Pull Request Prioritization Algorithm based on Acceptance and Response Probability

Muhammad Ilyas Azeem*†
*University of Chinese Academy of Sciences
Beijing 100049, China
azeem@itechs.iscas.ac.cn

Qiang Peng†
†Lab for Internet Software Technologies, Institute of Software, CAS
Beijing 100190, China
pengqiang@nfs.iscas.ac.cn

Qing Wang*†‡§
‡State Key Lab of Computer Science, Institute of Software, CAS, §Corresponding author
Beijing 100190, China
wq@itechs.iscas.ac.cn

***Abstract*—Pull requests (PRs) prioritization is one of the main challenges faced by integrators in pull-based development. This is especially true for large open-source projects where hundreds of pull requests are submitted daily. Indeed, managing these pull requests manually consumes time and resources and may lead to delays in the reaction (i.e., acceptance or response) to enhancements or bug fixes suggested in the codebase by contributors. We propose an approach, called `AR-Prioritizer` (Acceptance and Response based `Prioritizer`), integrating a PRs prioritization mechanism that considers these two aspects. The results of our study demonstrate that our approach can recommend top@5, top@10, and top@20 most likely to be accepted and responded pull requests with Mean Average Precision of 95.3%, 89.6%, and 79.6% and Average Recall of 40%, 65.7%, and 92.9%. Moreover, `AR-Prioritizer` has outperformed the baseline models with a statistical significance in prioritizing the most likely to be accepted and responded to PRs.**

***Index Terms*—Pull requests, machine learning models, prioritization, pull-based development**

I. 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 participants contribute by submitting PRs (PRs) that can be reviewed and accepted or rejected [1], [2] rather than directly modifying the source code. However, popular GitHub projects such as `Kubernetes` and `Rails` receive tens or hundreds of PRs daily [3], [4]. Selecting what PRs to accept or to what PRs to respond becomes a challenge for the integrators [5], even more so since the aforementioned platforms do not provide mechanisms for prioritizing PRs when they are submitted.

A contributor creates a separate branch of a repository, e.g., to try new ideas or fix bugs without affecting the main branch [6]. 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 PR is created and integrators are requested to merge the changes into the master branch. Integrators have to manually inspect submitted PRs [7], [8] and decide whether to reject them, request the submitters to apply further changes [9], or merge them into the master branch. Given the high number of PRs, it is important to select those that deserve the integrators' attention, as reviewing PRs which are less likely to be accepted might lead to a waste of the integrators' time and resources [10].

However, in practice, PR prioritization is based on the integrator's experience and judgment [3], making this task error-prone and labor-intensive. Usually, integrators have to make a trade-off between selecting PRs for review and timely replying to the contributors. On the one hand, selecting a PR for review which is less likely to be accepted may lead to a waste of both the time and resources [10]. On the other hand, a lack of an integrator's response could frustrate the contributors causing them to lose interest in the project [11]. Indeed, contributors prefer to receive an immediate rejection for the PRs rather than not receiving any response [11].

Previous approaches to PR prioritization focus either on predicting the likelihood of response [12] or the likelihood of acceptance [1], [4], [9], [13]. However, specific PRs may be directly accepted and merged into the master branch without the need for initiating a PR discussion. As a matter of fact, this may occur when contributors apply minor fixes[1] or when the changes are aimed at fixing known issues[2]. We empirically assessed that on a total of 278,418 PRs analyzed in our study, 33,231 of them (i.e., 11.6%) were accepted without any discussion. In such cases, approaches exclusively based on response likelihood may fail in recognizing (and prioritizing) these useful PRs. On the other hand, exclusively relying on acceptance likelihood may lead to delays in obtaining feedback from integrator(s) on significant PRs having a lower initial probability of merging, with negative effects on the latency of these PRs [13]. For these reasons, we argue that PR prioritization strategies should be based on *predicted likelihood of acceptance and/or response* as this combination could help the integrators in deciding to review a particular PR and respond to it.

Thus, differently from prior work, we propose an approach, called `AR-Prioritizer` (`A`cceptance and `R`esponse based `Prioritizer`), which automatically ranks PRs according to their acceptance and response probabilities, with the goal of prioritizing all the important PRs that are worthy to be considered for merging. In particular, our prioritization approach

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

978-1-7281-8913-0/20/$31.00 ©2020 IEEE
DOI 10.1109/QRS51102.2020.00041

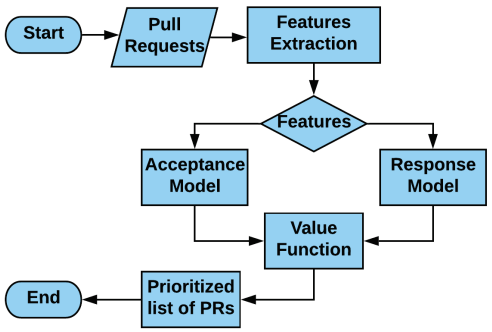

Fig. 1. The flow of the prioritization algorithm

is designed for giving the highest priority to PRs having *both* high acceptance and high response probabilities for supporting integrator(s) in providing feedback to contributors on PRs that are worthy to be accepted, this to minimize the time required to obtain updates from contributors on useful PRs. The name `AR-Prioritizer` is inspired by this idea of relying on both the response and acceptance probability for enabling the PRs prioritization. Fig. 1 illustrates the flow of our proposed algorithm. Specifically, two prediction models have been trained on features, extracted from the context of the PR, to predict the probability of PR acceptance and PR response on a given day. A high acceptance probability suggests that the specific PR is worthy to be accepted and, therefore, it may be reviewed first. Similarly, a high response likelihood highlights the importance of the specific PR and the need to receive updates from the contributor's side [12]. Our algorithm combines the probability of acceptance and the importance of the PRs to produce the prioritized PRs list.

The contributions of this paper include the following:

- We investigate the effectiveness of applying several machine learning techniques including Random Forest (RF), Logistic Regression (LR), Support Vector Machine (SVM), and XGBoost (XGB) on the classification performance in both the PR acceptance and response prediction tasks.
- We design a machine learning-based algorithm that relies on both acceptance and response likelihood to identify (and prioritize) the PRs that will be actually merged and responded, with a mean average precision of 89.6% and an average recall of 65.7% in top@10 PRs.
- We compare the results obtained by our approach with the ones achieved by baseline approaches based on either response [12] or acceptance probability [1].

The remainder of the paper is organized as follows. Related work is described in section II. Section III describes the approach used, Section IV presents the research questions investigated in this study and the research design, Section V describes the results and analysis, Section VI discusses the threats to validity of the study and Section VII concludes our work.

II. Related Work

Gousios *et al.* [3], [11] surveyed GitHub developers to investigate the practices and challenges faced by both the integrators and contributors in the PbD model. Their findings reveal that maintaining PR quality and PR prioritization are the two main challenges faced by the integrators in PbD [3]. Similarly, lack of integrator's responsiveness is the main challenge faced by the contributors in PbD [11]. Rahman *et al.* [14] study revealed that age, maturity, and the number of developers and their experience can affect the success and failure rates of PRs. Zhang *et al.* [15] conducted an exploratory study of @-mentions in PbD, demonstrating that @-mention is beneficial to the processing of PRs. Liu *et al.* [16] reported that usage of PbD helps to increase the social and development activities of projects, but it leads to prolonged bug fixing time.

Jiang *et al.* [17], 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.* [18] proposed an accurate (77.97%) reviewer recommendation algorithm based on file path similarity, while Yu *et al.* [19] extended three typical approaches used in bug triaging and code review for recommending reviewers in PbD. Recently, Ying *et al.* [20] proposed a PR reviewer recommendation approach (EARec) which considers developer expertise and authority, which outperforms previous state-of-art methods. Liao *et al.* [21] proposed a Topic-based Integrator Matching Algorithm (TIMA) to predict highly relevant collaborators (the core developers) as the integrator to incoming PRs.

Gousios *et al.* [1] in an exploratory study reported that the merge decision is mainly influenced by the specific recently modified code. Tsay *et al.* [4] 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.* [9], [13] found that PR latency is a complex issue and continuous integration is a dominant factor for PRs acceptance and latency.

van der Veen *et al.* [12] proposed a code review PRioritizer system that 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.

III. The AR-Prioritizer Approach

In this section, we introduce our machine learning-based PR prioritization approach called `AR-Prioritizer`. Fig. 1 shows the flow of the `AR-Prioritizer`. The four main

parts of the algorithm are discussed in detail in the following sections.

A. Feature Extraction

Features extracted for this study are based on previous studies on PRs acceptance [1], [9], PRs response [12], and also on the surveys conducted with GitHub developers [3], [11]. To select relevant features to train the machine learning models and to predict the Acceptance/Response probability of PRs, we extracted features from the following four dimensions. Features that have been introduced in this study are shaded with cyan color in the corresponding table of each dimension.

1) Project: We summarise the features related to the project of a given PR in Table A1 of Appendix A. Project age has been shown to be a statistically significant predictor of PR acceptance [4], [9]. 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 [14]. 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.

2) Pull request: The features, summarised in Table A2 of Appendix A, characterize the PR 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 [3]. Moreover, churn-related metrics have been shown to be statistically significant predictors for PR acceptance [9]. PRs submitted from the same branch may receive a quick response [12]. Time till the first response is a statistically significant predictor for PR acceptance [9]. The PRs submission day may also affect its acceptance [22]. 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 [10], [11], [23]. Besides, the description of the PR can be helpful in software maintenance and software comprehension tasks [11], [24]. Therefore, the title and body text are transformed into word embedding using Word2Vec and used as features. 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 [25].

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 [4]. Moreover, reviewers prefer contributors with high-profile having a successful history [3]. The success of the contributor is described by features like previous PRs, contributions, and acceptance rates [1]. 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 A3 of Appendix A.

4) Integrator: The integrators take into consideration the code review while evaluating the quality of the PRs [3]. The number of discussion comments and review comments have an impact on the latency and acceptance of the PR [1]. Similarly, the number of participants who take part in the discussion also has an impact on the acceptance and latency of the PRs [1]. Previous studies show that '@' mentions are beneficial to the processing of PR by reducing the delay time in the developer's collaboration [15]. Table A4 of Appendix A, shows the features extracted from the integrator's dimension.

5) Target labels: The target labels represent the dependent variables that we have used in our dataset. The `PR_accept` is a binary variable that represents whether or not a given PR is accepted. Similarly, `PR_response` is also a binary variable representing whether or not the given PR has received a response from the integrator on the current day.

B. Acceptance Probability Prediction

Acceptance probability refers to the probability that the PR can be merged into the central repository. Acceptance probability is relevant for the integrator's perspective, as they prefer to inspect the most likely to be accepted and merged into the central repository, rather than the PR that may be aborted and closed later.

In this paper, the Acceptance probability prediction problem is transformed into a binary classification problem. The SVM, Logistic Regression, XGBoost, and Random Forest classifiers have been used for enabling the prediction.

C. Response Probability Prediction

Response probability refers to the probability that the PR will receive a user update on the current day or not. The PRs with the highest probability to be updated are considered as the important ones [12]. PRs prioritization just based on PR acceptance may always put some PRs which are most likely to be accepted at the top of the prioritized list. This may cause PR starvation because some PRs which are less likely to be accepted but need response will always be put at the end of the prioritized list. This can cause a lack of timely response to the contributors and a consequent loss of contributors' interest in participating in the project [3].

The Response probability prediction problem is also transformed into a binary classification problem. The Response probability prediction depends on the way the training data is generated. In this study, training data are generated using a one-day sliding window. This means that the Response probability model will predict the probability of PRs to be responded during the current day, this to present the PRs that will need immediate attention to the integrator(s).

D. Value Function

To overcome the problems deriving from considering only one of the acceptance and response dimensions, we propose the usage of a value function that combines the two probabilities, thus ranking on top PRs that will be likely accepted and considered important for integrators. The input of the value function is the acceptance and response probabilities, and the output is the score of the PR in the prioritizing list. The higher the score of a PR, the higher is its priority in the resulting list. The scoring function considered in our approach is the following:

$$f(x_1, x_2) = \frac{x_1 + x_2}{2} \quad (1)$$

where x_1 is the predicted acceptance probability and x_2 is the predicted response probability.

IV. Research Design

In this section, we present our research questions and all the steps that we followed to evaluate our approach.

A. Research Questions

We investigate the performance and benefits of the proposed approach by addressing the following research questions:

- **RQ1**: What is the performance of machine learning classifiers, trained with within-project training strategy, in predicting pull-request acceptance and response?
- **RQ2**: What is the performance of machine learning classifiers, trained with cross-project training strategy, in predicting pull-request acceptance and response?
- **RQ3**: How good is the best performing model in comparison to the baseline model in both the prediction tasks?
- **RQ4**: To which extent is `AR-Prioritizer` able to prioritize useful PRs?
- **RQ5**: Does `AR-Prioritizer` outperform the baseline models in prioritizing the most likely to be accepted and responded to PRs?

B. Dataset Overview

We have selected popular GitHub projects, in terms of stars [26], 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.

The time span of the PRs starts from the creation time of the project until February 2018. GitHub REST API V3 has been used to crawl the data from the projects. Table I shows the main statistics about the selected projects and the dataset.

1) Projects Selection: In order to obtain high-quality experiment data and select projects that are still active and avoid personal and toy projects [27], and target those projects which receive tens or hundreds for PRs every day and integrators face issues in the selection of quality PRs, we applied the following "criterion sampling" [28].

1) Projects that are developed in strict accordance with the process of the pull-based development model.

TABLE I
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

TABLE II
Distribution of the two datasets used in this study.

Dataset	**Accepted PRs**	**Rejected PRs**
Accept	199,474	78,944
Accept_Train	176,209	72,808
Accept_Test	21,478	5,669
Dataset	**Responded PRs**	**Not Responded PRs**
Response	203,566	560,005
Response_Train	177,937	499,866
Response_Test	24,293	58,060

2) Projects with more than 13 open PRs per day. We applied this filter to target projects having a large number of open PRs rate per day.
3) Projects with more than 1000 contributors, to increase PRs heterogeneity.

C. Datasets Construction

Our dataset consists of 278,418 PRs data extracted from the 19 projects given in table I. 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 **Accept** dataset contains all the features discussed above with a binary target variable `PR_accept` which describes whether or not the given PR is accepted. For the **Response** dataset, the training data are generated using a one-day sliding window. This means that each PR's lifetime (from submission till closing time) is divided into days and during that day it is checked whether or not it gets a response from the integrator, represented by the target variable `PR_response`.

D. Evaluation Measurements

To evaluate the PRs acceptance and response prediction performance of `AR-Prioritizer` we use the traditional information retrieval metrics: precision, recall, F-measure, accuracy, and AUC. F-measure also called the F_1 score is the harmonic mean of precision and recall. Accuracy describes the proportion of correctly classified samples out of total samples.

The AUC (Area Under Curve), most commonly defined as the area under the ROC [29] curve. As given in Table II, the

distribution of the datasets is imbalanced. That is why we have used AUC because it is more robust towards the imbalanced datasets [29]. To assess the prioritization effectiveness of `AR-Prioritizer` we used the following metrics:

1) Mean Average Precision: Mean Average Precision (MAP) for a set of queries is the mean of the average precision scores for each query [30]. 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:

$$P(j) = \frac{\sum_{k:\pi(k)\leqslant\pi(j)} y_k}{\pi(j)} \tag{2}$$

$$AP_i = \frac{\sum_{j=1}^{n_i} P(j) * y_j}{\sum_{j=1}^{n_i} y_j} \tag{3}$$

$$MAP = \frac{\sum_{i=1}^{n} AP_i}{n} \tag{4}$$

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

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.

$$Recall = \frac{k}{j}, \tag{5}$$

where k is the number of positive samples in top@N, and j is the total number of positive samples.

$$AR = \frac{\sum_{i=1}^{n} Recall_i}{n}, \tag{6}$$

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

E. Empirical studies

1) Experiment I (RQ1): The aim of Experiment I is to select the best classifier among the four classifiers: Logistic Regression [31], SVM [32], Random Forest [33], and XGBoost [34] to use in our approach. We selected these classifiers as they have been used in the previous studies on PRs [1], [12], [35]. Hence, all four classifiers are trained for both the acceptance and response probability prediction tasks using within-project training strategies. The datasets **Accept_Train** and **Response_Train**, described in Table II, are used for training and **Accept_Test** and **Response_Test** for testing.

The PRs belonging to each project in the training datasets (**Accept_Train** and **Response_Train**) have been used for training and the PRs of the corresponding project in the testing datasets (**Accept_Test** and **Response_Test**) have been used for testing.

Furthermore, to improve the performance of the prediction models we performed the following steps: scaled all the features for the two classifiers SVM and Logistic Regression in both the cases i.e. acceptance and response prediction training and testing, and tuned the hyper-parameters during training.

2) Experiment II (RQ2): In experiment II, again all the models have been trained, with cross-project training strategy, using 10-fold cross-validation on the datasets: **Accept** and **Response** respectively. 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 both the datasets according to their closing time. Then the whole training 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 in both the prediction tasks.

3) Experiment III (RQ3): The goal of experiment III is to compare our best performing models in PR acceptance and response prediction with the baseline models respectively. For PR acceptance prediction, Gousios *et al.* [1] model has been used as a baseline model. In this study, we have replicated their model using our own dataset and the same set of features except for test code churn, links to other PRs, and the number of test's lines of code as these features are not available in our dataset.

For PR response prediction, we have used PRioritizer, proposed by Veen et al. [12], as a baseline model for PR response prediction. PRioritizer predicts whether or not a given PR will receive a response in the following time window (day). We have replicated the PRioritizer on our own dataset using the same features set except feature has test code which is not available in our dataset.

4) Experiment IV (RQ4): The goal of experiment IV is to assess the performance of `AP-Prioritizer` on PRs prioritization. In the context of this experiment, the best performing classifier in experiments I and II was selected as the final classifier to consider for the prediction in `AP-Prioritizer`. `AP-Prioritizer` is tested on a statistically significant set of 400 PRs. Each PR in the test set could have one of the following possible statuses assigned: 'Not-Accepted and Not-Responded', 'Accepted and Not-Responded', 'Not-Accepted and Responded', 'Accepted and Responded', depending on its actual acceptance and actual replies. The distribution of the four statuses is the same in the selected test set of 400 PRs. The test set is divided into ten equal samples where each sample contains 40 PRs.

We are interested in understanding the extent to which `AP-Prioritizer` can give the highest priority to the PRs that are likely to be both accepted and responded to. As discussed in Section I, having these PRs prioritized allows integrator(s) to timely reply to these PRs that deserve to be merged and, consequently, to minimize the time required to obtain updates from contributors. Thus, MAP and AR values are computed by considering the PRs in the 'Accepted and Responded' status as the positive samples, while the PRs in all

the other statuses are considered as the negative samples. MAP and AR are computed for each sample on top@5, top@10, and top@20 PRs.

5) Experiment V (RQ5): The aim of experiment V is to compare the results of `AP-Prioritizer` with the ones achieved by two baseline approaches (as discussed in section IV-E3): (i) PRioritizer [12], which prioritizes PRs *exclusively* relying on their predicted response likelihoods, and (ii) Gousios *et al.* [1] model, which prioritizes PRs *exclusively* relying on their predicted acceptance probabilities. For comparing the results obtained by the different approaches, we make use of the MAP and AR computed for the top 5, 10, and 20 PRs across the ten samples of the statistically significant test set. Again MAP and AR values are computed by considering the PRs in the 'Accepted and Responded' status as the positive samples, while the PRs in all the other statuses are considered as the negative samples.

V. Results

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

A. RQ1: What is the performance of machine learning classifiers, trained with within-project training strategy, in predicting pull-request acceptance and response?

Fig. 2 shows the results for RQ1. AUC for each project has been calculated on the testing datasets. The left and right subgraphs show the results of the PRs acceptance and response prediction models respectively. Overall, the four classifiers have performed well in both the prediction task. In the case of accept prediction task, XGB has outperformed the rest of the three classifiers and have achieved an average AUC value of 89.5%. AUC values of all the XGB models trained on each project are highest except for the project OpenCV where all the classifiers achieved the same AUC of 94%. RF could achieve an average AUC of 86% in the acceptance prediction task and stood second in performance. LR and SVM achieved almost similar average AUC values with a negligible difference.

Similarly to the acceptance prediction models results, XGB performed well than the rest of the three classifiers and secured an average AUC value of 88.8%. XGB model trained on the project React achieved the highest AUC of value 96.9% and lowest AUC value (82.5%) on project Salt. All the XGB models achieved the highest AUC values than the models of the rest of the three classifiers. RF models achieved very similar AUC values to the XGB models with a very little difference of 1% in average AUC. However, XGB models have achieved higher results in other evaluation metrics like accuracy and F-Measure. Therefore, XGB is selected as the best classifier. Analogous to the acceptance prediction task, LR and SVM models secured approximately similar results. Overall, all the classifiers have produced acceptable results in the response prediction task.

The Kruskal-Wallis tests, with ($p<1.046\text{E}^{-8}$) and ($p<1.384\text{E}^{-7}$) for acceptance and response prediction respectively, show that the differences between the results of the four classifiers are statistically significant in both the prediction tasks. Besides, pairwise Mann-Whitney tests show that XGB and RF produce different results (with statistical evidence) from the one achieved through LR and SVM models in both the prediction tasks. However, the differences between XGB and RF are not statistically significant.

> **RQ1 Summary:** XGB has outperformed the rest of the classifiers SVM, RF, and LR in both the prediction tasks. However, the difference in the performance of the XGB and RF models is not statistically significant. Using the within-project training strategy, XGB has achieved the highest average AUC values of 86% and 88.8% in acceptance and response prediction tasks respectively. SVM and LR models have shown poor performance in both the prediction tasks.

B. RQ2: What is the performance of machine learning classifiers, trained with cross-project training strategy, in predicting pull-request acceptance and response?

Table III shows the results of the four classifiers trained, with a cross-project training strategy, using 10-fold cross-validation. Similarly to the results of the within-project training strategy, in the cross-project training strategy, all the classifiers performed well in both the acceptance and response prediction tasks.

XGB outperformed the rest of the classifiers in the acceptance prediction task with the highest results in all the evaluation metrics. The RF stood second in the acceptance prediction task with precision value (87.8%) almost equal to the one achieved by the XGB model (88.4%). The performance of the rest of the two classifiers is quite similar but the LR classifier obtained negligibly high results in all the evaluation metrics. Overall, based on the results of all the evaluation metrics both the LR and SVM models are also acceptable for cross-project PR acceptance prediction tasks.

Results of the Kruskal-Wallis test ($p<5.241\text{E}^{-8}$) shows that the performance of the four classifiers in predicting PR acceptance is statistically significant. The subsequent pairwise Mann-Whitney test (with p-value adjusted using Holm's correction procedure [36]) shows that XGB classifier performance is significantly better than the rest of the models with a large effect size ($\delta = -1$).

In the case of response prediction task again XGB classifier outperformed the rest of the classifiers by achieving the best results in AUC (98.8%), precision (88.9%), F-measure (73.6%), and accuracy (87.5%) evaluation measures. Similarly to the XGB classifier, RF classifier produced better results in the four evaluation metrics (AUC, recall, F-Measure, and Accuracy) and stood second in performance. LR and SVM secured third and fourth position with negligible differences in evaluation metrics values. All the four classifiers performed well in the cross-project PR response prediction task.

Similarly to the PR acceptance prediction models, Kruskal-Walis test ($p<5.202\text{E}^{-9}$) shows that the performance of the four classifiers in PR response prediction is statistically significant. Pairwise Mann-Whitney test shows that the performance of

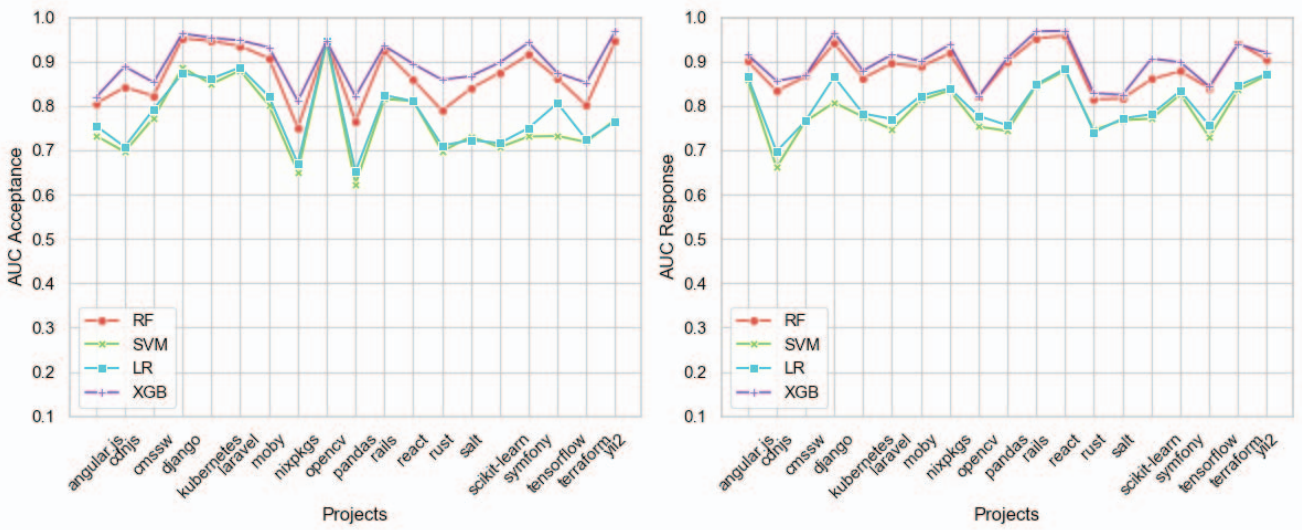

Fig. 2. AUC score of acceptance and response prediction models trained using within-project strategy

TABLE III
ACCEPTANCE & RESPONSE PREDICTION MODELS TRAINED USING CROSS-PROJECT TRAINING STRATEGY

Acceptance Prediction Models					
Model	**AUC**	**Precision**	**Recall**	**F-Measure**	**Accuracy**
Random forest	0.865	0.878	0.875	0.876	0.822
SVM	0.800	0.839	0.873	0.855	0.788
Logistic Regression	0.804	0.841	0.880	0.859	0.793
XGBoost	0.906	0.884	0.934	0.908	0.864
Response Prediction Models					
Model	**AUC**	**Precision**	**Recall**	**F-Measure**	**Accuracy**
Random forest	0.881	0.626	0.763	0.687	0.808
SVM	0.802	0.741	0.364	0.481	0.788
Logistic Regression	0.807	0.768	0.356	0.473	0.790
XGBoost	0.906	0.889	0.630	0.736	0.875

the XGB classifier is statistically better than the rest of the model with a large effect size ($\delta = -1$).

RQ2 Summary: XGB model, trained using cross-project training strategy, has again outperformed the rest of the three classifiers in both the prediction tasks with a statistical significance. All the four classifiers trained using cross-project training strategy performed well in both the prediction tasks.

C. RQ3: How good is the best performing model in comparison to the baseline model in both the prediction tasks?

The goal of RQ3 is to compare the performance of the best performing models, produced as a result of RQ1 and RQ2, with the baseline models in both the prediction tasks respectively. We have used Gousios *et al.* [1] as a baseline model for acceptance prediction task and Veen *et al.* [12] model called PRioritizer as a baseline model for the response prediction task. The comparison has been made both within and cross-project training strategies.

TABLE IV
COMPARISON OF XGBOOST MODELS (ACCEPTANCE & RESPONSE) WITH BASELINE MODELS TRAINED USING CROSS-PROJECT TRAINING STRATEGY.

Acceptance Prediction Models					
Model	**AUC**	**Precision**	**Recall**	**F-Measure**	**Accuracy**
XGBoost	0.906	0.884	0.934	0.908	0.864
Gousios et al	0.776	0.849	0.793	0.819	0.744
Response Prediction Models					
Model	**AUC**	**Precision**	**Recall**	**F-Measure**	**Accuracy**
XGBoost	0.906	0.889	0.630	0.736	0.875
PRioritizer	0.843	0.598	0.702	0.644	0.786

Fig. 3 illustrates the results of best performing models, i.e. XGBoost (XGB) models, and the baseline models in both the prediction tasks. The XGB models outperformed the baseline models in both the prediction tasks trained using a within-project strategy. Results of the Kruskal-Walis test shows that XGB models performed better than the baseline models, trained using within-project training strategy, in both the acceptance (p<7.478E^{-7}) and response ($p = 0.002$) prediction tasks with a statistical significance.

Similarly, Table IV illustrates the results achieved by the XGB models and the corresponding baseline models trained using a cross-project training strategy. The results show that the XGB models have performed better than the baseline models. Kruskal-Walis test results show that XGB models have outperformed the baseline models in both the PR acceptance ($p = 0.000$) and response ($p = 0.000$) with a statistical significance.

RQ3 Summary: Our best performing models, i.e. XGB, have significantly outperformed the baseline models Gousios *et al.* and PRioritizer in acceptance and response prediction tasks respectively. Similarly, XGB models performed well in both the training strategies.

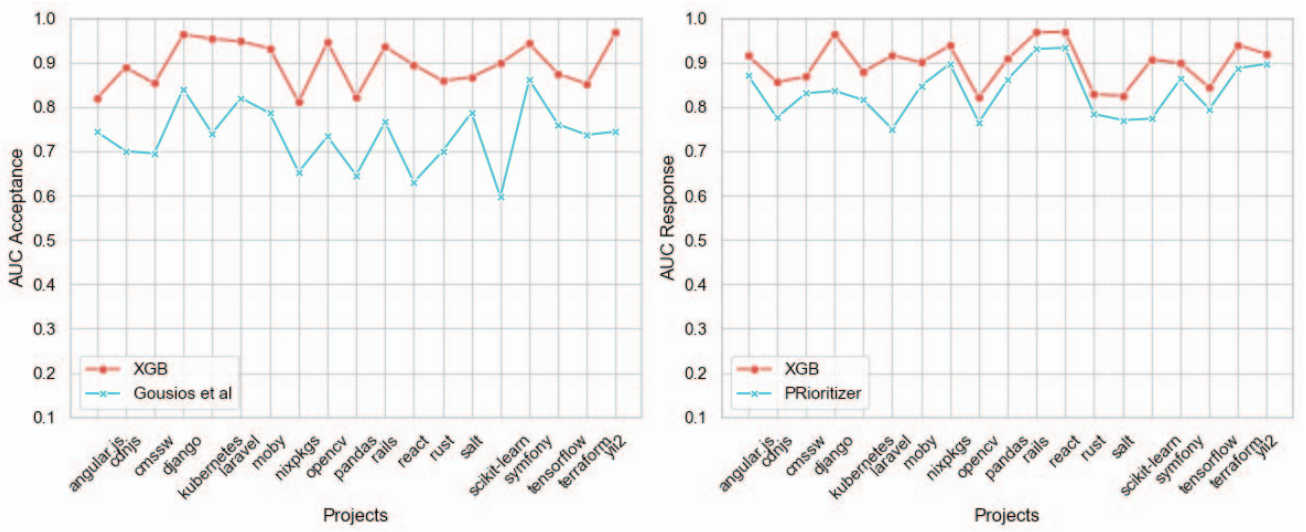

Fig. 3. Comparison of best performing models in both the acceptance and response prediction tasks with the baseline models.

D. RQ4: To which extent is AR-Prioritizer able to prioritize useful PRs?

RQ4 aims to assess the extent to which AR-Prioritizer can prioritize the PRs that are most likely to be accepted and get responded to. Table V shows the MAP and AR values achieved by AR-Prioritizer on the statistically significant test set of 400 PRs. The MAP and AR metrics are computed for top@5, 10, and 20 prioritized PRs.

AR-Prioritizer obtains above 95.3% of the average MAP in prioritizing the top@5 PRs in the 10 samples reported in Table V. In particular, our prioritization approach achieves promising results, especially for seven of the analyzed samples (i.e., Sample_0, Sample_1,..., and so on with MAP-5 equal to 1.000) in which AR-Prioritizer is able to recommend 5 out of 5 PRs that are worthy to be accepted and responded. This allows integrator(s) to timely review and responds to the recommended PRs with a low probability of wasting time on PRs that are less likely to be accepted. In addition, for the top@10 and top@20 PRs of the analyzed samples, AR-Prioritizer is able to prioritize PRs with an average MAP of 89.6% and 79.6% respectively. Hence, AR-Prioritizer can successfully recommend almost 9 out of 10 and 16 out of 20 most likely to be accepted and responded to PRs.

Concerning the AR metric, AR-Prioritizer achieves best results in the case of top@20 prioritized PRs, i.e. average AR of 92.9%, across the 10 studied samples, illustrated in Table V. For top@10 PRs, AR-Prioritizer gets an average AR of 65.7%, while in the case of top@5 prioritized PRs, it can achieve only 40% of average AR across the 10 samples. This means that, for the majority of studied samples, in the top@10 PRs recommended by AR-Prioritizer at maximum the 35% of PRs that deserve to be accepted and responded will not appear.

TABLE V
MAP AND AR OBTAINED BY THE AR-PRIORITIZER.

Model	Mean Average Precision			Average Recall		
	MAP-5	MAP-10	MAP-20	AR-5	AR-10	AR-20
Sample_0	1.000	0.970	0.898	0.455	0.727	1.000
Sample_1	1.000	0.911	0.768	0.364	0.545	0.909
Sample_2	1.000	0.892	0.814	0.364	0.636	0.909
Sample_3	0.888	0.874	0.836	0.444	0.778	0.889
Sample_4	0.888	0.842	0.743	0.444	0.667	1.000
Sample_5	1.000	1.000	0.643	0.375	0.375	0.875
Sample_6	1.000	0.839	0.817	0.333	0.889	1.000
Sample_7	0.756	0.719	0.656	0.375	0.750	1.000
Sample_8	1.000	1.000	0.913	0.556	0.667	0.778
Sample_9	1.000	0.916	0.869	0.333	0.533	0.933

RQ4 Summary: Results show that AR-Prioritizer can prioritize the most likely to be accepted and responded PRs in the top@k categories with the best average MAP (top@5 = 95.3%, top@10 = 89.6%, top@20 = 79.6%). In the case of AR, the AR-Prioritizer produced the best results for the top@20 prioritized PRs i.e. 92.9%. However, one can select the top@10 PRs where the AR-Prioritizer achieved better results for both the evaluation metrics i.e. MAP = 89.6% and AR = 65.7%.

E. RQ5: Does AR-Prioritizer outperform the baseline models in prioritizing the most likely to be accepted and responded to pull requests?

In Fig. 4 the top@10 MAP and AR values obtained by AR-Prioritizer and the two baseline approaches are reported. It is important to be noted that AR-Prioritizer has outperformed the baseline models in all the top@k categories.

AR-Prioritizer achieves the best MAP values compared to the PRioritizer and Gousios model in all the analyzed samples except sample_6 and sample_9 where PRioritizer showed slightly better results. Gousios model produces poor

MAP results in prioritizing the most likely to be accepted and responded to PRs across all the studied samples. Specifically, on average, the combination of response and acceptance likelihoods allows to achieving MAP results (i) 19% higher than the ones achieved by PRioritizer (response likelihood prediction), and (ii) 52% higher than the ones achieved by Gousios model (acceptance likelihood prediction).

Analogously, `AR-Prioritizer` outperformed the baseline models in top@10 AR in all the analyzed samples except sample_5 where the Gousios model achieved slightly better AR value. Again the Gousios model performed poorly in achieving top@10 AR across the 10 samples. `AR-Prioritizer` on average achieved top@10 AR 17.4% higher than PRioritizer and 34.8% higher than the Gousios model.

We used the Mann-Whitney U test (with the p-level fixed to 0.05) to check if the differences in the MAP and AR results achieved by `AR-Prioritizer` and the baseline models are statistically significant, and Cliff's d effect size [37], to characterize the eventual differences. As we performed multiple tests, we adjusted our p-values using Holm's correction procedure [36]. While significant differences are observed between the MAP results obtained by `AR-Prioritizer` and the ones achieved by the PRioritizer ($p = 0.0210$), the MAP values obtained by `AR-Prioritizer` and the ones achieved by the Gousios model are different with statistical evidence ($p = 0.0005$) and large effect-size ($\delta = 0.81$). It is worth noticing that similar results have been obtained, in terms of MAP for top@5 and top@20 PRs. Similarly, `AR-Prioritizer` performed better with statistical significance in terms of top@10 AR results than the PRioritizer ($p = 0.0373$) and Gousios model ($p = 0.0013$) with large effect size ($\delta = 0.78$). Looking at such results, we can conclude that in our approach the prioritization is mainly driven by the response likelihood prediction while the acceptance dimension could serve to optimize the final rankings.

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 `AR-Prioritizer`. To this purpose, we selected the top 20 PRs from two randomly selected samples, as shown in Table A5 of Appendix A. The samples are provided to all the considered approaches to prioritize them. The ranks assigned by each approach along with the original labels for acceptance and response of the PRs are reported in Table A5. As it can be observed from Table A5, in Sample Zero, while the Gousios model and the PRioritizer put on the top of the list PRs that would not be accepted (i.e., `kubernetes-56129`, `symfony-24937`, and `framework-22137`) and PRs that would not be responded (i.e., `opencv-9746`, and `cmssw-21143`), respectively, `AR-Prioritizer` gives the highest priorities to the PRs that deserve to be both accepted and responded (i.e., `kubernetes-54373` and `kubernetes-59490`, etc.). Similarly, in Sample One, while `AR-Prioritizer` correctly prioritizes the top four PRs that deserve to be both accepted and responded (`kubernetes-55977`, `rails-30757`, and `cmssw-21947`), the PRioritizer approach puts at the second, fourth and fifth positions of the rank PRs that would be not accepted (`cmssw-21816`, `kubernetes-59142`, `cmssw-21041`) and the Gousios model gives the highest priority to a PR (`salt-45438`) which would be not responded.

RQ5 Summary: `AR-Prioritizer` has outperformed the baseline models in top@10 MAP and AR with a statistical significance. The prioritization is mainly driven by the response likelihood prediction while the acceptance dimension could serve to optimize the final rankings. Besides, `AR-Prioritizer` can better prioritize useful PRs than the baseline models exclusively based on either response or acceptance probability prediction.

VI. Threats to Validity

Threats to construct validity. To design `AR-Prioritizer`, 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 [1], [11], [12], 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 [38] 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 [27], 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 compared the AUC values obtained by different machine learning algorithms. For demonstrating that the differences in the results obtained by the different ML models are statistically significant, we used appropriate statistical procedures and effect size measures. In addition, to measure the performance of `AR-Prioritizer` we employed widely-used metrics in the information retrieval field, Mean Average Precision and Recall. Besides the mere statistical tests, randomly selected samples of PRs have been provided as input to both `AR-Prioritizer` and the baseline approaches. Although such analysis is limited to specific samples of our dataset, qualitative insights about the PRs that the approaches are able to prioritize is provided.

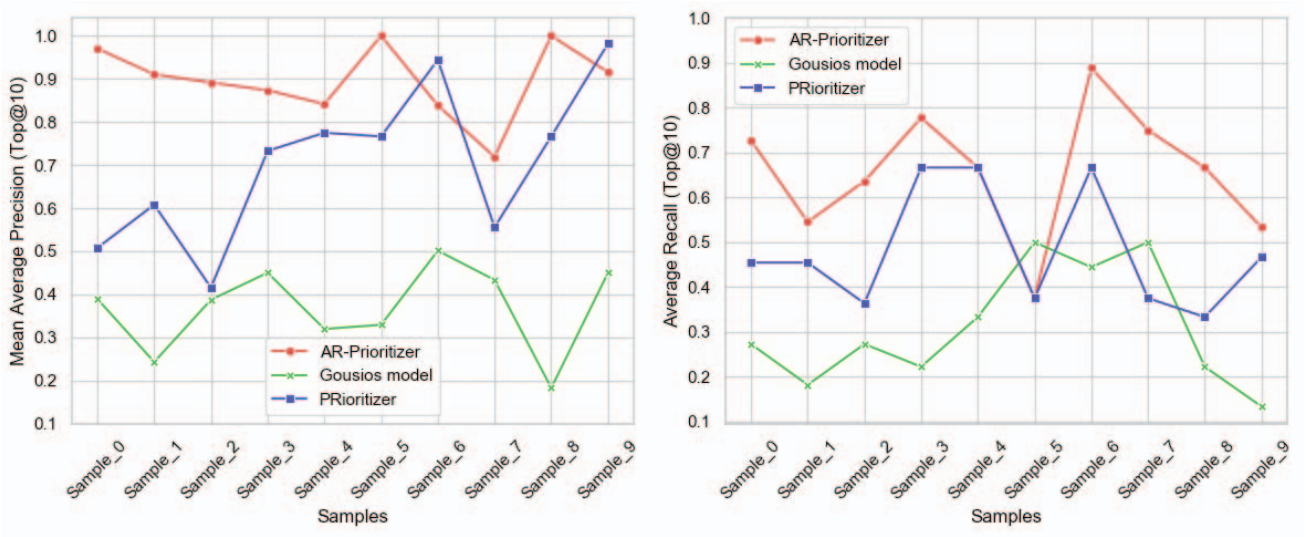

Fig. 4. Comparison of the top@10 MAP & AR achieved by `AR-Prioritizer`, PRioritizer, and Gousios model.

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, such projects were selected considering well-defined selection criteria (see Section IV-B1). In addition, intending to increase 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 to further projects mined from different platforms.

VII. Conclusion

In this study, we attempted to address one of the challenges occurring in pull-based development, i.e. PRs prioritization. In particular, we proposed a novel approach, named `AR-Prioritizer`, which prioritizes pull requests relying on both the acceptance and the response prediction of the PRs. For this purpose, we have trained four classifiers: Logistic Regression, SVM, Random Forest, and XGBoost, to predict pull request acceptance and response and select the best performing classifier to use in our approach. Results have shown that the XGBoost classifier has outperformed the rest of the three classifiers in both the prediction tasks. The best performing classifier i.e. XGBoost has been used to build the proposed prioritization algorithm.

The `AR-Prioritizer` can recommend top@5, top@10, and top@20 most likely to be accepted and responded to pull requests with Mean Average Precision (MAP) of 95.3%, 89.6%, and 79.6% and Average Recall (AR) of 40%, 65.7%, and 92.9%. Moreover, `AR-Prioritizer` has outperformed the baseline models with a statistical significance in prioritizing the most likely to be accepted and responded to PRs. Besides, a comparison with the results achieved by two baseline models relying on *either* response probability *or* acceptance likelihood prediction has shown that `AR-Prioritizer` can better prioritize PRs categories, and that is worthy to be both accepted and responded. `AR-Prioritizer` produced promising results and we expect it can reduce the efforts and time consumed by the integrator in the pull requests selection process.

Future studies will be aimed at investigating further aspects that can be related to the PRs prioritization. In this context, we aim to integrate our PRs prioritization approach into coding platforms supporting pull-based development (e.g. GitHub, Bitbucket, etc.), thus surveying real integrators and contributors in order to (i) evaluate the usefulness of `AR-Prioritizer`, and (ii) discover additional factors that can be used to improve the prioritization performed by our approach. Finally, we also plan to improve the value function, by designing larger empirical experiments to establish the best performing combination of weights for response and acceptance probabilities.

Acknowledgment

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] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 345–355. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568260

[2] E. T. Barr, C. Bird, P. C. Rigby, A. Hindle, D. M. German, and P. Devanbu, "Cohesive and isolated development with branches," in *Fundamental Approaches to Software Engineering*, J. de Lara and A. Zisman, Eds. Springer Berlin Heidelberg, 2012, pp. 316–331.

[3] G. Gousios, A. Zaidman, M. Storey, and A. v. Deursen, "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, May 2015, pp. 358–368.

[4] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: ACM, 2014, pp. 356–366. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568315

[5] Y. Wang and D. Redmiles, "Cheap talk, cooperation, and trust in global software engineering," *Empirical Software Engineering*, vol. 21, no. 6, pp. 2233–2267, Dec 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9407-3

[6] Understanding the github flow. https://guides.github.com/introduction/flow/. Accessed: 2019-07-23.

[7] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/567793.567795

[8] C. Bird and A. Bacchelli, "Expectations, outcomes, and challenges of modern code review." IEEE, May 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/

[9] Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang, "Determinants of pull-based development in the context of continuous integration," *Science China Information Sciences*, vol. 59, no. 8, p. 080104, Jul 2016. [Online]. Available: https://doi.org/10.1007/s11432-016-5595-8

[10] Y. Fan, X. Xia, D. Lo, and S. Li, "Early prediction of merged code changes to prioritize reviewing tasks," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3346–3393, Dec 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9602-0

[11] G. Gousios, M. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *International Conference on Software Engineering (ICSE)*, 2016, pp. 285–296.

[12] E. v. d. Veen, G. Gousios, and A. Zaidman, "Automatically prioritizing pull requests," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 357–361.

[13] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, May 2015, pp. 367–371.

[14] M. M. Rahman and C. K. Roy, "An insight into the pull requests of github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 364–367. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597121

[15] Y. Zhang, G. Yin, Y. Yu, and H. Wang, "A exploratory study of @-mention in github's pull-requests," in *Asia-Pacific Software Engineering Conference*, 2014, pp. 343–350.

[16] J. Liu, J. Li, and L. He, "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, June 2016, pp. 313–322.

[17] J. Jiang, Y. Yang, J. He, X. Blanc, and L. Zhang, "Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development," *Information and Software Technology*, vol. 84, pp. 48 – 62, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491630283X

[18] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE 2014. New York, NY, USA: ACM, 2014, pp. 119–122. [Online]. Available: http://doi.acm.org/10.1145/2593702.2593705

[19] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204 – 218, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584916000069

[20] H. Ying, L. Chen, T. Liang, and J. Wu, "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)*, May 2016, pp. 29–35.

[21] Z. Liao, Y. Li, D. He, J. Wu, Y. Zhang, and X. Fan, "Topic-based integrator matching for pull request," *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, 2017.

[22] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, p. 15, May 2005. [Online]. Available: https://doi.org/10.1145/1082983.1083147

[23] Z. Liu, X. Xia, C. Treude, D. Lo, and S. Li, "Automatic generation of pull request descriptions," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 176–188.

[24] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github," *Inf. Softw. Technol.*, vol. 74, no. C, p. 204?218, Jun. 2016. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.01.004

[25] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *International Conference on Software Engineering*, 2016, pp. 404–415.

[26] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct 2016, pp. 334–344.

[27] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán, and D. E. Damian, "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*, 2014, pp. 92–101. [Online]. Available: https://doi.org/10.1145/2597073.2597074

[28] M. Patton, *Qualitative Evaluation and Research Methods*. Newbury Park: Sage, 2002.

[29] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2005.10.010

[30] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval the Concepts and Technology Behind Search*. DBLP, 2011.

[31] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.

[32] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, p. 1565, 2006.

[33] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[34] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[35] Y. Yu, H. Wang, G. Yin, and C. X. Ling, "Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration," in *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2014, pp. 335–342.

[36] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.

[37] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[38] A. Mohamed, L. Zhang, J. Jiang, and A. Ktob, "Predicting which pull requests will get reopened in github," in *25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018*, 2018, pp. 375–385. [Online]. Available: https://doi.org/10.1109/APSEC.2018.00052

APPENDIX

TABLE A1
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 in 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

Note: The shaded features are introduced in this study

TABLE A2
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	The 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 the creation of the PR until the first response from the integrator.
latency_after_response	The time duration in minutes after the first response till the 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?

Note: The shaded features are introduced in this study

TABLE A3
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, a core member of the project organization?
contributions	The number of contributions made by 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.

Note: The shaded features are introduced in this study

TABLE A4
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?

Note: The shaded features are introduced in this study

TABLE A5
TOP 20 PULL REQUESTS FROM TWO RANDOMLY SELECTED SAMPLES FOR THE COMPARISON OF AR−PRIORITIZER AND BASELINE MODELS.

Sample Zero					
PR_ID	Labels		AR-Prioritizer	Gousios model	PRioritizer
	Accept	Response	Rank	Rank	Rank
kubernetes-54373	TRUE	TRUE	1	18	7
kubernetes-59490	TRUE	TRUE	2	3	2
nixpkgs-33394	TRUE	TRUE	3	20	1
nixpkgs-32172	TRUE	TRUE	4	8	11
tensorflow-13279	TRUE	TRUE	5	13	15
nixpkgs-35998	TRUE	TRUE	6	14	10
kubernetes-56129	FALSE	TRUE	7	16	5
rust-44965	TRUE	TRUE	8	6	19
tensorflow-15603	TRUE	TRUE	9	12	6
framework-22137	FALSE	TRUE	10	19	4
symfony-24937	FALSE	TRUE	11	7	3
kubernetes-60503	TRUE	TRUE	12	1	14
kubernetes-54819	TRUE	FALSE	13	15	17
kubernetes-56648	TRUE	TRUE	14	9	13
kubernetes-57180	FALSE	TRUE	15	2	16
kubernetes-58827	FALSE	TRUE	16	17	20
pandas-19909	TRUE	TRUE	17	4	8
opencv-9746	TRUE	FALSE	18	5	12
tensorflow-15770	FALSE	TRUE	19	10	9
cmssw-21143	TRUE	FALSE	20	11	18
Sample One					
PR_ID	Labels		AR-Prioritizer	Gousios model	PRioritizer
	Accept	Response	Rank	Rank	Rank
kubernetes-57017	TRUE	TRUE	1	11	1
rust-47700	TRUE	TRUE	2	8	11
rust-46378	TRUE	TRUE	3	18	8
symfony-26072	TRUE	TRUE	4	4	5
kubernetes-59142	FALSE	TRUE	5	6	6
cmssw-21041	FALSE	TRUE	6	20	9
pandas-18809	TRUE	TRUE	7	17	3
rust-45549	TRUE	TRUE	8	3	10
yii2-14819	FALSE	TRUE	9	1	2
react-11541	FALSE	TRUE	10	14	7
cmssw-21816	FALSE	TRUE	11	2	4
kubernetes-53466	TRUE	TRUE	12	19	19
tensorflow-14043	FALSE	TRUE	13	15	12
cdnjs-11769	TRUE	TRUE	14	16	15
kubernetes-56298	TRUE	FALSE	15	12	17
kubernetes-54279	TRUE	TRUE	16	5	18
cmssw-21784	TRUE	FALSE	17	9	16
salt-45438	TRUE	FALSE	18	13	14
kubernetes-57245	FALSE	TRUE	19	7	20
kubernetes-56382	TRUE	TRUE	20	10	13