution flow), NC (No categories identified).

Submit the changes was the category with the highest number of paragraphs per CONTRIBUTING file, appearing in 2,192 projects. The *Deal with the code* category represented the second highest average of paragraphs per CONTRIBUTING file and the second in the number of projects, being identified in 1,660 projects with an average of 4 paragraphs per file. *Contribution Flow* was the category with the third highest frequency, appearing in 1,648 projects, with an average of only two paragraphs per project. A similar phenomenon happened with *Build local workspace* (1,162 projects; 2 paragraphs/project). *Talk to the community* (513 projects) and *Choose a task* (332 projects) were in the lowest positions.

Regarding the frequency of categories per project, not all categories are covered by the CONTRIBUTING files (see Figure 8). From our set of 2,274 OSS projects, we identified 729 with content related to four of the six categories (32%), 603 related to 3 (27%), and 411 files containing only two categories (18%). For 287 projects, we identified information about five categories, and for only 65 projects (6%), the classifier identified information about all six categories. On the lower bound, only one category of information was identified for 165 projects. We also found 14 projects where no categories were identified. In a manual inspection of their CONTRIBUTING files, we detected that none of them present any information that could be mapped to any of the six categories, validating the analysis made by the classifier. While some presented ways to report an issue, others contained links to contribution guidelines elsewhere (some on the GitHub wiki, others outside GitHub).

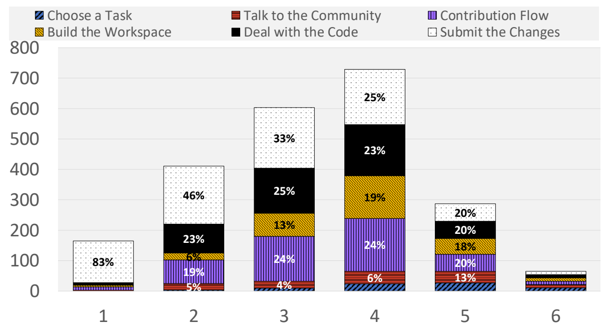

Figure 8: Distribution of categories per CONTRIBUTING file predicted. The percentages represent the proportion of each category in the respective subset of files.

The distribution of categories is in line with the distribution of 500 projects manually annotated during the qualitative analysis, providing further evidence of the adequacy of the classifier. The only differences are that no projects analyzed had zero categories of information and the *Contribution flow* category had a slightly higher average of paragraphs per CONTRIBUTING file.

In summary, more than 50% of the CONTRIBUTING files present information pertaining to fewer than 3 categories of barriers faced by newcomers, while only 15% present information classified in 5 or 6 different categories. These results—in addition to the fact that more than 60% of the projects collected do not have a CONTRIBUTING file (Table 1)—evidence that this highly relevant resource for new contributors is still inadequate for mitigating barriers faced by newcomers. In particular, the lack of content about Choosing a task (CT) and Building the workspace (BW) is crucial and may hinder onboarding and lead to dropouts [67, 74].

Answer to RQ2: Most CONTRIBUTING files focus on the final stages of the contribution process. Categories containing information such as how to submit the changes and deal with the code are the most frequent, while information about choosing a task and contacting the community is often missing.

8 DISCUSSION

Lack of essential information for newcomers. In our study, we noticed that many projects do not provide primary information that new contributors may need when attempting to contribute to a project. This was highlighted in previous literature [71, 74] and evidenced in our analysis based on the number of categories of information covered per project in Figure 8 and Table 2. Most projects had a maximum of 3 out of 6 categories covered in their CONTRIBUTING files. This suggests that OSS projects might not satisfy newcomers' needs in terms of documentation when considering the categories defined by the literature. Some of the most critical barriers faced by newcomers [74, 78] are not covered by the CONTRIBUTING files. Table 2 shows that only 23% of the files analyzed had information about how to Choose a task, and 28% of them presented some information about how to Build their local workspace. The "curse of expertise" [24], i.e., the inherent cognitive bias stemming from the deep familiarity with the subject matter, may hamper project maintainers' ability to accurately evaluate the comprehensiveness and clarity of their documentation. Our results can shed light on the gaps in the existing documentation from the perspective of barriers commonly faced by newcomers.

A more critical problem is also evidenced in Table 1, showing that ≈ 65% of the projects in our sample (more than 6,000 in absolute numbers) do not have a CONTRIBUTING file available in their repositories. Although some projects prefer to use other resources to explain their contributing process (e.g., *Valhalla* [83] uses a section in their README file), many popular repositories do not contain any orientation for newcomers, even though they are open to external submissions (e.g., *Google Sanitizers* [32], *Microsoft PHPSQL* [50], *NVIDIA NCCL* [51]).

Most files focus on the contribution process's final steps. In Figure 7, we show the average number of paragraphs identified per category in the projects of our qualitative analysis. The results suggest that the category with the highest number of paragraphs is Submit the Changes, followed by Contribution Flow and Deal with the Code. Although Dealing with the code focuses on the more general steps of the project, Submitting the changes and Dealing with the code are intended to be relevant for newcomers in the later stages of their contribution, after they selected a task, built their workspace, and established communication with the project's community. This result suggests that projects tend to focus more on the last stages of the contribution, assuming newcomers already know how to implement their contribution.

Implications for practice and research. As a result of this study, we also implemented a web tool to provide feedback to project maintainers about their CONTRIBUTING files [19]. The maintainer only needs to input their project URL, and our tool reviews the project's CONTRIBUTING file using our classification model. The tool provides a chart showing the distribution of paragraphs per category of information, a discussion about the dominant categories (i.e., the highest number of paragraphs) and weak categories (i.e., the lowest number of paragraphs), and a comparison of the input project with other popular repositories on GitHub. In addition, the tool provides a clear description for each category when the report is presented to the user, highlighting why they are important. The report provided by the tool also suggests CONTRIBUTING files that maintainers could use as inspiration to enrich a specific faulty category. We envision the proposed tool as a starting point to support better documentation files.

This tool could be particularly useful to community builders and managers who oversee a non-trivial number of projects. Those playing these roles need information about the content of CONTRIBUTING files in multiple projects in the ecosystem to take action. The classifier may support their efforts by providing insights into the types of information available for each project.

The tool is also an important step toward implementing automated on-demand developer documentation, which automatically parses documentation and generates responses to user queries [66], and smart assistants [16]. These tools need to parse existing documentation and classify information in order to provide adequate assistance to newcomers.

From the research perspective, our study helps to understand how the current content of CONTRIBUTING files addresses newcomers' needs. Our work can be extended to evaluate the content quality of the CONTRIBUTING files, which may help newcomers find appropriate documentation. Future work can also investigate the subcategories of Steinmacher et al.'s model [73].

9 LIMITATIONS AND DESIGN DECISIONS

In this section, we present our work's limitations and trade-offs for research design decisions.

Using the most popular projects from GitHub. We focused our study on GitHub and the results may not generalize for the whole OSS universe. Nevertheless, GitHub is arguably the most popular OSS hosting platform. Additionally, the selected projects may not generalize to GitHub as well, since our projects were selected based on their programming language and popularity. Still, there may be projects that are not exactly software projects in our sample, like "algorithms" and "awesome lists"—in a manual analysis, these projects correspond to ≈1% of our sample. We acknowledge that a more diverse set of projects would potentially bring more data points with different styles. However, focusing on more popular projects and on GitHub brings more confidence about the relevance of the CONTRIBUTING files analyzed. We opted to keep a more trustful set of projects, rather than expanding the data points.

Unit of analysis. Our approaches to selecting, filtering, analyzing, and classifying documentation files were based on prior studies [8, 26, 61]. Still, our decision to choose paragraphs as the unit of analysis instead of lines or larger chunks of text could impact our results. We attempted to use lines as units of analysis, but they did not provide enough context to identify the categories during manual analysis since the information in CONTRIBUTING files is highly contextual. Paragraphs provide enough context for identifying the categories, and Markdown provides a standard approach to split the content into paragraphs (i.e., blank lines).

Other approaches to determining the content of CONTRIBUTING files could also be used. For example, a classification model based on section names could be a great alternative for our decision. We decided to make our classification based on paragraphs and not on section names for the same reasons that we did not use lines as units of analysis. We also decided to keep duplicated paragraphs from distinct projects in our dataset, as CONTRIBUTING files from different projects may follow similar guidelines. We ran LinearSVC without the duplicated paragraphs, and the performance was similar to our final model (precision: 0.612, recall: 0.621).

It is also worth mentioning that we only analyzed the content available on the CONTRIBUTING files. We did not explore any external links from these files or resources they reference; we also did not check textual, HTML, .ris, or other types of files containing contribution guidelines. We analyzed 95 README files from (randomly selected) projects that we dismissed because of the absence of the CONTRIBUTING file; only three had links to external guidelines, and six had sections related to contribution. This could have limited the conclusions made for projects such as *Apple Swift* [3], and others highlighted in Section 7, whose contribution file only contained directions to other sources of documentation. Future studies are encouraged to (i) analyze one level of depth using the links available in the CONTRIBUTING files, and (ii) understand how to use the proposed approach to refactor contributing guidelines contained on README and other textual files onto CONTRIBUTING files.

Representativeness of contributors' perspective. To assess the quality of our classification model, we invited participants with programming expertise to answer questions in which they judged a set of predictions made by our classifier. Although we introduced a tutorial at the beginning of the questionnaire, we cannot guarantee that the answers given by the respondents represent the perspective of contributors or the correctness of the predicted categories. To mitigate this problem, we not only asked the participants to match the categories definition with their names in the early stages of the survey but also included an attention check question into our set of questions to ensure participants did not randomly assign answers for them. Once again, our choice was guided by the trustfulness of the data points. We kept only a small set of answers, which can be considered more reliable than having more data points and losing reliability.

Coverage of the categories and information. We decided to use a pre-existing set of categories to label our dataset according to the barriers newcomers could face. We acknowledge that the categories analyzed may not cover all the information a newcomer may need when contributing to a project. However, the set of categories resulted from several studies investigating problems associated with documentation files in the context of OSS repositories [76, 77]. We opted to classify our data using a validated set of categories rather than explore potential new categories.

Construction of the classifier. To build a classification model from scratch, a set of design decisions were made throughout the process. We understand that other strategies could have been adopted in building our model (e.g., the use of additional pre-trained models) and that the decisions made may have an influence on the performance reported in this study. To mitigate this issue, we compared our classifier with chatGPT in Section 6.1, and trained a supervised model with the same dataset using FastText [21] (precision: 0.653, recall: 0.653). Both strategies presented a similar or worse performance than our final classifier. We also undertook an ablation study to determine the impact of the heuristic and statistical features. Two models were constructed using the same configurations as our final classifier: one solely with statistical features (precision: 0.657, recall: 0.664) and the other with only heuristic features (precision: 0.414, recall: 0.493). Both models exhibited performance comparable to, or less than, our final estimator.

10 CONCLUSION

A primary documentation resource for newcomers embarking on open source software projects is the CONTRIBUTING file. Located within repositories, these files outline the project's contribution guidelines. While many OSS communities utilize CONTRIBUTING files to orient newcomers, the comprehensiveness of their content was largely unexplored.

In this paper, we investigate the extent to which CONTRIBUTING files address the onboarding barriers newcomers face in OSS projects. Drawing upon a barrier model from existing literature [74], we manually analyzed CONTRIBUTING files from 500 projects. Our findings indicate a notable lack of information: 90% of the projects lacked content in at least two of the six information categories, with 79% missing details in three or more categories. Notably, our manual review revealed that over 75% of the projects failed to include guidance on task selection and workspace setup, two key barriers for newcomers as highlighted by Steinmacher et al. [74].

We also built a machine learning model designed to automatically classify the information from CONTRIBUTING files from other projects and thereby help projects identify missing information in their files. Overall, the classifier performed well in this multiclass problem, with an overall precision of 0.655 and a recall of 0.662. The performance was good for four out of the six categories of information (F1 $\geq$ 0.61): Build local workspace, Deal with the code, Talk to the community, and Submit the changes. Exceptions were How to choose a task and Contributing flow, with low recall (< 0.3) and F1 of 0.379 and 0.345, respectively.

In summary, our findings indicate that many OSS projects need to improve the comprehensiveness of their 'CONTRIBUTING' files to better cater to newcomers. Evaluating 2,274 projects using our machine learning model, our results echoed the findings from our qualitative assessment: 84% of the projects lacked content in at least two of the six information categories, and 52% were deficient in three or more categories. To assist with this issue, we developed an online tool designed to offer feedback to project maintainers about how their 'CONTRIBUTING' files address onboarding challenges, ensuring that the communities are better equipped to welcome and nurture their next generation of contributors.

11 DATA AVAILABILITY

The artifacts used in this paper are available on Zenodo [20].

ACKNOWLEDGMENTS

This work is partially supported by FAPESP (grant #2019/12743-4), CNPq/MCTI/FNDCT (#408812/2021-4), MCTIC/CGI/FAPESP (#2021/06662-1), and NSF (2236198, 2247929, 2024561, and 2303042).

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1199–1210.

[2] Amazon. 2023. Amazon Mechanical Turk (Website). https://www.mturk.com/ [Accessed on Aug-2023].

[3] Apple. 2023. Apple Swift (CONTRIBUTING.md). https://github.com/apple/swift/blob/main/CONTRIBUTING [Accessed on Aug-2023].

[4] Tasneem Batool, Mostafa Abuelnoor, Omar El Boutari, Fadi Aloul, and Assim Sagahyroon. 2021. Predicting Hospital No-Shows Using Machine Learning. In *2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*. 142–148. https://doi.org/10.1109/IoTaIS50849.2021.9359692

[5] Ismaïl Biskri and Sylvain Delisle. 2002. Text classification and multilinguism: Getting at words via n-grams of characters. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2002), Orlando (Florida, USA)*, Vol. 5. 110–115.

[6] Giuseppe Bonaccorso. 2017. 12.2.4.2 Tf-idf Vectorizing. In *Machine Learning Algorithms*. Packt Publishing.

[7] Giuseppe Bonaccorso. 2017. 2.1.1.1 One-vs-All. In *Machine Learning Algorithms*. Packt Publishing.

[8] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 334–344.

[9] Norman M Bradburn, Seymour Sudman, and Brian Wansink. 2004. *Asking questions: the definitive guide to questionnaire design–for market research, political polls, and social and health questionnaires*. John Wiley & Sons.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[11] Seung Youn Chyung, Katherine Roberts, Ieva Swanson, and Andrea Hankinson. 2017. Evidence-based survey design: The use of a midpoint on the Likert scale. *Performance Improvement* 56, 10 (2017), 15–23.

[12] Giuseppe Ciaburro and Prateek Joshi. 2019. 2.9.4 There's More... In *Python Machine Learning Cookbook (2nd Edition)*. Packt Publishing.

[13] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[14] Maria Eduarda Rosa da Silva, Giovani Gracioli, and Gustavo Medeiros de Araujo. 2022. Feature Selection in Machine Learning for Knocking Noise detection. In *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 1–8. https://doi.org/10.1109/SBESC56799.2022.9964726

[15] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. 2021. What makes a great maintainer of open source projects?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 982–994.

[16] James Dominic, Jada Houser, Igor Steinmacher, Charles Ritter, and Paige Rodeghero. 2020. Conversational bot for newcomers onboarding to open source projects. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 46–50.

[17] Günhan Dündar and Mustafa Berke Yelten. 2020. 3.6.2 Resampling. In *Modelling Methodologies in Analogue Integrated Circuit Design*. Institution of Engineering and Technology.

[18] Omar Elazhary, Margaret-Anne Storey, Neil Ernst, and Andy Zaidman. 2019. Do as i do, not as i say: Do contribution guidelines match the github contribution process?. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 286–290.

[19] Fronchetti et al. 2023. Contributing Files (Website). https://contributing.streamlit.app/ [Accessed on Aug-2023].

[20] Fronchetti et al. 2023. Replication Package (Zenodo Repository). https://zenodo.org/record/8270217 [Accessed on Aug-2023].

[21] Facebook. 2023. FastText (Website). https://fasttext.cc/ [Accessed on Aug-2023].

[22] Fabian Fagerholm, Alejandro Sanchez Guinea, Jay Borenstein, and Jürgen Münch. 2014. Onboarding in open source projects. *IEEE Software* 31, 6 (2014), 54–61.

[23] Fabian Fagerholm, Alejandro S Guinea, Jürgen Münch, and Jay Borenstein. 2014. The role of mentoring and project characteristics for onboarding in open source software projects. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.

[24] Matthew Fisher and Frank C Keil. 2016. The curse of expertise: When more knowledge leads to miscalibrated explanatory insight. *Cognitive science* 40, 5 (2016), 1251–1269.

[25] Karl Fogel. 2009. *How To Run A Successful Free Software Project - Producing Open Source Software*. CreateSpace, Scotts Valley, CA.

[26] Felipe Fronchetti, Igor Wiese, Gustavo Pinto, and Igor Steinmacher. 2019. What attracts newcomers to onboard on oss projects? tl; dr: Popularity. In *IFIP International Conference on Open Source Systems*. Springer, 91–103.

[27] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 109–119.

[28] Johannes Fürnkranz and Peter A Flach. 2003. An analysis of rule evaluation metrics. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 202–209.

[29] GitHub. 2020. GitHub Octoverse. https://octoverse.github.com/credits/ [Accessed on Jun-2023].

[30] GitHub. 2022. GitHub Flavored Markdown Specs Paragraphs. https://github.com/gfm/#paragraphs [Accessed on Jun-2023].

[31] GitHub. 2022. Setting guidelines for repository contributors. https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/setting-guidelines-for-repository-contributors [accessed on Jun-2023].

[32] Google. 2023. Google Sanitizers (GitHub Repository). https://github.com/google/sanitizers [Accessed on Aug-2023].

[33] Kazi Amit Hasan, Marcos Macedo, Yuan Tian, Bram Adams, and Steven Ding. 2023. Understanding the Time to First Response In GitHub Pull Requests. In *Intl. Conference on Mining Software Repositories (MSR 2023)*.

[34] Hideaki Hata, Taiki Todo, Saya Onoue, and Kenichi Matsumoto. 2015. Characteristics of sustainable oss projects: A theoretical and empirical study. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, 15–21.

[35] Helena Holmstrom, Eoin Ó Conchúir, J Agerfalk, and Brian Fitzgerald. 2006. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*. IEEE, 3–11.

[36] Mohammad Hossin and Md Nasir Sulaiman. 2015. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process* 5, 2 (2015), 1.

[37] Ammar Ismael Kadhim. 2018. An Evaluation of Preprocessing Techniques for Text Classification. *International Journal of Computer Science and Information Security* 16, 6 (2018).

[38] Frank Kane. 2017. 9.7 TF-IDF. In *Hands-on Data Science and Python Machine Learning*. Packt Publishing.

[39] Jacob Krüger, Sebastian Nielebock, and Robert Heumüller. 2020. How Can I Contribute? A Qualitative Analysis of Community Websites of 25 Unix-Like Distributions. In *Proceedings of the Evaluation and Assessment in Software Engineering*. 324–329.

[40] Imbalanced Learn. 2023. Imbalanced Learn (Website). https://imbalanced-learn.org/stable/ [Accessed on Aug-2023].

[41] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 187–197.

[42] Timothy C Lethbridge, Janice Singer, and Andrew Forward. 2003. How software engineers use documentation: The state of the practice. *IEEE software* 20, 6 (2003), 35–39.

[43] Jiawei Li and Iftekhar Ahmed. 2023. Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality. (2023).

[44] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1* (Philadelphia, Pennsylvania) *(ETMTNLP '02)*. Association for Computational Linguistics, USA, 63–70.

[45] Yuzhan Ma, Sarah Fakhoury, Michael Christensen, Venera Arnaoudova, Waleed Zogaan, and Mehdi Mirakhorli. 2018. Automatic classification of software artifacts in open-source applications. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 414–425.

[46] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2017. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software* 126 (2017), 57–84.

[47] Gerardo Matturro, Karina Barrella, and Patricia Benitez. 2017. Difficulties of newcomers joining software projects already in execution. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 993–998.

[48] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. 2018. Open source barriers to entry, revisited: A sociotechnical perspective. In *Proceedings of the 40th International conference on software engineering*. 1004–1015.

[49] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application programming interface documentation: what do software developers want? *Journal of Technical Writing and Communication* 48, 3 (2018), 295–330.

[50] Microsoft. 2023. Microsoft PHPSQL (GitHub Repository). https://github.com/microsoft/msphpsql [Accessed on Aug-2023].

[51] NVIDIA. 2023. NVIDIA NCCL (GitHub Repository). https://github.com/NVIDIA/nccl [Accessed on Aug-2023].
[52] Fred Nwanganga and Mike Chapple. 2020. 9.1.1.1 k-Fold Cross-Validation. In *Practical Machine Learning in R*. John Wiley & Sons.
[53] Open Source Guides. 2022. Open Source Guides – Starting an Open Source Project. https://opensource.guide/starting-a-project/ [Accessed on Jun-2023].
[54] OpenAI. 2023. ChatGPT (Website). https://chat.openai.com/ [Accessed on Aug-2023].
[55] Susmita Hema Padala, Christopher John Mendez, Luiz Felipe Dias, Igor Steinmacher, Zoe Steine Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Dale Simpson, Margaret Burnett, et al. 2020. How gender-biased tools shape newcomer experiences in oss projects. *IEEE Transactions on Software Engineering* (2020).
[56] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How can i improve my app? classifying user reviews for software maintenance and evolution. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 281–290.
[57] Yunrim Park and Carlos Jensen. 2009. Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 3–10.
[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[59] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. 2016. More common than you think: An in-depth study of casual contributors. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, Vol. 1. IEEE, 112–123.
[60] Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, Sonia Haiduc, Barbara Russo, and Michele Lanza. 2017. Automatic identification and classification of software development video tutorial fragments. *IEEE Transactions on Software Engineering* 45, 5 (2017), 464–488.
[61] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of GitHub README files. *Empirical Software Engineering* 24, 3 (2019), 1296–1327.
[62] Pooja Rani, Sebastiano Panichella, Manuel Leuenberger, Andrea Di Sorbo, and Oscar Nierstrasz. 2021. How to identify class comment types? A multi-language approach for class comment classification. *Journal of systems and software* 181 (2021), 111047.
[63] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. 155–165.
[64] Brittany Reid, Markus Wagner, Marcelo d'Amorim, and Christoph Treude. 2022. Software Engineering User Study Recruitment on Prolific: An Experience Report. *arXiv preprint arXiv:2201.05348* (2022).
[65] Martin P Robillard and Yam B Chhetri. 2015. Recommending reference API documentation. *Empirical Software Engineering* 20, 6 (2015), 1558–1586.
[66] Martin P Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, et al. 2017. On-demand developer documentation. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 479–483.
[67] Fabio Santos, Bianca Trinkenreich, João Felipe Pimentel, Igor Wiese, Igor Steinmacher, Anita Sarma, and Marco A Gerosa. 2022. How to choose a task? Mismatches in perspectives of newcomers and existing contributors. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*.
[68] CJ Satish and M Anand. 2016. Software documentation management issues and practices: A survey. *Indian Journal of Science and Technology* 9, 20 (2016), 1–7.
[69] Scikit-learn. 2023. Cross-validation: evaluating estimator performance (Documentation). https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html [Accessed on Jun-2023].
[70] Francesco Setragno, Massimiliano Zanoni, Augusto Sarti, and Fabio Antonacci. 2017. Feature-based characterization of violin timbre. In *2017 25th European Signal Processing Conference (EUSIPCO)*. 1853–1857. https://doi.org/10.23919/EUSIPCO.2017.8081530
[71] Dan Sholler, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye, and Greg Wilson. 2019. Ten simple rules for helping newcomers become contributors to open projects. *PLoS computational biology* 15, 9 (2019), e1007296.
[72] Spacy. 2023. Rule Based Matching (Documentation). https://spacy.io/usage/rule-based-matching [Accessed on Aug-2023].
[73] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*. 1379–1392.
[74] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. 2016. Overcoming Open Source Project Entry Barriers with a Portal for Newcomers. In *ICSE '16*. Association for Computing Machinery, New York, NY, USA, 273–284.
[75] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A Gerosa. 2018. Almost there: A study on quasi-contributors in open source software projects. In *Proceedings of the 40th International Conference on Software Engineering*. 256–266.
[76] Igor Steinmacher, Marco Aurélio Graciotto Silva, and Marco Aurélio Gerosa. 2014. Barriers faced by newcomers to open source projects: a systematic review. In *IFIP International Conference on Open Source Systems*. Springer, 153–163.
[77] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.
[78] Igor Steinmacher, Christoph Treude, and Marco Aurelio Gerosa. 2018. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software* 36, 4 (2018), 41–49.
[79] Kathryn T Stolee and Sebastian Elbaum. 2010. Exploring the use of crowdsourcing to support empirical studies in software engineering. In *Proceedings of the 2010 ACM-IEEE international symposium on Empirical software engineering and measurement*. 1–4.
[80] Xin Tan, Yiran Chen, Haohua Wu, Minghui Zhou, and Li Zhang. 2023. Is It Enough to Recommend Tasks to Newcomers? Understanding Mentoring on Good First Issues. *arXiv preprint arXiv:2302.05058* (2023).
[81] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A first look at good first issues on GitHub. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 398–409.
[82] Jalaj Thanaki. 2017. 5.3.4.1 Understanding TF-IDF. In *Python Natural Language Processing*. Packt Publishing.
[83] Valhalla. 2023. Valhalla (GitHub Repository). https://github.com/valhalla/valhalla [Accessed on Aug-2023].
[84] Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. 2014. How social Q&A sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 342–354.
[85] Sathiyamoorthi Velayutham. 2020. 3.5.1 Precision. In *Handbook of Research on Applications and Implementations of Machine Learning Techniques*. IGI Global.
[86] S Vijayarani, Ms J Ilamathi, and Ms Nithya. 2015. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks* 5, 1 (2015), 7–16.
[87] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. 2022. Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks. *ACM Transactions on Computer-Human Interaction* 29, 2 (2022), 1–33.

Received 2023-02-02; accepted 2023-07-27