Pull request prioritization algorithm

February 5, 2019

1 Introduction

We proposed a machine learning based algorithm to prioritize the pull requests (PRs) based on their quality. The algorithm is based on two key factors: the **Accept** and **Response** probabilities of the pull requests. The Accept probability means the probability that the pull request will be accepted or not. The Response probability means the probability that the pull request will be responded in the given time frame or not. In our case, the time frame for response is one day.

2 Approach

In this section, we explain the steps that we followed to develop the PRs prioritization algorithm. Pull requests have been extracted from 16 popular GitHub projects, the details are given in Table 1. Figure 1 shows the steps we followed to develop the PRs prioritization algorithm.

2.1 Features Extraction

Features are extracted from five entities, Projects, Issue, Integrator, User, and Author (Contributor), interacting with the pull request. In total 62 features have been extracted including some textual features like title, body, review comments and user comments etc. We used Word2Vec to preserve the semantic meaning to these textual features. Description of all the extracted features is given in Table 5 in the Appendix.

2.2 Accept Probability Prediction

The life cycle of the PR consists of three states: Open, Merged and Closed. When a new PR is created it is in the Open state then the state is changed to Merged state if it does not contain any conflict with the existing code otherwise the pull request is aborted and the state is changed to Closed state. The Accept probability of a PR means the probability that the PR will be Merged or not. The Accept probability problem is transformed into a binary classification

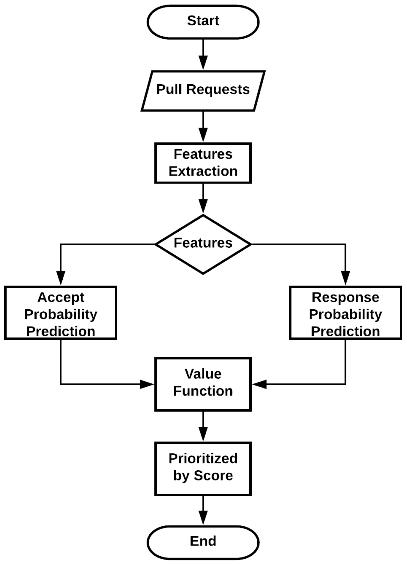


Figure 1: Flow Chart of Prioritizing Method

Table 1: GitHub projects used in this study

Projects	**Open pull requests per day**	**% of open pull requests**
Kubernetes	529.3646	26%
Nixpkgs	235.663	37%
Cmssw	135.5359	43%
Tensorflow	125.4807	50%
Rails	116.2873	55%
Rust	100.674	60%
Symfony	82.66298	64%
Pandas	81.51934	68%
React	80.50276	72%
Salt	76.45304	76%
Scikit-Learn	68.75691	79%
Yii2	67.81215	82%
Cdnjs	65.61326	85%
Terraform	65.38122	88%
Moby	63.94475	92%
Django	61.92265	95%

problem and four classifiers SVM, Logistic Regression, XGBoost, and Random Forest classifiers have been used for the prediction. The model with the best performance is selected to develop our PR prioritization algorithm.

2.3 Response Probability Prediction

Response Probability means the probability that the PR will be responded in the specified time frame. In our case, the time frame is one day. This means that a given PR will be responded on a given day or not. Similar to the Accept probability, the Response probability problem is also transformed into a binary classification problem. The same four classifiers have been used to predict the response probability of the PRs. The best performing model has been used as the ultimate model for our PRs prioritization algorithm.

2.4 Prioritization Algorithm

The proposed PR prioritization algorithm takes the two probabilities, i.e. the Accept and Response, as input and produces a priority score for a pull request in the resulting list. The following value function is used to give a score to each pull request.

$$f(x_1, x_2) = e^{x_1} + e^{x_2} \tag{1}$$

here, x_1 is Accept probability and x_2 is Response probability. The value function assigns the two factors the same weights. It means Accept probability and Response probability have the same degree of influence on the final result. Thus, by leveraging a two-factor ranking based on prediction and synthesis, the algorithm prioritizes the high-quality pull requests that require immediate response of the integrator. The proposed prioritization method can replace the manual selection process of pull requests, which can greatly save the time and efforts of the integrators in pull requests selection.

2.5 Evaluation Measurements

Area Under Cure: The AUC (Area Under Curve), most commonly defined as the area under the ROC [1] curve, have been used in this study to identify the best classifier for pull requests Accept and Response prediction tasks. There are many other evaluation metrics, such as Logloss, Accuracy, Precision, etc. which can be used for the evaluation of classifier, but we have used AUC because it is more robust towards the imbalance datasets [1].

The AUC is calculated as follows:

$$AUC = \frac{\sum_{i\epsilon PositiveClass} Rank_i - \frac{M*(M+1))}{2}}{M*N} \tag{2}$$

The following two measures have been used to evaluate the performance of the proposed prioritizing framework.

Mean Average Precision: Mean Average Precision (MAP) for a set of queries is the mean of the average precision scores for each query [2]. It is a measurement that considers both positive and negative cases and their sort order. The more positive cases that are placed before the negative cases, the greater the value of MAP. It is calculated as follows:

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

$$P(j) = \frac{\sum_{k:\pi(k) \leqslant \pi(j)} y_k}{\pi(j)} \tag{4}$$

$$MAP = \frac{\sum_{i=1}^{n} AP_i}{n} \tag{5}$$

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

Recall: Recall refers to the percentage of correct samples that have been retrieved to the total number of correct samples. We used Recall to measure the proportion of positive samples in the TOP-N sample.

$$Recall = \frac{M}{N} \tag{6}$$

Where M is the number of positive samples in TOP-N, and N is the total number of positive samples.

Table 2: Datasets used in the experiments

Dataset	**Description**
All-Accept	All the features of the 16 projects are extracted from PROMPT dataset along with the label representing whether the pull request is accepted or not
All-Response	All the features of the 16 projects are extracted from PROMPT dataset along with the response label representing that the pull request will be responded within the specified time frame or not
B201709-Accept	A part of All-Accept dataset with pull requests made since the projects are created till September 2017.
B201709-Response	A part of All-Response dataset will responses for the pull requests made before September 2017
A201708-Test	A part of All-Accept dataset with pull requests made between August 2017 & February 2018

3 Dataset Creation

We used PROMPT dataset to perform our experiments. Table 1 shows the sub-datasets that we extracted from the PROMPT dataset for our experiments.

3.1 Experiment I

During experiment I, we trained four classifiers SVM, Logistic Regression, Random forest, and XGBoost on All-Accept and All-Response datasets to predict the Accept and Response probability of the pull requests. To ensure the validity of the prediction, 10-fold cross validation have been used and the training process was repeated 10 times. The results of the experiment I are given in Table 3. As it is evident from Table 3 that XGBoost has outperformed the rest of the classifier and has achieved the highest AUC values of 0.8966 and 0.8734 in the Accept and Response prediction tasks respectively.

Table 3: AUC of Experiment Result

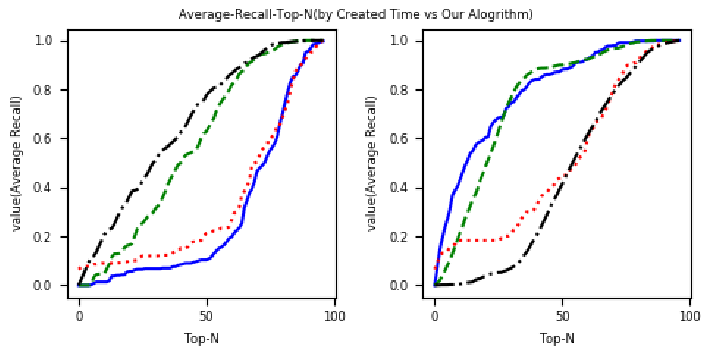
Model	**Accept**	**Response**
SVM	0.5474	0.5045
LR	0.6290	0.5915
XGBoost	0.8966	0.8734
Random Forest	0.7626	0.7522

3.2 Experiment II

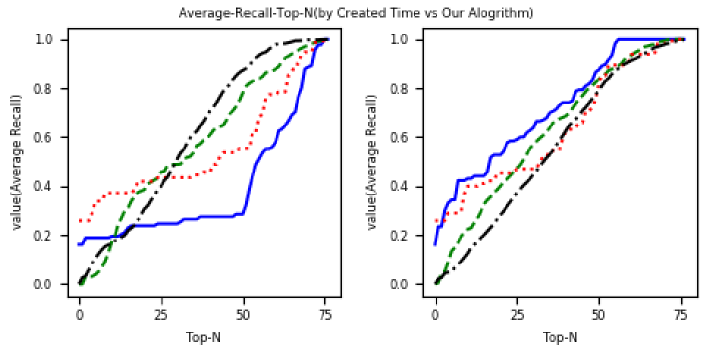
As XGBoost produced the best results in both the prediction tasks, therefore we selected XGBoost to develop our prioritization algorithm. During experiment II, two models have been trained on datasets B201709-Accept and B201709-Response using XGBoost classifiers to predict the Accept and Response probabilities. The output of the two models is then pass through the value function to produce the priority score for each pull request. The algorithm has been tested using the test dataset A201708-Test. The test dataset contains pull requests made in the six months September 2017 to February 2018. The results produced by the algorithm are then compared with a list of pull requests sorted on their date of creation. Preliminary experiments of our prioritization algorithm produce some promising results, given in Table 4, and outperforms the sorting facility of the GitHub.

3.3 Preliminary results and analysis

The proposed prioritization algorithm is evaluated on 16 popular GitHub projects. The prioritization algorithm results are compared with the existing sorting facility provided by the GitHub for the pull requests i.e. sorting pull requests based on the date created.



(a) Project: Symfony-2018-01



(b) Project: Yii-2018-01

Figure 2: Average Recall Curve of pull requests for randomly selected months

Figure 2 shows the average recall curves of the pull requests made in the month of January 2018 vs the prioritizing result list of length N. The left sub-graph represents the average recall curves of the pull requests produced by sorting the pull requests using their creation time, while the right sub-graph represents the average recall curves of pull requests produced by our proposed algorithm. The “accept and response” curve, the blue solid-line, represents the average recall of the pull requests that are accepted and responded to by the integrator on the same day on which they have been created by the contributors. The “accept and not response” curve, the green dashed-line, represents the average recall of the pull requests that are not responded but accepted in some time in the future whereas “not accept and response” curve, the red dotted-line, represents the average recall of the pull requests that are not accepted but responded by the integrators on the same day of their creation. The last curve “not accept and not response”, the black dot-dashed-line, represents the average recall of the pull requests that are neither accepted nor responded by the integrator on the day of their creation. For these experiments, we have defined the first three curves of the pull requests "accept and response", "accept and not response", and "not accept and response" as a positive sample and "not accept and not response" as a negative sample.

It is evident from figure 2 that the average recall increases as N increases in both the left and right sub-graphs of each project. In our case, the high average recall of positive sample means that the proposed algorithm returned most of the high-quality pull requests on the top of the list. As discussed in the previous paragraph that our positive sample consists of the three curves of the pull requests. Our priority is to put the "accept and response" on the top then either "accept and not response" or "not accept and response" and then put the low-quality pull requests "not accept and not response" at the end of the pull requests list.

In average the prioritization algorithm produced good results but there are some cases in which the results are low as shown in Table 4. For project Symfony, our algorithm produced an average MAP of 68% and an average Recall of 44%, which means that the algorithm can successfully prioritize 6 pull requests out of 10 on top of the list. Which we believe can reduce the effort of the integrators during the review process of the pull requests. On the other hand, our algorithm produced lower results i.e. an average MAP of 29% and an average Recall of 40% for the project Yii2. The reason for the low result is due to the less response rate received by the pull requests form the integrators. As the results are calculated on daily basis, maybe on some days of the week the integrator even do not respond to any of the pull requests. Due to which overall results of the prioritization are affected. Besides, it is worth noting that the algorithm has produced both higher MAP and average recall for both the projects in the month of February 2018. We investigated the reason behind the good results and we found that all the project got the highest acceptance and response rate in the month of February 2018. This may be because the more recent the pull requests are submitted the more response they receive from the integrators and hence get accepted.

Table 4: Mean Average Precision and Average Recall for the six months from September 2017 to February 2018 predicted by the prioritization algorithm

Projects	**Mean Average Precision**			**Average Recall**		
	Top-5	**Top-10**	**Top-20**	**Top-5**	**Top-10**	**Top-20**
symfony	0.74	0.68	0.61	0.29	0.44	0.61
rust	0.66	0.60	0.53	0.06	0.12	0.21
pandas	0.65	0.61	0.52	0.21	0.32	0.47
nixpkgs	0.60	0.57	0.51	0.06	0.11	0.18
kubernetes	0.58	0.55	0.49	0.03	0.05	0.09
moby	0.56	0.52	0.48	0.22	0.38	0.59
react	0.56	0.52	0.48	0.35	0.48	0.62
rails	0.54	0.50	0.44	0.22	0.31	0.47
cmssw	0.51	0.49	0.45	0.07	0.13	0.25
salt	0.50	0.47	0.43	0.13	0.21	0.31
tensorflow	0.43	0.41	0.37	0.07	0.12	0.22
terraform	0.40	0.38	0.34	0.31	0.47	0.61
scikit-learn	0.34	0.34	0.32	0.20	0.42	0.65
yii2	0.30	0.30	0.28	0.28	0.40	0.50
cdnjs	0.28	0.27	0.24	0.12	0.21	0.40
django	0.23	0.25	0.24	0.14	0.23	0.42

References

[1] 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

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

A Description of all the features given in the PROMPT database

Table 5: Features extracted from the pull request and its context

Entity	Feature	Description
Project	Project_Name	Name of the project
	Comments_Per_Closed_PR	Average comments of pull requests of the project
	Additions_Per_Week	Number of lines added per week to the project
	Deletions_Per_Week	Number of lines deleted per week from the project
	Team_Size	The size of the project team
	Merge_Latency	The average time of pull request from open to merged of the project
	Friday	The rate of lines changed in the project on Friday
	Sunday	The rate of lines changed in the project on Sunday
	Monday	The rate of lines changed in the project on Monday
	Churn_Average	Average number of lines added and deleted by pull requests
	Stars	The number of stars of the project
	Tuesday	The rate of lines changed in the project on Tuesday
	Project_Age	The age of the project
	Wednesday	The rate of lines changed in the project on Wednesday
	Saturday	The rate of lines changed in the project on Saturday
	File_Touched_Average	Average Number of total files touched in every pull request
	Forks_Count	The number of forks of the project
	Watchers	The number of Watchers of the project
	Thursday	The rate of lines changed in the project on Thursday
	Close_Latency	The average time of pull request from open to close of the project
	Contributor_Num	The number of contributors to 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 project
	Language	the Language of project
Comments	Pull_Request _ID	Pull_Request _ID is used as a foreign key referencing to the corresponding pull request
	Participants_Count	The number of participants in one pull request
	Comments_Embedding	The embedding of all the comments
	Comments_Count	The number of comments
	Last_Comment_Mention	Does last comment contain '@'?
Pull_Request	Pull_Request _ID	Pull_Request_ID is used as a primary key in the table
	Project_Name	Project_Name is used as a foreign key referencing to the corresponding project
	Body	Embedding of body text in the pull request
	Intra_Branch	Are two branches of two projects the same?
	Mergeable_State	Is this pull request in a mergeable state?
	Assignees_Count	The number of assignees in the pull request
	Label_Count	The number of labels in the pull request
	Files_Changed	The number of changed files in the pull request
	Contain_Fix_Bug	Does the pull request fixes a bug?
	Wait_Time	The waiting time of the pull request
	Day	The current day
	Deletions	Number of lines deleted from the pull request
	Additions	Number of lines added to the pull request
	Rebaseable	Is pull request rebaseable?
	Mergeable	Is pull request merge-able?
	Title	The embedding of the title text
	Commits_PR	The number of commits of a pull request
	Workload	The number of open pull requests
	url	The URL of each pull request
	PR_Latency	The time between the creation and closing of a pull request
	Commits_Average	Average number of commits per pull request
Contributor	Pull_Request _ID	Pull_Request _ID is used as a foreign key referencing to the corresponding pull request
	Private_Repos	The number of private repos of the creator
	Followers	Number of Followers of the creator
	Closed_Num	Number of closed pull requests of the creator
	Contributor	Role of the creator in the project
	Public_Repos	Number of public repos of the creator
	Organization_Core_Member	Is creator the core member of the project organization?
	Contributions	Number of contributions of the creator
	User_Accept_Rate	The rate of the merged pull requests of the creator
	Accept_Num	Number of the merged pull requests of the creator
	Closed_Num_Rate	Number of the closed pull requests of the creator
	Prev_PRs	Number of the previous pull requests created the creator
	Following	The number of followers of the creator
Reviews	Pull_Request _ID	Pull_Request _ID is used as a foreign key referencing to the corresponding pull request
	Review_Comments	Embedding of the review comments
	Review_Comments_Count	Number of review comments
Issue	Pull_Request _ID	Pull_Request _ID is used as a foreign key referencing to the corresponding pull request
	Point_To_IssueOrPR	Does the pull request aim to solve an issue or like other pull requests?
	Open_Issues	The number of open issues
Response	Pull_Request _ID	Pull_Request _ID is used as a foreign key referencing to the corresponding pull request
	response_label	A Boolean field represents whether a pull request is responded on a given day