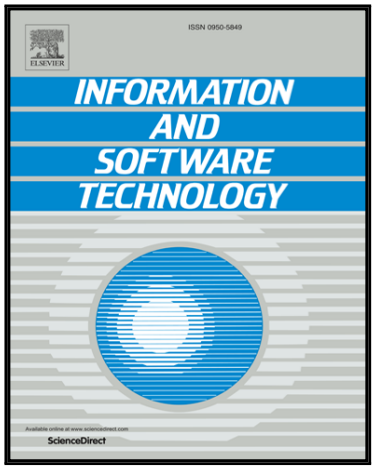
Journal Pre-proof



Unveiling the Technical Roles of GitHub Users

João Eduardo Montandon, Marco Tulio Valente, Luciana L. Silva

PII: S0950-5849(20)30227-5
DOI: https://doi.org/10.1016/j.infsof.2020.106485
Reference: INFSOF 106485

To appear in: *Information and Software Technology*

Received date: 5 March 2020
Revised date: 27 October 2020
Accepted date: 4 November 2020

Please cite this article as: João Eduardo Montandon, Marco Tulio Valente, Luciana L. Silva, Unveiling the Technical Roles of GitHub Users, *Information and Software Technology* (2020), doi: https://doi.org/10.1016/j.infsof.2020.106485

Unveiling the Technical Roles of GitHub Users

João Eduardo Montandon[a], Marco Tulio Valente[b], Luciana L. Silva[c]

[a]*Technical College (COLTEC), Federal University of Minas Gerais, Belo Horizonte, Brazil*
[b]*Department of Computer Science, Federal University of Minas Gerais, Belo Horizonte, Brazil*
[c]*Department of Computer Science, Federal Institute of Minas Gerais, Ouro Branco, Brazil*

Abstract

Context: Modern software development demands high levels of technical specialization. These conditions make IT companies focus on creating cross-functional teams, such as frontend, backend, and mobile developers. In this context, the success of software projects is highly influenced by the expertise of these teams in each field. **Objective:** In this paper, we investigate machine-learning based approaches to automatically identify the technical roles of open source developers. **Method:** For this, we first build a ground truth with 2,284 developers labeled in six different roles: backend, frontend, full-stack, mobile, devops, and data science. Then, we build three different machine-learning models used to identify these roles. **Results:** These models presented competitive results for precision (0.88) and AUC (0.89) when identifying all six roles. Moreover, our results show that programming-languages are the most relevant features to predict the investigated roles. **Conclusion:** The approach proposed in this paper can assist companies during their hiring process, such as by recommending developers with the expertise required by job positions.

Keywords: Technical Roles, Technical Expertise, Developers Profiles, Machine Learning, GitHub

1. Introduction

Software development is a human-intensive activity, which makes the success of software projects highly influenced by the quality and expertise of their development workforce [22]. Besides, software systems are complex engineering artifacts, which demand high levels of technical specialization in different areas [14]. These conditions make IT companies focus on creating cross-functional teams with experts in several positions, such as databases, security, frontend design, backend design, mobile apps, etc.

Recently, these companies are increasingly using social coding platforms—such as GitHub—to search for suitable candidates to their opened positions [37, 15, 53]. Previous works have proposed solutions to use the data of such platforms to discover some skills of these candidates. They mainly focus on identifying which technologies do software developers most work on, such as libraries, frameworks, and languages [56, 59, 64, 58, 30, 19, 66, 65, 45]. A few others have dedicated to discerning developers' soft skills, including communication, teamwork, responsibility, etc [56, 53, 2]. Lastly, some studies aimed on identifying the roles of developers in open source projects, like bug fixer, bug triager, core developer, and tester [67, 52, 11, 21, 29, 1, 31, 20].

Usually, IT companies organize their development teams accordingly to the technology the developers master, e.g., *frontend*, *backend*, *mobile*, and others [44]. For instance, *frontend* developers are specialized on the application's interface, as opposed to *backend* who are responsible for core features. In this context, we currently lack approaches for inferring developers' technical roles. Therefore, our key goal in this paper is to *identify the technical roles played by developers using information available in open source platforms*.

Email addresses: joao.montandon@dcc.ufmg.br (João Eduardo Montandon), mtov.dcc.ufmg.br (Marco Tulio Valente), luciana.lourdes.silva@ifmg.edu.br (Luciana L. Silva)

Does the Identification of Technical Roles Matter?

We define technical roles as the ones derived from expertise on particular programming technologies (e.g., programming languages) and/or architectural components (e.g., frontend frameworks). In this sense, distinguishing candidates' technical roles are important as they represent a good proxy for the technologies and techniques each one master [44]. For instance, data scientists might have a deeper understanding of data analyzing techniques. By contrast, frontend developers should master Web APIs concepts, such as RESTFul APIs, AJAX, etc. In fact, source code hosting platforms—e.g., GitHub—are currently looking for alternatives to make their users profile richer by better expressing their skills, accomplishments, and interests.[1]

Indeed, technical roles are one of the first aspects considered by recruiters when hiring developers for a specific position. To provide evidence of this claim, we inspected the jobs listed by Stack Overflow Jobs[2]—the part of the Q&A forum where companies post job offers—on October 25, 2018. Our key finding can be summarized as follows:

> 3,234 out of 5,027 Stack Overflow job posts (64%) include in their description at least one of the technical roles investigated in this paper.

Additionally, we conducted a preliminary survey to better understand how GitHub is used when hiring software developers. For this, we contacted all Stack Overflow users self-described as technical recruiters who were active in the platform since 2018, and that publicly provided their e-mail for contact. We emailed these users with a brief survey where we asked:

> *Do you often use GitHub in your hiring process? If you do, please select the options that best describe your usage (you can mark multiple options).*

We provided four distinct options for the second question, which were formulated after consulting the literature about software expertise [37, 53, 7, 45]. To avoid possible bias introduced by the order of the options, we configured our survey system to present them in random order for each contact. In total, we contacted 30 recruiters and received 7 answers (23% response rate). Six developers confirmed they use GitHub for hiring purposes. The survey results are summarized in Figure 1.

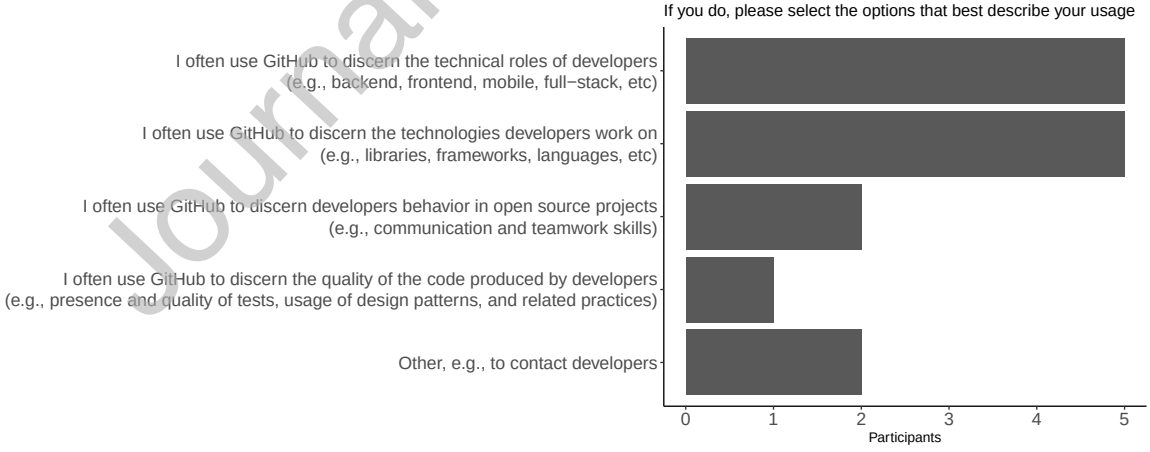


Figure 1: Survey responses from the recruiters that use GitHub in their hiring process.

Considering our intention in this paper, the main finding of this survey can be summarized as follows:

[1] `https://twitter.com/natfriedman/status/1133700043695886336`
[2] https://stackoverflow.com/jobs

> Five out of six technical recruiters use GitHub to discern the *technical roles* of the candidates, which is exactly our central goal in this paper.

Although not directly linked with our goals here, another frequent usage of GitHub includes identifying which technologies do developers work on (also with 5 answers) and to discern developers' behavior in open source projects (2 answers).

Proposed Study

In this study, we investigate an approach centered on supervised machine-learning techniques to identify developers' technical roles by considering their contributions to public GitHub projects. More specifically, we infer developers' technical roles by using features extracted from their public GitHub profiles (e.g., programming languages, short bio, etc) and their GitHub projects. We first build a ground truth with the technical roles of 2,284 GitHub developers. Then, we executed the proposed technique to identify developers in six popular technical roles:[3] *backend*, *frontend*, *full-stack*, *mobile*, *devops*, and *data science*. Lastly, we evaluated our approach by answering four research questions.

(RQ.1) How accurate are machine learning classifiers on identifying developers' technical roles? Our best model scored meaningful results for precision (0.75) and AUC (0.70), while it reported lower ones for recall and F1 (0.49 and 0.59, respectively). However, in the context of this work, precision-based metrics gain more importance, as technical recruiters are more interested in correctly identifying candidates that fit their needs (i.e., precision) instead of identifying them (i.e., recall). When considered independently, our models provide the best results for *data science* and *frontend* roles (0.86 and 0.77; precision). By contrast, we observed lower results for *backend* (0.62; precision).

(RQ.2) What are the most relevant features to distinguish technical roles? Features associated with programming languages are relevant for all roles. Individually, *data science*'s top-10 most relevant features have the largest representativeness rate: 33.2%. By contrast, *backend* top-10 features scores only 6.8% of the total.

(RQ.3) Do technical roles influence each other during classification? In this RQ, we investigate the gains achieved with the use of Classifiers Chains in our problem. Suppose that a developer is predicted as having a role R_1. After making this prediction, R_1 is used as input to the classifiers of R_2, R_3, R_4, and R_5 roles (assuming we analyze five roles, as in our first three RQs). Specifically, we check the improvements achieved with Classifier Chains by setting up and training 120 different classification models (i.e., covering all possible permutations of five roles). Overall, such models present minor improvements in recall (+0.05) at the cost of precision scores (−0.05) when compared to the ones in *RQ.1*.

(RQ.4) How effectively can we identify full-stack developers? Differently from the technical roles analyzed in the first three RQs, the *full-stack* role is derived from the combined expertise in *backend* and *frontend* technologies. Therefore, we conducted a separated study to identify *full-stack* developers and verify how they impact the classification of *backend* and *frontend* developers. The proposed classifier performed very well when identifying FullStack, achieving 0.99 for precision and 0.71 for recall. Moreover, the addition of FullStack developers significantly improved the identification of both Backend and frontend—0.87 and 0.86 for precision, respectively—after specific adjustments.

Our Contributions

We show that—based on high-level features retrieved from developers' public profiles and projects in GitHub—it is possible to infer major technical roles commonly used in industry to recognize software developers. Secondly, we make publicly available our ground truth with the roles of 2,284 GitHub developers. This ground truth can motivate and support further research in the area.

[3] Accordingly to https://insights.stackoverflow.com/survey/2018

Table 1: Regular expressions used to identify technical roles.

Role	Regular Expression	Examples
Backend	`/\bback.{0,1}end\b/i`	*Back end*, *backend*
Frontend	`/\bfront.{0,1}end\b/i`	*Frontend*, *front-end*
DevOps	`/\bdev.{0,1}ops\b/i`	*DevOps*, *dev-ops*
DataScience	`/\bdata.{0,1}scientist\b/i`	*Data Scientist*, *data scientist*
Mobile	`/\bmobile\b/i`	*mobile*, *Mobile*

Paper Structure

The rest of this paper has nine sections. First, Section 2 documents the data and methods used in the study, including the ground truth creation, the features extracted from GitHub, and the machine learning setup. Section 3 presents the results for the first three research questions. Section 4 describes the exploratory study conducted on *full-stack* developers. In Section 5.3, we manually analyze some developers classified by our approach to qualitatively interpret the classification results. In Section 5 we discuss the implications to both practitioneers and software engineering community. Section 6 discusses threats to validity and Section 7 presents Related Work. Finally, Section 8 concludes this work.

2. Study Design

2.1. Technical Roles

The focus of this paper is to propose an approach to unveil the technical roles of GitHub developers. We used the Annual Developer Survey conducted by Stack Overflow in 2018 to select the roles considered in our study. More than 100,000 developers from 183 countries answered this 30-minute survey where, among other questions, they answered which technical roles they associate with the tasks they normally perform. In this paper, we study the top-4 most popular roles, according to this survey: Backend (58%), FullStack (48%), Frontend (38%), and Mobile (20%). Moreover, we included DevOps (10.4%) and DataScience (7.7%) roles in our analysis as both are also frequently featured in the top trending IT positions. Specifically, we first focus on five roles: Backend, Frontend, DevOps, DataScience and Mobile (*RQ.1*, *RQ.2*, and *RQ.3*). Due to its particularity, we investigate the FullStack role separately (*RQ.4*).

2.2. Ground Truth

As the proposed machine learning model relies on features extracted from GitHub to classify developers' technical roles, we fully avoided using GitHub's data to generate our ground truth. Instead, we relied exclusively on Stack Overflow's data to build this ground truth, since this data would not be used further in our study. Other works use data from Stack Overflow as a reliable source to assess developers' expertise [64, 54, 2]. On the other hand, after building the ground truth, our goal is to identify technical roles of GitHub users, i.e., by only considering GitHub data.

We followed a two-step process to build the ground truth, as explained next.

Stack Overflow Data Gathering: We used Stack Exchange Data Explorer (SEDE)[4] to collect Stack Overflow users with GitHub profiles. SEDE is a publicly available tool that allows querying data available in Stack Exchange platform, using a web interface. On June 2nd, 2020, we queried SEDE for Stack Overflow users who have a link to GitHub. This query returned 27,051 developers. We then extracted—through regular expressions—the GitHub username from the provided links, and used GitHub GraphQL API[5] to retrieve GitHub data (as described in Section 2.3) for developers with valid usernames. During this procedure, we discarded developers with URLs pointing to invalid GitHub usernames. We ended up with a dataset composed of 24,889 developers.

Labeling: In this step, we analyzed the profile information provided by Stack Overflow to label developers' roles. Figure 2 depicts an example of Stack Overflow's user profile. Among the information available in his profile, this

[4] `https://data.stackexchange.com/`
[5] `https://developer.github.com/v4/`

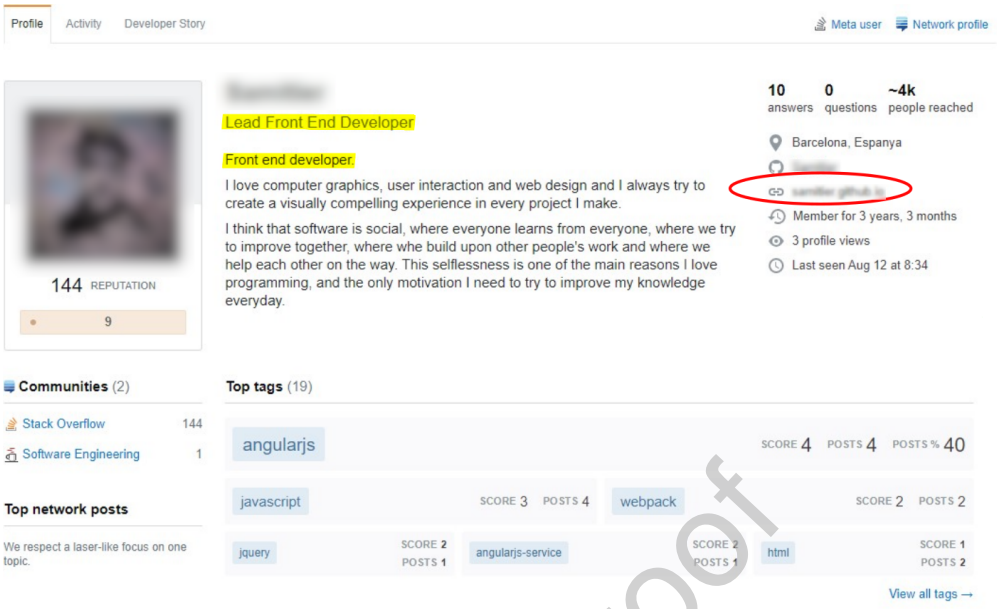


Figure 2: Example of Stack Overflow developer profile. The data used in the ground truth creation is highlighted.

developer described himself as a "Lead Front End Developer". Therefore, we relied on developers' description text in Stack Overflow to determine their roles. More specifically, we elaborated five distinct regular expressions to identify each role, as described in Table 1. These regular expressions were configured to consider spaces, e.g., *Back-end*, *Front end*) and to ignore case (e.g., *Mobile* and *mobile*. Considering the profile in Figure 2, our labeling process classify him as FRONTEND ("*Lead Front End Developer.*" and "*Front end developer.*").

Selected Developers: By following the aforementioned steps, 1,802 developers were labeled with at least one technical role. We discarded 140 (7.7%) developers with less than five GitHub public repositories. The reason is that our models depend on developers being active on GitHub to infer their roles. Lastly, 1,662 developers were included in our ground truth, containing 2,022 roles assignments.

FRONTEND is the role with more developers (820), followed by MOBILE (453) and BACKEND (450). Table 2 shows how these labels are distributed. As we can observe, 1,340 developers (80%) are associated with just one role. Most of them are FRONTEND (545; 33%), or MOBILE (352; 21%). Considering the 287 developers who have two or more roles, the majority are both BACKEND and FRONTEND (198; 69%). By contrast, DATASCIENCE is the one with the smallest intersection with other roles, i.e., one developer in all situations. 32 developers were labeled in three roles, most of them in BACKEND, FRONTEND, and MOBILE (23; 72%). Only three developers were assigned in four roles. Finally, we found no developers who self-labeled themselves in all five roles.

2.3. Data Collection

After using Stack Overflow's data to leverage our ground truth, we collected GitHub's data for each developer in this golden set. This data will be later transformed into a list of features to feed our prediction models (see Section 2.4). To provide a code agnostic solution (and therefore analyze repositories in multiple programming languages), we collected mostly textual data about the developers' profiles and the projects they own.[6] The data we collected fits into six distinct categories:

- *Programming Language:* Previous works report that programming languages are a reliable proxy to assess developers' expertise area [37, 38, 53, 24, 7]. For each developer, we first retrieved her list of projects, and then extracted both *commits* and the *main programming language* of each one. Next, we derived the commits

[6]It is important to note that we have discarded forked projects in our analysis.

Table 2: Distribution of developers among the analyzed roles.

# Roles	Role					# Devs.
	Backend	Frontend	Mobile	DevOps	DataScience	
One	✓					178
		✓				545
			✓			352
				✓		119
					✓	146
Two	✓	✓				198
	✓		✓			29
	✓			✓		9
	✓				✓	1
		✓	✓			39
		✓		✓		4
			✓	✓		5
			✓		✓	1
				✓	✓	1
Three	✓	✓	✓			23
	✓	✓		✓		7
	✓	✓			✓	1
	✓		✓	✓		1
Four	✓	✓	✓	✓		3

information into three dimensions: (a) the total number of commits; (b) the number of commits performed by the developer (i.e., she is the author); and (c) the rate between both of them. Lastly, we grouped the projects accordingly to the programming language collected earlier and averaged the resulting values. We ended up with the following features for each developer, in each language: *Language (total)*, *Language (author)*, and *Language (rate)*.

- *Short Bio:* Short description of the developers' main activities, manually provided by each one. We decided to include this category in our analysis as some developers use this information to describe their field of activity, e.g., "I develop iOS/Android apps with Swift/Kotlin language".
- *Projects' Names:* These names may contain keywords referencing specific technologies used in the projects, e.g., *android-data-binding-1*, *docker-example*, etc. As observed in previous works, these technologies are also indicators of expertise [58, 26, 54, 45]. Therefore, for each developer, we collected the name of her GitHub projects and merged them into a single data point.
- *Projects' Topics:* For the same reason as above, topics are generally used by GitHub developers to declare technologies and tools used by their projects, e.g., *ionic*, *gulp*, and *webpack*. We also collected the GitHub topics of the developers' projects and merged them into a single data point as well.
- *Projects' Descriptions:* This information can also provide clues about the technologies used by GitHub projects [27]. For instance, the description "a React.js contact manager" indicates that *ReactJS* is used in the project. For this reason, we collected the short description of each developer's project and merged them into a single data point.
- *Projects' Dependencies:* We decided to include the list of dependencies since 3rd-party libraries are largely used in modern software systems [59, 27, 30, 45]. For each project, we extracted its list of dependencies using a GitHub GraphQL endpoint.[7] Next, we assembled these features by summing up the number of times each dependency was referenced in each developer's projects. We initially collected data on 17,556 dependencies, but we restricted ourselves to the top-1,000 most used ones as they are present in 81% of all analyzed projects.

[7] `https://developer.github.com/v4/object/dependencygraphmanifest/`

Table 3: Dataset with five target labels.

Developer		BACKEND	FRONTEND	MOBILE	DEVOPS	DATASCIENCE
D_1	$\cdots$	1	1	0	0	0
D_2	$\cdots$	1	0	0	1	0
D_3	$\cdots$	0	0	0	0	1
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
D_n	$\cdots$	0	0	1	0	1

2.4. Selected Features

To transform the data described in Section 2.3 into a list of features, which can be used as input for a machine learning algorithm, we applied the following transformation steps:

Correlation Analysis: Initially, we ended up with 477 and 1,000 features for *Programming Languages* and *Projects' Dependencies* categories, respectively. Due to this high number, we followed a correlation analysis procedure to remove the ones with high correlations in each category. For this, we used *corr* function from *pandas*[8] Python library to generate a Spearman correlation matrix. Then, we identified pairs of features with a correlation greater than 0.7, as previously adopted in the literature [8]; in such cases, we discarded the feature that has the highest correlation rate when compared to the other ones. In total, we discarded 260 features for *Programming Languages*. Generally these correlations happened with features belonged to the same language, but covering different domains, e.g., *C (author)* and *C (rate)*, *CSS (author)* and *CSS (rate)*, etc. In most cases, we maintain the *rate*-based ones since they usually have the lowest correlation rate with the others. As for *Projects' Dependencies*, we discarded 202. Differently, the discarded dependencies are more diverse, including specific (*ionic-native-statusbar*, a specific plugin for ionic framework) and general (*unicorn*, an http webserver) purpose libraries.

Bag-of-Words: Machine learning algorithms do not deal with text directly. For this reason, we used a bag-of-words to transform each textual category into a list of features. Bag-of-words is a simple and efficient technique to extract features from text [25]. In summary, each word is mapped to a feature that describes its frequency in a document. As in other studies [33, 55, 10], we performed the following steps to apply this technique on *Short Bio*, *Projects' Names*, *Projects' Topics*, and *Projects' Descriptions*. First, we manually stripped out HTML tags, punctuation, and numbers through regular expressions. Then, we removed stop words (e.g., of, him, the, and, etc.) using the default English list provided by *sklearn.text* module. Finally, we used *TfidfVectorizer*[9] class from *sklearn* library to apply bag-of-words using the TF-IDF outcome as feature values. Following common guidelines for text processing [50], we considered only words in a certain document frequency range: [0.04, 0.15] for *Projects' Descriptions*, [0.03, 0.25] for *Projects' Names*, [0.01, 0.25] for *Projects' Topics*, and [0.01, 0.20] for *Short Bio*. These limits were defined after executing Random Forest classifier with 100 different bag-of-words configurations by randomly selecting different document frequency values in each configuration. We selected the configuration which presents improvements in most evaluated metrics. The result of each configuration is available in our replication package for further reference.

After these steps (i.e., *Correlation Analysis* and *Bag-of-Words*), we ended up with 1,471 features, including 798 from *Projects' Dependencies* category, 217 from *Programming Languages*, 169 from *Projects' Descriptions*, 155 from *Projects' Names*, 69 for *Short Bio*, and 63 from *Projects' Topics*.

2.5. Machine Learning Setup

Multi-label Problem: Usually, classification problems rely on a single-label, i.e., each instance is associated with a single label. On the other hand, a multi-label classification problem can associate more than one label to each instance [62, 10]. Particularly, our dataset is a five-label classification problem since there are five distinct roles each developer can be an expert on. Table 3 illustrates this dataset. In this example, developer D_1 is associated with

[8] `https://pandas.pydata.org/`

[9] `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer`

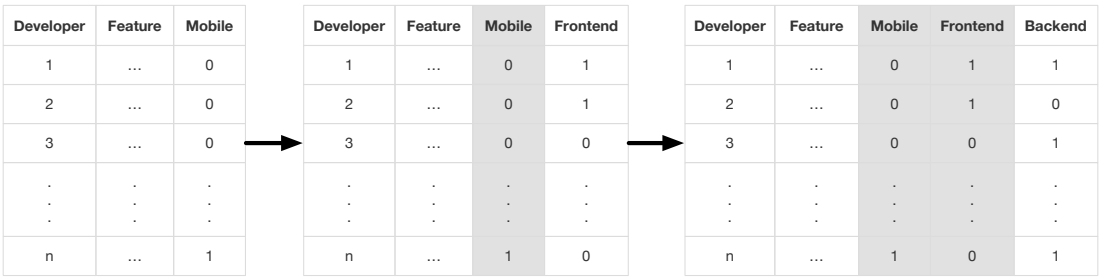


Figure 3: Multi-label classification using Classifier Chains.

Backend and Frontend roles. Likewise, developer D_2 is tied with Backend and DevOps. On the other hand, D_3 is associated just with DataScience.

Problem Transformation: We can approach a multi-label classification problem in different ways. For instance, we can transform a multi-label dataset into a single-label one [35, 51]. Alternatively, some algorithms work directly with multi-label datasets [62]. In this paper, we use two common transformation techniques for dealing with multi-label datasets:

1. *Binary Relevance (BR):* This technique splits the original dataset into multiple binary classification problems, i.e., one binary classification problem for each label [61, 35]. Then, such models are fitted independently. The results can be shown either independently (one result for each model) or aggregated (weighted average of all models). In this study, we calculated the aggregated results using a *micro-average* strategy, as it is recommended for multilabel problems.[10]
2. *Classifier Chains (CC):* This technique links the binary classifiers along a chain in a way that each classifier uses results from earlier ones in its predictions [51]. The goal is to take advantage of eventual dependencies that might exist among target labels. Figure 3 illustrates a classification scenario using *CC* considering three technical roles: Mobile, Frontend, and Backend. Each model includes previous predictions as features and propagates its results along the chain. Figure 3a presents the first model in the chain (Model I). As we can see, this model makes its predictions using only the available features. Model II (Figure 3b) predicts its values using all features plus Model I predicted labels. Finally, the process is propagated to Model III, using all features plus both Models I and II predicted labels. In this way, *CC* tackles the label independence problem faced by *BR* [36, 69].

Machine Learning Classifier: We initially decided to use Random Forest [13] and Naive Bayes [39] classifiers to train and test our models for identifying the technical roles of GitHub users. We selected Random Forest due to its robustness to noise and outliers [60]. Moreover, Random Forest was successfully used in many application areas, including software engineering problems [42, 47, 49, 18, 8, 10]. As we are dealing mostly with textual information, we also decided to include Naive Bayes in our study as this classifier generally presents good results in textual scenarios, such as spam filtering [43] and news classification [40]. We used the implementation provided by *scikit-learn* [46] for both classifiers along with 10-fold cross-validation to select the best model. Basically, cross-validation splits the data into k folds (in our case, $k = 10$), where $k - 1$ folds are used to fit the model and the remaining one is used to test the predictions.

Evaluation Metrics: We use six metrics to evaluate the quality of the model's predictions: precision, recall, F1-score, AUC (Area Under the Curve), Jaccard Coefficient, and Hamming Loss. Precision measures the correctness, while recall measures the completeness of the model predictions. F1-score is the harmonic mean of precision and recall.

[10] `https://scikit-learn.org/stable/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel`

AUC represents the area under the Precision-Recall (PR) curve. Unlike ROC, PR curves are more recommended for assessing unbalanced datasets [50]. Jaccard Coefficient—also known as multi-label accuracy [12]—is the division of the number of correctly predicted labels by all true labels. Hamming Loss is the ratio of wrong labels by the number of labels [23]. As it is a loss measure, the lower the value, the better the model. Finally, we compared all metrics against a Stratified baseline. This baseline considers the dataset distribution to perform its predictions. For instance, in a dataset where 10% of the samples are positive labels, this baseline limits its positive predictions to 10% as well.

3. Results

(RQ.1) How accurate are machine learning classifiers in identifying technical roles?

Table 4a presents the general results for the first three roles using Binary Relevance (*BR*). The Random Forest classifier presented the best results overall, scoring 0.77 for precision and 0.71 for AUC. Moreover, apart from recall, Random Forest outperformed Naive Bayes for all evaluated metrics. Finally, both classifiers performed significantly better than the Stratified Baseline. The baseline results are, for example, only 0.33 for precision.

Table 4b presents the classification results for each role, separately. Overall, Random Forest presented better results for precision when compared to Naive Bayes, and both outperformed the Stratified Baseline. Regarding Random Forest, the classifier presented high precision rates—i.e., above 0.7—for 4 out of 5 roles: DataScience (0.86), Mobile (0.78), Frontend (0.77) and DevOps (0.70). On the other hand, only Frontend achieved results as good as for recall (0.78) and F1 (0.77). Further, the classifier scored poor results especially for Backend role (precision, recall, and F1 all equal to 0.28).

Metric	Stratified Baseline	Random Forest	Naive Bayes
Precision	0.33	0.77	0.51
Recall	0.32	0.49	0.62
F1	0.33	0.59	0.56
AUC	0.41	0.71	0.59
Jaccard Coeff.	0.20	0.42	0.39
Hamm. Loss	0.32	0.16	0.24

(a) Overall results

Role	Precision	Recall	F1
Stratified Baseline			
Backend	0.28	0.28	0.28
Frontend	0.48	0.46	0.47
Mobile	0.28	0.28	0.28
DevOps	0.08	0.06	0.07
DataScience	0.09	0.09	0.09
Random Forest			
Backend	0.62	0.12	0.18
Frontend	0.77	0.78	0.77
Mobile	0.78	0.38	0.51
DevOps	0.70	0.13	0.20
DataScience	0.86	0.66	0.74
Naive Bayes			
Backend	0.33	0.40	0.36
Frontend	0.69	0.82	0.75
Mobile	0.51	0.46	0.47
DevOps	0.24	0.47	0.32
DataScience	0.45	0.85	0.58

(b) Results for each role

Table 4: Binary Relevance results.

When it comes to Naive Bayes, we did not identify any score above 0.7 for precision; Frontend has the highest one (0.69), followed by Mobile (0.51) and DataScience (0.45). In fact, we observed such good results in three scenarios only: Frontend and DataScience for recall (0.82 and 0.85, respectively), and Frontend for F1 (0.75). Even though, Naive Bayes was superior to the baseline in all scenarios.

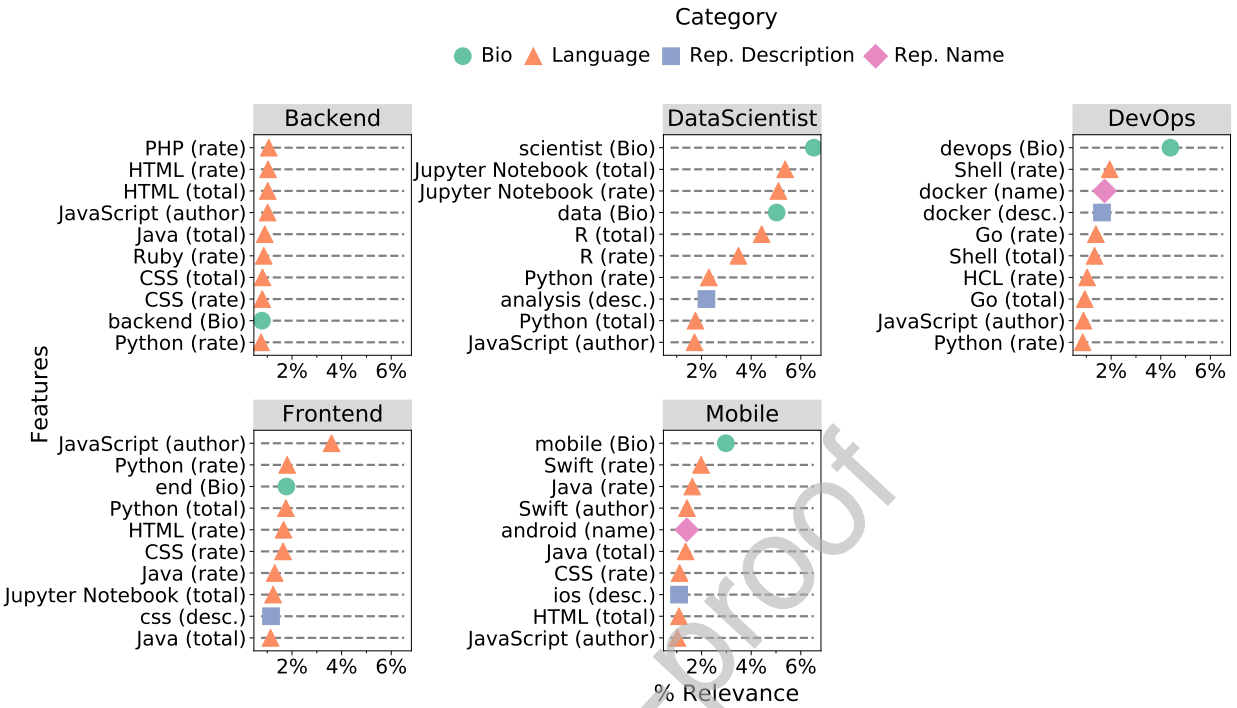


Figure 4: Most relevant features for each technical role. Colors and shapes represent each feature category: developers bio ((*Bio*), green circle), projects' descriptions ((*desc.*), blue square), projects' names ((*name*), pink diamond), and programming languages ((*rate*, *author*, and *total*), orange triangle)

> Random Forest presented the best results overall (e.g., 0.77, precision; 0.71, AUC), outperforming both Naive Bayes and the baseline. When analyzed individually, DataScience and Frontend scored the best results for most metrics (e.g., 0.86 and 0.77; precision), while Backend showed the worse ones (e.g., 0.62; precision).

(RQ.2) What are the most relevant features to distinguish technical roles?

Figure 4 shows the top-10 most relevant features by technical role. To identify these features, we executed the Random Forest classifier for each role independently, using the same parameters as in *RQ.1*. We then selected the top-10 features with the highest value from the feature relevance ranking provided by each model. In Figure 4, the colors and shapes identify the category of each feature (programming languages, projects' names, projects' descriptions, projects' topics, projects' dependencies, or short bio). Furthermore, we annotate the category associated with each feature in its description. Finally, we normalized the ranking with respect to the feature with the highest value.

Features associated with *programming languages* are largely predominant for all five roles, representing 38 out of 50 features: 7 for DataScience, DevOps, and Mobile; 8 for Frontend, and 9 for Backend. From these, 19 (50%) are rate-based, 13 (34%) consider the total number of commits, and 6 (16%) include only the commits performed by the author. In other words, 66% of the relevant *programming languages* features do take into account actual developers' contributions.

Next, *short bio* appears as the second most frequent with 6 occurrences, followed by *projects' descriptions* and *projects' names* (3 and 2 occurrences, respectively). Nevertheless, *short bio* shows up as the most important feature in three roles. Most of these features directly describes the role that is analyzed: *scientist* for DataScience, *devops* for DevOps, *mobile* for Mobile, and *backend* for Backend. Surprisingly, *projects' topics* and *projects' dependencies* are not present in any ranking position.

When we analyze the results for each role, we see that DataScience has the highest relevant features. Six features stand out with more than 3% of relevance rate: *scientist (Bio)* (6.5%), *Jupyter Notebook (total)* (5.4%), *Jupyter Notebook (rate)* (5.1%), *data (Bio)* (5.0%), *R (total)* (4.4%), and *R (rate)* (3.5%). Although these values may sound low, our model includes 1,662 features. Also, these features are rather specific to the DataScience field; from them only *Jupyter Notebook (total)* is also present in other roles (Frontend).

For Mobile, *mobile (Bio)* stands out with 3.0% of relevance rate, followed by *Swift (rate)* (2.0%) and *Java (rate)* (1.6%). More specifically, three out of 10 features are directly associated to the iOS development platform (*Swift (rate)*, *Swift (author)*, and *ios (desc.)*). Likewise, other three features are linked to the Android platform: *Java (rate)*, *android (name)*, and *Java (total)*. The features related to iOS and Android development represent 4.5% and 4.3% of the top-10 listed features, respectively.

Regarding DevOps, we observe that *devops (Bio)* plays a prominent position in the ranking, with 4.4%. The next ones are *Shell (rate)* (1.9%), *docker (name)* (1.7%), and *docker (desc.)* (1.6%). Frontend ranking presents a similar distribution, where *JavaScript (author)* is the prominent feature with 3.6%, whereas the next five are valued between 1.8% (*Python (rate)*) and 1.6% (*CSS (rate)*).

Finally, we observe that no feature stands out in Backend ranking. The most relevant feature is *PHP (rate)*, with 1.1%, while *Python (rate)* is the 10th feature in the ranking, with 0.8%. Even though, 5 out of the top-10 features are linked with backend development: *PHP (rate)*, *Java (total)*, *Ruby (rate)*, *backend (Bio)*, and *Python (rate)*.

> Features related to *programming languages* are predominant for all five roles. In DataScience role, six features stand out: *scientist (Bio)*, *Jupyter Notebook (total)*, *Jupyter Notebook (rate)*, *data (Bio)*, *R (total)*, and *R (rate)*. On the other hand, fewer features are presented such importance in other roles: *devops (Bio)* for DevOps, *mobile (Bio)* for Mobile, *JavaScript (author)* for Frontend. Lastly, no feature stands out from the others for Backend.

(RQ.3) Do technical roles influence each other during classification?

As explained in Section 2.5, Classifier Chains (*CC*) is a technique to deal with multi-label classification problems. When using this technique, Binary Classifiers are ordered in such a way that each model uses previous predictions as input. The rationale is that, given two labels *A* and *B*, if *A* influences *B* then we might want to use *A* to improve *B*'s predictions.

Particularly, we executed the *CC* technique with Random Forest for all possible role combinations. In total, the classifier was executed for 5! combinations, i.e., 120 different combinations. Figure 5 presents the results in six different graphics, one for each metric considered in this study. For comparison reasons, the results are reported in relation to *RQ.1*, which is used as a baseline in this RQ and it is represented by a dashed line in the figure. The horizontal axis represents each *CC* configuration and the vertical axis the number of points each metric has above or below the baseline. This means that the *CC* technique performed better than the baseline when its results are above the dashed line, or worse otherwise.

Overall, all models show a minor increase of recall up to almost +0.05 (configurations 72, 89, and 96), at a cost of lower precision (–0.05; configurations 47, 80, 93, and 113). We also observe minor improvements for both F1 and Jaccard Coefficient (+0.02, configurations 60, 72, and 96; both), while AUC presented a minimal decrease (−0.01, configurations 80 and 83). By contrast, we did not find any relevant change to the Hamming Loss score. Interestingly, the highest improvements for recall, F1, and Jaccard—and also some of the minor setbacks for precision—happened between configurations 49 and 60, when Mobile is classified first, followed by Backend or Frontend.

> The results reveal that including technical roles predictions to the classifying process does not bring significant improvements. Overall, we obtained minor improvements for recall, Jaccard, and F1 on the cost of precision. These results are more consistent when Mobile is classified first, followed by Backend or Frontend.

4. Understanding the FullStack Role

Differently from the previously analyzed roles, FullStack developers are defined by their ability to work through the whole application stack. In fact, the industry sees a FullStack developer as someone who can perform both

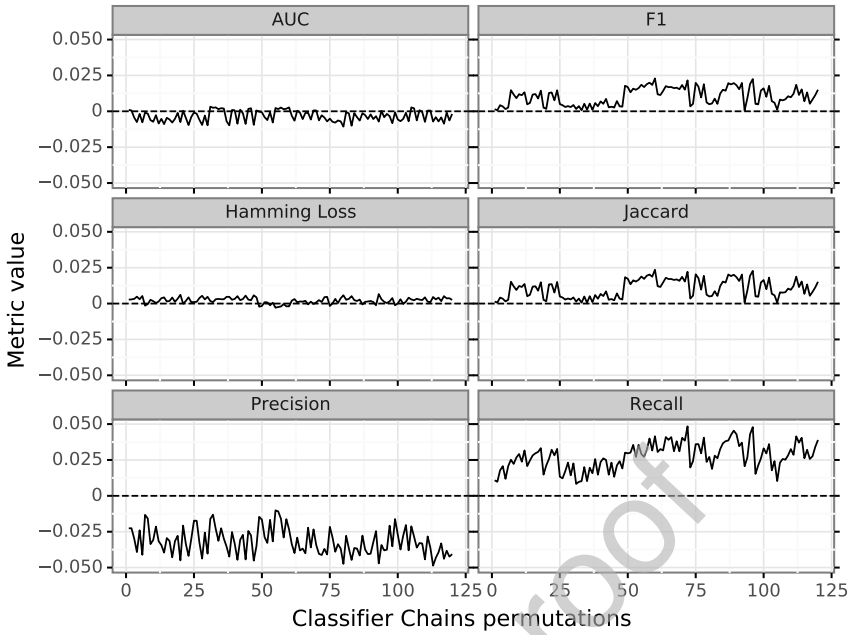


Figure 5: Overall results using Classifier Chain, reported in relation to the *RQ.1* baseline (i.e., dashed line).

Frontend and Backend tasks. The following quote, retrieved from a popular Medium post,[11] illustrates this point:

> *Being a Full-Stack developer [. . .] means that you are able to work on both sides and understand what is going on when building an application.*

Due to this particularity, we decided to investigate the FullStack role in a specific section. First, we extended the ground truth described in Section 2.2 to include developers self-identified as FullStack on Stack Overflow.[12] The new ground truth is composed by 2,284 developers; 853 of them are FullStack. From these, 783 were not labeled either as Backend or Frontend, which contradicts the FullStack's definition. To fix this inconsistency, we adapted the labeling process to consider every FullStack developer as an expert in both Backend and Frontend roles.

After applying the same process described in Sections 2.3 and 2.4, this extended dataset ends up with 1,567 features: 819 from *Projects' Dependencies* category, 219 from *Programming Languages*, 212 from *Projects' Descriptions*, 146 from *Projects' Topics*, 101 from *Projects' Names*, and 70 from *Short Bio*.

(RQ.4) How effectively can we identify full-stack developers?

Table 5 presents the classification results for Random Forest using the Binary Relevance approach (*RQ.1*) and including FullStack. Besides, we included the results for each of the five previous roles, along with the difference for the one described in RQ.1.

Overall, all metrics reported significantly better results after including FullStack developers in the ground truth. Precision, recall and F1 increase to 0.88 (+0.11), 0.69 (+0.20), and 0.77 (+0.18), respectively. Likewise the new AUC, Jaccard Coefficient, and Hamming Loss reach 0.89 (+0.18), 0.69 (+0.10), and 0.13 (−0.03), respectively.

Considering each role, we observe that FullStack presented very high results: its precision is 0.99, recall is 0.71, and F1 is 0.83. When it comes to the other roles, we noticed an interesting improvement for both Frontend and Backend. Frontend's value for F1 increased from 0.77 to 0.87 (+0.10), mainly due to recall improvement (from 0.78

[11] `https://medium.com/coderbyte/a-guide-to-becoming-a-full-stack-developer-in-2017-5c3c08a1600c`, Accessed at 2019-04-04.

[12] We used the following regex to label developers as FullStack: `/\bfull.{0,1}stack\b/i`.

Table 5: Binary Relevance results after including FullStack, Random Forest only.

Role	Precision	Recall	F1
FullStack	0.99	0.71	0.83
Backend	0.87 (+0.25)	0.63 (+0.51)	0.73 (+0.55)
Frontend	0.86 (+0.09)	0.89 (+0.11)	0.87 (+0.10)
Mobile	0.80 (+0.03)	0.34 (−0.04)	0.47 (−0.04)
DevOps	0.75 (+0.05)	0.06 (−0.07)	0.11 (−0.09)
DataScience	0.86 (+0.00)	0.62 (−0.04)	0.71 (−0.03)
Overall[13]	0.88 (+0.11)	0.69 (+0.20)	0.77 (+0.18)

to 0.89; +0.11). Most important, Backend gained 55 points in F1 when compared to the one in *RQ.1* (from 0.28 to 0.73). Recall is also largely responsible for this difference as it scored 0.63 against 0.28 before; an increase of 51 points. By contrast, we observe a minor negative impact in F1 for Mobile (−0.04), DevOps (−0.09), and DataScience (−0.03). Even so, the precision score for these roles has either increased (Mobile, +0.03; DevOps, +0.05) or stayed the same (DataScience).

We accredit this side effect as a consequence of expanding the dataset with the new FullStack developers. Particularly, in our second dataset, we found several FullStack developers with missing Backend or Frontend labels (more precisely, 783 developers, as we mentioned before). In other words, once they are FullStack, for these developers it is implicit that they should be also viewed as Backend or Frontend. Consequently, after labelling FullStack developers as Backend and Frontend we provided new information to the classifiers, which allowed them to improve their predictions.

> The proposed classifier performed very well when identifying FullStack developers (*precision* = 0.99). Moreover, FullStack developers have an interesting collateral effect, since they contributed to improving results of both Backend (+25%) and Frontend (+9%).

5. Discussion

In this section, we start discussing the implications of our work to both academia and practitioners (Section 5.1). Then, we argue about how the results for precision and recall should be interpreted in the context of this study (Section 5.2). Lastly, we analyze the performance of our approach by inspecting developers profiles in three different scenarios (Section 5.3).

5.1. Implications

We shed light on the importance of unveiling technical roles; an expertise topic not yet extensively investigated by the software engineering community. More specifically, we performed two preliminary studies to demonstrate the importance of this topic. First, we inspected posts from Stack Overflow Jobs platform and find out that 64% of 5,027 offers are for one of the six technical roles studied in this paper. Second, we conducted a survey with technical recruiters to understand which characteristics do they look for in GitHub profiles. Five out of six recruiters indicated they search for clues to discern the *technical roles* of the candidates.

Besides, the proposed machine learning approach has most of its usage in hiring processes. In this context, when hiring software developers, technical recruiters can benefit from the technical roles inferred by the proposed models. This usage might occur in two main ways. First, by *proactively* identifying developers with the skills expected by existing job positions in the company. Secondly, by *reactively* evaluating the profile of candidates who have already applied to existing job positions, to assure they have the expected skills. In both cases, the inferred technical roles should be used as a piece of additional information during the hiring process, which certainly includes other selection instruments, such as technical interviews, reference letters, etc.

[13] Overall value is calculated using the same strategy as described in Section 2.5.

5.2. A Note on Precision and Recall

In general, our approach showed its effectiveness in revealing the technical roles of software developers given their GitHub profiles. For instance, our model scored 0.88 for precision and 0.89 for AUC when considering all six roles analyzed in this study (see Table 5); by contrast we achieved a lower result for recall (0.69). When analyzed individually, we observe that the precision for each role ranges from 0.75 (DevOps) to 0.99 (FullStack). In fact, except for DevOps all other roles achieved at least 80% of precision. On the other hand, only Frontend reached similar value for recall (0.89); the other ones remained between 0.06 (DevOps) and 0.71 (FullStack).

Essentially, precision answers the following question *"What proportion of positive identifications was actually correct?"*, whereas recall seeks to answer *"What proportion of actual positives was identified correctly?"*.[14] In the context of technical recruiters, we advocate that precision is more important than recall as they normally need to identify a small set of candidates for the existing positions; i.e., they do not need to locate all developers matching their job positions in the GitHub universe. Furthermore, companies avoid at most a false positive (a bad hiring), as it is more expensive [41, 9]. On the other hand, we understand that such limitation might represent an issue for software developers, as qualified developers can eventually be discarded for not scoring meaningful results in the features considered in this study.

5.3. Manual Analysis

We manually inspected developers profiles to analyze their predictions so we can better understand the proposed classification results. Specifically, we analyze developers in three distinct scenarios: correct classifications (True Positive); wrongly classified in a given role (False Positive); and not classified in a given role, despite being so (False Negative).

5.3.1. True Positive Scenario

Developers in this group have had their roles correctly predicted by our approach. In this scenario, we inspected developer D_{844}, classified as Mobile Developer. D_{844} owns 22 public GitHub projects, most of them related to mobile development; four projects have *Swift* as their main programming language, and other five have the word *Android* in their description. Moreover, another four projects are implemented in Java and contain experiments about chart animations in Android. In total, 19 out of 22 projects are directly related to mobile development. In his personal website, D_{844} describes himself as "Android & iOS Engineer — App Maker". Through D_{844}'s GitHub page, we can also reach his LinkedIn profile and find out he has been working with mobile development since 2014.

5.3.2. False Positive Scenario

This group contains developers wrongly predicted as experts in any of the analyzed roles. We inspected the profile of D_{1341}, which is not a DevOps developer, but was predicted as such. In his LinkedIn profile, D_{1341} identifies himself as a "Data Engineer", also mentioning he works in this position since 2016. D_{1341} received most endorsements in *Python* (19), *Data Science*, and *Machine Learning* (18, both). By contrast, D_{1341} has few endorsements in DevOps-specific technologies: one for *AWS* and two for *Linux*. He also obtained certifications in "Machine Learning" and "Scalable Microservices with Kubernetes". D_{1341} is very active on GitHub, performing 711 contributions in the last year. This developer owns 30 projects in 13 different languages, most of them in *Python* (five projects). Further, *Docker* appears in the description of two other projects described as an assistant tool for data analysis. Therefore, due to his limited experience with DevOps-based tools and languages, D_{1341} should not be classified as a DevOps.

5.3.3. False Negative Scenario

Developers are considered False Negative when they are not classified as experts in a given role, despite being so. For this scenario, we examined D_{68}, who is a Frontend developer, but was not identified as such. In his short bio, D_{68} describes himself as a "Frontend Engineer". On GitHub, this developer has performed 1,018 contributions over the last year, where 51% of them were commits. Even though, D_{68} is the owner of only 12 projects; five of them are directly related to frontend technologies (e.g., *JavaScript*, *vue.js*, etc). The remaining projects are implemented

[14] Questions retrieved from `https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall`

in other languages, such as *Java*, *PHP*, and *Python*. Moreover, his projects' descriptions do not mention the use of Frontend technologies. In fact, five projects do not contain any description at all. In other words, the *programming languages* and *short bio* used by D_{68} are the only features directly related to Frontend, but their presence was not enough to label him correctly.

False negative predictions gather our attention as they directly impact the recall's performance. As our classifiers scored lower values for this metric in Mobile and DevOps roles, we decided to extend this analysis by inspecting 10 new randomly selected developers, five in each role. Overall, we observe these developers do not hold specific information to classify them in such roles. For instance, the five selected DevOps developers maintain 70 projects in total. However, only one is mainly implemented using *Shell Script*, which is the second most relevant feature for this role (see Figure 4). One developer mentioned the *devops* keyword in his bio, but this was not reinforced by the other relevant features; only one of his 18 projects is implemented using a relevant language (*Python*). Similar behavior is noted when analyzing Mobile developers. The majority of the projects analyzed are not written in Mobile-based languages, such as *Swift* and *Java*. Interestingly, most of them are implemented with Frontend-based ones, such as *JavaScript* and *TypeScript*. We analyzed this characteristic further and found out that some of these projects rely on cross-platform Mobile frameworks—e.g., *cordova*, *ionic*, etc—which are not in the top-10 most relevant features. In fact, two out of the five developers explicitly stated they are Frontend developers with experience in building mobile web apps. Lastly, relevant keywords like *android*, *ios*, and *mobile*, are indeed mentioned in the repositories of one developer but in a format that prevents their extraction by our bag-of-words tokenization process, e.g., *iOSUIAutomation*, *MyFirstAndroid*, etc.

6. Threats to Validity

The following issues are possible threats to our results:

Target Roles: We analyzed six technical roles in this paper. Although they represent a restricted number of roles, we selected the ones among the most popular, based on a recent large-scale survey conducted by Stack Overflow. Moreover, they cover 64% of the jobs recently listed at Stack Overflow jobs, as mentioned in the Introduction.

Ground Truth: Our dataset is restricted to developers who have profiles on both Stack Overflow and GitHub. Besides, the ground truth is based on developers' public activities on GitHub. Evidently, they represent a subset of all activities of several developers. This limitation may increase false negatives in particular scenarios. For example, a developer is *mobile* but the publicly available data characterizes him/her as *backend* developer. A potential direction for tackling this problem could be the usage of developers' activities from additional data sources. Furthermore, we followed an automated process to label the 2,284 developers in the ground truth. This process is subjected to bias since it relies on the description provided by the developer on Stack Overflow. Further work should consider a semi-automatic labeling strategy [63] or semi-supervised learning techniques [16] to evaluate the accuracy of our labeling process.

Multi-label Classification: Other classification techniques can be applied in multi-label problems, e.g., Label Powerset [28]. However, we rely on the two most used techniques to handle multi-label classification. Moreover, we used two well-known classification algorithms—Random Forest and Naive Bayes. Despite that, further work should consider other classification algorithms, such as XGBoost [17].

Thresholds: As usual in machine learning studies, we acknowledge that our results depend on different thresholds, which are used for example to discard correlated features (Pearson ≥ 0.7), to limit the number of features used in projects' descriptions ([0.04, 0.15]), projects' names ([0.01, 0.25]), projects' topics ([0.01, 0.25]), and short bio ([0.01, 0.20]), as reported in Section 2.4. As a general guideline, we always use conservative thresholds. Furthermore, we experimented with other threshold values in all cases and selected the ones that presented the best results.

7. Related Work

Related work approaches the expertise of software developers in two major standpoints. The first one, known as *domain expertise*, address expertise in the context of specific projects, ignoring developers' technical background. For instance, these works involve the identification of source code file owners [68, 21, 1], core developers [29, 31, 5, 6], or organizational and contribution roles [4, 11, 20]. The second category, known as *technical expertise*, focuses on

revealing developers' technical abilities. Most studies in this second group focus on identfying expertise in 3rd-party technologies [45, 59, 65, 26, 27, 64], source code quality [37, 38], and soft skills [7].

7.1. Domain Expertise

Avelino et al. [6] relied on developers commit activity to estimate truck-factors (i.e., a source code authorship metric). Furthermore, they tested their approach against 119 open source systems to identify their core developers. Honsel et al. [29] use multi-dimensional Hidden Markov Models to predict the contribution behavior of developers. For each developer in a project, they collect data (commits, bug comments, and mailing lists) and classify them into *core, major*, and *minor* classes. Previously, Alonso et al. [3] build a rule-based model to derive developers' expertise based on the frequency of their commits. Their model links a list of categories—extracted from the directory structure of each project—to each developer based on the frequency of changes they perform in each directory. Da Silva et al. [21] introduced a technique to leverage expertise in system's artifacts at source code method-level. The authors rely on a GPU-based approach to leverage such expertise information. Kagdi et al. [32] mine software repositories to recommend developers who master a particular component of a project. Other works investigated expertise in the context of organizational and contribution roles [11, 20]. For example, Bhattacharya et al. [11] analyzed the use of graph models to classify developers on seven organizational roles: *patch tester, assist, triager, bug analyst, core developer, bug fixer,* and *patch-quality improver*. Lastly, Yu and Ramaswamy [68] also mined software repositories to predict developers' roles, but they focus on the interactions among them (e.g., frequency of co-editing). Note that the works in this subsection aim at recommending developers to maintain specific projects. This scenario fits well in the OSS development field. On the other hand, the IT industry is also interested in identifying developers' technical abilities regardless their previous knowledge in specific projects [48, 34, 56].

7.2. Technical Expertise

In fact, most related work addresses the technical expertise problem towards *3rd-party technologies*. In recent work, we [45] proposed two methods to assess the expertise level of 575 developers in three popular JavaScript libraries. Basically, we used features collected from GitHub activity—e.g., commits frequency, code churn, etc.—to classify developers into three expertise levels: *novice*, *intermediate*, and *expert*. Previously, Teyton et al. [59, 57] proposed a search engine to locate—based on syntactical changes—developers with experience on some technical skills, e.g., *Java*, *Linux*, *Apache*, etc. Similarly, Wan et at. [65] developed a probabilistic model to assess developers' coding abilities on software technologies. Lastly, Greene and Fischer [26] also proposed a tool that extracts and presents developers' skills data from GitHub, including skills on programming languages, libraries, and frameworks. Other authors proposed matching the technologies that developers use with the ones required in specific tasks. For example, Hauff and Gousios [27] extracted user activity from GitHub and job advertisements to recommend matching open jobs to developers. They leverage a vector space model—using data extracted from GitHub profiles and job advertisements of a large UK job portal—and rely on cosine similarity to match developers and jobs. Venkataramani et al. [64] proposed a model that, by using developers' activity information on GitHub, recommends Stack Overflow questions for these developers to answer. By contrast, our work focus on detecting developers' expertise into a higher granularity level, i.e., technical roles.

Other works approached technical expertise differently. For exmaple, Agrawal et al. [1] used clustering techniques to analyze which technical factors are more relevant for *backend* developers and *testers*. Differently from our work, the authors consider only commits, timestamps, and file names, whereas we collect a wider set of features, i.e., programming languages, developers' bio, and projects' metadata. Other researchers investigated the importance of *source code quality* when analyzing candidates for opened positions. Marlow et al. [37, 38] interviewed both developers and recruiters to understand which information is more relevant when evaluating online profiles in developer communities. In general, interviewees reported some cues used by them when assessing these profiles, including, which programming languages and libraries do developers use, as well as the readability of their code. Lastly, there are studies mentioning the importance of *soft skills* in the software development environment. Baltes and Diehl [7] presented a conceptual theory structuring the relevant factors in software development. Among the several abilities listed in their work, the authors highlighted the importance that proper personal skills—such as communication, critical thinking, and communication—has on software development.

8. Conclusion

The increasing complexity and relevance of modern software systems are fostering the specialization of software developers in particular components and technologies. As a result, when hiring developers, companies usually do not look for developers with a broad range of skills, but for ones who can work with specific technologies and in specific tasks. Motivated by this context, in this paper we described an approach centered on supervised machine learning to predict six widely popular technical roles of developers nowadays: Backend, Frontend, Mobile, FullStack, Data-Science, and DevOps. Using features extracted from the public profiles of GitHub users, we obtained great results for identifying all six roles in terms of precision (0.88) and AUC (0.89). By contrast, we observed lower results for DevOps and Mobile regarding recall, e.g., 0.06 and 0.34, respectively. Even so, we believe that this study can offer good assistance to technical recruiters as, in their context, identifying correct candidates (precision) is more relevant. As future work, we plan to extend our research in the following lines: (a) improve the classification process by adding new features (e.g., frameworks) and by testing others classifiers (e.g., XGBoost [17]); (b) validate our results (and the practical usage of our approach) through a large-scale survey with real developers, and (c) Implement a browser plug-in to inform the technical roles of GitHub existing profiles.

Replication Package: Our data and our scripts—in a Jupyter Notebook format—are publicly available at the following URL: `https://doi.org/10.5281/zenodo.3986172`.

References

[1] Agrawal, K., Aschauer, M., Thonhofer, T., Bala, S., Rogge-Solti, A., and Tomsich, N. (2016). Resource Classification from Version Control System Logs. In *IEEE International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 249–258.

[2] Ahmed, I. (2018). What Makes a Good Developer ? An Empirical Study of Developers ' Technical and Social Competencies. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 319–321.

[3] Alonso, O., Devanbu, P. T., and Gertz, M. (2008). Expertise Identification and Visualization from CVS. In *International Working Conference on Mining Software Repositories (MSR)*, pages 125–128.

[4] Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *International Conference on Software Engineering (ICSE)*, pages 361–370.

[5] Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2017). Assessing code authorship: The case of the Linux kernel. In *13th International Conference on Open Source Systems (OSS)*, pages 151–163.

[6] Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2019). Measuring and analyzing code authorship in 1+118 open source projects. *Science of Computer Programming*, 1(1):1–34.

[7] Baltes, S. and Diehl, S. (2018). Towards a theory of software development expertise. In *Symposium on the Foundations of Software Engineering (FSE)*, pages 1–14.

[8] Bao, L., Xing, Z., Xia, X., Lo, D., and Li, S. (2017). Who Will Leave the Company?: A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report. In *Working Conference on Mining Software Repositories (MSR)*, pages 170–181.

[9] Behroozi, M., Parnin, C., and Barik, T. (2019). Hiring is Broken: What Do Developers Say About Technical Interviews? In *Visual Languages and Human Centric Computing (VLHCC)*, pages 1–9.

[10] Beyer, S., Macho, C., Pinzger, M., and Di Penta, M. (2018). Automatically classifying posts into question categories on stack overflow. In *International Conference on Program Comprehension (ICPC)*, pages 211–221.

[11] Bhattacharya, P., Neamtiu, I., and Faloutsos, M. (2014). Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 11–20.

[12] Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning Multi-label Scene Classification. *Pattern Recognition*, 37(9):1757 – 1771.

[13] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

[14] Brooks Jr, F. P. (1995). *The mythical man-month*. Addison-Wesley.

[15] Capiluppi, A., Serebrenik, A., and Singer, L. (2013). Assessing technical candidates on the social web. *IEEE Software*, 30(1):45–51.

[16] Chapelle, O., Schlkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. Cambridge, MIT Press.

[17] Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794.

[18] Coelho, J., Valente, M. T., Silva, L. L., and Shihab, E. (2018). Identifying Unmaintained Projects in GitHub. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 14–21.

[19] Constantinou, E. and Kapitsaki, G. M. (2016). Identifying Developers' Expertise in Social Coding Platforms. *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, (1):63–67.

[20] Constantinou, E. and Kapitsaki, G. M. (2017). Developers expertise and roles on software technologies. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 365–368.

[21] Da Silva, J. R., Clua, E., Murta, L., and Sarma, A. (2015). Niche vs. Breadth: Calculating Expertise Over Time Through a Fine-grained Analysis. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 409–418.

[22] DeMarco, T. and Lister, T. (1999). *Peopleware: Productive Projects and Teams*. Dorset House, 2 edition.

[23] Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On Label Dependence and Loss Minimization in Multi-label Classification. *Machine Learning*, 88(1):5–45.
[24] Ford, D., Barik, T., Rand-Pickett, L., and Parnin, C. (2017). The tech-talk balance: what technical interviewers expect from technical candidates. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 43–48.
[25] Goldberg, Y. (2017). *Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies)*. Morgan & Claypool Publishers, 1th edition.
[26] Greene, G. J. and Fischer, B. (2016). CVExplorer: identifying candidate developers by mining and exploring their open source contributions. In *International Conference on Automated Software Engineering (ASE)*, pages 804–809.
[27] Hauff, C. and Gousios, G. (2015). Matching GitHub Developer Profiles to Job Advertisements. In *Working Conference on Mining Software Repositories (MSR)*, pages 362–366.
[28] Herrera, F., Charte, F., Rivera, A. J., and del Jesus, M. J. (2016). *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Springer.
[29] Honsel, V., Herbold, S., and Grabowski, J. (2016). Hidden Markov Models for the Prediction of Developer Involvement Dynamics and Workload. In *International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, pages 1–10.
[30] Huang, W., Mo, W., Shen, B., Yang, Y., and Li, N. (2016). CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities. In *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 87–92.
[31] Joblin, M., Apel, S., Hunsen, C., and Mauerer, W. (2017). Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. In *International Conference on Software Engineering (ICSE)*, pages 164–174.
[32] Kagdi, H., Hammad, M., and Maletic, J. I. (2008). Who can Help me with this Source Code Change? In *International Conference on Software Maintenance (ICSM)*, pages 157–166.
[33] Kim, S., E. James Whitehead, J., and Zhang, Y. (2008). Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196.
[34] Litecky, C., Aken, A., Ahmad, A., and Nelson, H. J. (2008). Mining for Computing Jobs. *IEEE Software*, pages 78–85.
[35] Luaces, O., Díez, J., Barranquero, J., del Coz, J. J., and Bahamonde, A. (2012). Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 1(4):303–313.
[36] Madjarov, G., Kocev, D., Gjorgjevikj, D., and Deroski, S. (2012). An Extensive Experimental Comparison of Methods for Multi-label Learning. *Pattern Recognition*, 45(9):3084–3104.
[37] Marlow, J. and Dabbish, L. (2013). Activity Traces and Signals in Software Developer Recruitment and Hiring. In *Conference on Computer Supported Cooperative Work (CSCW)*, pages 1–11. ACM Press.
[38] Marlow, J., Dabbish, L., and Herbsleb, J. (2013). Impression Formation in Online Peer Production : Activity Traces and Personal Profiles in GitHub. In *Conference on Computer Supported Cooperative Work (CSCW)*, pages 117–128.
[39] Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417.
[40] McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *International Conference on Machine Learning*, pages 41–48.
[41] McDowell, G. (2015). *Cracking the Coding Interview: 189 Programming Questions and Solutions*. CareerCup, LLC.
[42] Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., and Zimmermann, T. (2013). Local Versus Global Lessons for Defect Prediction and Effort Estimation . *IEEE Transactions on Software Engineering*, 39(6):822–834.
[43] Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes – which naive bayes? In *Conference on Email and Anti-Spam (CEAS)*, pages 1–9.
[44] Montandon, J. E., Politowski, C., Silva, L. L., Valente, M. T., Petrillo, F., and Guéhéneuc, Y.-G. (2020). What Skills do IT Companies look for in New Developers? A Study with Stack Overflow Jobs. *Information and Software Technology*, pages 1–6.
[45] Montandon, J. E., Silva, L. L., and Valente, M. T. (2019). Identifying Experts in Software Libraries and Frameworks among GitHub Users. In *International Conference on Mining Software Repositories (MSR)*, pages 1–12.
[46] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
[47] Peters, F., Menzies, T., and Marcus, A. (2013). Better Cross Company Defect Prediction . In *Working Conference on Mining Software Repositories (MSR)*, pages 409–418.
[48] Prabhakar, B., Litecky, C. R., and Arnett, K. (2005). IT skills in a tough job market. *Communications of the ACM*, 48(10):91–94.
[49] Provost, F. and Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine learning*, 42(3):203–231.
[50] Provost, F. and Fawcett, T. (2013). *Data Science for Business: What You Need to Know About Data Mining and Data-analytic Thinking*. O'Reilly Media, Inc., 1st edition.
[51] Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier Chains for Multi-label Classification. *Machine Learning*, 85(3):333–359.
[52] Robles, G., Gonzalez-Barahona, J. M., and Herraiz, I. (2009). Evolution of the core team of developers in libre software projects. In *Working Conference on Mining Software Repositories (MSR)*, pages 167–170.
[53] Sarma, A., Chen, X., Kuttal, S., Dabbish, L., and Wang, Z. (2016). Hiring in the global stage: Profiles of online contributions. In *International Conference on Global Software Engineering (ICGSE)*, pages 1–10.
[54] Saxena, R. and Pedanekar, N. (2017). I Know What You Coded Last Summer. In *Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 299–302.
[55] Shirabad, J. S., Lethbridge, T. C., and Matwin, S. (2003). Mining the maintenance history of a legacy software system. In *International Conference on Software Maintenance (ICSM)*, pages 95–104.
[56] Singer, L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. (2013). Mutual assessment in the social programmer ecosystem. In *ACM Computer Supported Cooperative Work (CSCW)*, pages 103–116.
[57] Teyton, C., Palyart, M., Falleri, J.-R., Morandat, F., and Blanc, X. (2014a). Automatic Extraction of Developer Expertise. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10.

[58] Teyton, C., Palyart, M., Falleri, J.-R., Morandat, F., and Blanc, X. (2014b). Automatic Extraction of Developer Expertise Categories and Subject Descriptors. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10.

[59] Teyton, Cedric and Falleri, Jean-Rémy and Morandat, Floréal and Blanc, X. (2013). Find your Library Experts. In *Working Conference on Reverse Engineering (WCRE)*, pages 202—-211.

[60] Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What are the Characteristics of High-rated Apps? A Case Study on Free Android Applications. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 301–310.

[61] Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). *Mining Multi-label Data*, pages 667–685. Springer.

[62] Tsoumakas, G., Katakis, I., and Vlahavas, I. (2011). Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089.

[63] Vajda, S., Rangoni, Y., and Cecotti, H. (2015). Semi-automatic ground truth generation using unsupervised clustering and limited manual labeling: Application to handwritten character recognition. *Pattern Recognition Letters*, 58:23–28.

[64] Venkataramani, R., Gupta, A., Asadullah, A., Muddu, B., and Bhat, V. (2013). Discovery of technical expertise from open source code repositories. In *International Conference on World Wide Web (WWW)*, pages 97–98.

[65] Wan, Y., Chen, L., Xu, G., Zhao, Z., Tang, J., and Wu, J. (2018). SCSMiner: Mining Social Coding Sites for Software Developer Recommendation with Relevance Propagation. *World Wide Web*, pages 1–21.

[66] Wang, Z., Sun, H., Fu, Y., and Ye, L. (2017). Recommending crowdsourced software developers in consideration of skill improvement. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 717–722.

[67] Ye, Y. and Kishida, K. (2003). Toward an understanding of the motivation open source software developers. In *International Conference on Software Engineering (ICSE)*, pages 419–429.

[68] Yu, L. and Ramaswamy, S. (2007). Mining CVS Repositories to Understand Open-Source Project Developer Roles. In *International Workshop on Mining Software Repositories (MSR)*, pages 1–10.

[69] Zhang, M.-L. and Zhou, Z.-H. (2014). A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.