SCIENCE CHINA
Information Sciences

• LETTER •

July 2021, Vol. 64 179105:1–179105:3
https://doi.org/10.1007/s11432-018-9823-4

Predicting accepted pull requests in GitHub

Jing JIANG, Jiateng ZHENG, Yun YANG, Li ZHANG & Jie LUO*

State Key Laboratory of Software Development Environment, Beihang University, Beijing 100083, *China*

Received 26 June 2018/Revised 29 September 2018/Accepted 29 December 2018/Published online 15 April 2021

Citation Jiang J, Zheng J T, Yang Y, et al. Predicting accepted pull requests in GitHub. Sci China Inf Sci, 2021, 64(7): 179105, https://doi.org/10.1007/s11432-018-9823-4

Dear editor,

Various open source software hosting sites, such as Github, provide support for pull-based development and allow developers to flexibly and efficiently make contributions [1]. In GitHub, contributors fork the project's main repository of a project, and independently change their codes. When a set of changes is ready, contributors create pull requests and submit them to the main repository. Any developer can provide comments and exchange opinions regarding pull requests [2, 3]. Developers freely discuss whether the code style meets the quality standard, repositories require modification, or submitted codes are of high quality. According to the comments, the contributors can modify codes. Members of the core team of the project, here on, referred to as integrators, are responsible for inspecting the submitted code changes, identifying issues (e.g., vulnerabilities), and deciding whether to accept pull requests and merge these code changes into the main repository [1]. Integrators act as the guardians of the project quality.

As projects gain popularity, the volume of incoming pull requests increases, heavily burdening the integrators [1]. There has been previous study [4] about integrator recommendation for pull requests. Besides recommending integrators, inspecting code changes consumes much of the integrators' time and effort. An accepted pull request prediction approach can help the integrators by allowing them to either enforce an immediate rejection of code changes or allocate more resources to overcome the deficiency. The integrators can reduce time wasted on the inspection of rejected code changes, and focus on more promising ones, which will eventually be merged.

In this study, we propose an accepted pull request prediction approach named XGPredict, which builds an XGBoost classifier based on the training dataset of a project to predict whether pull requests will be accepted.

First, we extract the code, text, and project features. The code features measure the characteristics of the modified code in projects. Text features are extracted from the natural language descriptions of pull requests, such as the title and body. The project features mainly analyze recent development history in projects. The aforementioned features can be automatically and immediately extracted when pull requests are submitted.

Based on these features, we leverage the XGBoost classifier to predict accepted pull requests. XGPredict computes the acceptance and rejection probabilities of pull requests. If the acceptance probability is higher than or equal to the rejection probability, the pull request is predicted as accepted. If the acceptance probability is lower than the rejection probability, the pull request is predicted as rejected. XGPredict not only returns the prediction results but also provides the acceptance and rejection probabilities of pull requests, which helps integrators to make decisions in a code review.

In the following part, the XGBoost classifier is introduced at first. Then, the code, text and project features are described. Finally, the framework of the accepted pull request prediction approach, XGPredict is introduced.

XGBoost. XGBoost (extreme gradient boosting) is a supervised machine learning algorithm that implements a process called boosting to yield accurate models [5]. The XGBoost inputs comprise pairs of training examples $(\boldsymbol{x}_0, y_0)$, $(\boldsymbol{x}_1, y_1)$, …, $(\boldsymbol{x}_n, y_n)$, where $\boldsymbol{x}$ is a vector of the features describing the example and y is its label. The output of XGBoost is the predicted value $\hat{y}$. XGBoost aims to minimize the loss function L, which indicates the difference between $\hat{y}$ and y.

XGBoost uses a decision tree boosting algorithm to minimize the loss function L. It sequentially builds an ensemble of decision tree models. XGBoost uses a greedy algorithm to choose the feature, which achieves the minimum loss function L based on the current tree structure, as the node to split the tree. In particular, XGBoost enumerates the possible tree structures q. The greedy algorithm commences from a single leaf and iteratively adds branches to the tree. We assume that I_L and I_R are the instance sets of the left and right nodes after the split. H_L and H_R are the loss functions of the left and right subtrees, respectively. G_L and G_R are the gradients of the loss function based on the left and right subtrees, respectively. λ is a penalty for introducing new leaf nodes, and γ is the complexity cost of introducing new leaf nodes. Letting $I = I_L \cup I_R$, the loss function after the split is given as $L_{\text{split}} = \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} + \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$.

XGBoost uses the formula specified above to evaluate the

* Corresponding author (email: luojie@nlsde.buaa.edu.cn)

feature candidates and choose a feature as the node that minimizes the loss function. XGBoost iteratively chooses features and sequentially builds an ensemble of decision tree models. When the tree reaches a preset maximum depth, the iteration stops and the model is finally built.

Code features. Code features mainly measure the characteristics of code modified in a pull request.

Weißgerber et al. [6] observed that the size of a patch impacts its acceptance. We use features src_churn, src_addition, src_deletion, num_commits and files_changes to quantify the size of a pull request. src_churn is the number of lines changed by the pull request. src_addition is the number of lines added by the pull request. src_deletion is the number of lines deleted by the pull request. num_commits is the number of commits in the pull request. files_changes is the number of files touched by the pull request.

Furthermore, Pham et al. [7] found that presence of tests in pull requests increased the integrators' confidence in accepting them. Therefore, we use feature has_test to investigate whether a pull request contains test files.

Text features. In previous studies, text features were used to recommend reviewers who decided to accept or reject code changes. Further, we consider text features to capture the textual characteristics of messages in pull requests. Gousios et al. [8] studied the impacts of comments on pull requests' acceptance. Comments are added after the submission of pull requests. When pull requests are submitted, we immediately predict acceptance, and we do not consider comments because they are not yet created. We mainly study the text in the titles and bodies of the pull requests.

In GitHub, developers are required to write the titles and briefly describe the pull requests. Unlike the titles, the bodies are not necessary; they are used to introduce the details of pull requests. We use feature has_body to measure whether the pull requests have bodies. Some developers insert links in titles or bodies that redirect to related information, such as other pull requests, discussion in Stack Overflow, or technical websites. Links may provide further information about pull requests. We use feature text_link to decide whether pull requests include links.

Previous studies used text similarity as a feature to build machine learning models in reviewer recommendation. We also use text similarity as a feature to predict which pull requests would be accepted. The idea is that related pull requests are often described similarly, and they may have similar code review results. So the pull requests with the high text similarity may have similar code review results. Feature text_similarity_merged uses cosine similarity to compute the text similarity between a current pull request and previous pull requests that were accepted in the last three months. Feature text_similarity_rejected computes the text similarity between a current pull request and previous pull requests that were rejected in the last three months.

Project features. These features quantify receptivity of the project to pull requests. Following previous studies [8], we consider feature commits_files_touched to measure whether a pull request modifies files that are actively developed recently in the project. In particular, commits_files_touched is the number of total direct commits in the files touched by a pull request three months before the time at which the pull request was created.

Various open-source projects may have varying standards for evaluating pull requests. The stringency of recent code reviews may influence the evaluation results; therefore, we use features file_rejected_proportion, last_10_merged, last_10_rejected, and last_pr in the prediction of accepted pull requests. file_rejected_proportion is the percentage of previous rejected pull requests in files touched by the pull request. last_10_merged is the number of merged pull requests in 10 recent pull requests. last_10_rejected is the number of rejected pull requests in 10 recent pull requests. last_pr evaluates whether the most recent pull request is rejected.

Framework. The framework of XGPredict comprises two phases: a model-building phase and a prediction phase.

In the model-building phase, we compute code, text, and project features for each pull request in a training dataset. Then, we use a weighted vector to represent each pull request; each element in this vector corresponds to the value of a feature. There are only two possible results for the pull request evaluation: accepted or rejected. The prediction of accepted pull requests can be transformed into a binary classification problem, which is often addressed by machine learning techniques [9]. According to the pull request features and their known evaluation results in the training datasets, we build the classifier XGBoost based on the training dataset of a project. XGBoost is a scalable end-to-end tree boosting method [5], that assigns a label (accept or reject) to a data point (in our case: a pull request).

In the prediction phase, we use XGPredict to predict accepted pull requests. First, XGPredcit extracts the code, text and project features. Then, it processes these features into the XGBoost classifier built during the model-building phase and computes the acceptance and rejection probabilities of the pull requests. If the acceptance probability is higher than or equal to the rejection probability, an accepted pull request is predicted; otherwise, the pull request is predicted as rejected. XGPredict not only returns prediction results but also provides the acceptance and rejection probabilities of the pull requests, which helps the integrators to make decisions.

Threats to validity. Threats to external validity relate to the universality of our study. XGPredict is mainly designed for GitHub, and it is not known if this approach can be generalized to other OSS platforms. In the future, we plan to study other platforms, and explore the usability of XGPredict.

Threats to construct validity relate to the degree to which the construct being studied is affected by experimental settings. We extract features from code, text and project dimensions. We have not studied all the potential features applicable to the prediction of the accepted pull requests. In future work, we will study more potential features. For example, some open-source software projects adopt continuous integration in code reviews. In future work, we plan to consider the features of continuous integration and explore the changes int the prediction performance.

Conclusion. In this study, a novel approach for predicting the accepted pull requests based on the XGBoost machine learning algorithm, called XGPredict, is proposed. We extract code, text and project features for pull requests in GitHub and build an XGBoost classifier to predict the accepted pull requests for each project. The proposed XGPredict approach is useful for predicting the accepted pull requests and improving the code review process.

Acknowledgements This work was supported by National Key Research and Development Program of China (Grant No. 2018YFB1004202), National Natural Science Foundation of China (Grant No. 61672078), and State Key Laboratory of Software Development Environment (Grant No. SKLSDE-

2018ZX-12).

References

1 Gousios G, Zaidman A, Storey M A, et al. Work practices and challenges in pull-based development: The integrators perspective. In: Proceedings of the 37th ICSE, Florence, 2015. 1–11

2 Yu Y, Wang H M, Yin G, et al. Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? Inf Softw Tech, 2016, 74: 204–218

3 Yu Y, Yin G, Wang T, et al. Determinants of pull-based development in the context of continuous integration. Sci China Inf Sci, 2016, 59: 080104

4 Zanjani M B, Kagdi H, Bird C. Automatically recommending peer reviewers in modern code review. IEEE Trans Softw Eng, 2016, 42: 530–543

5 Chen T Q, Guestrin C. Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16), New York, 2016. 785–794

6 Weißgerber P, Neu D, Diehl S. Small patches get in! In: Proceedings of MSR, Leipzig, 2008. 67–75

7 Pham R, Singer L, Liskin O, et al. Creating a shared understanding of testing culture on a social coding site. In: Proceedings of ICSE, San Francisco, 2013. 112–121

8 Gousios G, Pinzger M, Deursen A. An exploratory study of the pull-based software development model. In: Proceedings the 36th ICSE, Hyderabad, 2014. 345–355

9 Xia X, David L, Wang X Y, et al. A comparative study of supervised learning algorithms for re-opened bug prediction. In: Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR). New York: IEEE, 2013. 331–334