Action-based Recommendation in Pull-request Development

Muhammad Ilyas Azeem[*]
University of Chinese Academy of Sciences
Beijing, China
azeem@itechs.iscas.ac.cn

Sebastiano Panichella
Zurich University of Applied Science
Switzerland
panc@zhaw.ch

Andrea Di Sorbo
University of Sannio
Italy
disorbo@unisannio.it

Alexander Serebrenik
Eindhoven University of Technology
The Netherlands
a.serebrenik@tue.nl

Qing Wang[†‡§]
University of Chinese Academy of Sciences
Beijing, China
wq@itechs.iscas.ac.cn

ABSTRACT

Pull requests (PRs) selection is a challenging task faced by integrators in pull-based development (PbD), with hundreds of PRs submitted on a daily basis to large open-source projects. Managing these PRs manually consumes integrators' time and resources and may lead to delays in the *acceptance, response,* or *rejection* of PRs that can propose bug fixes or feature enhancements. On the one hand, well-known platforms for performing PbD, like GitHub, do not provide built-in recommendation mechanisms for facilitating the management of PRs. On the other hand, prior research on PRs recommendation has focused on the likelihood of either a PR being accepted or receive a response by the integrator. In this paper, we consider both those likelihoods, this to help integrators in the PRs selection process by suggesting to them the appropriate actions to undertake on each specific PR. To this aim, we propose an approach, called *CARTESIAN* (a*C*ceptance *A*nd *R*esponse classifica*T*ion-based requ*ES*t *I*dentifc*A*tio*N*) modeling the PRs recommendation according to PR actions. In particular, *CARTESIAN* is able to recommend three types of PR actions: *accept*, *respond*, and *reject*. We evaluated *CARTESIAN* on the PRs of 19 popular GitHub projects. The results of our study demonstrate that our approach can identify PR actions with an average precision and recall of about 86%. Moreover, our findings also highlight that *CARTESIAN* outperforms the results of two baseline approaches in the task of PRs selection.

[*]Also with Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing 100190, China.
[†]Also with Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing 100190, China.
[‡]Also with State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing 100190, China.
[§]Corresponding author

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; **Collaboration in software development**; **Software development process management**.

KEYWORDS

Pull Requests recommendation, Machine learning, Software maintenance and evolution

ACM Reference Format:
Muhammad Ilyas Azeem, Sebastiano Panichella, Andrea Di Sorbo, Alexander Serebrenik, and Qing Wang. 2020. Action-based Recommendation in Pull-request Development. In *International Conference on Software and System Processes (ICSSP '20), October 10–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3379177.3388904

1 INTRODUCTION

Popular platforms for open-source software development like GitHub, BitBucket, and GitLab support *Pull-based Development* (PbD), i.e., the software development process where rather than directly modifying the source code participants contribute by submitting pull requests (PRs) that can be reviewed with final decision, including PR *acceptance, response, and rejection* [2, 14]. However, popular GitHub projects, such as `Kubernetes` and `Rails`, receive tens or hundreds of pull requests daily [16, 34]. Selecting what pull requests to accept or respond becomes a challenge for the integrators [37], and this is complicated by the absence of mechanisms for prioritizing pull requests in PbD platforms.

PbD starts when a contributor creates a separate branch of a repository, e.g., to try new ideas or fix bugs without affecting the main branch.[1] Once a branch has been created, the contributor starts adding, editing or deleting files and making commits to the new branch. The branch commits track the contributor's progress on the branch; once all the changes are performed, a new pull request is created and integrators are requested to merge the changes into the master branch. Integrators have to manually inspect submitted pull requests [4, 23] and decide whether to reject them, request the submitters to apply further changes [43], or merge them into the master branch. This inspection process consumes a large amount of integrators' time [9, 10, 16, 39], with effort being

[1]https://guides.github.com/introduction/flow/

also spent on reviewing low-quality (i.e., unlikely to be accepted) pull requests [10]. Thus, in PbD it is of critical importance to select PRs that deserve integrators' attention.

Popular platforms for performing PbD, like GitHub, do not provide built-in recommendation mechanisms for identifying PRs worthy of the integrators' attention. Hence, the PbD process efficiency is compromised, delaying the integrators' actions and frustrating the contributors [15]. Indeed, contributors prefer to receive an immediate rejection for the pull requests rather than not receiving any response [15].

Previous approaches to PR recommendation focus either on predicting the likelihood of response [35] or the likelihood of acceptance [14, 34, 40, 43]. However, approaches focusing exclusively on response likelihood fail in recognizing PRs that are directly accepted and merged into the master branch without the need for discussion. For example, this may occur when contributors apply minor fixes[2] or when the changes are performed to fix known issues[3]. We empirically assessed that on a total of 278,418 PRs analyzed in our study, 33,231 PRs (i.e., 11.6%) were accepted without any discussion. Similarly, relying exclusively on acceptance may lead to misidentification of PRs that can be accepted after "shepherding".

For these reasons, we argue that PR recommendation strategies should be based on a more fine-grained classification of PRs: `accept`, `respond`, and `reject`. We define the three classes as follows: `accept` are PRs accepted without any discussion; `respond` are PRs accepted after discussion with the contributors; and finally, `reject` are PRs which have not been accepted. To automate the classification of PRs we propose an approach, called *CARTESIAN* (a*C*ceptance *A*nd *R*esponse classifica*T*ion-based requ*ES*t *I*dentifc*A*tio*N*). From a practical point of view, *CARTESIAN* can be used by integrators to have PRs clustered/classified according to the aforementioned categories. In this way they can prioritize the PR to address, depending on the actions they want to take. We note that *CARTESIAN* does not deal with the order of PRs given.

To evaluate *CARTESIAN* we pose the following questions:

- **RQ1**: To which extent is *CARTESIAN* able to identify the actions that should be undertaken by integrators on PRs?
- **RQ2**: To which extent can our approach be used to prioritize useful PRs?

Based on the evaluation of 19 GitHub projects (selection of the project done as indicated in Section 4), we conclude that *CARTESIAN* can recommend the right actions on PRs with an average precision and recall of 86%. Moreover, *CARTESIAN* significantly outperforms the baseline approaches in prioritizing PRs that will be actually accepted and gives high priority to PRs that integrators consider useful.

Paper structure. The remainder of the paper is organized as follows. After reviewing related work in Section 2, we describe the proposed approach in Section 3. Section 4 discusses the details of the empirical study carried out to answer our research questions, and in Section 5 we present and analyze the results. In Section 6 we identify the threats that could affect the validity of our work and conclude, outlining future research directions, in Section 7.

[2]See https://github.com/facebook/react/pull/12377
[3]See https://github.com/facebook/react/pull/12358

2 RELATED WORK

2.1 Empirical Studies of Pull Requests

Gousios *et al.* [15, 16] investigated and analyzed the PbD model by surveying GitHub developers, discussing the practices and challenges faced by both the integrators and contributors in the pull-based development model. Their findings reveal that maintaining PR quality and PR prioritization are the two main challenges faced by the integrators in PbD [16]. Similarly, lack of integrator's responsiveness is the main challenge faced by the contributors in PbD [15]. Rahman *et al.* [29] provided a comparative study involving 78 GitHub projects, investigating successful and unsuccessful pull requests, and found that age, maturity, and the number of developers and their experience can affect the success and failure rates of PRs. Zhang *et al.* [44] conducted an exploratory study of @-mentions in PbD, demonstrating that @-mention is beneficial to the processing of pull-requests. Liu *et al.* [21] conducted a larger study on 461 GitHub projects, reporting that usage of PbD helps to increase the social and development activities of projects, but it leads to prolonged bug fixing time.

2.2 Reviewer/Integrator Recommendation

Jiang *et al.* [18], proposed to support PbD by recommending appropriate reviewers for the PRs, discovering that the *activeness* is the most important attribute in the reviewer recommendation. In a similar direction, Thongtanunam *et al.* [33] proposed an accurate (77.97%) reviewer recommendation algorithm based on file path similarity, while Yu *et al.* [42] extended three typical approaches used in bug triaging and code review for recommending reviewers in PbD. Recently, Ying *et al.* [39] proposed a PR reviewer recommendation approach (EARec) which considers developer expertise and authority, which outperforms previous state-of-art methods. Liao *et al.* [20] proposed a Topic-based Integrator Matching Algorithm (TIMA) to predict highly relevant collaborators (the core developers) as the integrator to incoming PRs.

2.3 Pull requests acceptance/merge/latency prediction

Gousios *et al.* [14] conducted an exploratory study to identify the features that influence the PRs decision to merge and the time to process it: merge decisions mainly influenced by the specific recently modified code, while time to merge is affected by the developers experience and project characteristics (e.g., size and openness to external contributors). Tsay *et al.* [34] studied the social and technical factors behind the PR's acceptance. Their results suggest that highly discussed PRs are much less likely to be accepted and mature projects are more conservative when evaluating PRs. Similarly, Yu *et al.* [40, 43] explored the factors that affect PRs acceptance and latency in the context of continuous integration using regression modeling. Results show that latency is a complex issue and continuous integration is a dominant factor for PRs acceptance and latency.

2.4 Pull Request Recommendation

van der Veen *et al.* [35] proposed a code review PRioritizer system which uses a machine learning model to predict whether the current

PR will receive user updates in the following time window. The main limitation of the PRioritizer is that it uses daily user updates but does not consider the acceptance probability of the PRs. Due to this limitation, PRs that are not likely to be accepted but with a high probability to be updated by users may receive a higher priority compared to more useful PRs (i.e., the ones that would be actually accepted). Hence, our approach mitigates this limitation of PRioritizer by also considering the acceptance likelihood of each PR, thus reducing the probability that the integrator(s) focus on PRs having a low probability to be accepted. Our paper proposes to support integrators/developers to make actions, considering the actual PRs feedback. This idea was also explored in similar work, concerning the actions to take, when considering user feedback [26, 27, 32, 46].

3 *CARTESIAN* APPROACH

In this section, we introduce our machine learning-based PR recommendation approach called *CARTESIAN*. Differently from previous studies, *CARTESIAN* categorizes the PRs into three classes: `accept`, `respond`, and `reject`.

3.1 Feature Extraction

Features extracted for this study are based on previous studies on PRs [14, 43] and surveys conducted with GitHub developers [15, 16]. We extracted features from four aspects of a PR: *Project*, *Contributor (Author)*, *Integrator*, and *Pull Request* itself.

3.1.1 Project. We summarise the features related to the project of a given PR in Table 1. Project age has been shown to be a statistically significant predictor of PR acceptance [34, 43]. Furthermore, Tsay et al. have shown that team size and project popularity (stars) *negatively* influence the likelihood of a PR to be accepted: the more popular the project is, the less easy it is for a contributor to influence it by submitting a PR. We expect other popularity measures such as the number of watchers and the number of forks to affect the acceptance likelihood as well. Programming language and domain of the project may also have an influence on the success and failure of the PRs [29]. Finally, to understand the overall working rhythm of the project we further include such measures as additions and deletions of lines of code per week, the average merge latency of the PRs in the project and the average number of commits per PR.

3.1.2 Pull request. The features, summarised in Table 2, characterize the pull request information, e.g., its size or age. Gousios et al. observed that integrators take into account the size of the PR when evaluating the quality of the contribution [16]. Moreover, churn-related metrics have been shown to be statistically significant predictors for PR acceptance [43]. PRs submitted from the same branch may receive a quick response [35]. Time till the first response is a statistically significant predictor for PR acceptance [43]. The PRs submission day may also affect its acceptance [31]. Finally, it is plausible that the description of the PR (i.e., *title* and *body*) may contain information influencing the integrators' decision to accept a PR or to respond to it. Therefore, the title and body text are transformed into word embedding using Word2Vec, as explained in Section 3.2.

Table 1: Features extracted from the Project dimension.

Feature	Description
project_age	The age of the project measured in months.
team_size	The size of the project's team
stars	The numbers of stars of the project.
file_touched_average	The average number of total files touched in every PR.
forks_count	The number of forks of the project.
watchers	The number of watchers of the project.
language	The programming language in which the project is written.
project_domain	The domain of the project e.g. compiler, web framework, etc.
contributor_num	The number of contributors to the project.
comments_per_closed_PR	Average comments on pull requests of the project.
additions_per_week	The number of lines added per week to the project.
deletions_per_week	The number of lines deleted per week from the project.
merge_latency	The average time of the pull request in days from the open state to merged state of the project
week_days	The rate of lines changed in the project on each day of the week. Each day has been used then as a feature.
churn_average	The average number of lines added and deleted by the pull requests.
close_latency	The average time of pull request in days from the open state to the close state of the project.
comments_per_merged_PR	The average number of comments in merged pull requests.
project_accept_rate	The rate of merged pull requests of the project.
workload	The number of open pull requests still open in the project at the creation time of the examined PR.
commits_average	The average number of commits per pull request.
open_issues	The number of open issues

3.1.3 Author/Contributor. Status of the author in the community in general, represented by the number of followers, and specifically in the project, as reflected by the "contributor" status, have been shown to be statistically significant for predicting PR acceptance [34]. Moreover, reviewers prefer contributors with high-profile having a successful history [11, 16]. The success of the contributor is described by features like previous PRs, contributions, and acceptance rates [14]. Some projects assign priority to core member's contributions over external members. We have added some new features which may complement the existing features including the contributor's number of public repositories, rate of closed PRs. The complete list of features related to the developer authoring the PR is given in Table 3.

3.1.4 Integrator. The integrators take into consideration the code review while evaluating the quality of the PRs [16]. The number of discussion comments and review comments has an impact on the latency and acceptance of the PR [14]. Similarly, the number of participants who take part in the discussion also has an impact on the acceptance and latency of the PRs [14]. The at_mention feature represents whether the last comment message contains any reference to a developer. Previous studies show that '@' mentions are beneficial to the processing of PR by reducing the delay time in the developer's collaboration [44]. Table 4 shows the features extracted from the integrator's dimension.

Table 2: Features extracted from the Pull Request dimension.

Feature	Description
title	The embedding of the title text.
body	Embedding of the body text in the pull request.
intra_branch	Are the source and target repositories the same?
mergeable_state	Is this pull request in a mergeable state?
assignee_count	The number of the assignee in the pull request.
label_count	The number of labels given to the pull request.
files_changed	The number of changed files in the pull request.
contain_fix_bug	Does the pull request fix a bug?
wait_time	The waiting time of the pull request.
day	The current day of the pull request.
src_churn	Number of lines added to or deleted from the pull request.
commits_PR	The total number of commits in a pull request.
is_responded	Is the PR is responded by the integrator.
first_response_time	It is the time duration in minutes from creation of the PR till the first response from the integrator.
latency_after_response	The time duration in minutes after the first response till closing time of the PR.
PR_latency	The pull request latency is the time in minutes between the creation and closing of a pull request.
title_word_count	The number of words in the title of the examined PR.
body_word_count	The number of words in the body of the examined PR.
point_to_issueOrPR	Does the pull request aim to solve an issue or like other pull requests?

3.2 Generate Word Embedding

We extracted all the textual information from the PRs of the 19 projects, including 'Title', 'Body', and 'Review-Comments'. Then we eliminate the stop words[13] and apply stemming [28]. After pre-processing the textual information, we used all the remaining text for Word2Vec training to obtain the word embedding matrix of the whole vocabulary in our text [22]

Word2Vec, a neural network-based model, generates low dimensional word vectors, called "word embedding", from a corpus that can retain the semantics of the text [38]. In this study, we have used a variation of Word2Vec called Skip-gram model since it is more accurate in recent studies involving development tasks [38][6]. Since different text fields may have a different number of words, we transformed the text into a single word vector by averaging all the word vectors of all the words in the text. More formally, given a text that contains n words, suppose w_i is the word vector of the i^{th} word in the text. A text embedding t is generated as follows:

$$t = \frac{\sum_{i=0}^{n} w_i}{n}$$

3.3 The Classification model

A system that recommends the right actions (`accept`, `respond`, or `reject`) on PRs could be beneficial for the integrators. Especially in situations where integrators have restricted time to review a given number of PRs. Such recommendation of actions on PRs can help them to schedule their review process accordingly. In case the workload is higher, then the integrator can first focus on the `accept` PRs recommended by the model, because such PRs could be related to (i) minor bug fixes, (ii) new minor features, or (iii) refactoring activities. In a quick go through the integrator can accept the recommended PRs which can substantially reduce the workload in less amount of time. The `respond` class models PRs that are acceptance worthy but need discussion. Thus, indicating these PRs to integrators can allow them to give timely responses to the contributors so that high-quality PRs could be merged faster into the master branch. Recommending the action `reject` on PRs, could prevent the integrators from wasting time in reviewing low-quality PRs which are less likely to be accepted.

CARTESIAN models the PR recommendation as a multi-class problem. In particular, our approach identifies the likely right actions that integrators should undertake on each specific PRs. Seven different classifiers, discussed in section 5.1, are trained on the features discussed above, to select the best among them to build *CARTESIAN*.

Table 3: Features extracted from the Contributor dimension

Feature	Description
followers	The number of followers of the contributor who has submitted the pull request.
closed_num	The number of closed pull requests of the contributor.
is_contributor	A binary variable is True if the creator is a contributor.
public_repos	The number of public repositories of the contributor
is_core_member	Is the creator, the core member of the project organization?
contributions	The number of contributions to the current contributor.
user_accept_rate	The rate of the merged PRs of the contributor up to the creation of the examined PR.
accept_num	The total number of the merged PRs of the contributor.
closed_num_rate	The rate of the closed PRs of the contributor.
prev_PRs	The number of the previous PRs created the contributor.
following	The number of GitHub users followed by the contributor.

Table 4: Features extracted from the Integrator dimension

Feature	Description
line_comments_count	The number of line comments in the source code of the examined PR.
comments_word_count	The number of words in the review and discussion comments.
participants_count	The number of participants in the examined PR.
num_comments	The number of reviews and discussion comments on the examined PR.
at_mention	Does the last comment of the PR contain '@' mentions?

4 EXPERIMENTAL DESIGN

The main goal of this study is to assess the effectiveness of *CARTESIAN* in identifying the right actions to take on PRs, as well as its

Table 5: GitHub projects selected

Projects	#PRs	#Watchers	#Stars	#Contributors	#Forks	OPRs/day
Kubernetes (Go)	35672	2306	33873	1995	11639	529.36
Nixpkgs (Nix)	27792	133	2291	2210	2595	235.67
Cmssw (C++)	21089	76	527	1990	2389	135.54
Tensorflow (C++)	6729	7619	93054	1562	59180	125.48
Rails (Ruby)	20305	2623	3900 5	4464	15423	116.29
Rust (Rust)	24566	1231	27100	2348	4705	100.68
Symfony (PHP)	16255	1276	17009	2094	5986	82.66
Pandas (Python)	8272	810	13504	1311	5316	81.52
React (JavaScript)	6521	5611	90956	1254	16809	80.50
Salt (Python)	28598	598	8684	3089	3943	76.45
Scikit-Learn (Python)	5390	2059	26640	1202	13044	68.76
Yii2 (PHP)	5939	2059	26640	1202	13044	67.81
Cdnjs (JavaScript)	7281	246	6078	1576	3384	65.61
Terraform (Go)	7758	797	11448	1434	3659	65.38
Moby (Go)	18543	3322	48079	1950	13664	63.94
Django (Python)	9606	1951	32522	1844	13462	61.92
Opencv (C++)	7455	2001	23002	1154	16149	38.25
Angular.js (JavaScript)	7557	4369	58127	1808	28469	29.04
Laravel (PHP)	13090	1013	11283	2001	5197	13.47

usefulness when used to prioritize PRs. In this section, we present all the steps that we followed to construct the dataset, the evaluation metrics which have been used to evaluate the performance of our approach, and the experiments conducted.

4.1 Study Subject

GitHub is the world's largest open-source platform hosting open source project repositories. We extracted data of 19 projects hosted on GitHub, as reported in Table 5. We collected PR data from the creation time of the projects until February 2018. We used the REST API V3 interface provided by GitHub to crawl the data. The following steps are followed to prepare the data.

4.1.1 *Project selection.*

We have selected popular GitHub projects, in terms of stars [5], which are still active and have a long history on average 7 years. These projects are diverse as they are written in different programming languages and they belong to different domains including data science frameworks, web frameworks, operating systems, compiler, and others. Besides, to obtain high-quality experimental data, we have used the following additional selection strategy:

- Projects that are developed in strict accordance with the process of the pull-based development model.
- Projects with more than 13 open pull requests per day. We applied this filter to target projects having a large number of open pull requests rate per day.
- Projects with more than 1000 contributors, to increase PRs heterogeneity.

The selected projects along with their characteristics are reported in Table 5.

4.2 Datasets Construction

Our dataset consists of 278,418 pull requests data extracted from the 19 projects given in table 5. These PRs are extracted since the projects are started until February 2018. We have divided the dataset into train and test datasets as follows: PRs which are closed before September 1, 2017, are used as the train data and PRs submitted after September 1, 2017, till February 28, 2018, are used as the test data. The rationale of this choice is to make have in training and test set PR spanned in two different years. The distribution of the three classes of PRs in our dataset is: `accept` (11.6%), `respond` (60%), and `reject` (28.4%). The distribution shows that the dataset is imbalanced.

Replication package: We make available in our replication package[4] the scripts and all the raw data used for this research.

4.3 Evaluation Measurements

To answer our RQ1 and evaluate the classification performance of *CARTESIAN* we use the traditional information retrieval metrics: precision, recall, F-measure and Accuracy. F-measure also called the F_1 score is the harmonic mean of precision and recall.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Accuracy describes the proportion of correctly classified samples out of total samples. The general formula for accuracy calculation is given below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative.

To answer our RQ2 and assess the prioritization effectiveness we used the following metrics:

4.3.1 *Mean Average Precision.*

Mean Average Precision (MAP) for a set of queries is the mean of the average precision scores for each query [1]. It is a measure that considers both positive and negative cases and their sort order. The more positive cases that are placed before the negative ones, the greater is the value of MAP. It is computed as follows:

$$P(j) = \frac{\sum_{k:\pi(k)\leqslant\pi(j)} y_k}{\pi(j)}$$

$$AP_i = \frac{\sum_{j=1}^{n_i} P(j) * y_j}{\sum_{j=1}^{n_i} y_j}$$

$$MAP = \frac{\sum_{i=1}^{n} AP_i}{n},$$

where y_j indicates whether the j^{th} element in the sorted order is a positive example and $\pi(k)$ is the sorting position.

4.3.2 *Average Recall.*

Average Recall (AR) for a set of queries is the mean of the Recall scores for each query. Recall refers to the percentage of correct samples that have been retrieved to the total number of correct samples. We used AR to measure the proportion of positive samples in the top@N sample for all queries: $Recall = \frac{k}{j}$, where k is the number of positive samples in top@N, and j is the total number of positive samples.

$$AR = \frac{\sum_{i=1}^{n} Recall_i}{n},$$

where $Recall_i$ is the Recall of query i and n is the total number of queries.

[4]https://github.com/Tools-Demo/cart_package

4.4 Experiments

4.4.1 Experiment I (RQ1). The aim of the experiment I is to evaluate the extent to which our approach can be used to predict the action that integrators should take on PRs. Thus to pursue this goal we have used different machine learning classifiers. Specifically, we trained seven classifiers: Logistic Regression [36], SVM [25], Random Forest [17], Decision Trees, Naive Bayes, K-Nearest Neighbor and XGBoost [7] to recommend the right actions on PRs and the model with highest f_1 score and accuracy is selected as the ultimate model to develop *CARTESIAN*. We selected these classifiers as they have been used in previous studies on PRs [14, 35, 41]. Hence, all seven classifiers are trained using a 10-fold cross-validation strategy. As the PRs are processed in a time frame, PRs that are already closed cannot be predicted using PRs submitted after them (predicting past PRs). Therefore, we have sorted the PRs in the dataset according to their closing time. Then the whole dataset is split into 11 equal folds e.g. fold1, fold2, ... fold11. Each model is trained on the previous fold(s) and tested on the following fold. During iteration 1, each model is trained on fold1 and tested on fold2. Similarly, during iteration 2 each model is trained on fold1+fold2 and tested on fold3 and so on. The scores of the ten iterations are then averaged to get the final results.

To select the best features among all the features in the training dataset, we have used the importance score of each feature [12]. To get the final set of features, we trained and tested the model by iteratively removing features by their importance, recording the precision, recall, and accuracy along the way. The iteration is stopped when the accuracy starts decreasing than the one achieved with all the features.

4.4.2 Experiment II (RQ2). To assess how helpful is *CARTESIAN* in prioritizing useful PRs, we performed two steps. Firstly, we compared *CARTESIAN* with baseline models, the prioritizing criteria studied by Gousios et al. [16]. Gousios et al. [16] surveyed 749 integrators and they concluded from their survey that integrators prioritize the PRs based on their age and size. They further explained that the integrators prefer to treat PRs in first-in-first-out priority. Similarly, PRs having fewer modifications in terms of line of code added and/or deleted are given high priority. We have used these two criteria as baselines and created two models, namely the FIFO model and the Sized-Based Model (SBM). In the case of FIFO model, the test sample is prioritized based on the creation date/time of the PRs and then the MAP and AR are calculated, whereas in the case of Sized-based model the test samples are prioritized based on the size (i.e., number of lines added and deleted) of the PRs. On the other hand, *CARTESIAN* sorts the PRs in the test sample according to the identified classes, by moving (i) on top all the PRs classified as `accept`, and (ii) soon after the PRs belonging to the `respond` class.

To evaluate the prioritization performance we use Mean Average Precision (MAP) and Average Recall (AR) on the top@20 PRs prioritized by each considered approach. In particular, for MAP and AR computation, a PR is considered a positive sample if it has been actually accepted and merged in the project (either directly or after discussion), while it is a negative sample in case of rejection.

Secondly, we qualitatively analyzed the top@20 PRs, from each subsample, recommended by *CARTESIAN* to investigate whether they could be useful for the integrators. We have followed the same procedure for coding and thematic analysis as performed by Gousios et al. [14]. The first author examined 100 PRs (title and description of PRs) and identified their types. Another author then examined the next 100 PRs to validate the types of recommended PRs. The two examined samples are merged after the cross-validation to obtain the final samples.

5 RESULTS

In this section, we report and discuss the main results achieved in our empirical study.

5.1 RQ1: To which extent is *CARTESIAN* able to identify the actions that should be undertaken by integrators on PRs?

The goal of this research question is to assess the performance of the proposed approach in automatically recommending the actions that should be taken on each PRs. To this aim, we compare the results obtained by several machine learning (ML) algorithms, that have proven to be useful in prior studies on PRs classification or prioritization [14, 35, 41]. Table 6 reports the precision, recall, F_1 score, and accuracy achieved by the aforementioned ML algorithms trained by using the features discussed in Section 3.1 and the 10-fold cross-validation strategy. More specifically, for each considered ML algorithm, we report the precision, recall, and F_1 score results achieved in each class, along with the average values of those metrics and the accuracy score. The 'Class' column represents the three classes of PRs.

As shown in Table 6, XGBoost outperforms the other classifiers in correctly identifying the actual actions on PRs, achieving the highest values in all the considered metrics. Besides, XGBoost achieves the highest values of precision, recall, and F_1-score for all the classes of the PRs. While the Random Forest classifier obtains performance that is comparable to the one achieved through the XGBoost technique, the other algorithms perform significantly worse, with the Naive Bayes algorithm that achieves the worst results. It is worth noticing that all the models can better identify the PRs belonging to the `accept` category than the ones belonging to the other classes. This could depend on the fact that our dataset is unbalanced and the reason behind the success of XGBoost and Random Forest models could be connected with the fact that both of them are tree-based ensemble classifiers and can work well with imbalanced data [8, 30].

With the aim of investigating the most important features in the training dataset, we followed the procedure defined in [12]. In particular, we exploited the feature's importance as computed by the XGBoost classifier, which provides the **Gain** of each feature. The Gain associated with a specific feature represents the relative contribution of the corresponding feature in the construction of the boosted decision trees. More specifically, we ran the XGBoost classifier 50 times on the training dataset, recording the feature's importance along the way. The average of Gain values obtained in the 50 iterations is used as the final feature's importance score. To get the final set of relevant attributes, after ranking them according to the importance scores, we trained and tested the model by iteratively removing features (starting from the least relevant ones) and

Table 6: Results of the seven classifiers trained using 10-fold cross-validation technique.

Model	Class	Precision	Recall	F_1 Score	Accuracy
XGBoost	accept	0.87	0.93	0.90	**0.86**
	response	0.93	0.98	0.95	
	reject	0.81	0.66	0.72	
	Average	**0.86**	**0.86**	**0.86**	
Random Forest	accept	0.86	0.91	0.88	**0.84**
	response	0.91	0.99	0.95	
	reject	0.75	0.63	0.68	
	Average	**0.84**	**0.84**	**0.84**	
Decision Tree	accept	0.80	0.95	0.87	**0.81**
	response	0.90	0.98	0.94	
	reject	0.78	0.42	0.54	
	Average	**0.81**	**0.81**	**0.79**	
Logistic Regression	accept	0.82	0.82	0.82	**0.76**
	response	0.72	0.90	0.74	
	reject	0.64	0.53	0.58	
	Average	**0.77**	**0.76**	**0.75**	
SVM	accept	0.76	0.84	0.79	**0.70**
	response	0.25	0.15	0.13	
	reject	0.57	0.56	0.56	
	Average	**0.66**	**0.70**	**0.67**	
K-Nearest Neighbor	accept	0.73	0.70	0.71	**0.63**
	response	0.48	0.47	0.46	
	reject	0.48	0.52	0.50	
	Average	**0.64**	**0.63**	**0.63**	
Naive Bayes	accept	0.71	0.72	0.71	**0.60**
	response	0.36	0.91	0.51	
	reject	0.60	0.20	0.30	
	Average	**0.65**	**0.60**	**0.58**	

recording the precision, recall, and accuracy along the way. After removing 39 less important attributes, we observed that by only considering 31 features we were able to achieve the same results like the ones reported in Table 6. The final list of selected features along with their average Gain value is given in Figure 1.

As highlighted by the feature selection process, `num_comments` (i.e., the number of review and discussion comments), `is_contributor` (i.e., the role of the PR submitter), and `participants_count` (i.e., the number of participants in the discussion) are the most relevant features for identifying the actions that should be undertaken on PRs. Therefore, looking at Figure 1, we can conclude that the classification is largely driven by features in the Contributor and Integrator dimensions.

RQ1 Summary: *CARTESIAN* can automatically recommend the actions to undertake on PRs with average precision, recall, f_1-score, and accuracy of 86%, using XGBoost as the underlying machine learning algorithm. The classification accuracy is largely driven by features in the Contributor and Integrator dimensions.

5.2 RQ2: To which extent can our approach be used to prioritize useful PRs?

To investigate the usefulness of our approach when adopted in a practical context, we evaluate the performance achieved by *CARTESIAN* when used to prioritize PRs. In particular, we compare the

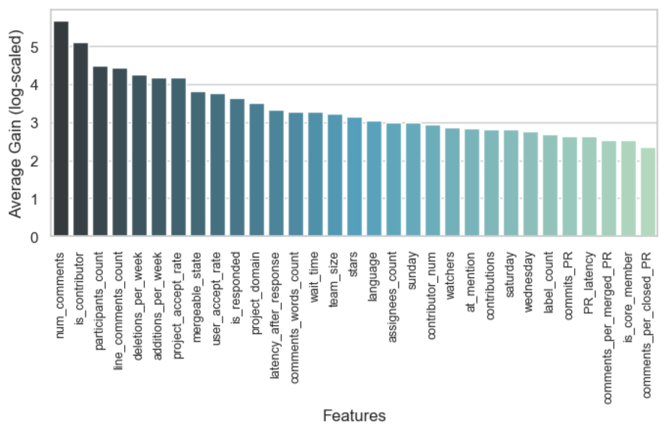

Figure 1: Features selected as a result of feature selection process.

performance of *CARTESIAN* with two baseline models; first-in-first-out (FIFO), which prioritizes PRs depending on the submission date, and Sized-Based Model (SBM), which rewards PRs having a smaller size (number of lines added and deleted). To this aim, a statistically significant sample of 380 PRs extracted from our dataset has been used for comparison purposes. This sampling was necessary to have an equal proportion of PRs from each class. Then, the test sample has been shuffled and divided into 10 equal subsamples (sample_0, sample_2..., sample_9), where each subsample contains 38 PRs. The proportion of PRs from each class in the subsamples is random, to better reflect real situations. Each subsample has been provided to *CARTESIAN* to predict the *action* for each PR and then the PRs in each sub sample are prioritized according to the predicted actions, as detailed in Section 4.4.2. Similarly, PRs in each subsample are prioritized using the baseline models. MAP and AR for top@20 PRs have been computed for all three models and the results are shown in Figure 2. *CARTESIAN* outperforms the baseline models in both MAP and AR. We used the Mann-Whitney pairwise test (with the p-value adjusted using the Benjamini & Yekutieli [3] method), to check if the differences in the MAP and AR results are statistically significant, and effect size measure to quantitatively characterize the eventual differences. Results of the aforementioned test show that *CARTESIAN* is better in prioritizing the most likely to be accepted PRs than the FIFO baseline model with statistical evidence (MAP: $p = 0.0005$, AR: $p = 0.0058$) and large effect size ($\delta = 0.9$). Similarly, the performance of *CARTESIAN* is significantly better than the SBM baseline model (MAP: $p = 0.0005$, AR: $p = 0.0058$) with large effect size ($\delta = 0.9$).

To qualitatively corroborate the quantitative results obtained, we also analyze the results produced by all the approaches and try to investigate the motivations behind the higher performance achieved by *CARTESIAN*. To this purpose, we inspected the top@20 PRs from the sample_0 and sample_5 prioritized by the three models as shown in Table 7. The 'Rank' column represents the rank given to each PR by the models and the 'Label' column represents the actual action undertaken on the PRs. Looking at Table 7 we can observe that in both the samples *CARTESIAN* has prioritized PRs

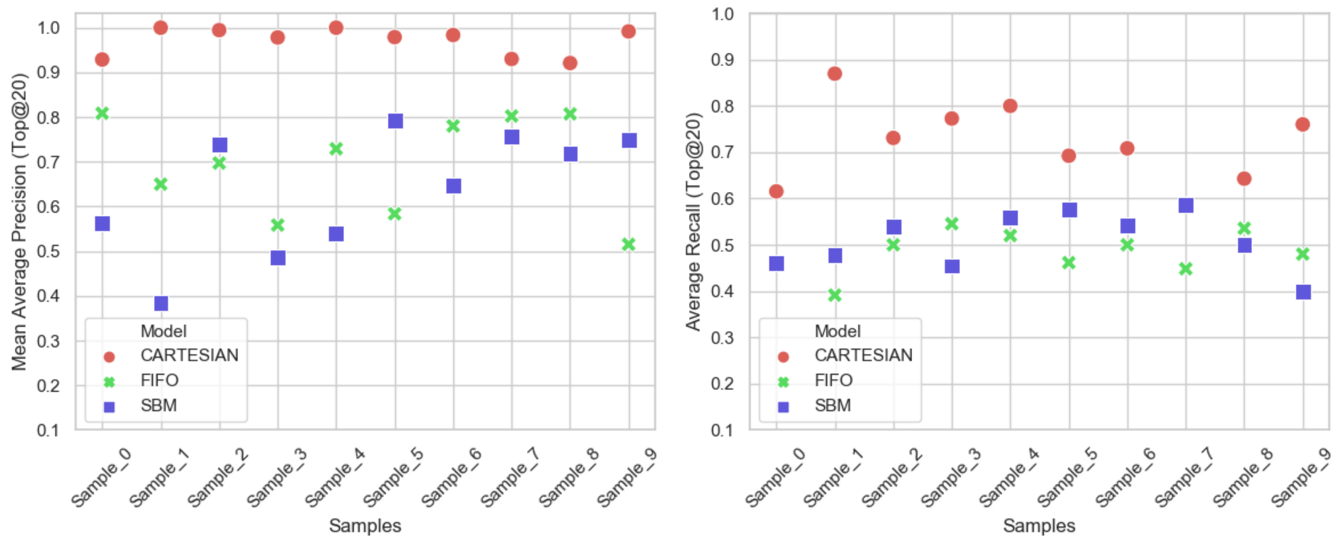

Figure 2: Top@20 MAP and AR achieved by CARTESIAN and baseline approaches.

that are most likely to be accepted and/or replied. In contrast the prioritization made by the baseline models can reward PRs that are likely to be rejected, and this could result in a waste of the integrators' time and efforts.

To gain further insights into our results and better understand the kinds of PRs that are rewarded by *CARTESIAN*, we also manually examined the top@20 PRs prioritized by our approach in each of the aforementioned samples (sample_0,..., sample_9). As prior work [16] showed that integrators give high priority to PRs that either fixes a bug or add new features, our aim was to understand whether *CARTESIAN* can prioritize those kinds of PRs. As reported in Table 8, most frequent types of PRs prioritized by our approach are bug fixes (24%) and new features (17%) which indicates that *CARTESIAN* is a valid support for giving high priority to PRs that integrators consider useful. Only 12.5% of the recommended PRs are Doc PRs, which are considered less important [45].

RQ2 Summary: *CARTESIAN* can significantly outperform FIFO and SBM baseline approaches in prioritizing PRs that are worthy to be either directly accepted or accepted after discussion. Moreover, the qualitative analysis results show that *CARTESIAN* is a valid support for giving high priority to PRs that integrators consider useful.

6 THREATS TO VALIDITY

Threats to construct validity. To design *CARTESIAN*, we used a set of features that may only partially model all the relevant aspects that integrators take into account when prioritizing PRs, as there may be further factors to which integrators actually pay attention that we not considered. To mitigate this issue, all the features considered in our work have been selected considering previous studies [14, 15, 35], which demonstrated the importance of such factors and their relevance to PRs prioritization.

Threats to internal validity. To construct our dataset, we collected PRs having the closed status assigned. However, some of the not-accepted or not-responded pull requests that we considered in our dataset could be eventually reopened in the future [24] and accepted after reconsideration. To partially alleviate this concern, we avoided collecting PRs that have been recently closed, considering PRs that are still closed after more than a year, this to reduce the probability of the PRs to be reconsidered. Moreover, to identify accepted PRs in our dataset, we rely on the GitHub's *Merged* attribute and some of the PRs may appear as non-merged even if they are actually merged [19], To mitigate this concern, we selected projects for which the percentage of merged pull requests with respect to the number of closed pull requests seems reasonable. Moreover, we manually checked whether the majority of the PRs in the selected projects are merged through GitHub's pull request merge facilities.

Threats to conclusion validity. To answer our RQ1, we (i) compared the classification performance obtained by different machine learning algorithms, by evaluating widely-used metrics in the information retrieval field, and (ii) followed a well-defined procedure to evaluate the importance of features for performing PRs classification. Also, to measure the effectiveness of *CARTESIAN* when adopted for prioritizing PRs (RQ2), we compared the Mean Average Precision and Average Recall values achieved by *CARTESIAN* and two baselines approaches on top-ranked PRs. For demonstrating that *CARTESIAN* outperforms baseline models with statistical evidence, we used appropriate statistical procedures and effect size measures. Besides, we manually inspected the PRs prioritized by our approach to gain qualitative insights into the types of PRs that *CARTESIAN* is able to reward.

Threats to external validity. The main threat to external validity could be related to the specificity of our dataset in which we collected PRs from a subset of projects from Github. Such projects could not be adequately representative of all the open-source projects adopting pull-based development processes. To address this issue,

Table 7: Top 20 PRs from two random samples prioritized by the three models.

Sample Zero						
Rank	CARTESIAN		FIFO		SBM	
	PR_ID	Label	PR_ID	Label	PR_ID	Label
1	react-12154	accept	scikit-learn-9686	respond	tensorflow-12885	reject
2	nixpkgs-30865	accept	nixpkgs-28991	respond	cmssw-20468	respond
3	nixpkgs-33594	accept	tensorflow-12885	reject	opencv-9957	reject
4	nixpkgs-31286	accept	salt-43438	accept	framework-21236	accept
5	symfony-24156	accept	cmssw-20468	respond	nixpkgs-28991	respond
6	nixpkgs-30733	accept	rust-44639	respond	symfony-25249	reject
7	nixpkgs-28933	accept	framework-21236	accept	framework-23320	accept
8	tensorflow-16637	accept	rust-44836	reject	nixpkgs-35821	reject
9	salt-46136	accept	django-9141	accept	nixpkgs-30974	accept
10	framework-20979	accept	nixpkgs-29892	reject	nixpkgs-34920	reject
11	nixpkgs-29438	accept	kubernetes-53753	respond	rust-44639	respond
12	salt-45419	respond	rails-30868	accept	kubernetes-53753	respond
13	pandas-18177	respond	moby-35220	accept	kubernetes-58934	reject
14	opencv-10697	respond	tensorflow-13877	reject	nixpkgs-33259	accept
15	pandas-18620	reject	nixpkgs-30696	accept	nixpkgs-29892	reject
16	nixpkgs-29309	respond	opencv-9957	reject	nixpkgs-35459	respond
17	nixpkgs-32471	reject	nixpkgs-30974	accept	nixpkgs-33658	respond
18	cmssw-21910	respond	rust-45776	reject	rust-44836	reject
19	tensorflow-15081	respond	kubernetes-56659	reject	kubernetes-57087	respond
20	rust-45853	respond	symfony-25249	reject	django-9141	accept
Sample Five						
Rank	CARTESIAN		FIFO		SBM	
	PR_ID	Label	PR_ID	Label	PR_ID	Label
1	react-12075	accept	nixpkgs-28861	reject	cdnjs-11977	reject
2	nixpkgs-33452	accept	cmssw-20350	reject	nixpkgs-33423	respond
3	opencv-10738	accept	nixpkgs-29120	respond	framework-22673	accept
4	framework-21982	accept	salt-43434	accept	pandas-18990	accept
5	framework-21718	accept	nixpkgs-29486	respond	nixpkgs-35503	accept
6	nixpkgs-35503	accept	kubernetes-53229	respond	nixpkgs-33452	accept
7	framework-22716	accept	kubernetes-53389	respond	tensorflow-16570	respond
8	nixpkgs-31005	accept	kubernetes-53472	reject	salt-46203	accept
9	framework-22673	accept	cdnjs-11977	reject	kubernetes-60252	reject
10	salt-46203	accept	framework-21718	accept	opencv-10738	accept
11	salt-45260	accept	nixpkgs-30963	reject	framework-21982	accept
12	pandas-18990	accept	nixpkgs-31005	accept	framework-21718	accept
13	salt-43434	accept	pandas-18102	reject	kubernetes-57403	respond
14	kubernetes-53389	respond	framework-21982	accept	kubernetes-53229	respond
15	cmssw-22353	reject	nixpkgs-31488	respond	nixpkgs-32742	respond
16	kubernetes-53229	respond	kubernetes-55730	respond	tensorflow-17026	reject
17	nixpkgs-32069	reject	django-9361	reject	nixpkgs-31005	accept
18	nixpkgs-33088	respond	pandas-18420	respond	tensorflow-15597	reject
19	nixpkgs-29486	respond	nixpkgs-32069	reject	cmssw-22353	reject
20	tensorflow-16570	respond	nixpkgs-32742	respond	salt-45260	accept

such projects were selected considering well-defined selection criteria (see Section 4.1.1). In addition, with the aim of increasing the heterogeneity of data, we selected projects having (i) different natures, (ii) developed by different developers' communities, and (iii) implemented through different programming languages. Finally, our results are based on projects developed on GitHub, thus it is unclear whether they can be generalized to further platforms for open-source software development. For this reason, in the future, we plan to evaluate our approach on further projects mined from different platforms.

7 CONCLUSION

In this study, we proposed *CARTESIAN*, an automated approach to help integrators selecting useful PRs when the pull-based software development process is adopted. In particular, our approach is able to automatically recommend the *actions* (i.e., `accept, respond, reject`) that integrators should undertake on each specific PRs. We evaluated (i) the performance achieved by *CARTESIAN* in identifying such PR actions on a dataset containing more than 270,000 pull-requests extracted from 19 different projects, as well as (ii) the effectiveness of our approach when used to prioritize PRs. Results

Table 8: Types of pull requests prioritized by CARTESIAN in top@20.

Type of PR	Short Description	%
Bug fix	PRs that fix bug or bugs	24
New feature	PRs that add new features to the project.	17
Version update	PRs which updates an existing package to a new version or update builds.	16.5
Refactoring	PRs the refactor the existing code especially removing unused code snippets, duplicated code, renaming methods, and update the code to make it more human-readable.	13.5
Doc PRs	PRs which update the documents of the project.	12.5
Tests	PRs which either add a new test for an existing method/class, fix or update existing unit tests.	4.5
Other	These are the PRs which could not be classified due to lack of information	12

of our study show that (i) *CARTESIAN* can identify PR actions with an average precision and recall of about 86%, and (ii) *CARTESIAN* can better prioritize the PRs that will be actually either directly accepted or accepted after discussion than both FIFO and Size-based approaches. A qualitative analysis of the PRs prioritized by our approach, allowed us to observe that many of those PRs are useful for integrators, as they are related to either bug fixes or implementations of new features.

Future studies will be aimed at investigating further aspects that can be related to the PRs recommendation. Specifically, our aim is to integrate our PRs recommendation approach into platforms supporting pull-based development (e.g. GitHub, Bitbucket, etc.), in order to (i) further evaluate the usefulness of *CARTESIAN*, and (ii) discover additional factors that can be used to improve the recommendation performed by our approach.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1403400. The first author of this paper is sponsored by the CAS-TWAS President's Fellowship for International Ph.D. students.

REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 2011. *Modern Information Retrieval the Concepts and Technology Behind Search.* DBLP.
[2] Earl T. Barr, Christian Bird, Peter C. Rigby, Abram Hindle, Daniel M. German, and Premkumar Devanbu. 2012. Cohesive and Isolated Development with Branches. In *Fundamental Approaches to Software Engineering*, Juan de Lara and Andrea Zisman (Eds.). Springer Berlin Heidelberg, 316–331.
[3] Yoav Benjamini and Daniel Yekutieli. 2001. The control of the false discovery rate in multiple testing under dependency. *Ann. Statist.* 29, 4 (08 2001), 1165–1188. https://doi.org/10.1214/aos/1013699998
[4] Christian Bird and Alberto Bacchelli. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. IEEE. https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/
[5] H. Borges, A. Hora, and M. T. Valente. 2016. Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International*

Conference on Software Maintenance and Evolution (ICSME). 334–344. https://doi.org/10.1109/ICSME.2016.31

[6] C. Chen, S. Gao, and Z. Xing. 2016. Mining Analogical Libraries in Q A Discussions – Incorporating Relational and Categorical Knowledge into Word Embedding. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 338–348. https://doi.org/10.1109/SANER.2016.21

[7] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[8] D. J. Dittman, T. M. Khoshgoftaar, and A. Napolitano. 2015. The Effect of Data Sampling When Using Random Forest on Imbalanced Bioinformatics Data. In *2015 IEEE International Conference on Information Reuse and Integration*. 457–463. https://doi.org/10.1109/IRI.2015.76

[9] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2019. Confusion in Code Reviews: Reasons, Impacts, and Coping Strategies. In *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, Xinyu Wang, David Lo, and Emad Shihab (Eds.). IEEE, 49–60. https://doi.org/10.1109/SANER.2019.8668024

[10] Yuanrui Fan, Xin Xia, David Lo, and Shanping Li. 2018. Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering* 23, 6 (01 Dec 2018), 3346–3393. https://doi.org/10.1007/s10664-018-9602-0

[11] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: how programmers *really* look at pull requests. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Rick Kazman and Liliana Pasquale (Eds.). ACM, 51–60. https://doi.org/10.1109/ICSE-SEIS.2019.00014

[12] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. 2010. Variable selection using random forests. *Pattern Recognition Letters* 31, 14 (2010), 2225 – 2236. https://doi.org/10.1016/j.patrec.2010.03.014

[13] K. V. Ghag and K. Shah. 2015. Comparative analysis of effect of stopwords removal on sentiment classification. In *2015 International Conference on Computer, Communication and Control (IC4)*. 1–6. https://doi.org/10.1109/IC4.2015.7375527

[14] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. ACM, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[15] G. Gousios, M. Storey, and A. Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In *International Conference on Software Engineering (ICSE)*. 285–296.

[16] G. Gousios, A. Zaidman, M. Storey, and A. v. Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 358–368. https://doi.org/10.1109/ICSE.2015.55

[17] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[18] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48 – 62. https://doi.org/10.1016/j.infsof.2016.10.006

[19] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2014. The promises and perils of mining GitHub. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*. 92–101. https://doi.org/10.1145/2597073.2597074

[20] Zhifang Liao, Yanbing Li, Dayu He, Jinsong Wu, Yan Zhang, and Xiaoping Fan. 2017. Topic-Based Integrator Matching for Pull Request. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference* (2017), 1–6.

[21] J. Liu, J. Li, and L. He. 2016. A Comparative Study of the Effects of Pull Request on GitHub Projects. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 313–322. https://doi.org/10.1109/COMPSAC.2016.27

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). arXiv:1310.4546 http://arxiv.org/abs/1310.4546

[23] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (July 2002), 309–346. https://doi.org/10.1145/567793.567795

[24] Abdillah Mohamed, Li Zhang, Jing Jiang, and Ahmed Ktob. 2018. Predicting Which Pull Requests Will Get Reopened in GitHub. In *25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018*. 375–385. https://doi.org/10.1109/APSEC.2018.00052

[25] William S Noble. 2006. What is a support vector machine? *Nature biotechnology* 24, 12 (2006), 1565.

[26] Sebastiano Panichella. 2018. Summarization techniques for code, change, testing, and user feedback (Invited paper). In *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018*, Cyrille Artho and Rudolf Ramler (Eds.). IEEE, 1–5. https://doi.org/10.1109/VST.2018.8327148

[27] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[28] Martin Porter. [n.d.]. The Porter stemmer Algorithm. http://tartarus.org/~martin/PorterStemmer/. Accessed October 23, 2019.

[29] Mohammad Masudur Rahman and Chanchal K. Roy. 2014. An Insight into the Pull Requests of GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. ACM, New York, NY, USA, 364–367. https://doi.org/10.1145/2597073.2597121

[30] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. 2010. RUSBoost: A Hybrid Approach to Alleviating Class Imbalance. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40, 1 (Jan 2010), 185–197. https://doi.org/10.1109/TSMCA.2009.2029559

[31] Jacek Sliwerski, Thomas Zimmermann, and Andreas Zeller. 2005. When Do Changes Induce Fixes? *SIGSOFT Softw. Eng. Notes* 30, 4 (May 2005), 1âĂŞ5. https://doi.org/10.1145/1082983.1083147

[32] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 499–510. https://doi.org/10.1145/2950290.2950299

[33] Patanamon Thongtanunam, Raula Gaikovina Kula, Ana Erika Camargo Cruz, Norihiro Yoshida, and Hajimu Iida. 2014. Improving Code Review Effectiveness Through Reviewer Recommendations. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering* (Hyderabad, India) *(CHASE 2014)*. ACM, New York, NY, USA, 119–122. https://doi.org/10.1145/2593702.2593705

[34] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India). ACM, New York, NY, USA, 356–366. https://doi.org/10.1145/2568225.2568315

[35] E. v. d. Veen, G. Gousios, and A. Zaidman. 2015. Automatically Prioritizing Pull Requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 357–361. https://doi.org/10.1109/MSR.2015.40

[36] Strother H Walker and David B Duncan. 1967. Estimation of the probability of an event as a function of several independent variables. *Biometrika* 54, 1-2 (1967), 167–179.

[37] Yi Wang and David Redmiles. 2016. Cheap talk, cooperation, and trust in global software engineering. *Empirical Software Engineering* 21, 6 (01 Dec 2016), 2233–2267. https://doi.org/10.1007/s10664-015-9407-3

[38] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. 2016. From Word Embeddings to Document Similarities for Improved Information Retrieval in Software Engineering. In *International Conference on Software Engineering*. 404–415.

[39] H. Ying, L. Chen, T. Liang, and J. Wu. 2016. EARec: Leveraging Expertise and Authority for Pull-Request Reviewer Recommendation in GitHub. In *2016 IEEE/ACM 3rd International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*. 29–35. https://doi.org/10.1109/CSI-SE.2016.013

[40] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. 2015. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 367–371. https://doi.org/10.1109/MSR.2015.42

[41] Y. Yu, H. Wang, G. Yin, and C. X. Ling. 2014. Who Should Review this Pull-Request: Reviewer Recommendation to Expedite Crowd Collaboration. In *2014 21st Asia-Pacific Software Engineering Conference*, Vol. 1. 335–342. https://doi.org/10.1109/APSEC.2014.57

[42] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204 – 218. https://doi.org/10.1016/j.infsof.2016.01.004

[43] Yue Yu, Gang Yin, Tao Wang, Cheng Yang, and Huaimin Wang. 2016. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences* 59, 8 (18 Jul 2016), 080104. https://doi.org/10.1007/s11432-016-5595-8

[44] Y. Zhang, G. Yin, Y. Yu, and H. Wang. 2014. A Exploratory Study of @-Mention in GitHub's Pull-Requests. In *Asia-Pacific Software Engineering Conference*. 343–350.

[45] Guoliang Zhao, Daniel Alencar da Costa, and Ying Zou. 2019. Improving the pull requests review process using learning-to-rank algorithms. *Empirical Software Engineering* 24, 4 (2019), 2140–2170.

[46] Y. Zhou, Y. Su, T. Chen, Z. Huang, H. C. Gall, and S. Panichella. 2020. User Review-Based Change File Localization for Mobile Applications. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.2967383