A Bot Identification Model and Tool Based on GitHub Activity Sequences

Natarajan Chidambaram, Alexandre Decan[1], Tom Mens

[a]*Software Engineering Lab, University of Mons, Mons, 7000, Belgium*

Abstract

Identifying whether GitHub contributors are automated bots is important for empirical research on collaborative software development practices. Multiple such bot identification approaches have been proposed in the past. In this article, we identify the limitations of these approaches and we propose a new binary classification model, called BIMBAS, to overcome these limitations. To do so, we propose a new ground-truth dataset containing 1,035 bots and 1,115 humans on GitHub. We train BIMBAS on a wide range of features extracted from the activity sequences of these GitHub contributors. We show that the performance of BIMBAS (in terms of precision, recall, F1 score and AUC) is comparable to state-of-the-art bot identification approaches, while being able to identify bots engaged in a wider range of activity types. We implement RABBIT, an open-source command-line bot identification tool based on BIMBAS. We demonstrate its ability to be used at scale, and show that its efficiency outperforms the state-of-the-art.

Keywords: bot identification, classification model, GitHub, collaborative software development

1. Introduction

Collaborative software development practices through social coding platforms such as GitHub enable developers to jointly engage in software development activities irrespective of their geographical location [1, 2]. Some of these activities can be quite repetitive and effort-intensive such as updating dependencies, reviewing code, checking code quality, testing, processing issues and pull requests, publishing releases, onboarding newcomers, and interacting with users [3, 4]. This justifies the need for tools to automate these tasks and activities, such as automated workflows and development bots. This article focuses on the latter automation practice in the context of GitHub.

GitHub distinguishes different types of actors that can perform activities on GitHub. Actors of type Organization are used to control a collection of repositories, members and teams, but will be ignored in the current paper. Actors of type Bot perform activities on behalf of their associated GitHub Apps, which are automated tools that extend GitHub functionality. Actors of type User are associated to individual contributors that use their GitHub account to own repositories and

Email addresses: natarajan.chidambaram@umons.ac.be (Natarajan Chidambaram), alexandre.decan@umons.ac.be (Alexandre Decan), tom.mens@umons.ac.be (Tom Mens)

[1]F.R.S.-FNRS Research Associate

generate new content (such as commits, issues, pull requests, comments, reviews and many more). While this actor type is mainly intended for **humans**, nothing prevents it to be used to automate repetitive activities (which we will refer to as **bot accounts**). While **bot actors** (serving Apps) have "[bot]" in their name and are trivial to detect through the Users API, GitHub does not allow distinguishing *humans* from *bot accounts*, justifying the need for bot identification approaches.

In the remainder of this paper, we reserve the term **bots** to collectively refer to *bot actors* and *bot accounts*, and the term **contributors** to collectively refer to *humans* and *bots*.

Bots have been shown to belong to the top contributors in certain repositories [5]. Their prevalence [6] is challenging for researchers conducting quantitative socio-technical analyses on software repositories since neglecting the presence of bots might lead to biased or incorrect conclusions [7, 8]. The ability to identify bots is also important for empirical studies about the role played by bots in collaborative software development [3]. Last but not least, communities and funding organisations can benefit from bot identification tools to correctly recognise, accredit and sponsor human activity [9].

The inability to distinguish bot accounts from humans in GitHub has lead to the proposal of several bot identification approaches [10, 11, 12, 13, 14]. However, each approach has specific shortcomings, such as focusing on a limited subset of activities, the need for a substantial amount of API queries or data to be downloaded, the use of computationally costly features to identify bot accounts, or even the absence of a publicly available tool or script to execute the approach on recent accounts and repositories. These limitations make existing bot identification approaches difficult to use in practice for identifying large sets of contributors, highlighting the need for a bot identification tool that can be used at scale. Therefore, the current article addresses the following goals:

G1. Creating a ground-truth dataset of bots and humans. Developing a new bot detection approach requires a ground-truth dataset containing a large amount of humans and bots. The construction of a new dataset is motivated by the fact that older existing datasets are either restricted to a limited set of event types, or not sufficiently accurate. Also, creating such a dataset takes time and effort. We propose a manually curated ground-truth dataset of 2,150 contributors that were active on GitHub as of 3 May 2024. This dataset contains 1,115 humans and 1,035 bots (of which 242 are Apps and 793 are bot accounts).

G2. Identifying the limitations of existing bot identification approaches. We found four GitHub bot identification approaches in the research literature that can be applied in practice, either because they rely on simple heuristics that are easy to implement; or because they come with a documented implementation or even a directly installable and usable tool. We study the internal workings of each approach and identify their limitations. We execute these approaches on the ground-truth dataset to compare their performance in terms of precision, recall and F1 score. We also compare their efficiency in terms of amount of data downloaded, time taken to execute and number of required GitHub API queries.

G3. Creating a bot identification model based on activity sequences. To overcome the limitations identified in G2 we propose BIMBAS, a novel binary classification model based on activity sequences of GitHub contributors. BIMBAS relies on a wide range of features extracted from the contributor activity sequences to accurately detect bots. BIMBAS makes use of Gradient Boosting model and achieves a performance comparable to state-of-the-art bot identification approaches.

G4. Developing an efficient bot identification tool. To enable practitioners and researchers to benefit from the BIMBAS bot identification model we develop RABBIT, an open-source command-line-based bot identification tool. We compare RABBIT to the approaches identified for G2, in terms of precision, recall, execution time, data downloaded and number of API queries. Our results show that RABBIT can be used efficiently to identify bots in GitHub at scale.

The remainder of this article is structured as follows. Section 2 presents the related work on GitHub bot usage, analysis and identification. Sections 3, 4, 5 and 6 align with goals G1, G2, G3 and G4 respectively. Section 7 presents the main threats to validity of our research. Finally, Section 8 concludes.

2. Related Work

This section presents the related work on bots in collaborative software development on GitHub. Section 2.1 focuses on quantitative and qualitative research related to the use of bots, the tasks they carry out, the way they interact with human contributors, and the benefits, drawbacks and impact of using them. Section 2.2 provides an overview of the existing bot identification approaches that have been reported in the scientific literature.

2.1. Analysing the use of development bots

Based on developer perceptions of productivity [15], Storey et al. [16] reflected on the positive and negative impact of bots in social coding platforms. They identified the qualities of bots that can improve developer productivity. They found that bots are used to automate tedious tasks, help developers to keep up with the flow, improve decision-making by capturing and analysing data relevant to decisions, support team communication and task coordination. However, they also noticed that bots might reduce team interaction, bring interruptions and distractions, and might not accommodate cultural changes in the organisation. Erlenhov et al. [17] interviewed 21 developers and performed an online survey with another 111 developers to identify three personas among bots, focusing on autonomy, chat interfaces and smartness. They found that bots do not tend to go beyond simple automation tools and chat interfaces. This highlighted the absence of smart, general-purpose bots with a potential transformative effect on software projects.

Ghorbani et al. [18] qualitatively analysed the characteristics affecting developer preferences for interacting with bots in PRs. They formulated 13 questions to collect data based on contributor experiences of using bots in their respective communities. By interviewing 12 participants from four different open-source communities they identified seven themes on how software developers perceive software bots: attitude, autonomy, persona, task, feelings, project norm, and role. Among these, they found autonomy and persona to exert more influence in shaping developer perception of bots. To further study this influence of autonomy and persona, they conducted surveys among 56 participants and recommended that developers should have options to: (i) scale the autonomy of bots; (ii) select and change bot personae; and (iii) improve project-specific feedback on bot behaviour and developer preferences.

Wessel et al. [3] analysed the usage of 48 different bots in 93 GitHub projects and identified 12 different tasks performed by bots, such as ensuring license agreements, reviewing code, welcoming newcomers, running automated tests and creating issues and pull requests (PR). To analyse the changes in PR characteristics before and after bot adoption, they looked into PRs belonging

to 44 projects for a duration of 6 months before and 6 months after bot adoption. They found statistically significant differences in terms of number of commits, number of changed files, number of comments and time to close PR. Furthermore, through surveys involving 205 participants (developers and integrators), they identified 16 challenges in using bots in PRs (e.g., bots may take wrong actions and provide non-comprehensive/poor feedback) and reported 23 improvements (e.g., enhancing user interaction and improving code reviews). In another study, Wessel et al. [19] interviewed 21 practitioners to identify challenges that need to be taken into account when developing bots that interact in PRs. They classified challenges into 25 categories (e.g., introducing noise, providing non-comprehensive feedback, difficulty in setting up configuration files, ensuring that the bot is properly performing its intended task, and restricted bot actions). To mitigate the noise introduced by bots, they developed a meta-bot that summarises and customises other bots' actions to reduce information overload incurred by their use [20]. They found 22 design strategies and grouped them into four categories for developing their meta-bot and one category for modifying GitHub's interface: (a) information management (e.g., summarisation of bot comments); (b) newcomer assistance (e.g., welcoming message); (c) notification management (e.g., schedule bot notification); (d) spam and failure management (e.g., bug reports); and (e) platform support (e.g., separating bot comments).

Wyrich et al. [21] analysed the PRs created and commented by humans and bots to understand the difference in priority given to PRs created by bots and humans. They found that PRs created by humans received faster response and 72.53% of them were merged. PRs created by bots took significantly more time to receive a response and only 37.38% of them were merged, even though they contained fewer changes on average than PRs of humans. Zhang et al. [22] quantitatively studied the factors influencing PR latency and the change in these factors with a change in context (e.g., time, project and developer). They identified 47 features that influence PR latency. They classified these features into developer characteristics (e.g., *Is this the developer's first PR?* and *Do contributor and integrator have the same affiliation?*), project characteristics (e.g., team size and project age), and PR characteristics (e.g., whether it is bug fix, and the number of PR comments). As they found a widespread usage of bots in PRs, they conducted a case study to identify the impact of bots on PR latency. Executing an existing bot identification tool to identify bots in 3.3M+ PRs belonging to 11K+ GitHub projects obtained from GHTorrent, they observed that more than one out of three PR comments were made by bots. Further, they found that the presence of comments posted by humans were more important than those of bots in explaining PR latency.

Khatoonabadi et al. [23] qualitatively studied the potential benefits and drawbacks of using stale bot for pull-based development. By observing PRs for a duration of two years (1 year before and 1 year after adopting stale bot) in 20 large and popular open-source projects, they concluded that: (i) stale bot usage widely varies across projects (in terms of intervening, warning and closing PRs); (ii) stale bot adoption is associated with faster reviews, resolutions, fewer updates in merged PRs, and a decreased number of active project contributors; and (iii) stale bot tends to be active in more complex PRs (in terms of #commits, #files changed and #lines changed), PRs submitted by first time contributors and PRs with a lengthy review process. Overall, they found that stale bot can help projects deal with a backlog of unresolved PRs and also improve the PR review process, but may negatively affect project newcomers.

2.2. Bot identification approaches

Several bot identification approaches have been proposed in the literature, either as an explicit contribution or implicitly as part of the data filtering process of empirical studies. We provide a

brief overview of these approaches here. A more in-depth explanation of the most prominent bot identification approaches will be presented in Section 4.

A straightforward approach is to rely on a predefined list of bots that have been manually identified and published as ground-truth dataset [6, 10, 21]. Such ground truths are indispensable for empirical research, and for developing and evaluating new bot identification models. However, they should not be applied "in practice" since they are inevitable incomplete (by construction) and therefore can only be used to find a subset of all potential bots that may be present. Moreover, ground truth datasets become outdated over time, including bots that may no longer exist today.

Another straightforward approach is a Name-Based Heuristic (henceforth abbreviated to NBH) based on simple regular expressions to detect whether a contributor name contains specific substrings (e.g., "bot" or "automate"). This heuristic has been used with different variations in the research literature [21, 24].

Several bot identification approaches have been proposed based on machine learning classification models. Dey et al. [11] developed BIMAN, an ensemble model combining three different models to identify bots. The first model, called BIN (for Bot In Name) is a variant of NBH to check for the "bot" substring preceded and/or followed by non-alphabetic characters (e.g., sre-bot, github-actions[bot]). The second model relies on similarity in commit messages, based on the assumption that the textual variation in commit messages is lower for bots than for humans. The third model is a Random Forest classifier trained on six features related to the modifications made to files in commits: (i) number of files changed by author across commits; (ii) number of unique file extensions in all commits; (iii) standard deviation of number of files per commit; (iv) mean number of files per commit; (v) number of unique projects commits have been associated with; and (vi) median number of projects commits have been associated with. BIMAN can be applied on any GitHub account. However, if the account is not involved in committing, then only the BIN model will be used.

Golzadeh et al. [10] developed BoDeGHa, a tool that relies on a Random Forest classification model to identify bots in a given repository based on the repetitive comments they create in issues and PRs. Chidambaram et al. [25] improved BoDeGHa by relying on the wisdom of the crowd principle to address the diverging predictions that BoDeGHa could make for the same contributor when applied to multiple repositories. Golzadeh et al. [12] proposed BoDeGiC, a variation of BoDeGHa to identify bots based on their commit messages.

Abdellatif et al. [13] proposed BotHunter, a tool that relies on a Random Forest classification model to detect bots based on the GitHub account login, GitHub profile information, account activity based on GitHub events, and similarity in comments and commit messages. Inspired by previous bot identification approaches, Liao et al. [14] proposed BDGOA, a Random Forest classifier that uses features related to GitHub account information, account activity based on GitHub event sequences, and text similarity in PR, issue, and commit comments, and commit and release messages.

Chidambaram et al. [26] proposed to consider 24 different activity types as a basis for features to distinguish bots from human contributors. Among others, they observed statistically significant differences in their number of activity types, the time taken to perform these activities, the dispersion in the number of activity types performed in each repository, and the median time taken to switch between repositories to perform another activity. They suggested to leverage these and related features to provide a new activity-based bot identification model, which is the purpose of the current article.

3. G1: Creating a ground-truth dataset of bots and humans

Our first research goal is to create a ground-truth dataset of active bots and humans that will serve as the basis for creating a new bot identification model (G3) and tool (G4). The construction of a new dataset is motivated by the fact that older existing datasets are either restricted to a limited set of GitHub event types, or not sufficiently accurate.

To reach goal G1, we started from three existing data sources to increase the likelihood of finding bots. We complemented this with the top contributors of several popular repositories. Since we aim to use the ground-truth dataset as a basis for a bot identification model (goal G3) based on contributor activity sequences, we impose as an inclusion filter that contributors need to have been recently active on GitHub.[2]

To ensure the accuracy of our ground-truth dataset we only include contributors that are manually checked by two raters. To do so, we applied a multi-rater labelling process to determine the contributor type. Two raters independently inspected the contributors' GitHub profiles, as well as the activities they performed on GitHub. Based on this information they labelled the contributor as either *bot* or *human* and discussed together in case of disagreement. If not enough information was available to come to a decision, the account was discarded.

As a starting point for our new ground-truth dataset we relied on the dataset of Chidambaram et al. [4], initially containing 350 bots and 620 humans that were manually labelled. Applying the inclusion filter we retained 271 bots and 501 humans.

To further complement our ground-truth dataset, we considered the dataset used by Wyrich et al. [21] in the context of an empirical study of the difference between bots and humans in the proportion of PRs being merged by them. They proposed a dataset of 4,654 bots, but a large majority of them (86.1%) did not pass our inclusion filter. Following the multi-rater labelling process we manually labelled the remaining 645 contributors, resulting in 506 bots and 139 humans.

Cardoen et al. [27] provided a dataset of the commit histories of GitHub Actions workflow files in 32K+ repositories. The dataset contains the names of more than 62K contributors. Given the automated nature of GitHub Actions workflows, we expect this dataset to contain many bots. Ignoring all the contributors that were no longer active, we started by analysing all the contributors having the substring "bot" in their name. Following the multi-rater labelling process, we identified 178 bots and 45 humans. To further expand our list of bots, we applied a state-of-the-art identification tool [13] on randomly selected contributors. Considering only the contributors identified as bots, we followed the multi-rater labelling process until we reached 72 bots (leading to a total of 250 bots). We complemented them with 205 humans to reach an equal amount of 250 humans.

Golzadeh et al. [5] reported that many bots are among the top contributors in GitHub repositories. Driven by this insight, we selected 10 popular GitHub repositories known by the authors[3] and analysed the top 30 contributors (as reported by GitHub) in each of them. After ignoring duplicates and removing contributors who are no longer active, we followed the multi-rater labelling process and identified and included another 8 bots and 225 humans in our dataset.

Table 1 summarises the final ground-truth dataset we obtained after following all these steps. Overall, the dataset includes 2,150 contributors, of which 1,115 humans and 1,035 bots. 791 of

[2]The exact definition and rationale of "active" will be provided in Section 5.1.

[3]Each of the considered repositories contained more than 500 watchers, more than 8k forks, and more than 25k stars.

Table 1: GitHub contributors included in the ground-truth dataset

data sources	total	#bots	#humans
1. Chidambaram et al. [4]	772	271	501
2. Wyrich et al. [21]	645	506	139
3. Cardoen et al. [27]	500	250	250
4. Top contributors in 10 popular GitHub repositories	233	8	225
total	**2,150**	**1,035**	**1,115**

these bots are bot accounts, while the remaining 242 are Apps. It has been made publicly available at `https://zenodo.org/records/12588134` to be used by researchers and practitioners.

4. G2: Identifying the limitations of existing bot identification approaches

Research goal **G2** aims to compare the performance and efficiency of the most prominent approaches in order to identify their main limitations. Based on these limitations, goal **G3** (that will be presented in Section 5) will come up with an improved bot identification model that mitigates the identified limitations.

4.1. Excluded bot identification approaches

Section 2.2 provided an overview of the bot identification approaches for GitHub that have been proposed in the scientific research literature. For practical reasons, we could only include the following approaches in our comparison: NBH, BoDeGHa, BoDeGiC and BotHunter.

We excluded BIMAN [11] since it requires as input specific files that need to be obtained from the *World of Code* [28] dataset. This dataset contains the commits, blobs, trees and folders of open-source git software repositories. However, the *World of Code* dataset is not publicly available since access should be granted on an individual basis. Because of this practical limitation of BIMAN, we could not include it in our comparison.

We excluded BDGOA [14] since we did not find any mention of a replication package, dataset or executable tool that could be used to replicate or evaluate the proposed model. As such, we were not able to replicate the approach and could not include it as part of our comparison.

4.2. Experimental setup

To compare the selected bot identification approaches, we will use a *test set* corresponding to 40% of the contributors contained in the ground-truth dataset. The remaining 60% of contributors are part of the *training set* that is reserved for training the new BIMBAS model that will be proposed in **G3**. We use stratified splitting to preserve the proportion of bots and humans in the training and test set. Overall, the *training set* includes 621 bots and 669 humans, whereas the *test set* includes 414 bots and 446 humans.

Note that some approaches (BoDeGHa and BoDeGiC) work at the repository level, i.e., they require as input a repository and optionally a set of contributor names to identify which of the repository contributors are bots. By default, all contributors to the given repository will be analysed. If a contributor is active in multiple repositories, the prediction made by these approaches may depend on the repository that has been selected for analysis. For our experiment we selected the repository on which the contributor was the most active recently based on the GitHub Events API.[4]

[4] `https://docs.github.com/en/rest/activity/events#list-public-events-for-a-user`

To evaluate the performance of the considered bot identification approaches, we rely on the usual metrics of precision (P), recall (R), F1-score and their weighted counterparts. The use of the weighted variant is motivated by the fact that our dataset is slightly imbalanced (51.9% humans and 48.1% bots). Table 2 recalls the definitions of the performance metrics, with TP referring to the number of *true positives* (bots that are correctly classified as bots), TN the number of *true negatives* (humans that are correctly classified as humans), FP the number of *false positives* (humans that are wrongly classified as bots) and FN the number of *false negatives* (bots that are wrongly classified as humans).

Table 2: Definition of precision, recall, F1-score and their weighted variants.

population	**precision** (P)	**recall** (R)	**F1-score** ($F1$)
bots (B)	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{2*P(B)*R(B)}{P(B)+R(B)}$
humans (H)	$\frac{TN}{TN+FN}$	$\frac{TN}{TN+FP}$	$\frac{2*P(H)*R(H)}{P(H)+R(H)}$
weighted ($B \cup H$)	$\frac{P(B)*\|B\|+P(H)*\|H\|}{\|B\|+\|H\|}$	$\frac{R(B)*\|B\|+R(H)*\|H\|}{\|B\|+\|H\|}$	$\frac{2*P(B\cup H)*R(B\cup H)}{P(B\cup H)+R(B\cup H)}$

Table 3 summarises the performance metrics for each considered bot identification approach. We additionally report the number of "unknown" contributors, i.e., those contributors whose type could not be determined because of intrinsic limitations of the considered approach. The presence of unknown contributors has a direct effect on the recall: a higher number of unknowns will lead to a lower recall.

Table 3: Performance of considered bot identification approaches on 860 contributors.

	bots				**humans**				**weighted**		
approach	P	R	F1	unknown	P	R	F1	unknown	P	R	F1
NBH	.803	.671	.732	0	.735	.848	.788	0	.768	.763	.761
BoDeGHa	.918	.512	.657	169	.861	.457	.597	223	.891	.486	.629
BoDeGiC	.812	.271	.406	278	.747	.159	.262	349	.785	.224	.346
BotHunter	.967	.928	.947	1	.937	.971	.954	0	.952	.950	.950

Table 4: Efficiency of considered bot identification approaches on 860 contributors

approach	**data downloaded**	**time**	**API queries**
NBH	-	0.01 sec	-
BoDeGHa	3.83 GB	7.7 h	10,222
BoDeGiC	23.3 GB	23.1 h	-
BotHunter	0.261 GB	20.8 h	37,240

Table 4 reports on the efficiency of the considered approaches in terms of amount of data downloaded and execution time. Each approaches was executed on the same system with an Intel Xeon W-1290P 3.7 GHz CPU processor running Fedora 34 (Server Edition). The downloaded data is measured using the network monitoring tool NetHogs (version 0.8.7). We also count, for those approaches relying on GitHub, the number of API queries used. Such information is quite relevant, since GitHub imposes an API rate limit of maximum 5,000 queries per hour. Exceeding this limit results in a waiting time until the query limit is reset by GitHub. For the approaches that use the GitHub API, we report on this waiting time.

4.3. NBH: Name-Based Heuristic

The name-based heuristic (NBH) naively identifies a contributor as *bot* if the substring "bot" appears somewhere in its name. Many variations of this heuristic have been used in research literature [24, 21] (e.g., only considering substring "bot" at the end of the name, considering related terms like "automate", requiring that the substring is preceded or followed by a non-alphabetic character, and so on). However, Golzadeh et al. [29] executed this approach on 540 contributors belonging to 27 repositories and obtained a recall of $R = 0.520$ for detecting bots. This reveals an important limitation of this heuristic, namely that it yields many false negatives, i.e., bots that do not contain the substring "bot" in their name (e.g., bors, micronaut-build, id-jenkins, strapi-cla). Conversely, this heuristic leads to false positives when humans have the string "bot" as part of their name. For example, the last names Cabot and Abbott are not uncommon for humans. For instance, NBH falsely identified 68 humans as bot due to the presence of "bot" in their name, leading to a recall of $R = 0.848$ for humans. Line NBH of Table 3 summarises the performance results, with an overall weighted precision $P = 0.768$ and recall $R = 0.763$, confirming the presence of many false positives and false negatives.

From Table 3, one can observe that the recall for bots ($R = 0.671$) is higher than the recall of $R = 0.520$ that was observed by Golzadeh et al. [29]. This is because the recall for NBH depends only on the proportion of bots that have 'bot' in their name. So, the recall reported for this approach in Table 3 is an overestimation since, by construction, our dataset (in Section 3) contains a higher proportion of such bots.

On the positive side, the NBH approach is extremely efficient in time and memory. Since it only relies on contributor names, it does not require any other data to be downloaded or any API queries to be executed, implying that bots can be identified almost instantly.

4.4. BoDeGHa

Golzadeh et al. [10] developed a bot identification tool that uses a Random Forest binary classification model to identify bots. It works at the level of individual GitHub repositories, i.e., it only considers the activities of contributors within that repository. For each contributor, BoDeGHa retrieves their issue and pull request comments and compute five features: (i) the string distance between comments[5], (ii) the number of comment patterns (sets of very similar comments), (iii) the (Gini) inequality between comment patterns, (iv) the total number of comments, and (v) the number of empty comments.

A limitation of BoDeGHa is that it restricts itself to issue and pull request comments, making it unable to detect bots that do not engage in such activities. Even when a contributor is engaged in such activities, BoDeGHa requires at least 10 comments (by default) to provide a prediction. This explains why BoDeGHa could not give a prediction for 392 contributors (169 bots and 223 humans) during its execution on the test set (see Table 4).

While evaluating BoDeGHa, we identified some overly restrictive condition to avoid exceeding the API rate limit. We relaxed this condition in order to optimise BoDeGHa's execution and created a pull request that has been merged into BoDeGHa's GitHub repository [6]. We executed this improved version BoDeGHa with its default parameters on the contributors of the test set. It took 7.7 hours (464 minutes), required 10,222 GitHub API queries and downloaded 3.83 GB of data to provide its predictions. A reason for BoDeGHa's high execution time is that it depends on

[5] using a combination of Levenshtein and Jaccard distance

[6] `https://github.com/mehdigolzadeh/BoDeGHa/pull/25`

the combination of features that are computationally costly (calculating Levenshtein and Jaccard distance). Overall, BoDeGHa achieved a weighted precision of $P = 0.891$. Since BoDeGHa could make a prediction only for 468 contributors, it achieved a very low recall of $R = 0.486$.

4.5. BoDeGiC

Golzadeh et al. developed BoDeGiC [12] in a follow-up work and alternative to BoDeGHa. BoDeGiC uses an approach that is similar to the one of BoDeGHa, but applied to commit messages rather than to issue and pull request comments. Unlike BoDeGHa, BoDeGiC does not take a GitHub repository as input but can work directly with a given (local) git repository. By doing so, it only requires data stored in the .git folder to make its predictions and does not need to use GitHub API.

We executed BoDeGiC on the contributors of the test set. Since cloning all the repositories is time-consuming and requires downloading a large amount of data, and since BoDeGiC only requires the commit messages, we adapted the `git` command used to clone repositories to exclude blobs. [7] With this command, for example, the *servo/servo* repository on GitHub took only 10.5 seconds to be cloned (and 114MB) while cloning this repository with all blobs would have taken 128 seconds (and 1.14GB). The process of cloning all considered repositories took one hour while predicting the type of contributor took 22.1 hours, so in total it took 23.1 hours to execute and required 23.3GB of data. BoDeGiC was able to provide a prediction for 233 contributors. It could not provide a prediction for the remaining 627 contributors (278 bots and 349 humans) because they do not reach the minimum 10 commits required by BoDeGiC to make a prediction. Although, BoDeGiC achieved an overall weighted precision of $P = 0.785$, it has very low recall of $R = 0.224$ due to the latter reason.

4.6. BotHunter

Abdellatif et al. [13] proposed BotHunter, a Python script that executes a bot identification model based on a Random Forest classifier. BotHunter takes as input the name of a contributor, and relies on the GitHub API to retrieve its data, as is the case for BoDeGHa.

The model uses 19 features (three of which are in common with BoDeGHa and BIMAN) to identify bots based on profile information (*account login*, *account name*, *tag* and *bio*, *number of followers* and *number of followings*) and account activity (total number of repository, issue, PR and commit events, unique number of repository, issue, PR and commit events, median events per day and median response time). Among these considered features, the top five most important features were *account name*, *account login*, *number of followers*, *issue/PR comments similarity*, and *median events per day*. Two of these top features (*account name* and *account login*) are actually variants of the NBH approach presented in Section 4.3, since both features check for the presence of substring "bot" or "automate".

While evaluating BotHunter, we discovered that it retrieves only 30 events per API call while the GitHub API allows to retrieve up to 100 events at once. We therefore adapted BotHunter to retrieve up to 100 events per API call, thus reducing the number of API queries and reducing its execution time since the hourly API rate limit is reached less frequently. We created a pull request of these modifications to the GitHub repository of BotHunter[8] which was accepted by the repository maintainer and is now integrated in the latest release of BotHunter.

[7] `git clone --filter=blob:none --no-checkout <repo>`

[8] `https://github.com/ahmad-abdellatif/BotHunter/pull/5`

We applied this modified version of BotHunter on the contributors of the test set. With the notable exception of a contributor that no longer exists on GitHub, BotHunter was able to provide predictions for all the other contributors. Table 3 shows that BotHunter exhibits the best performance, reaching a precision of $P = 0.952$ and recall of $R = 0.950$. On the other hand, Table 4 shows that BotHunter consumed 37,240 queries, downloaded 261 MB data and took 20.8 hours to provide the predictions. This long execution time includes the 50 minutes of waiting time due to the fact that BotHunter reached the GitHub API rate limit seven times.

4.7. Summary

All the considered approaches to detect bots were found to suffer from several limitations that make them difficult to use in practice, and more specifically at large-scale. On the one hand, NBH is fast to determine the type of a contributor but exhibits low precision and recall compared to more advanced approaches. On the other hand, these more advanced approaches, while being more accurate, require a large amount of data and take a lot of time when applied on hundreds or thousands of contributors, either because they need a lot of API queries or because the features they need to make their decisions are costly to compute. This makes them impractical to be used at scale. Additionally, many of the considered approaches require, or specifically target, contributors to be involved (at least) in specific activity types (e.g., committing for BoDeGiC or commenting for BoDeGHa), making them unable to accurately determine the type of the contributors that are not involved in these activities. To some extent, deciding which approach should be used is basically a trade-off between efficiency (at scale) and performance.

These limitations highlight the need for a more practical bot identification approach that: (i) exhibits a good performance in terms of precision and recall; (ii) does not restrict itself to specific activity types; (iii) does not require a large amount of data nor API queries to remain below the hourly rate limit; (iv) is fast to compute by avoiding the use of computationally intensive features.

In the remaining of this article, we propose BIMBAS, an approach based on activity sequences able to accurately detect bots involved in various activity types, and RABBIT, an efficient tool (in terms of execution time, API queries and amount of data) implementing the BIMBAS bot detection approach. We will show that RABBIT does not suffer from the above-mentioned limitations while being able to accurately detect bots and humans.

5. G3: Creating BIMBAS, a bot identification model based on activity sequences

We identified the limitations of the existing bot identification approaches in Section 4. The goal of the current section is to develop a new model detecting whether a contributor is a *bot* or a *human* that: (i) exhibits a performance comparable to the state-of-the-art; (ii) can be used to predict contributors that are involved in other (or more) activity types than the usual commit-, issue- or PR-related activities; (iii) requires a low amount of data to be downloaded; and (iv) can be implemented as part of an efficient tool that is able to classify thousands of contributors in a limited amount of time.

The current section is structured as follows. Section 5.1 explains how to construct activity sequences based on the events provided by the GitHub API. Section 5.2 details the features we compute from the activity sequences and presents their rationale. Section 5.3 explains the procedure that we follow to impute missing values, select the best classification model, and eliminate unimportant features. Based on the output of this process, we propose BIMBAS, a classification

model distinguishing bots and humans based on their activities. Section 5.4 evaluates the performance of BIMBAS on the test set. Section 5.5 discusses the most important features contributing to the predictions made by BIMBAS. Section 5.6 discusses the misclassified cases. Finally, Section 5.7 summarises the resulting model.

The replication package containing the material for creating and evaluating BIMBAS is available online. [9]

5.1. Extracting activity sequences

In order to develop a new bot identification tool that will not require a lot of data and can consider a wide range of activity types to give a prediction for many contributors, we propose a novel approach that is entirely based on activity sequences.

The GitHub Events API allows one to retrieve up to the last 300 public events that were generated by a contributor during the last 90 days. These low-level events will then be converted to fine-grained activity sequences [4]. Relying on activity sequences has two main benefits: (i) events can be retrieved from the GitHub API using at most three API queries, implying the hourly API rate limit will not be reached before around 1,666 contributors; and (ii) the activities that can be obtained from these events cover a wide range of all possible activities a contributor can do on GitHub, implying that we will be able to categorize more contributors, even those not active in committing or commenting. This section explains the procedure that we followed to retrieve the recent events from GitHub Events API for all the contributors present in our dataset and convert them to activities sequences.

For each contributor, we queried the GitHub Events API, as of 3 May 2024, and retrieved 376,638 events performed by 2,150 contributors (194,863 by 1,035 bots and 181,775 by 1,115 humans). These events correspond low-level events, for example, IssuesEvent is generated whenever a contributor opens, closes or reopens an issue. In order to identify the corresponding fine-grained activities, one has to refer to the *action* field present in the *payload* field of the event provided by GitHub Events API. Based on this value, the IssuesEvent can correspond to *Opening issue*, *Closing issue* or *Reopening issue*. We followed the process presented by Chidambaram et al. [4] to convert the low-level event types provided by the API into 24 more fined-grained activity types. These activity types include *Pushing commit*, *Opening PR*, *Closing issue*, *Publishing release*, *Creating tag*, and so on. In this paper, we ignored all GitHub contributors that performed less than five GitHub events, as they would not provide enough information for a bot identification model to make any conclusive decision. This resulted in a total of 337,246 activities performed by the 2,150 contributors of the ground-truth dataset. 182,218 of these activities were performed by 1,035 bots, while 155,028 activities were performed by 1,115 humans.

5.2. Selecting features

Chidambaram et al. [26] already observed many differences between the activities made by bots and those made by humans. They suggested five behavioural features that capture the differences between bots and humans: the number of activity types, the inequality of number of activity types across repositories, and the inequality of time between consecutive activities. In addition to this, they also observed a significant difference in the number of repositories contributed to by bots and humans, and in the median time for them to switch between repositories.

[9] `https://github.com/natarajan-chidambaram/BIMBAS_RABBIT_replication_package`

Figure 1: Generic pipeline for training and evaluating the classification model.

We took these five features as an initial set of features. We extended this feature set by considering a wide range of counting metrics related to contributors (e.g., their number of activities, number of activity types, and number of repositories contributed to) as well as temporal metrics related to their activity sequences (e.g., duration between consecutive activities in a repository, and time difference between consecutive activities of different activity types). Several of these metrics are computed at the level of a single repository (e.g., NTR) or a single activity type (e.g., NAT) and therefore have to be aggregated. Since we do not know in advance which aggregation functions will be the most useful for the model, we aggregate them using mean and median for central tendency, std (standard deviation) and IQR (inter-quartile range) for dispersion, and Gini for inequality. This leads us to a total of 45 features, of which 5 are non-aggregated and 8*5 are aggregated features. The entire list of considered features is reported in Table 5, together with the rationale for selecting these features.

5.3. Model selection

In this section we explain the process we followed to come up with a classification model for detecting bots. More specifically, Section 5.3.1 explains how we handle and impute missing values for the features we selected. Section 5.3 details the approach we followed to identify the best classification model (classifier and its hyperparameters). Section 5.3.3 explains the methodology followed to remove the features that do not contribute to the predictions made by the classification model.

5.3.1. Handling and imputing missing values

Some machine learning classifiers do not support the presence of missing values. Since the values of several features cannot be computed for contributors that are exclusively working in a single repository (e.g., DAAR) or exclusively performing a single activity type (e.g., DCAT), we apply a two-step process to impute such missing values and make the model aware of this. More specifically, we (1) replace missing values with the median value of the corresponding feature, and (2) add a Boolean indicator to signal to the model whether a missing value was imputed [30]. Combining the imputed value and the Boolean indicator allows improving model performance [31]. This two-step process is part of the model pipeline, i.e., the missing values are computed based on training data only, to avoid the model becoming contaminated by unseen data. Fig. 1 depicts this generic pipeline, the "Estimator" step abstracting the actual model in use.

5.3.2. Identifying a classification model

To differentiate bots from humans based on their activity sequences, we rely on a classification model based on a binary classifier since we only have two classes (*bot* and *human*).

Table 5: List of considered features and the intuition behind them.

Counting metrics:

Acronym	Feature	Intuition
NA	Number of Activities	Since bots are automated agents, we expect them to produce more activities than humans as they do not suffer from the same limitations as humans (e.g., the need for sleep).
NT	Number of (activity) Types	Bots are expected to perform a smaller range of activity types than humans since they are likely to be specialised towards specific tasks.
NR	Number of Repositories contributed to	Bots are expected be involved in more repositories than humans.
NOR	Number of Owners of Repositories contributed to	Many bots are expected to be used by a small number of repository owners, as they may have been developed or configured by those owners specifically for their repositories. Humans on the other hand, have the freedom to decide which repositories they contribute, regardless of who owns the repository.
ORR	Owner-Repo Ratio = $\frac{\#NOR}{\#NR}$	We expect many bots to be used by small number of repository owners that use these bots in most of their repositories. In contrast, human accounts can be freely involved in multiple repositories belonging to a wide range of repository owners.

Aggregated metrics: Each of the metrics below is aggregated using five aggregation functions: mean, std, median, IQR, and Gini.

Acronym	Feature	Intuition
NAR	Number of Activities per Repository	The number of activities per repository might be high for bots as they are intended to perform repetitive tasks and can work continually without needing breaks.
NAT	Number of Activities per Type	The number of activities per activity type might be higher for bots as they are specialised in performing specific activity types in repositories.
NCAR	Number of Consecutive Activities in a Repository	Bots do not suffer from context switching, hence they are expected to switch more easily and more frequently between different repositories.
NTR	Number of (activity) Types per Repository	We expect bots to be specialised in the activities they do within repositories, hence the number of activity types they have across repositories is more likely to be constant.
DCAR	Duration of Consecutive Activities in a Repository	If bots tend to switch between repositories, the time spent in a repository for carrying out consecutive activities might be lower than for humans.
DAAR	Duration between Activities Across Repositories	Since bots do not suffer from context switching, we expect them to take less time to have activities in multiple repositories than humans.
DCA	time Difference between Consecutive Activities	Since bots are automated scripts, they may be very fast in carrying out their next activity after the previous one. The same bot might even work in parallel in multiple, not necessarily related, repositories.
DCAT	time Difference between Consecutive Activities of different Types (in hours)	Since bots do not suffer from context switching, we expect them to switch more swiftly between activities of different types.

We selected seven different classifier types that are commonly used and have the ability to perform binary classification: Decision Tree [32], Random Forest [33], Support Vector Machines [34], Gradient Boosting [35], XGBoost [36], Linear Discriminant Analysis [37], and Gaussian Naive Bayes [38].

All the considered classifier types accept a set of hyperparameters that can be tuned to increase the model performance. We followed a grid-search 10-fold cross-validation hyperparameter tuning process [39] to find, for each classifier type, the values that should be used for these hyper-

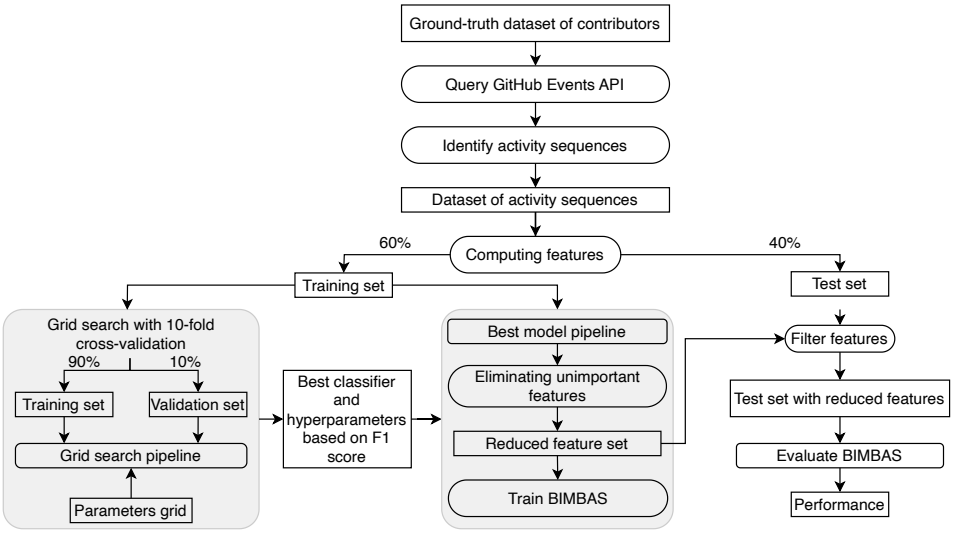

Figure 2: Overview of the process followed to identify the best model, eliminate features and to evaluate BIMBAS.

parameters. In total, we consider 13,021 combinations of classifier types and hyperparameter values. The process is illustrated in Fig. 2 (leftmost gray box).

To train and evaluate these 13K+ models, we used the features corresponding to activity sequences of each contributor present in the training set. The training set contains 669 (out of 1,115) humans and 621 (out of 1,035) bots, i.e., 60% of all the contributors. We followed a 10-fold cross-validation process, relying on a stratified shuffle split strategy to maintain similar proportions of humans and bots within each fold, and we measured the resulting performance of each model based on the usual performance measures of weighted precision (P), recall (R), F1 score, and area under ROC curve (AUC).

Table 6 reports on the results of this process. To allow us to interpret and compare the performance of the models, we also included NBH a baseline, task-specific model. Since we cannot list all the considered combinations, we report for each type of classifier on the results obtained by the best combination of hyperparameters in terms of weighted F1 score, as this score reflects the precision and recall of a model through a single, easy to compare value. Among all considered combinations, Gradient Boosting is the best overall performer, with the highest F1 score (for bots, humans, and weighted overall), the highest precision for bots, and the highest AUC score.

5.3.3. Eliminating features

We trained and evaluated the models on a set of 45 initial features. However, not all these features have the same importance during the classification process, and some of them may be redundant or may contribute little or not at all to the decision of the model. To identify which features can be safely removed without affecting model performance, we rely on the well-known *recursive feature elimination* (RFE) technique [40]. RFE aims to identify and eliminate the least important features (i.e., those that do not contribute much to the model performance) by recursively considering smaller and smaller sets of features.

Table 6: Performance of the best model for each considered classifier type, in descending weighted F1 score

	bots			humans			weighted			
classifier type	P	R	F1	P	R	F1	P	R	F1	AUC
Gradient Boosting	**.923**	.939	**.930**	.944	**.925**	**.934**	**.934**	**.932**	**.932**	**.970**
Random Forest	.897	**.947**	.921	**.948**	.899	.922	.924	.922	.922	.969
XGBoost	.900	.935	.917	.938	.903	.920	.920	.919	.919	.967
Decision Tree	.891	.931	.909	.933	.891	.911	.913	.910	.910	.924
Linear Discriminant Analysis	.874	.916	.893	.921	.876	.897	.898	.895	.895	.961
Support Vector Machines	.820	.806	.812	.823	.833	.827	.822	.820	.820	.833
Gaussian Naive Bayes	.865	.597	.705	.710	.912	.798	.785	.760	.753	.893
NBH (baseline model)	.765	.669	.714	.725	.810	.765	.744	.742	.740	.739

We applied RFE in a 10-fold cross-validation loop on the training set. At the end of the process, RFE identified the following seven features that can be removed without any compromise on the model performance: NR, DCA_{IQR}, NAR_{std}, NTR_{IQR}, $\text{NCAR}_{\text{median}}$, $\text{NCAR}_{\text{Gini}}$, $\text{DCAR}_{\text{Gini}}$.

5.4. Training and evaluating BIMBAS

The grid search cross-validation explained in Section 5.3.2 allowed us to identify the best classifier type (Gradient Boosting) and its corresponding hyperparameters. In Section 5.3.3, we identified that 38 features are important for the model to exhibit good performance. In this section, we introduce, train and evaluate BIMBAS, a "Bot Identification Model Based on Activity Sequences". BIMBAS implements the selected classifier and its hyperparameters, and takes as input the 38 identified features.

Table 7: Performance comparison of BIMBAS against existing approaches on the test set of 860 unseen contributors.

	bots			humans			weighted		
approaches	P	R	F1	P	R	F1	P	R	F1
NBH	.803	.671	.732	.735	.848	.788	.768	.763	.761
BoDeGHa	.918	.512	.657	.861	.457	.597	.891	.486	.629
BoDeGiC	.812	.271	.406	.747	.159	.262	.785	.224	.346
BotHunter	.967	.928	.947	.937	.971	.954	.952	.950	.950
BotHunter without NBH	.984	.587	.735	.722	.991	.836	.848	.797	.787
BIMBAS	.883	.911	.897	.915	.888	.901	.899	.899	.899

BIMBAS is trained on the full training set (60% of all contributors). To assess its performance on unseen data, and therefore to validate BIMBAS, we applied it on the test set containing the remaining 40% contributors (i.e., 414 bots and 446 humans). Table 7 reports on the results of BIMBAS on this test set. Since the test set is the same than the one we used in Section 4, we also report on the results obtained by the other bot detection approaches for comparison.

BIMBAS correctly identified most of the bots (377 out of 414) with a precision $P = 0.883$ and a recall $R = 0.911$. Similarly, most of the humans (396 out of 446) that are present in the test set were correctly identified by BIMBAS with a precision $P = 0.915$ and recall $R = 0.888$. Overall, BIMBAS reaches a precision $P = 0.899$ and a recall $R = 0.899$, making it the second most performant bot detection approach.

However, in Section 4.3 we mentioned that the good performance of NBH could be explained by the high proportion of bots in the ground-truth dataset that have "bot" in their name. The most important features of BotHunter (e.g., *account name* and *account login*) are variations of NBH, relying on the presence of "bot" or "automate" [13]. This is likely the reason why BotHunter correctly identified 97.7% of the bots that have "bot" in their name, whereas it identified only

66.7% of the bots that do not. To get a better understanding of the actual performance of BotHunter on less obvious cases of bot contributors, we executed a modified version of BotHunter that no longer relies on the presence of "bot" or "automate" to make its decision.[10] The results are provided in Table 7 as "BotHunter without NBH". As anticipated, this variant of BotHunter has more difficulties to identify bots, with a decrease of recall for bots from $R = 0.928$ to $R = 0.587$, a decrease in overall recall from $R = 0.950$ to $R = 0.797$ and a decrease in overall precision from $P = 0.952$ to $P = 0.848$. This confirms that the performance of BotHunter heavily depends on the NBH-related features to detect bots and is less effective in detecting non-obvious cases of bots. On the other hand, BIMBAS makes no distinction between contributors based on their names, and bases its decision exclusively on the activity sequences. As such, its performance does not depend on the presence of "bot" in the name, nor of any other substring.

5.5. Feature importance

In order to identify the most contributing features (among the 38 remaining ones), we applied the *permutation importance* [33] technique on the test set. This model inspection technique measures the importance of each feature by randomly shuffling the values of one feature at a time, and measuring how this affects the performance of the model. We applied this technique in a 10-fold cross-validation setting on the test set, meaning that each feature is shuffled 10 times and the resulting F1 scores are aggregated.

Table 8: Top five important features for distinguishing bots from humans in test set, reporting their median values and effect size.

		median		**effect size**	
acronym	**feature**	bots	humans	δ	interpretation
NT	number of activity types	4.0	10.0	.725	large
NOR	number of owners of repositories contributed to	1.0	4.0	.554	large
$\text{DCAT}_{\text{median}}$	median time difference between consecutive activities of different types	0.003 h	0.125 h	.482	large
$\text{NAT}_{\text{median}}$	median number of activities per type	29.5	6.0	.676	large
NAT_{mean}	mean number of activities per type	45.0	14.9	.703	large

Table 8 reports on the 5 most important features, i.e., on the 5 features that have the higher impact in terms of F1 score when shuffled. This table also reports on the median value of these features, distinguishing between bots and humans. To determine whether there is a statistically significant difference between bots and humans for these features, we performed Mann-Whitney U tests [41]. The null hypothesis, stating there is no difference between the two populations, was consistently rejected with a significance level $\alpha = 0.001$ after controlling for family-wise error rate with the Bonferroni-Holm method [42]. The effect size of these tests, based on Cliff's δ [43], reveals a *large* difference for these features between bots and humans.

5.6. Analysing the misclassifications

The evaluation of BIMBAS revealed that is performing well on the test set. Nevertheless, BIMBAS misclassified 37 out of 414 bots as humans (FN) and 50 out of 446 humans as bots (FP). We manually tried to find possible reasons for these misclassifications.

[10] It did not suffice to simply change the names of those contributors, since BotHunter requires the exact contributor name to retrieve its data.

A first observation was that many misclassified contributors performed very few activities. For example, eclipse-metro-bot and arduino-ci-script-bot are bots that respectively have only 6 and 8 activities. This lack of data makes it difficult for BIMBAS to determine if their behaviour is closer to humans or bots. To quantify the impact of the number of activities on misclassifications, we measured the proportion of misclassified contributors in function of the number of withheld activities for each contributor. Fig. 3 shows this proportion, revealing that a lower number of activities coincides with a higher proportion of misclassified contributors. For instance, when applied on the latest eight activities of each contributor, 25% of the contributors are misclassified. When applied on the latest 25 activities, 16% of the contributors are misclassified. In contrast, the proportion of misclassified contributors does not exceed 10% starting from 107 activities.

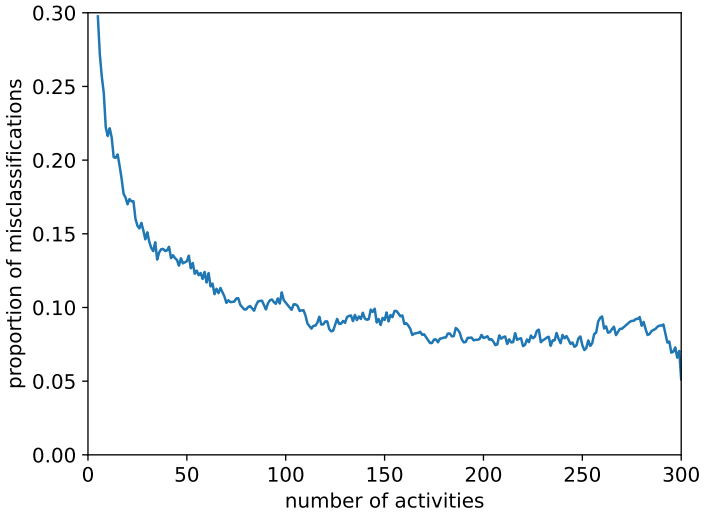

Figure 3: Proportion of misclassified contributors in function of number of considered activities

Another reason for misclassified bots is that they are taking a long time to switch between activity types ($DCAT_{median}$), although they are involved in multiple activity types (NT) and the mean and median number of activities per activity type (NAT_{median} and NAT_{mean}) is similar to that of humans. For example, bot-gradle is a bot that performed 244 activities belonging to 9 different activity types, but its $DCAT_{median}$ is 0.307 whereas its NAT_{median} and NAT_{mean} are 7 and 21.375 respectively. Comparing these values with the corresponding median for bots and humans in Table 8 conveys that it is difficult for BIMBAS to classify them as bots.

We also found instances of misclassified bots whose main or only purpose is to mirror (e.g., migrate, copy-paste or translate) human activities coming from other sources (e.g., another repository or another bug tracking system). For example, zx2c4-bot is a mirroring bot that creates and deletes tags and branches, closes PRs, and pushes commits to GitHub that were made on an external git repository. Such bots are difficult to distinguish from humans, given that each of their mirrored activities actually originate from some human activity.

On the other hand, one of the misclassified humans performed 300 activities belonging to a single activity type (pushing commits). While it is expected to have humans involved in this activity

type, it is unlikely for them to be involved in only a single activity type, since the median number of activity types for humans is 1.0 (see NT in Table 8). Indeed, the test set included only one human contributor that was exclusively pushing commits, so it can be considered to be an outlier in the class of humans. Another observation is that six misclassified humans have a median time difference between consecutive activity types ($\mathsf{DCAT}_{\mathsf{median}}$) of 0.001h, which corresponds more to bot behaviour than to human behaviour (see in Table 8).

5.7. Summary

We proposed BIMBAS, a novel bot identification approach based on activity sequences. To create BIMBAS, we followed a grid-search 10-fold cross-validation on the training set and compared the results obtained by 13K+ combinations of classifiers and their hyperparameters. Through this process, we found Gradient Boosting and its associated hyperparameters to be the top performer. We applied the RFE technique to remove features that do not contribute to the model performance, retaining 38 features. We then evaluated the performance of BIMBAS on the test set.

Overall, the performance of BIMBAS is comparable to the best existing bot identification approaches, making it a good candidate to be implemented as part of a tool. This will be the goal of the next section, in which we present RABBIT, an open source tool implementing BIMBAS. We will show that RABBIT outperfoms the existing state-of-the-art approaches in terms of efficiency.

6. G4: Developing RABBIT, an efficient bot identification tool

In order to allow researchers and practitioners to use BIMBAS, the activity-based bot identification model proposed and evaluated in Section 5, we propose RABBIT, an efficient executable command-line tool to detect bots on GitHub. Section 6.1 introduces RABBIT and explains its functioning. Section 6.2 evaluates and compares its efficiency against the existing bot identification approaches of Section 4, showing that RABBIT addresses their main limitations. Finally, Section 6.3 summarizes our contribution.

6.1. Implementation of RABBIT

RABBIT is a recursive acronym for "RABBIT is an Activity-Based Bot Identification Tool". It provides a command-line interface to use the BIMBAS model trained on the full dataset. It is released as an open source project on GitHub[11] under the Apache 2.0 License. Release 2.2.0 of RABBIT was used for the experiments in the current paper.[12] RABBIT can be installed using Python's package manager pip with `pip install git+https://github.com/natarajan-chidambaram/rabbit`.

The functioning of RABBIT is schematically represented in Fig. 4. The mandatory input is a list of contributor names. These names can be provided directly on the command line or through a text file. RABBIT also provides several optional parameters to change its default behaviour (number of queries, number of events, etc.) The output can be displayed on the standard output (by default) or stored in a CSV or JSON file.

RABBIT will assign one of five possible types to each provided contributor name: *Human*, *Bot*, *Organization*, *Unknown* or *Invalid*. To do so, the tool follows the process depicted in the middle

[11] `https://github.com/natarajan-chidambaram/rabbit`

[12] An earlier version of RABBIT implemented an XGBoost classification model based on a more limited set of features and trained on a significantly smaller dataset [44].

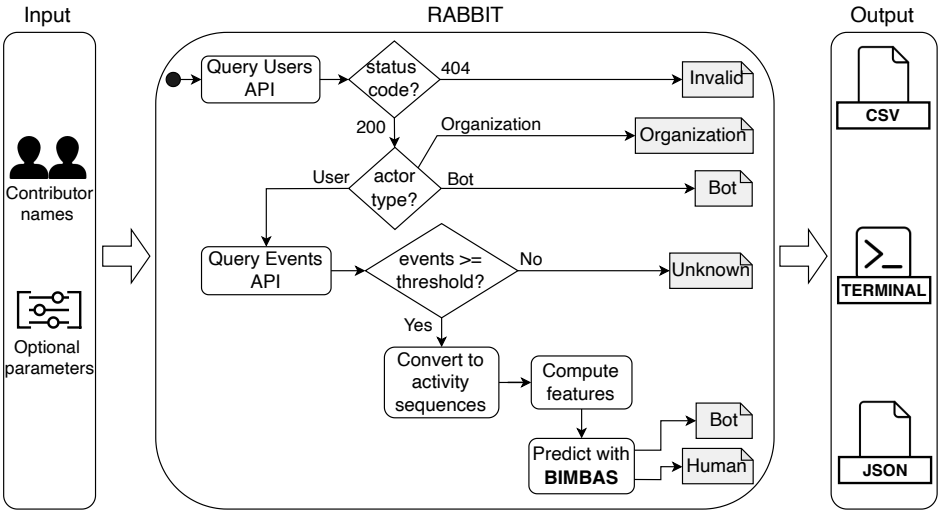

Figure 4: Schematic representation of RABBIT.

part of Fig. 4. The process starts by querying GitHub's Users API[13] to extract the value stored in the "type" field. The contributor type is predicted as *Invalid* if the contributor name does not exist on GitHub. If the contributor does exist, but the value in the "type" field is not "User" (e.g., it will be "Bot" for GitHub Apps, and "Organization" for organisations), this value is provided as output without needing any further processing. Only if the "type" field is "User", RABBIT queries the Events API[14] to extract up to 300 events (using at most 3 API queries) performed by the contributor during the last 90 days (a constraint imposed by the API). The contributor type will be *Unknown* if the number of events obtained does not reach the minimum threshold set by RABBIT (which is 5 events by default). If enough events can be retrieved, the extracted event sequence is converted into an activity sequence (see Section 5.1), the features are computed for this activity sequence (see Section 5.2), and the BIMBAS classification model is used to predict the contributor type as either *Bot* or *Human*.

Along with the contributor type, RABBIT also reports on the *confidence* of its decision. The confidence score is based on the *probability* associated to each prediction made by BIMBAS. Given that BIMBAS provides the probability for a contributor to be a "bot", the confidence score is computed as $|probability - 0.5| * 2$. Listing 1 shows an example of the execution of RABBIT with a list of 8 contributor names provided through the input file `names.txt`.

```
% rabbit --input-file names.txt
          contributor          type              confidence
  github-actions[bot]          Bot               1.0
       johnpbloch-bot          Bot               0.932
      openssl-machine          Bot               0.714
            ritchie46          Human             0.926
     juliaregistrator          Bot               0.875
```

[13] https://api.github.com/users/CONTRIBUTORNAME
[14] https://api.github.com/users/CONTRIBUTORNAME/events

```
gvanrossum.     Human          0.960
    google      Organization   1.0
  renovate      Unknown        –
    gh-ci       Invalid        –
```

Listing 1: Example of RABBIT usage and output.

6.2. Comparing RABBIT's efficiency with existing approaches

RABBIT benefits from the good model performance of BIMBAS, while at the same time addressing all limitations reported in Section 4 for the existing bot identification approaches. For instance, RABBIT supports a wider range of activity types and is able to determine the type of many more contributors than BoDeGHa or BoDeGiC.

To evaluate the efficiency of RABBIT, we applied it on the same test set of 860 contributor names as those that were used for evaluating the other bot detection approaches in Section 4. To determine the type of these 860 contributors, RABBIT required 22 minutes and 112 MB of downloaded data. Only 2,426 API queries were required to make all predictions, staying well below GitHub API rate limit of 5,000 queries per hour and per API key. By extrapolation, this means that RABBIT can process 1,772 contributors on average before reaching the rate limit.

Table 9: Efficiency comparison of RABBIT against existing approaches on the test set of 860 unseen contributors.

approaches	data downloaded	time	API queries
NBH	-	0.01 sec	-
BoDeGHa	3.83 GB	7.7 h	10,222
BoDeGiC	23.3 GB	22.1 h	-
BotHunter	0.261 GB	20.8 h	37,240
RABBIT	0.112 GB	22 m	2,426

To put the evaluation results of RABBIT in perspective, Table 9 compares its efficiency to the existing bot identification approaches, highlighting their limitations in terms of execution time, data downloaded, and number of required API queries. In terms of execution time, RABBIT is more than an order of magnitude faster than BoDeGHa (21×), BotHunter (57×) and BoDeGiC (60×). RABBIT requires considerably less data to be downloaded compared to BoDeGHa (34×) and BoDeGiC (208×). RABBIT uses an order of magnitude less API queries than other approaches relying on the GitHub API, namely BoDeGHa (4×) and BotHunter (15×).

6.3. Summary

We implemented RABBIT as a command-line tool to allow researchers and practitioners to use BIMBAS in practice to identify bots in GitHub repositories. RABBIT is more efficient than previous bot identification approaches. While still achieving a comparable performance, it can be used for considerably larger sets of contributors since it runs an order of magnitude faster, uses an order of magnitude less API queries and requires less data to be downloaded.

Based on one's specific need, other bot identification approaches could be favored. If accuracy is not a crucial factor, then NBH should be favored since it is really easy to implement and the fastest bot identification approach so far. If higher model performance would be preferred over a faster execution time, for example in the context of some empirical research study, BotHunter might be favored since it had less misclassified cases on our test set than RABBIT. However, as explained in Section 5.4, this is likely to be a consequence of the high proportion of bots that

have "bot" in their name in the ground-truth dataset, and the strong reliance of BotHunter on the NBH heuristic.

Creating an ensemble model that combines the strengths of different bot identification models (e.g., combining BotHunter and BIMBAS) would likely reduce the number of misclassifications that are inherent to any bot detection approach. However, such as an ensemble model would nullify the benefits of using RABBIT for its efficiency, so we see little value in doing so.

7. Threats to validity

We follow the structure recommended by Wohlin et al. [45] to discuss the main threats to validity of our research, and their potential consequences.

Construct validity examines the relationship between the theory behind the experiments performed and the observations found. This threat is mainly related to correctness of the dataset used in the experiments. A possible such threat is that contributors in the ground-truth are not labelled correctly. This situation is very unlikely to happen since we followed a multi-rater labelling process that resulted in an almost perfect inter-rater agreement (Cohen's kappa $\kappa = 0.91$ [46]). However, we cannot exclude that the ground-truth dataset contains so-called *mixed accounts* (i.e., accounts having a combination of human and bot activities) [8].

Another threat to construct validity is that the ground-truth dataset is biased, by construction, towards bots having "bot" in their name: 67.1% of all bots in the dataset (i.e., 694 out of 1,035) contain this substring in their contributor name, which is likely considerably more than what one could expect in practice. While this higher proportion of bots having "bot" in their name can not affect the performance of BIMBAS (and therefore of RABBIT) since it does not rely on this feature to detect bots, it may have led to an overestimation of the performance of bot identification approaches such as NBH and BotHunter that make use of that feature.

Internal validity concerns choices and parameters of the experimental setup that could affect the results of the observations. We strived to follow machine learning best practices during model construction, training, testing and evaluation. We relied on a state-of-the-art machine learning library (namely sklearn) to conduct the experiments. As such, our experimental setup is unlikely to have biased the results we obtained.

Another internal threat to validity is the criterion we used to select the repositories for executing BoDeGHa and BoDeGiC. We selected, for each contributor, the repository in which the contributor was the most active (in terms of events generated). A different selection criterion could lead to different performance results and to a different number of "unknown" predictions. However, none of these approaches explicitly define guidelines to decide on a repository whenever a contributor is active in more than one repository.

External validity concerns the degree to which the conclusions we derived are generalisable outside the scope of this study. The main threat to external validity is that BIMBAS was created and evaluated with activity sequences obtained from public GitHub repositories (because GitHub's API only returns public events). Since activities in private repositories may differ in their type, frequency and order we cannot make any claims on the performance of BIMBAS on contributors in private repositories. Similarly, BIMBAS cannot be applied "as is" to other collaborative development platforms (e.g., GitLab, Gitea or BitBucket). Even though such platforms are mostly based on the same principles, and even if the technical process we followed to create BIMBAS is likely to be applicable to these platforms, there could be differences in the APIs, the activity pace, activity types of contributors, and the ways bots interact with repositories and

contributors. As a consequence, it is very likely that a new classification model would need to be trained to take these differences into account.

Conclusion validity concerns whether the conclusions derived from the analysis are reasonable. Since our conclusions are mostly based on quantitative observations and are supported by the usual performance metrics to evaluate machine learning classifiers, they are unlikely to be affected by such threats.

8. Conclusion

Several automated bot identification approaches have been proposed for detecting bots that engage in automated activities in GitHub. They address the absence of a direct way to distinguish regular *human accounts* from *bot accounts* that tend to automate repetitive, effort intensive and error-prone activities.

Accurate detection of bots is important for researchers conducting empirical analyses, and for developer communities and funding organisations to correctly recognise and accredit human contributions. However, efficiency concerns make it challenging to apply existing bot identification approaches at large scale, either because they rely on computationally expensive features, because they need to download a lot of data, or because they have to wait for the API rate limit to replenish.

To overcome these limitations, we proposed the BIMBAS model and its accompanying implementation, RABBIT. BIMBAS is a classification model relying on activity sequences obtained from GitHub events to determine whether a given contributor is a bot or a human. BIMBAS relies on a Gradient Boosting binary classifier involving 38 features related to the activity sequences of a contributor to accurately determine its type. The performance of BIMBAS is comparable to the state-of-the-art bot detection approaches. RABBIT makes BIMBAS available as a command-line tool allowing to detect bots on considerably larger sets of contributors, since it runs an order of magnitude faster, uses an order of magnitude less API queries and requires less data to be downloaded than existing approaches.

Acknowledgement

This work is supported by Service Public de Wallonie Recherche under grant number 2010235 - ARIAC by DigitalWallonia4.AI, by the Fonds de la Recherche Scientifique – FNRS under grant numbers J.0147.24, T.0149.22, and F.4515.23.

References

[1] L. Dabbish, C. Stuart, J. Tsay, J. Herbsleb, Social coding in GitHub: Transparency and collaboration in an open software repository, in: International Conference on Computer Supported Cooperative Work (CSCW), ACM, 2012, pp. 1277–1286. doi:10.1145/2145204.2145396.

[2] J. M. Costa, M. Cataldo, C. R. de Souza, The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools, in: SIGCHI Conference on Human Factors in Computing Systems, ACM, 2011, pp. 3151–3160. doi:10.1145/1978942.1979409.

[3] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, M. A. Gerosa, The power of bots: Understanding bots in OSS projects, International Conference on Human-Computer Interaction (CHI) (2018). doi:10.1145/3274451.

[4] N. Chidambaram, A. Decan, T. Mens, A dataset of bot and human activities in GitHub, in: International Conference on Mining Software Repositories (MSR), IEEE/ACM, 2023, pp. 465–469. doi:10.1109/MSR59073.2023.00070.

[5] M. Golzadeh, T. Mens, A. Decan, E. Constantinou, N. Chidambaram, Recognizing bot activity in collaborative software development, IEEE Software 39 (5) (2022) 56–61. doi:10.1109/MS.2022.3178601.
[6] Z. Wang, Y. Wang, D. Redmiles, From specialized mechanics to project butlers: The usage of bots in open source software development, IEEE Software 39 (5) (2022) 38–43. doi:10.1109/MS.2022.3180297.
[7] M. Golzadeh, A. Decan, T. Mens, On the effect of discussions on pull request decisions, in: Belgium-Netherlands Software Evolution Workshop (BENEVOL), Vol. 2605, CEUR Workshop Proceedings, 2019.
[8] N. Cassee, C. Kitsanelis, E. Constantinou, A. Serebrenik, Human, bot or both? A study on the capabilities of classification models on mixed accounts, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2021. doi:10.1109/ICSME52107.2021.00075.
[9] C. Hauff, G. Gousios, Matching GitHub developer profiles to job advertisements, in: Working Conference on Mining Software Repositories (MSR), 2015, pp. 362–366. doi:10.1109/MSR.2015.41.
[10] M. Golzadeh, A. Decan, D. Legay, T. Mens, A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments, Journal of Systems and Software 175 (2021). doi:10.1016/j.jss.2021.110911.
[11] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, A. Mockus, Detecting and characterizing bots that commit code, in: International Conference on Mining Software Repositories (MSR), ACM, 2020, pp. 209–219. doi:10.1145/3379597.3387478.
[12] M. Golzadeh, A. Decan, T. Mens, Evaluating a bot detection model on git commit messages, in: Belgium-Netherlands Software Evolution Workshop (BENEVOL), Vol. 2912, CEUR Workshop Proceedings, 2020.
[13] A. Abdellatif, M. Wessel, I. Steinmacher, M. A. Gerosa, E. Shihab, BotHunter: An approach to detect software bots in GitHub, in: International Conference on Mining Software Repositories (MSR), 2022, pp. 6–17. doi:10.1145/3524842.3527959.
[14] Z. Liao, X. Huang, B. Zhang, J. Wu, Y. Cheng, BDGOA: A bot detection approach for GitHub OAuth apps, Intelligent and Converged Networks 4 (3) (2023) 181–197. doi:10.23919/ICN.2023.0006.
[15] A. N. Meyer, T. Fritz, G. C. Murphy, T. Zimmermann, Software developers' perceptions of productivity, in: International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 19–29. doi:10.1145/2635868.2635892.
[16] M.-A. Storey, A. Zagalsky, Disrupting developer productivity one bot at a time, in: International Symposium on Foundations of Software Engineering (FSE), ACM, 2016, pp. 928–931. doi:10.1145/2950290.2983989.
[17] L. Erlenhov, F. G. de Oliveira Neto, P. Leitner, An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective, in: Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM, 2020, pp. 445–455. doi:10.1145/3368089.3409680.
[18] A. Ghorbani, N. Cassee, D. Robinson, A. Alami, N. A. Ernst, A. Serebrenik, A. Wasowski, Autonomy is an acquired taste: Exploring developer preferences for GitHub bots, in: International Conference on Software Engineering (ICSE), IEEE, 2023, pp. 1405–1417. doi:10.1109/ICSE48619.2023.00123.
[19] M. Wessel, I. Wiese, I. Steinmacher, M. A. Gerosa, Don't disturb me: Challenges of interacting with software bots on open source software projects, International Conference on Human-Computer Interaction (CHI) 5 (2021). doi:10.1145/3476042.
[20] M. Wessel, A. Abdellatif, I. Wiese, T. Conte, E. Shihab, M. A. Gerosa, I. Steinmacher, Bots for pull requests: The good, the bad, and the promising, in: International Conference on Software Engineering (ICSE), IEEE/ACM, 2022, pp. 274–286. doi:10.1145/3510003.3512765.
[21] M. Wyrich, R. Ghit, T. Haller, C. Müller, Bots don't mind waiting, do they? Comparing the interaction with automatically and manually created pull requests, in: International Workshop on Bots in Software Engineering (BotSE), 2021, pp. 6–10. doi:10.1109/BotSE52550.2021.00009.
[22] X. Zhang, Y. Yu, T. Wang, A. Rastogi, H. Wang, Pull request latency explained: an empirical overview 27 (6) (2022) 126. doi:10.1007/s10664-022-10143-4.
[23] S. Khatoonabadi, D. E. Costa, S. Mujahid, E. Shihab, Understanding the helpfulness of stale bot for pull-based development: An empirical study of 20 large open-source projects, ACM Transactions on Software Engineering and Methodology 33 (2) (2023). doi:10.1145/3624739.
[24] H. Y. Lin, P. Thongtanunam, C. Treude, W. Charoenwet, Improving automated code reviews: Learning from experience, in: International Conference on Mining Software Repositories (MSR), 2024. doi:10.1145/3643991.3644910.
[25] N. Chidambaram, A. Decan, M. Golzadeh, Leveraging predictions from multiple repositories to improve bot detection, in: International Workshop on Bots in Software Engineering (BotSE), IEEE, 2022. doi:10.1145/3528228.3528403.
[26] N. Chidambaram, A. Decan, T. Mens, Distinguishing bots from human developers based on their GitHub activity types, in: Seminar on Advanced Techniques & Tools for Software Evolution (SATToSE), Vol. 3483, CEUR Workshop Proceedings, 2023.

[27] G. Cardoen, T. Mens, A. Decan, A dataset of GitHub actions workflow histories, in: International Conference on Mining Software Repositories (MSR), ACM, 2024. doi:10.1145/3643991.3644867.
[28] Y. Ma, T. Dey, C. Bogart, S. Amreen, M. Valiev, A. Tutko, D. Kennard, R. Zaretzki, A. Mockus, World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data, Empirical Software Engineering 26 (2021) 1–42. doi:10.1007/s10664-020-09905-9.
[29] M. Golzadeh, A. Decan, N. Chidambaram, On the accuracy of bot detection techniques, in: International Workshop on Bots in Software Engineering (BotSE), IEEE, 2022. doi:10.1145/3528228.3528406.
[30] S. van Buuren, Flexible Imputation of Missing Data, Champan and Hall, 2012. doi:10.1201/b11826.
[31] M. Van Ness, T. M. Bosschieter, R. Halpin-Gregorio, M. Udell, The missing indicator method: From low to high dimensions, in: SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2023. doi:10.1145/3580305.3599911.
[32] S. Safavian, D. Landgrebe, A survey of decision tree classifier methodology, IEEE Transactions on Systems, Man, and Cybernetics 21 (3) (1991) 660–674. doi:10.1109/21.97458.
[33] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32. doi:10.1023/A:1010933404324.
[34] R. G. Brereton, G. R. Lloyd, Support vector machines for classification and regression, Analyst 135 (2) (2010) 230–267. doi:10.1039/B918972F.
[35] J. H. Friedman, Greedy function approximation: A gradient boosting machine, The Annals of Statistics 29 (5) (2001) 1189–1232.
[36] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: International Conference on Knowledge Discovery and Data Mining (KDD), ACM, 2016, pp. 785–794. doi:10.1145/2939672.2939785.
[37] T. Hastie, R. Tibshirani, J. H. Friedman, J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer, 2001. doi:10.1007/978-0-387-21606-5.
[38] T. F. Chan, G. H. Golub, R. J. LeVeque, Updating formulae and a pairwise algorithm for computing sample variances, in: Symp. Computational Statistics (COMPSTAT), Springer, 1982. doi:10.1007/978-3-642-51461-6_3.
[39] I. H. Witten, E. Frank, Data mining: practical machine learning tools and techniques with Java implementations, SIGMOD Rec. 31 (1) (2002) 76–77. doi:10.1145/507338.507355.
[40] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines 46 (1) 389–422. doi:10.1023/A:1012487302797.
[41] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Ann. Math. Statist. 18 (1) (1947) 50–60. doi:10.1214/aoms/1177730491.
[42] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (2) (1979) 65–70.
[43] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions, Psychological bulletin 114 (3) (1993) 494. doi:10.1037/0033-2909.114.3.494.
[44] N. Chidambaram, T. Mens, A. Decan, Rabbit: A tool for identifying bot accounts based on their recent GitHub event history, in: International Conference on Mining Software Repositories (MSR), ACM, 2024. doi:10.1145/3643991.3644877.
[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer, 2012.
[46] C. Jacob, A coefficient of agreement for nominal scales, Educational and Psychological Measurement 20 (1) (1960) 37–46. doi:10.1177/001316446002000104.