Tell Me Who Are You Talking to and I Will Tell You What Issues Need Your Skills

Fabio Santos,[1] Jacob Penney,[1] João Felipe Pimentel,[1] Igor Wiese,[2] Igor Steinmacher,[1] Marco A. Gerosa[1]

[1]*Northern Arizona University, Flagstaff, AZ, USA,*
[2]*Universidade Tecnológica Federal do Paraná, Campo Mourão, PR, Brazil*

{fabio_santos, jacob_penney, joao.pimentel}@nau.edu, igor@utfpr.edu.br, {igor.steinmacher, marco.gerosa}@nau.edu

***Abstract*—Selecting an appropriate task is challenging for newcomers to Open Source Software (OSS) projects. To facilitate task selection, researchers and OSS projects have leveraged machine learning techniques, historical information, and textual analysis to label tasks (a.k.a. issues) with information such as the issue type and domain. These approaches are still far from mainstream adoption, possibly because of a lack of good predictors. Inspired by previous research, we advocate that label prediction might benefit from leveraging metrics derived from communication data and social network analysis (SNA) for issues in which social interaction occurs. Thus, we study how these "social metrics" can improve the automatic labeling of open issues with API domains—categories of APIs used in the source code that solves the issue—which the literature shows that newcomers to the project consider relevant for task selection. We mined data from OSS projects' repositories and organized it in periods to reflect the seasonality of the contributors' project participation. We replicated metrics from previous work and added social metrics to the corpus to predict API-domain labels. Social metrics improved the performance of the classifiers compared to using only the issue description text in terms of precision, recall, and F-measure. Precision (0.922) increased by 15.82% and F-measure (0.942) by 15.89% for a project with high social activity. These results indicate that social metrics can help capture the patterns of social interactions in a software project and improve the labeling of issues in an issue tracker.**

***Index Terms*—Labels, Tags, Skills, Human Factors, Mining Software Repositories, Social Network Analysis, Open Source Software, Machine Learning**

I. Introduction

Contributors to OSS projects struggle to select an appropriate task from the issue tracker [1]. The information available in a project's open task lists can be too large [2]–[4]. Labeling issues can facilitate task selection [5]. However, community maintainers report that manually labeling issues is challenging and time-consuming [6].

Recent studies that automatically label issues are limited to classifying the type of the issue (e.g., feature, bug, etc.) [7]–[14]. A notable exception is the work of Santos et al. [15], which shows that labeling issues with API domains is feasible and can benefit developers in choosing their tasks. APIs typically encapsulate modules that offer functionality in a specific domain (e.g., user interface, security, cloud, testing, etc.), hiding functionality details. If the contributors know which API domains are required to work on each issue, they may choose tasks that better match their skillset.

In this paper, we replicate Santos et al.'s [15] work, investigating how metrics derived from the communication among developers (a.k.a. social metrics) can improve the performance of the classifiers. Santos et al.'s [15] achieve 75% precision, and such a high number of false positives may discourage practical adoption.

Given the sociotechnical nature of software engineering [16], we hypothesize that social metrics capture communication patterns that can help predict API-domain labels. Social metrics have shown promising results in other contexts. For example, Wiese et al. [17] evidenced the usefulness of the communication context and communication roles to predict co-changes. Zimmermann and Nagappan [18] predicted defects using developer network metrics. Meneely et al. [19] used social network analysis to predict software failures. The intuition behind the research is that experts in a specific domain have a greater chance to join the conversations where given expertise is involved. Therefore, communication patterns might reveal the importance of the commenters in the social structure surrounding the software project, and conversations may help predict API domains needed to solve open issues.

We aim to answer the following research questions:

RQ1. To what extent can social metrics improve the prediction of API-domain labels?

RQ2. To what extent can we transfer learning among projects using social metrics to predict the API-domain labels?

By understanding how communication context and social network analysis play a role as predictors, we aim to improve the API-domain label prediction model, increasing its precision, recall, and F-measure. More broadly, we want to shed light on how social aspects of software development relate to the technical knowledge involved in the tasks.

II. Related Work

We split the related work into issue labeling and social metrics in prediction models.

a) Labeling issues: Labeling issues is a relevant strategy to assist newcomers in finding a suitable issue to start [20]. It is not surprising that several studies have proposed ways to automatically label issues [7], [9], [11]–[13], [21]. Zhou et al. [13] used two-stage processing and a Naive Bayes (NB) classifier to predict the priorities of bug reports. Kallis et al. [9] mined the textual description of the issues to classify the issues into bug, feature, or question. Xia et al. [12] mined text data from information sites and tag questions using preexisting labels. Recently, Santos et al. [15] explored labeling the issues with API domains—high-level categories of APIs declared in the source code (e.g., "User Interface" (UI), "Machine Learning" (ML), "Test", etc.). Santos et al. conducted a single-project case study to investigate the feasibility of automatically labeling issues with API domains to facilitate contributors' task selection [15]. That study showed promising results, with precision, recall, and F-measure of 0.755, 0.747, and 0.751, respectively. The authors also analyzed how developers perceive the API-domain labels. The study with developers showed that API-domain labels are helpful in choosing tasks—developers considered these labels more often than the existing project labels and other pieces of information. In this study, we replicate Santos et al.'s [15] work by adding social metrics to the prediction models to potentially improve the performance of the classifiers for issues in which social interaction occurs (i.e., developers commenting on the issue).

b) Using social metrics in prediction models: Social metrics have been the object of software engineering research in different contexts. In previous work, researchers mined data from discussions in issue trackers to predict co-changes [17], categorized studies on how social networks impact software quality [22], and explained software characteristics based on the organization structure [23]. For example, Wiese et al. [17] mined social data to build a model to predict co-changes of source code files, obtaining low rates of false negatives and false positives and accuracy from 0.89 to 1.00. The work of Kikas et al. [24] predicted issue lifetime based on social metrics, such as the number of comments and actors. They mined data from 4000 projects and found that social metrics complement the textual description of titles and bodies. Differently from these studies, we propose to enrich the set of predictors used in Santos et al.'s [15] study to improve label prediction.

III. Definition of Social Metrics and Hypotheses

Following Wiese et al.'s [17] work, which leveraged social metrics from systematic literature reviews [25]–[27], we organized social metrics into three categories: Communication Context, Developer's Role in Communication, and Network Properties. In this section, we discuss the metrics we used in the paper. We bring more details on how we operationalized these metrics later in Section IV-A3.

A. Communication context

The communication context category involves aspects from the discussion around issues. For example, Meneely et al. [19] used metrics from historical communication contexts to predict software failures. In our study, following Wiese et al.'s work [17], the communication context variables include the number of issue comments, the number of commenters, and wordiness. The intuition behind this idea is that comments, participants, and words might be correlated with the nature of the tasks, which can be a proxy for some API domains. For example, issues that are more discussed or involve more people in an approval process might be related to some complex aspect of the software and, therefore, point to some architectural component or domain [28]. The opposite may also be considered: issues with few and short discussions may point to components that use API domains that usually require less explanation and discussion. We collected the communication context by issue (see Section IV-A3).

B. Developer's role in communication

The developer's role in communication category includes the betweenness centrality and closeness centrality metrics, which we abbreviate as "betweenness" and "closeness." These metrics aim to capture the developers' roles in communication [29]. Developers involved in a discussion have different values of betweenness and closeness, and some have a more central role while others have a more peripheral one [30]. The participation of central developers in discussing an issue may also be a proxy for the skill or relevance of the issue and may correlate with certain API domains.

Betweenness, as expounded by sociologist Linton Freeman, is defined as the sum of a point's "partial betweenness values for all unordered pairs of points where $i \neq j \neq k$", where "partial betweenness" is defined as "the probability that point p_k falls on a randomly selected geodesic linking p_i with p_j" [31], [32]. In short, it is the "[n]umber of times that a node acts as a bridge along the shortest path" between two other nodes in a network [17]. Linton provides the formula

$$C_B(p_k) = \sum_{i<j}^{n}\sum^{n} b_{ij}(p_k) \tag{i}$$

where $b_{ij}(p_k)$, the partial betweenness of point p_k between two nodes, is given by the formula

$$b_{ij}(p_k) = \frac{g_{ij}(p_k)}{g_{ij}}$$

where $g_{ij}(p_k)$ is "the number of geodesics linking p_i and p_j that contain p_k" and g_{ij} is "the number of geodesics linking p_i and p_j".

Closeness refers to "the number of steps required to access every other vertex from a given vertex" [33] and is given by the formula

$$C_c'(p_k) = \frac{n-1}{\sum_{i=1}^{n} d(p_i, p_k)} \tag{iii}$$

where $d(p_i, p_k)$ is "the number of edges in the geodesic linking p_i and p_k", and $\sum_{i=1}^{n} d(p_i, p_k)$ is the sum of "the geodesic distances from that point to all other points in the graph" [32].

Obtaining the metrics for developers' roles relies upon a matrix to represent the discussion in periods, as discussed in Subsection IV-A3.

C. Communication Network properties

Finally, the communication network properties category comprises communication graph metrics like the number of nodes, edges, diameter, and density obtained upon the communication network. The intuition behind using these metrics is aligned with the Task-Level view of Conway's Law, which conceptualized a unit of work as one that developers are engaged with, such as a task in the issue-tracking system [23]. In this case, the communication structure of an issue may map to the solution.

Mathematician and graph theory scholar Frank Harary [34] that "[t]he diameter $d(G)$ of a connected graph G is the length of any longest geodesic" [34, p. 14]. The formula, given by West [35, p. 71], is

$$max_{u,v \in V(G)} d(u,v) \tag{iv}$$

where $d(u,v)$ is the distance between two vertices in G.

The density of a simple, directed graph G is "calculated as the percentage of the existing connections to all possible connections in the network" [17], a ratio with an antecedent of the number of existing edges and a consequent of the total possible edges. Diestel gives the formula [36, p. 164]:

$$\|G\| \Big/ \binom{|G|}{2} \tag{v}$$

where $\|G\|$ and $|G|$ are the number of edges and vertices in the graph, respectively.

D. Hypotheses

Based on the categories of metrics presented here, we formulate the following hypotheses to complement our research questions and guide our study:

H_0^1**.** There is no difference in adding communication context metrics to predict API-domain labels.

Rationale: The communication context encapsulates information about the issue. Commenters with specific skills are more likely to participate in discussions where their skills are required. API domains may correlate with specific skills and knowledge.

H_0^2**.** There is no difference in adding the developer's role in communication metrics to predict API-domain labels.

Rationale: The role of each commenter within a social network constructed from the comments on the issue may reflect how much knowledge is needed to complete a task. API domains may correlate with the complexity and relevance of the issues.

H_0^3**.** There is no difference in adding communication network metrics to predict API-domain labels.

Rationale: The communication network architecture correlates with the software architecture [23], which in turn may correlate with the presence of API domains that are required to work on an issue.

IV. Method

We replicated Santos et al. [15]'s work according to the following dimensions. We followed the same operationalization as Santos et al. whenever possible with some slight adjustments to support various projects as discussed in this section. In terms of population, we included Santos et al.'s [15] case-study project (JabRef) in our study to allow direct comparison and added two other projects with different characteristics (Audacity and PowerToys) (see Section IV-A). In terms of protocol, to allow a fair comparison, we reproduced Santos et al.'s study as close as possible, adding only the calculation of the social metrics and the comparison of the results with and without it to the protocol. We used the same performance metrics, statistical tests, and thresholds as the original study. Finally, in terms of experimenters, the study was run by a slightly different experimenter team, with some intersection between the authors of both studies.

As we replicated the work from Santos et al. [15], we partially reproduced phases 1, 2, and 3 from that study. We followed the method presented in Figure 1. In phases 1 and 2, we collected and filtered closed issues and pull requests from the project repositories. As we needed the source code that solved each issue to identify the APIs and train the classifiers, we filtered out all the issues not linked to pull requests (Section IV-A). Then, we parsed the source code changed by each pull request, looking for APIs used in each artifact. We parsed the declarations based on the language of the projects. For JabRef, which was written in Java, we looked for "import" statements; for Audacity, written in C++, we parsed "include" statements; and for PowerToys, written in C#, we looked for "using" statements. Finally, we categorized all the APIs using the process described in Section IV-B.

In phase 3, we built the corpus using TF-IDF as our baseline, following the study by Santos et al. [15]. Finally, we split the dataset into training and testing sets, using ten cross-validations. Following Santos et al. [15], we employed MLS SMOTE to improve the rare labels in the imbalanced dataset (Section IV-D) and employed the Random Forest classifier to predict the API domains (Section IV-E).

In phase 4, we created the dataset with the social metrics. We mined the conversation data from the issues, as defined in Section III and illustrated in Section IV-A3. Next, we investigated how each metric impacted the predictions and ran the Random Forest classifier with the newly gathered data. Finally, we analyzed the performance considering groups of predictors divided into hypotheses (Section IV-F).

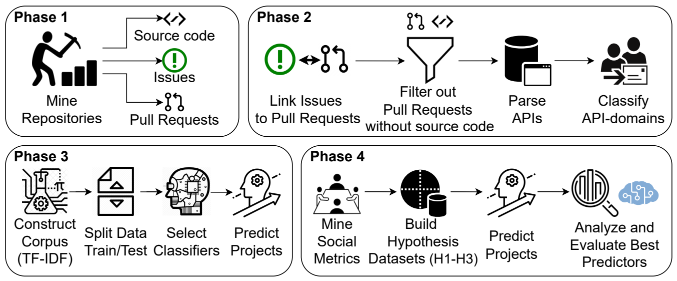

Fig. 1: Method Overview.

To answer RQ1 (To what extent can social metrics improve the prediction of API-domain labels?), we predict the API-domain labels with and without the social metrics to find the best precision, recall, and F-measure outcomes. We built datasets with the best social predictors ranked by feature importance.

We sought to predict the API-domain labels by building the datasets (Section IV-A) to evaluate the defined hypotheses (Section III-D). Therefore, we included in the dataset the social network and communication features related to H_0^1 - communication context; H_0^2 - developer's role in communication; and H_0^3 - communication network properties. Finally, we ran models configured with social metrics and one with Santos et al.'s [15] features (baseline) and compared them using the statistical tests mentioned in Section IV-F.

To answer RQ2 (To what extent we can transfer learning among projects using social metrics to predict the API-domain labels?), we ran a transfer learning experiment from the project that obtained the best results for the others to verify how transferable the predictions are. The projects' metrics (Section IV-F) were compared with the baseline.

A. Mining OSS Repositories

The mining stage was composed of the project selection, mining the issues and pull requests, and mining the social metrics.

1) Project Selection: We selected projects with different programming languages, high levels of popularity (at least 6K GitHub stars), and diverse goals. We also sought active projects with recently posted issues and solved pull requests (average monthly number of created issues and updated PRs > 5). Therefore, besides JabRef (from Santos et al.'s [15] work), we mined two other OSS projects: Audacity and PowerToys. We limited the number of projects to three since our study involved hiring experts to expand the API-domain labels adopted in the previous work [15].

The first project we mined was JabRef [37], an open-source bibliography reference manager. In Jan 2023, the project had 2.9k stars, 3.4k closed issues, 5.7k closed pull requests (PR), 17.7k commits, 37 releases, 2k forks, and 492 contributors.

Next, we mined Audacity and PowerToys. Audacity is an audio editor written in C++, which had 8.7k stars, 1.5k closed issues, 1.6k closed PRs, 16.5K commits, 31 releases, 2K forks, and 166 contributors. PowerToys offers a set of utilities for Windows and is written in C# and had 84.7k stars, 15k closed issues, 4.1k closed PRs, 6.4k commits, 73 releases, 4.9k forks, and 173 contributors.

2) Mining Issues and Pull Requests: We collected issues and PRs from the projects using the GitHub API. Our dataset includes title, description (body), comments, and submission date. We also collected the name of the files changed in each PR and the commit message associated with each commit (Figure 2 - A and B).

To train the model, we kept only the data from issues linked with merged and closed pull requests since we needed to map issue data to source code APIs through the files changed (Figure 2 - C and D). We searched for the symbol # followed by an issue number (a set of numeric characters) in the pull

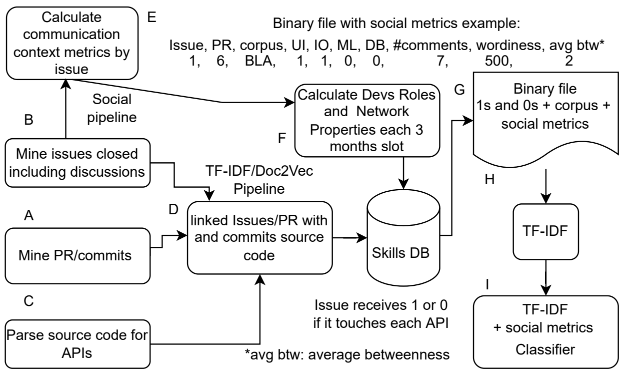
Calculate communication context metrics by issue
E
Binary file with social metrics example:
Issue, PR, corpus, UI, IO, ML, DB, #comments, wordiness, avg btw*
1, 6, BLA, 1, 1, 0, 0, 7, 500, 2
Social pipeline
B
Calculate Devs Roles and Network Properties each 3 months slot
F
G
Binary file 1s and 0s + corpus + social metrics
Mine issues closed including discussions
TF-IDF/Doc2Vec Pipeline
H
A
D
linked Issues/PR with and commits source code
Skills DB
TF-IDF
Mine PR/commits
C
Issue receives 1 or 0 if it touches each API
I
TF-IDF + social metrics Classifier
Parse source code for APIs
*avg btw: average betweenness

Fig. 2: Processing pipeline.

request title and body to identify the link between issues and PRs. We also filtered out issues linked to PRs without at least one source code file (e.g., those associated only with documentation or config files) since they do not provide the model with content related to any API.

3) Mining Social Metrics: For mining communication data, we built an extractor toll[1], which uses the PyGithub[2] library for interacting with the GitHub REST API v3 from Python and mine data from new endpoints. We also used igraph[3] and NetworkX[4] libraries for deriving the various social metrics.

All items in the **communication context** category refer to quantities counted by issues. The number of issue comments was gathered directly from the GitHub REST API. Issue wordiness was measured by finding how many words there are with a length greater than two in the aggregated text of the issue body and all issue comment bodies for an issue, following the example of Wiese et al. [17]. For issue commenters, we are interested in the quantity, which is unique. To find this value, we created a set of all commenter usernames, including the username of the issue author.

Drawing upon previous work [24], [38], we employ temporal periods and dynamic features, such as the evolving number of comments on an issue, to improve prediction models. For that purpose, we created directed, weighted graphs that map issue discussants and their conversations for all issues in a temporal period. A "temporal period" is an interval in which we partitioned the repository's history. For example, suppose the repository has existed for one year, and the chosen interval is three months. In that case, we separate the issues into four distinct temporal periods. The work from Kikas et al. [24] uses three months periods (slots) to measure the number of issues created and closed and commits created by contributors and by projects. In contrast, Wiese et al. [30] preferred six-month periods to predict co-changes. We have chosen three months as the duration for each temporal window to capture a period

[1]https://doi.org/10.5281/zenodo.7740450
[2]https://github.com/PyGithub/PyGithub
[3]https://github.com/igraph/python-igraph
[4]https://github.com/networkx/networkx

similar to the issue life cycle (90% of issues get closed in 96.4 days [24]).

For creating the communication network graphs, we used the same rules that Wiese et al. [30] used: nodes are discussants in an issue's conversation, edges are responses from one discussant to another, nodes and edges are added independently, and edges have weights representing the number of responses given. The original poster of the issue is not considered to have responded to anyone by creating the issue, and any response in the lifetime of the issue is considered a response to all prior discussants. We also disallow nodes from having self-edges. Like in previous work, the order or "directions" of discussants in a conversation is crucial because it affects the resulting metrics derived from a graph, such as betweenness and closeness.

We placed each issue in the repository's history into a temporal period. Then, we visited and processed each period of issues into a matrix. In the resulting adjacency matrices, each row represents a unique discussant, and each column represents the number of responses by that discussant to the discussant whose row index corresponds to the column index. To illustrate, let issue 1 be one which discussant A opened and to which discussant B responded once. Let issue 2 be in the same period as issue 1. Assume that B opened the issue, A responded twice to B, C responded once after (meaning that they responded to both A and B), and B responded last (meaning that they responded to both A and C).

$$Issue_1 = \begin{array}{cc} \begin{array}{cc} A & B \end{array} & \\ \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & \begin{array}{c} A \\ B \end{array} \end{array}$$

$$Issue_2 = \begin{array}{cc} \begin{array}{ccc} B & A & C \end{array} & \\ \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} & \begin{array}{c} B \\ A \\ C \end{array} \end{array}$$

After all issues in a period have resulting matrices, we used a method adapted from Yu et al. [39] to create a period matrix representing the total communication that transpired in all issues for the period combining all of the period's issue matrices. The resulting period matrix contains all discussants found in the issues for that period and aggregates the number of responses between them. In this scheme, each matrix corresponds to one issue communication model, while the combination matrix summarizes the communication model for the designated period.

To conduct this process, each issue matrix is visited, and the contents are merged with those of the period matrix. Like with the issue matrices, nodes representing discussants are only added if they are not already present. When merging edges from an issue matrix, an edge is added if it is not found. Otherwise, the weight of the edge, a whole number, is incremented by the weight found for that edge in the issue matrix. In the implementation, nodes are added by checking if the period matrix contains the unique identifier for a row/discussant. After all the discussants are integrated, all row contents, i.e., the edge weights indicating the number of responses between two developers, are integrated. Assuming that issues 1 and 2 are the only issues in the period, the resulting period matrix and the directed graph would be:

$$Issue_{1+2} = \begin{array}{cc} \begin{array}{ccc} A & B & C \end{array} & \\ \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} & \begin{array}{c} A \\ B \\ C \end{array} \end{array}$$

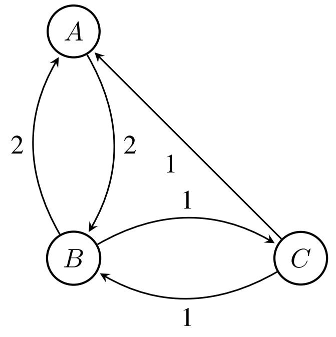

Fig. 3: The communication history example (issue 1+2)

Using igraph and NetworkX libraries, we produced corresponding graphs of these period matrices and retrieved metrics of interest from available function calls (Figure 2 - E-F).

The **developers' roles**, which include two centrality measures (betweenness and closeness) were collected by temporal period, and we computed the average, sum, and max value [30]. For each period, once we had the metrics computed, we applied those for each issue within the period, but only averaging, summing, or computing the max value with the individual betweenness and closeness for the discussants present on that issue.

The **communication network properties** were computed directly by counting the number of nodes and edges, in addition to the graph's diameter and density based on the matrix created by period or by issues. The size of a network is the number of nodes (unique discussants) in it. When it is computed by period, nodes are distinct from "issue discussants" communication context metric. The number of edges reflected the number of unique responses in the network for the given time interval—essentially, the sum of all weights in the aggregated adjacency matrix for a period and, thus, it is different from "issue comments". In this study, we opted to model the network properties by issues, and we removed the number of nodes from the model since it has the same value as unique discussants.

B. *Categorization of APIs*

We grouped the APIs into high-level categories since labeling the issues with dozens of APIs can harm the user interface and cause information overload. The categories provided ground truth to the supervised machine learning model. The categories were related to APIs imported in files that were changed in closed pull requests linked to closed issues. Santos et al. [15] manually categorized and customized the APIs for a single project (JabRef); therefore, it would not necessarily fit other projects. To make the set more generalizable, we recruited three experts—one specialized in each of the three main programming languages adopted in the projects (Expert1: 25 Years of Java; Expert2: 22 Years of Java and 10 of C++; Expert3: 18 years of Java and 12 Years of C#)—offering US$ 25.00 gift cards as a token of appreciation.

The categorization started with proposals from the experts to define generic API domain categories to encompass a broad range of projects (e.g., "UI", "IO", "DB", "ML", etc.). A card-sorting approach addressed disagreements via discussions until a consensus was reached. After four rounds (8 hours) of discussions, the experts reached an agreement and defined 31 API domains.

Next, we parsed the source code files to retrieve the libraries declared in "import" (java), "include" (C++), and "using" (C#) statements (Figure 2 - C). The intuition behind the API classification method is that libraries' namespaces often reveal architectural information and, consequently, their API domains [40], [41]. To identify the API domains for each library, we split all the API namespaces into packages. For instance, the API "com.oracle.xml .odbc.XMLDatabase" derived "com", "oracle", "xml", "odbc", and "XMLDatabase". Next, we eliminated the business domain name extensions (e.g., "org", "com"), country code top-level domain ("au", "uk", etc.), the project and company names ("microsoft", "google", "facebook", etc.). In the example, we kept the first package "xml", second package "odbc", and the class name: "XMLDatabase". The class name XMLDatabase was split into two words using the Python library `wordninja` [5], which employs word frequency and can identify concatenated words in texts. Therefore "XMLDatabase" was split into "XML" and "Database."

To facilitate the expert's work, for each package and class name, we identified how similar they were to the proposed API domains using an NLP Python package `spacy` [6]. Spacy is a multi-use NLP package that uses a cosine similarity function to compare the average of the word vectors to find similarities between sentences and tokens and can be trained with a software engineering vocabulary.

Next, the experts leveraged the NLP suggestions to classify the APIs. A card-sorting approach addressed any disagreement between the experts. We created lists. The first list contained all the aggregated first packages. The second list contained the aggregated second packages and so on. In addition to the package name, the list showed the NLP suggestions with the confidence levels (i.e., XML: IO=0.85; Lang=0.7; util=0.61). The experts evaluated the lists and accepted one suggestion or rejected all. In case of rejection, they evaluated the entire namespace of the package/class. By aggregating per package, we reduced the number of libraries evaluated by the experts. In the JabRef project, where the experts had to evaluate 1,692 libraries manually, aggregating it by the first and second packages resulted in only 137 and 45 packages, respectively. Two lists were sufficient to evaluate most libraries. Indeed, the volume of evaluations dropped significantly (to 10.8% in JabRef, 21.6% in PowerToys, and 45.1% in Audacity). In case of evaluation discrepancy between the first and second packages, the second package evaluation prevailed since it is less generic. For example, in the case above, the package "xml" may be evaluated as "IO" and "odbc" as "DB". When the experts accept both, "DB" prevails.

[5]https://pypi.org/project/wordninja/
[6]https://spacy.io/api/doc#similarity

C. Dataset Setup

Our dataset has one row for each issue, and the columns are composed of "1s" and "0s" for each API-domain column, indicating whether an issue has that API in the source code changed in the associated PR. The dataset also contains the corpus column with the concatenated text from the title, body, and comments on the issue.

The calculated metrics filled the columns of each issue. For example, suppose issue #1 was closed by PR #6, which had seven issue comments and 500 words, the average betweenness of the developers present in the discussant in that period was two. In addition, the files changed by PR #6 included libraries categorized as "UI" and "IO" by the experts, and the issue and PR text mined was "bib file does not load." Suppose the possible API domains are: "UI," "IO," "ML," and "DB." The dataset row contained the following values: issue #: 1, pr_linked number #: 6, corpus: "bib file does not load", #comments: 7, #wordiness: 500, avg betweenness: 2, UI: 1, IO: 1, ML: 0, DB: 0 (Figure 2 - G). The same logic applies to the remaining API-domain labels (31 possible) and metrics (described in Section III).

To transform the corpus into a feature set, we followed some studies [42]–[44] that applied TF-IDF—a technique for quantifying word importance in documents by assigning a weight to each word. To compute the TF-IDF weights, we followed the cleaning process used in the original work [15], which includes stemming and stop word removal. We also converted each word to lowercase and removed URLs, source code, numbers, templates, and punctuation. After applying TF-IDF, we obtained a vector of TF-IDF scores for each issue's word. The vector length is the number of terms used to calculate the TF-IDF, and each term was stored in a column with the TF-IDF weight (Figure 2 - H).

Given the recent popularity of different ways to represent documents for prediction models, we decided to evaluate the API-domain label prediction by replacing TF-IDF with Doc2Vec. Doc2Vec is a document representation method that translates text into a vector of features. Like TF-IDF, Doc2Vec was used in many studies where a corpus must be transformed into features such as document classification or sentiment analysis [45], [46]. Using the best setup found for RQ1, Doc2Vec was outperformed by TF-IDF (p=0.03 precision, p=0.00018 recall, and p=0.003 F-measure), and we kept TF-IDF only in the analysis. Despite being a classic NLP technique, TF-IDF usually overcomes newer techniques like Word2vec or Doc2Vec when the corpus is small and unstructured. Doc2Vec cannot identify the semantic and syntactic information of the words [47] in these situations. Many issues have small titles and unstructured or semi-structured bodies and comments. After removing templates (since their repetitive structure introduced noise and were not consistently used among the issues), code snippets, and URLs, we may reduce the number of words and lose the semantics and syntax in sentences.

D. Training and testing sets

To avoid overfitting, we ran each experiment ten times, using ten different training and test sets to match 10-fold cross-validation [48]. We used ShuffleSplit provided by `scikit-multilearn` package [49], a model selection technique that performs cross-validation for multi-label classifiers. To improve the dataset's balance, we used the SMOTE algorithm for the multi-label approach [50]. In the baseline study [15], SMOTE improved the F-measure by 6%.

E. Classifier

An issue may require the use of multiple APIs. Thus, we applied a multi-label classification approach, which has been used in software engineering for many purposes, such as classifying questions in Stack Overflow (e.g., [12]) and detecting types of failures (e.g., [51]) and code smells (e.g., [52]).

We used the Random Forest (RF) algorithm (Python `sklearn` package) since it had the best results in the baseline study [15]. We kept the following parameters: $criterion = 'entropy'$,$max\_depth = 50$, $min\_samples\_leaf = 1$, $min\_samples\_split = 3$,$n\_estimators = 50$.RF has been shown to yield good prediction results in software engineering studies [53]–[56] (Figure 2 - I).

F. Data Analysis

Our analysis started verifying the amount of issues each dataset has with num_comments > 3 and num_discussants > 2, since we need some level of social interaction to prospect relevant social metrics. We also tested for different thresholds (as presented in the results section). After computing all the social network metrics, the features' influence was compared to derive the best predictors. We used the feature_importances_ function present in the package `sklearn` to rank the features [49].

To evaluate the classifiers, we employed the following metrics [49]:

- **Precision** measures the proportion between the number of correctly predicted labels and the total number of predicted labels.
- **Recall** measures the percentage of correctly predicted labels among all correct labels.
- **F-measure** calculates the harmonic mean of precision and recall. F-measure is a weighted measure of how many correct labels are predicted and how many of the predicted labels are correct.

Since we are computing the metrics for a multi-label problem, we average the metrics above for the set of predicted labels regarding the ground truth. The metrics for each label can be calculated using different averaging strategies, for instance, the macro or micro [57]. The macro average is the arithmetic mean of all the per-label metrics. In contrast, the micro average, adopted by Santos et al. [15], is the global average metric obtained by summing TP, FN, and FP. We employed the micro to follow the baseline.

We ran statistical tests to verify whether the observed differences between groups of variables were statistically significant. We employed the Mann-Whitney U test to compare the classifier metrics, followed by Cliff's delta effect size test. The Cliff's delta magnitude was assessed using the thresholds provided by Romano et al. [58], i.e., $|d| < 0.147$ "negligible," $|d| < 0.33$ "small," $|d| < 0.474$ "medium," otherwise "large."

G. Data Availability

The source code and the data generated from our research are publicly available in a repository to help reproducibility[7].

V. Results

This section presents the results by research questions.

A. RQ1. To what extent can social metrics improve the prediction of API-domain labels?

TABLE I: Comparison - baseline X social metrics

Project/ Metric	**Audacity**		**JabRef**		**PowerToys**	
	Baseline	Social	Baseline	Social	Baseline	Social
Precision	**0.916**	0.897	**0.842**	0.812	**0.796**	0.788
Recall	0.884	**0.906**	**0.835**	0.812	**0.842**	0.833
F-Measure	0.899	**0.901**	**0.838**	0.811	**0.818**	0.809

When looking at all the data together, the results seemed not promising. We compared the baseline (dataset without the social metrics) and the dataset with the social metrics. As can be seen in Table I, only Audacity overperformed the baseline in recall and F-measure with a small difference.

Next, we went deeper and verified the average number of comments and, number of discussants, wordiness per issue, among others (Table II), as discussed in the method. Filtering Audacity and JabRef by num_comments > 3 (issues with at least some level of social interaction), our dataset was reduced to only 16 and 20 rows, respectively, while in PowerToys resulted in 207. The same occurred when filtering by wordiness > 300: while for Audacity and Jabref, we kept only 11 and 3 rows, respectively, for PowerToys, we kept 196. For num_discussants > 2 JabRef kept 20 rows, Audacity 18, and PowerToys 219. Since the social metrics require some developers to debate the issues, we continued the study with only the PowerToys dataset (PowerToys num_comments, wordiness, num_discussants, betweenness_avg, and betweenness_sum have normal distributions—Shapiro-Wilk test with p=1.83, 9.43, 3.45, 2.94, 3.04, respectively).

To understand the impact of each predictor in the predictions, we evaluated the model only with the social metrics (without the TF-IDF weights). We used the method "feature_importances_" from the package `sklearn.ensemble` [49] to rank the importance of each metric. The three most important predictors

[7]https://doi.org/10.5281/zenodo.7740450

TABLE II: Averages from social metrics in the studied datasets

Metric AVG/Project	Audacity	JabRef	PowerToys
Comments	1.41	1.73	**3.83**
Discussants	1.58	1.47	**2.23**
Wordiness	81.69	157.88	**297.91**
Edges	4.60	10.73	**31.38**
Density	0.73	0.58	**1.52**
Diameter	0.43	0.56	**0.70**
Betweenness avg	1077.72	2042.21	**208451.44**
Betweenness max	1418.27	2648.70	**350726.26**
Betweenness sum	1684.45	3304.60	**418619.28**
Closeness avg	0.56	**0.64**	0.56
Closeness max	0.60	**0.68**	0.61
Closeness sum	0.92	1.15	**1.25**

TABLE III: Feature Importances by Project

Feature	PowerToys	JabRef	Audacity
num_discussants	0.02908	0.04451	0.04451
num_comments	0.05488	0.03591	0.03591
wordiness	**0.26509**	**0.14283**	**0.57612**
edges	0.04970	0.03200	0.03650
diameter	0.03479	0.03462	0.01299
density	0.06147	0.02540	0.01758
betweenness_avg	0.08943	**0.12441**	0.04366
betweenness_max	0.07232	0.11745	0.04077
betweenness_sum	**0.09793**	**0.11788**	0.05049
closeness_avg	0.08836	0.11094	**0.06220**
closeness_max	0.06718	0.11440	0.04495
closeness_sum	**0.08977**	0.09964	**0.06166**

in the PowerToys project are wordiness (W), betweenness_sum (BS), and closeness_sum (CS). They also appear in the top three rank for the other projects (Table III).

Next, we filtered out the issues based on the three best PowerToys predictors (Table IV). We tested three different filters: wordiness > {300, 500, 700}, betweennness_sum > {450K, 600K, 800K, 1000K}, and closeness_sum > {1.5, 1.75, 2.0, 2.25}. We compared the baseline ("Non-Social") and the predictions without filtering ("Social"). Using the PowerToys dataset, we start from a threshold close to the average of the three mentioned metrics (Tables IV and V and Figure 4), increasing the value and filtering to a point where it produced a precision drop.

TABLE IV: Comparison - social metrics filtering - PowerToys

Filter	Number of issues	Precision	Recall	F-Measure
W300	196	0.904	0.938	0.920
W500	99	**0.922**	0.962	0.941
W700	56	0.909	**0.978**	**0.942**
BS450K	292	0.823	0.878	0.849
BS600K	243	0.834	0.895	0.863
BS800K	177	0.832	0.854	0.842
BS1000K	146	0.807	0.895	0.847
CS1.50	216	0.856	0.905	0.879
CS1.75	142	0.867	0.936	0.899
CS2.00	106	0.908	0.917	0.911
CS2.25	80	0.905	0.949	0.926

Table IV and Figure 4 show the results after filtering by wordiness, betweennness_sum, and betweennness_sum. Filtering with one feature, we obtained the best precision with wordiness > 500 (W500). The best recall and F-Measure were reached with wordiness > 700 (W700). Filtering with a combination of best features (e.g., W700+BS600K+CS2.00) reduced the dataset to a few rows, and, therefore, we discarded the results. We tested different settings with two and three filters, and we could not find better results.

Precision and F-measure tend to fall for PowerToys when the dataset reaches the lower bound (roughly below 100 issues), except when the dataset reaches a point where the filters reduce the feature space and ease the work of the machine-learning algorithm. We reject the hypothesis that the distributions are the same in precision and F-measure, using the Mann-Whitney test comparing Social (dataset without filters) x W500, W500 x BS600K, and only in F-measure for W500 x CS2.00, and W500 x W300 (Table V).

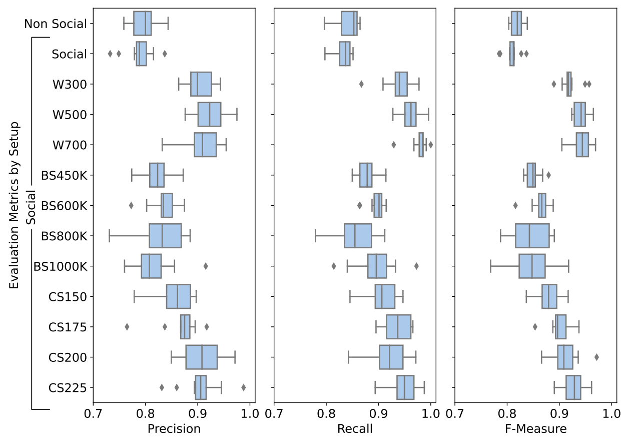

Fig. 4: Social metrics setups comparison: TF-IDF - PowerToys

TABLE V: Cliff's Delta for F-Measure and Precision: comparison of corpus models by setup - TF-IDF.

Corpus Comparison	Cliff's delta			
	Precision		F-measure	
Non Social x Social	0.18	'small'	0.28	'small'
Social: W300 x W500	-0.38	'medium'	-0.68	'large' *
Social: W500 x W700	0.16	'small'	-0.08	'negligible'
Social: BS450K x BS600K	-0.28	'small'	-0.5	'large'
Social: BS600K x BS800K	0.0	'negligible'	0.16	'small'
Social: BS800K x BS1000K	0.3	'small'	-0.12	'negligible'
Social: CS1.50 x CS1.75	-0.22	'small'	-0.48	'large'
Social: CS1.75 x CS2.00	-0.5	'large'	-0.22	'small'
Social: CS2.00 x CS2.25	-0.02	'negligible'	-0.36	'medium'
Social x W500	-1.0	'large' ***	-1.0	'large' ***
Social: W500 x BS600K	1.0	'large' ***	1.0	'large' ***
Social: W500 x CS200	0.23	'small'	0.64	'large' *

** $p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$*

Next, we compared the different hypotheses in predicting the API-domain labels using the PowerToys project.

Using the filter wordiness > 500 we predicted the API-domain labels using sets of social metrics regarding the defined hypotheses (Section III-D). Looking at Figure 5, we can see the results of the model created with all social metrics (and no filtering) had different distributions when compared to the model with all social features (Social x W500 – p=0.00018 for precision and F-measure) with large effect size. When comparing the model with all social metrics (W500) and the models based on H_0^1(H1W500), H_0^2, and H_0^3 had no statistical difference.Observing Table VI, we can see that all models created based on the hypotheses followed similar distributions. H_0^2 performed slightly better on precision, recall, and F-measure. H_0^2 model with six features conveyed almost the same result as the model with all features (Table V), thus should be preferred when the dataset is considerably large.

TABLE VI: Comparison of Hypotheses.

Hypotheses	Precision	Recall	F-Measure
H_0^1	0.913	0.957	0.934
H_0^2	**0.920**	**0.962**	**0.940**
H_0^3	0.909	0.961	0.934

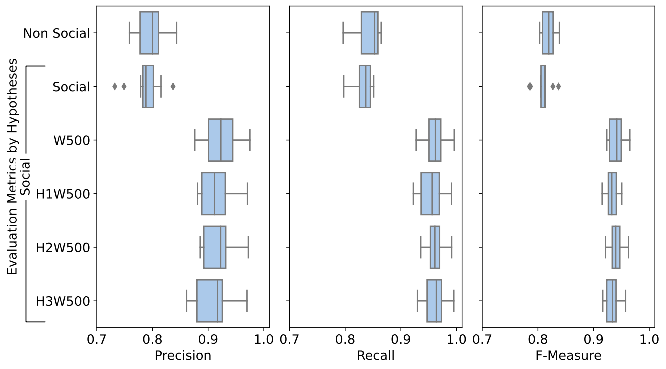

Fig. 5: Social metrics hypotheses setups comparison: PowerToys Project

RQ1 Summary. By using all features, it is possible to predict the API-domain labels with the social metrics and TF-IDF, filtering the dataset when wordiness > 500, with precision = 0.922, recall = 0.962, and F-measure = 0.941.

B. RQ2. To what extent can we transfer learning among projects using social metrics to predict the API-domain labels?

To answer RQ2, we verified whether it is possible to transfer learning from PowerToys to Audacity and JabRef. Transferring from PowerToys to Audacity and JabRef required us to normalize the labels in common for each pair of projects since we cannot predict for Audacity/JabRef what we cannot find in the PowerToys dataset.

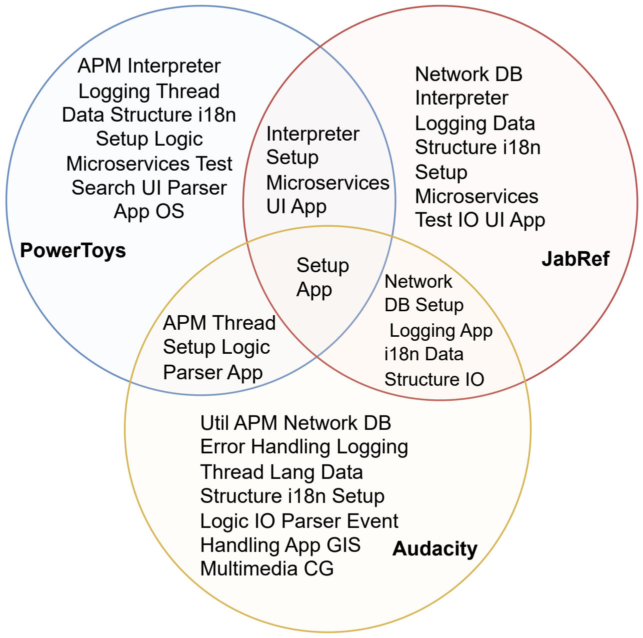
APM Interpreter Logging Thread Data Structure i18n Setup Logic Microservices Test Search UI Parser App OS
PowerToys
Interpreter Setup Microservices UI App
Network DB Interpreter Logging Data Structure i18n Setup Microservices Test IO UI App
JabRef
Setup App
Network DB Setup Logging App i18n Data Structure IO
APM Thread Setup Logic Parser App
Util APM Network DB Error Handling Logging Thread Lang Data Structure i18n Setup Logic IO Parser Event Handling App GIS Multimedia CG
Audacity

Fig. 6: Labels intersections from the studied projects.

The training dataset was filtered with wordiness > 500, repeating the best configuration found for RQ1. This makes some labels disappear from the training dataset. For example, although "i18n" is present in PowerToys and JabRef projects, no issue in the dataset prevails after filtering (Figure: 6).

Table VII shows the result of the transfer learning. The metrics are far behind the ones obtained by training and testing with the same dataset (Table VII).

TABLE VII: Comparison of corpus models with transfer learning.

Training	Test	Precision	Recall	F-Measure
PowerToys	Audacity	0.392	0.434	0.412
PowerToys	JabRef	0.328	0.643	0.434

RQ2 Summary. Transfer learning performed poorly and resulted in precision = 0.392, recall = 0.643, and F-measure = 0.434.

VI. Discussion

Do Social Metrics Matter?

Our results showed the social metrics are relevant to predict API-domain labels. The predictions using social metrics improved the precision of the models up to 15.82% and the F-measure up to 15.89% when filtering wordiness > 500). We could not observe an increase in the performance of the classifiers in the projects, with few comments and discussants actively participating in the tasks. The results are coherent with the intuition behind the research, as the predictions are based on the presence of social interaction.

We observed the increase of the relative counts in some labels after filtering—with emphasis on "Thread" from 0.02 to 0.12 and "APM" from 0.26 to 0.36. "Thread" issues have approximately six comments, 18 discussants, and 947 words on average, which is far above the PowerToys averages in Table II. "Thread" labels are present when the discussions are more protracted and might point to more specific characteristics of the issues (Table VIII).

To what extent does the model transfer learning?

Transfer learning is paramount when projects do not have enough data for training, not enough time to prepare the training

TABLE VIII: Labels distribution

Labels distribution	Label counts		Labels counts normalized	
Label names	Before filtering	After filtering	Before filtering	After filtering
Logging	6	1	0.008	0.01
Data Structure	13	1	0.01	0.01
Logic	14	7	0.01	0.07
Setup	337	49	0.46	0.49
Microservices	214	34	0.29	0.34
Test	6	1	0.008	0.01
App	502	75	0.69	0.75
Search	394	61	0.54	0.61
i18n	4	1	0.005	0.01
Parser	42	3	0.06	0.03
APM	201	36	0.26	0.36
UI	498	68	0.69	0.68
Thread	25	12	0.02	0.12
OS	521	75	0.71	0.75
Interpreter	22	4	0.02	0.04
rows	721	99		

dataset, the infrastructure is unavailable to develop their models, or the costs to run the models are prohibitive. Moreover, while we can access features from projects in the OSS communities to run API predictions, the same availability is not always possible in the industry. The source code may be restricted, and there is no way to prepare the ground truth with the APIs declared. Besides, the conversations are often protected by privacy laws. Therefore, despite the relevance of transfer learning, the results are far from the regular training and testing, and we should investigate ways to improve them. One possible reason is the diverse domains of the projects that predicted different API-domain labels.

How can software practitioners benefit from the results?

This approach can benefit developers who want to find an appropriate issue, which is the case of newcomers who have a hard time searching for a task [59]. Once the maintainers foment the communication in the projects' issue tracker, the discussions around issues, which hold the project skills, may be used to train our model to predict the API-domain labels and, thus, assist newcomers in picking issues and helping them learn about the tasks. Indeed, communication is a recipe for newcomers to "Keep the community informed about decisions" [5]. Reporting the advances on a task should attract comments from core members in charge of the module and with the skillset related to the task. Steinmacher et al. [5] guide newcomers to "Do not be afraid of the community". Reporting problems in appropriate community channels may lead contributors with the required skills to join the discussion. The expansion of communication to produce better social metrics will assist indirectly in breaking barriers defined by Steinmacher et al. [5]. The communication strategy is second in primacy to maintainers [20] and meets multi-teaming research whose hints include improving communication to address the coordination weakness, lack of member stability, and hierarchy in the highly-dynamic OSS projects [60].

VII. Threats to Validity and Limitations

Categorizing the API domain is one of the threats to the validity of this study. We recognize that different individuals may create different categorizations, which may introduce bias in our results. To mitigate this issue, we recruited three experienced developers: one expert for each of the programming languages used by the projects. A semi-automated approach supported these experts. A limitation of this approach is that NLP suggestions might not work as expected in languages like C++, where the information about the API packages is not present in the "include" declaration. For researchers interested in extending our work for projects from these languages, we recommend using another source of information beyond the API namespace, such as comments and documentation, to achieve proper library categorization.

Another concern is the number of issues in our dataset and the link between issues and PRs. To add an issue to the dataset, we need to link it to its solution submitted via pull request. By linking them, we identify the APIs used to create the labels and define our ground truth using the changed files. We manually inspected a random sample of issues to verify that the data was collected correctly and reflected what was shown in the issue tracker interface. When an issue is closed without leaving a trace on the ITS, we cannot track it using this method, and therefore the issue is discarded. The need to have a solution to the issue also introduces a bias to the generability of our results. We only tested the predictions for issues that had a linked PR to be able to establish the ground truth. When using our approach in practice to label open issues, some issues may not be related to code solutions or may have different characteristics than those that have a linked PR and may receive incorrect labels, decreasing the performance observed in our study.

We used the presence of an API in the file changed by a PR to define the ground truth of the API domains. The API may not necessarily affect the lines of code changed by a PR. Future work can explore other ways to link issues and API domains, increasing the accuracy of the labeling.

To avoid overfitting, we used a cross-validation method to randomize the training and testing samples. Despite the feature importance test pointing us to some features as the best predictors, this was observed when we isolated the social metrics as features. When mixing all the features (Social metrics + TF-IDF weights), the feature importance among the social metrics decreased.

Generalization is also a limitation. Predictions may differ for different projects and programming languages. In addition to the JabRef case study used by Santos et al. [15], we mined two other projects, improving the diversity of projects and programming languages. However, different results can be achieved when other projects and programming languages are considered.

We produced a more general set of API domain labels than Santos et al. [15]. These categories can potentially be transferable to other projects using the same APIs across multiple domains. Many projects adopt a typical architecture and frameworks (Hibernate, Spring, etc.), which makes them reuse similar sets of APIs. Creating more generic API-domain labels should help to generalize to other projects because standard APIs and third-party libraries account for up to 53% and 35%, respectively, of the total APIs used in projects [61]. Therefore, the chances that most APIs in a project had been previously categorized increase as our work is extended to new APIs and projects.

VIII. Implications

Implications for Researchers. The implications are manyfold. (i) Our work shows the importance of using social metrics as prediction factors, which is aligned with the idea that software engineering is a sociotechnical activity. Many studies use prediction models based solely on technical attributes and miss the opportunity to improve the results with social metrics. We expect our results to inspire further investigation of social metrics in other contexts and applications. (ii) Our results illustrate the value of replicating previous work not only to confirm previous findings but to extend with other ideas to improve the state of the art. (iii) The scientific community can extend our approach to predict contributors' skills using a similar approach as we used for issues. The community can also build tools that use our approach to recommend tasks according to contributors' skills and career goals (e.g., as proposed in the literature [62]). Automatic matchmaking can use the historical contributions data, when available, to compare the skills of tasks and contributors. We published the software, scripts, and data we used to facilitate replication, use, and extension. (iv) We used just one source of information to build our social metrics. We hope our results will inspire the software engineering community to explore the multitude of tools developers use to collaborate and use/propose new metrics and communication channels as information sources to calculate the metrics. Many prediction models end up not being used in practice due to suboptimal performance. Exploring ways to improve state of the art is paramount to transferring the scientific results into practice.

We understand that further studies are necessary before adopting our approach in practice. Our study is an essential step toward improving the predictions and testing the approach in different projects. In the following, we discuss some implications assuming the practical adoption of the approach.

Implications to new contributors. The social metrics can potentially improve label prediction, which increases the chance of adopting our approach in practice to facilitate task selection. The literature shows that newcomers find labels in the issue tracker useful to help find their tasks [63]. Facilitating the choice of a task—and consequently the onboarding of newcomers—is crucial since the literature has shown that they face various barriers and often give up contributing [5].

Implication for project maintainers. Our approach can be used by maintainers to automatically label issues on the issue tracker system of their projects. OSS maintainers are known to be busy, and issues end up being poorly labeled due to the lack of time to label them manually. Providing an approach that can automatically label issues has the potential of supporting their job, making the issues easier to find and enabling one to shortlist the issues by filtering those that match their skills with the tasks. Ultimately, this can facilitate the onboarding and engagement of contributors, which is vital for the sustainability of the projects [64], [65].

Educators. Educators often recommend their students contribute to OSS projects or adopt this activity as a course assignment. However, the literature has shown that students struggle to find suitable tasks [5]. Improving the automatic labeling of issues is a step toward improving task selection, which is very important for these students. Our labels are related to API domains, which map to skills students may want to develop (e.g., ML, security, etc.) or are comfortable with.

IX. Future Work

Future work should focus on generalizing the results, including more projects with different levels of social interactions. Other metrics also should be explored, for example, the Structural Holes [66]. Although PowerToys had better metrics averages, we can note JabRef had some best metrics averages: AVG and MAX Closeness (Table II). PowerToys project has a betweenness SUM far superior from the other projects. The network property features in PowerToys also are superior, but they ranked low in feature importance. The network properties should be better investigated to determine levels where they increasingly predict the API-domain labels.

Future work can also investigate different temporal periods to aggregate issues to calculate the social metrics derived from the Social Network—in this work, we only tested with three months intervals, following previous work [24]. To improve transfer learning predictions, future work can investigate proxy techniques [67] used in the literature to predict software engineering defects to conform source and target datasets for transfer learning. Another exciting investigation is identifying the contributor's API-domain labels based on the authors' historical PRs submitted and approved. Hence, we can automatically match contributors and tasks. To improve matchmaking, the API domains and related skills can be organized in a skill ontology [68]. Ontology matchers like AML [69] can be integrated into the pipeline for this goal.

X. Conclusion

Newcomers have difficulty choosing an issue when joining a new project. Several studies addressed this problem by evaluating the developers' expertise or the required skills in tasks to inform and recommend issues for a contribution. Towards this goal, APIs correlate to skills to update and fix source code, and knowing which ones are involved in a possible solution may assist a newcomer in picking an issue to unravel.

We investigated the performance of the social metrics to improve the previously proposed API-domain labels. We found a set of predictors able to enhance the model's performance. Subsequent studies should unravel ways to discover when a project is sensible to a specific metric threshold to avoid exhaustive possibilities. Our results suggest that the conversations on issues convene discussants interested or experts on those subjects. We substantially improve the API-domain label predictions up to 0.922 (precision), 0.978 (recall), and 0.942 (F-measure) using the conversation data when the project has enough participants and dialogs to create a consistent dataset. The outcomes of this study compared with Santos et al. [15] results reached an increase of 22.11% in precision, 30.9% in recall, and 25.4% in F-measure. In the PowerToys project, the results increased by 15.82% in precision and by 15.89% in F-measure. With these results in mind, maintainers and communities can encourage and leverage project communication to calculate social metrics. Better classifier performance can encourage the practical adoption of automatic issue labeling tooling [70].

Acknowledgments

This work is partially supported by the National Science Foundation under Grant numbers 1815503, 1900903, and 2236198. CNPq/MCTI/FNDCT grant #408812/2021-4 and MCTIC/CGI/FAPESP (grant #2021/06662-1)

References

[1] I. Steinmacher, T. Conte, and M. Gerosa, "Understanding and supporting the choice of an appropriate task to start with in open source software communities," in *International Conference on System Sciences*, USA, 2015.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiß, R. Premraj, and T. Zimmermann, "Quality of bug reports in eclipse," in *2007 OOPSLA workshop on eclipse technology eXchange*, 2007, pp. 21–25.

[3] L. Vaz, I. Steinmacher, and S. Marczak, "An empirical study on task documentation in software crowdsourcing on topcoder," in *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*, IEEE. Sao Carlos, Brazil: ACM, 2019, pp. 48–57.

[4] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[5] I. Steinmacher, C. Treude, and M. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, 2018.

[6] A. Barcomb, K. Stol, B. Fitzgerald, and D. Riehle, "Managing episodic volunteers in free/libre/open source software communities," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 1–1, 2022.

[7] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement? a text-based approach to classify change requests," in *CASCON*. Markham, ON, Canada: ACM, 2008, pp. 304–318.

[8] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE. IEEE, 2017, pp. 121– –130.

[9] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE. Cleveland, USA: IEEE, 2019, pp. 406–409.

[10] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in *9th International Conference on Information Communication and Management*. ACM, 2019, pp. 17–21.

[11] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *Asia-pacific software engineering conference*, vol. 2. IEEE, 2013.

[12] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Mining Software Repositories*. USA: IEEE, 2013.

[13] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, no. 3, pp. 150–176, 2016.

[14] Y. Zhu, M. Pan, Y. Pei, and T. Zhang, "A bug or a suggestion? an automatic way to label issues," *arXiv preprint arXiv:1909.00934*, vol. abs/1909.00934, 2019.

[15] F. Santos, I. Wiese, B. Trinkenreich, I. Steinmacher, A. Sarma, and M. A. Gerosa, "Can I solve it? Identifying apis required to complete oss tasks," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, IEEE. Madrid, Spain: IEEE, 2021, pp. 346–257.

[16] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, 2005.

[17] I. Wiese, R. Ré, I. Steinmacher, R. T. Kuroda, G. A. Oliva, C. Treude, and M. Gerosa, "Using contextual information to predict co-changes," *Journal of Systems and Software*, vol. 128, pp. 220–235, 2017.

[18] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *30th international conference on Software engineering*. ACM, 2008, pp. 531–540.

[19] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *16th International Symposium on Foundations of software engineering*. Atlanta, Georgia, USA: ACM, 2008, pp. 13–23.

[20] F. Santos, B. Trinkenreich, J. Pimentel, I. Wiese, I. Steinmacher, A. Sarma, and M. Gerosa, "How to choose a task? Mismatches in perspectives of newcomers and existing contributors," *Empirical Software Engineering and Measurement*, 2022.

[21] F. El Zanaty, C. Rezk, S. Lijbrink, W. van Bergen, M. Côté, and S. McIntosh, "Automatic recovery of missing issue type labels," *IEEE Software*, 2020.

[22] N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375–431, 2013.

[23] I. Kwan, M. Cataldo, and D. Damian, "Conway's law revisited: The evidence for a task-based perspective," *IEEE software*, vol. 29, no. 1, pp. 90–93, 2011.

[24] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *2016 IEEE/ACM 13th working conference on mining software repositories (msr)*, IEEE. Austin, TX, USA: ACM, 2016, pp. 291–302.

[25] I. Wiese, F. Côgo, R. Ré, I. Steinmacher, and M. Gerosa, "Social metrics included in prediction models on software engineering: a mapping study," in *International Conference on Predictive Models in Software Engineering*, 2014.

[26] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, 2011.

[27] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and software technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[28] J. Linåker, H. Munir, K. Wnuk, and C.-E. Mols, "Motivating the contributions: An open innovation perspective on what to share as open source software," *Journal of Systems and Software*, vol. 135, pp. 17–36, 2018.

[29] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *2009 20th International Symposium on Software Reliability Engineering*. Mysuru, Karnataka, India: IEEE, 2009, pp. 109–119.

[30] I. Wiese, R. Takashi, D. Nassif, R. Ré, G. Ansaldi, and M. Gerosa, "Using structural holes metrics from communication networks to predict change dependencies," in *International Workshop on Groupware*, Chile, 2014.

[31] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, 1977.

[32] L. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, 1978.

[33] S. Biçer, A. B. Bener, and B. Çağlayan, "Defect prediction using social network analysis on issue repositories," in *2011 International Conference on Software and Systems Process*, 2011, pp. 63–71.

[34] F. Harary, *Graph Theory*. Boulder, CO, USA: Westview Press, 1994.

[35] D. B. West *et al.*, *Introduction to graph theory*. Upper Saddle River, NJ, USA: Prentice Hall, 2001, vol. 2.

[36] R. Diestel, *Graph Theory*, 3rd ed. Springer, 2005.

[37] JabRef, "JabRef project," 2019. [Online]. Available: https://jabref.org/

[38] M. Rees-Jones, M. Martin, and T. Menzies, "Better predictors for issue lifetime," *arXiv preprint arXiv:1702.07735*, vol. 1702, no. 07735, 2017.

[39] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[40] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, 2009.

[41] A. Savidis and C. Savaki, "Software architecture mining from source code with dependency graph clustering and visualization," in *IVAPP*, 12 2021, pp. 179–186.

[42] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Instructional Conference on Machine Learning*, vol. 242. Canada: iCML, 2003.

[43] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *ICROIT*. India: IEEE, 2014.

[44] S. Vadlamani and O. Baysal, "Studying software developer expertise and contributions in stack overflow and github," in *International Conference on Software Maintenance and Evolution*. Australia: IEEE, 2020.

[45] D. Kim, D. Seo, S. Cho, and P. Kang, "Multi-co-training for document classification using various document representations: Tf–idf, lda, and doc2vec," *Information Sciences*, vol. 477, pp. 15–29, 2019.

[46] M. Bilgin and İ. F. Şentürk, "Sentiment analysis on twitter data with semi-supervised doc2vec," in *2017 international conference on computer science and engineering (UBMK)*. IEEE, 2017, pp. 661–666.

[47] D. Cahyani and I. Patasik, "Performance comparison of tf-idf and word2vec models for emotion text classification," *Bull. Electr. Eng. Inform.*, vol. 10, no. 5, 2021.

[48] F. Herrera, F. Charte, A. Rivera, and M. del Jesus, *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Springer, 2016.

[49] scikit-learn, "scikit-learn," https://scikit-learn.org/, accessed: 2022-06-27.

[50] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Mlsmote: approaching imbalanced multilabel learning through synthetic instance generation," *Knowledge-Based Systems*, vol. 89, pp. 385–397, 2015.

[51] Y. Feng, J. Jones, Z. Chen, and C. Fang, "An empirical study on software failure classification with multi-label and problem-transformation techniques," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, IEEE. Västerås, Sweden: IEEE, 2018, pp. 320–330.

[52] T. Guggulothu and S. A. Moiz, "Code smell detection using multi-label classification approach," *Software Quality Journal*, vol. 28, no. 3, pp. 1063–1086, 2020.

[53] D. Petkovic, M. Sosnick-Pérez, K. Okada, R. Todtenhoefer, S. Huang, N. Miglani, and A. Vigil, "Using the random forest classifier to assess and predict student learning of software engineering teamwork," in *2016 IEEE Frontiers in Education Conference (FIE)*, IEEE. Eire, PA, USA: IEEE, 2016, pp. 1–7.

[54] E. Goel, E. Abhilasha, E. Goel, and E. Abhilasha, "Random forest: A review," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 7, no. 1, pp. 251–257, 2017.

[55] T. Pushphavathi, V. Suma, and V. Ramaswamy, "A novel method for software defect prediction: hybrid of fcm and random forest," in *2014 International Conference on Electronics and Communication Systems (ICECS)*. Coimbatore, India: IEEE, 2014, pp. 1–5.

[56] S. Satapathy, B. Acharya, and S. Rath, "Early stage software effort estimation using random forest technique based on use case points," *IET Software*, vol. 10, no. 1, 2016.

[57] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," *Data mining and knowledge discovery handbook*, pp. 667–685, 2009.

[58] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *annual meeting of the Florida Association of Institutional Research*. Cocoa Beach, FL: FAIR, 2006, pp. 1–3.

[59] I. Steinmacher, T. U. Conte, and M. Gerosa, "Understanding and supporting the choice of an appropriate task to start with in open source software communities," in *2015 48th Hawaii International Conference on System Sciences*, IEEE. Kauai, USA: IEEE, 2015, pp. 5299–5308.

[60] P. Gupta and A. W. Woolley, "Productivity in an era of multi-teaming: The role of information dashboards and shared cognition in team performance," *ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–18, 2018.

[61] D. Qiu, B. Li, and H. Leung, "Understanding the API usage in Java," *Information and software technology*, vol. 73, pp. 81–100, 2016.

[62] A. Sarma, M. Gerosa, I. Steinmacher, and R. Leano, "Training the future workforce through task curation in an OSS ecosystem," in *2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Seattle, USA: ACM, 2016, pp. 932–935.

[63] J. W. D. Alderliesten and A. Zaidman, "An initial exploration of the "good first issue" label for newcomer developers," in *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2021, pp. 117–118.

[64] H. Horiguchi, I. Omori, and M. Ohira, "Onboarding to open source projects with good first issues: A preliminary analysis," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2021, pp. 501–505.

[65] I. Rehman, D. Wang, R. G. Kula, T. Ishio, and K. Matsumoto, "Newcomer oss-candidates: Characterizing contributions of novice developers to github," *Empirical Software Engineering*, vol. 27, no. 5, p. 109, 2022.

[66] R. Burt, *Structural Holes: The Social Structure of Competition*. USA: Harvard University Press, 1992.

[67] J. Nam, S. Pan, and S. Kim, "Transfer defect learning," in *International Conference on Software Engineering*, 2013.

[68] R. F. Calhau, C. L. Azevedo, and J. P. A. Almeida, "Towards ontology-based competence modeling in enterprise architecture," in *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2021, pp. 71–81.

[69] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz, and F. M. Couto, "The agreementmakerlight ontology matching system," in *International Conferences On the Move to Meaningful Internet Systems*, 2013.

[70] J. Vargovich, F. Santos, J. Penney, I. Steinmacher, and M. A. Gerosa, "Givemelabeledissues: An open source issue recommendation system," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR - Data and Tool Showcase)*, 2023.