Merge Time Prediction of Pull-Requests

Kyriakos Fragkeskos
K.Fragkeskos@student.tudelft.nl

Mengmeng Ye
M.Ye-1@student.tudelft.nl

Anelia Dimitrova
a.dimitrova@student.tudelft.nl

Department of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, Netherlands
2628 CD, Delft

ABSTRACT

Git-hub is one of the most popular web-based hosting services. It offers a wide range of functionalities for distributed revision control and source code management. One of the most interesting functionalities are the pull requests. Pull request gives the opportunity for everyone to make a contribution in a certain project and streamlines the process of contribution to that project as it is easy to understand for programmers and changes are implemented within the request itself, removing the necessity of manual comparison of code and automating the merging process. In this paper, we are going to answer the following research questions: Which features contribute most to the prediction of the time for merging, and How can we predict the merging time of a pull request? We find out that, the majority of the pull request are merged after the period of 1 hour. Using certain classifiers, we achieved to predict, for a given pull request, with accuracy up to 74% if it will merged in the next 12 hours after the submission of the pull request and with 85% accuracy if it will be merged in more that 1 week.

1. INTRODUCTION

Git hub is arguably one of the most famous web-based hosting services. Github offers a variety of functionalities meant to ease the use of source code management and sharing. These functionalities, are a valuable resource and descriptive features for a pull based system development [4].

The pull request is a tool, which assists developers to collaborate. For instance, a user has the ability to fork (clone) any open repository in GitHub, and download it on their machine (locally) and they are able to make any changes they want. After the addition of the changes, the user sends a pull request, which encapsulates all the changes (e.g commits, comments, type, deletions, insertions) the user did. This pull request is assessed by a main contributor of the project (code reviewer) and if the changes are satisfactory, then the pull request is merged with the main branch of the repository.

Unfortunately, there is no guarantee that a pull request will be assessed, whether it will be merged or discarded, or how long it will take to be evaluated or merged. Predicting the merge decision is not an easy task and thus, it requires finding the right approach and procedures and lot of training data in order to achieve this. [4] Using the features from, either the pull request, or traits from the entire repository and user to estimate the information, has gained widespread acceptance [4]. These techniques aim to find a descriptive subset of possible features and analyze them to classify whether a given request will be merged or not using machine learning algorithms or other predictive models.

Nevertheless, this approach [4], refers to a dataset that does not represent the entire population of projects on GitHub, because the dataset that they used, contains information about the projects with sufficient amount of history of pull requests. Therefore, the results may not be applicable or generalizable.

Our conjecture is that the use of a more representative dataset [5] will lead to more accurate and more generalizable outcomes. Based on that, we will try to predict the time period that is needed for a pull request to make the transition from "Open" to "Merged". Moreover, we will try to find and quantify the factors of the pull requests the project and the user that lead to an accurate prediction. A positive outcome will help programmers (contributors) to better schedule their time and efficiency to be more productive.

Hence, we conduct an analysis on the dataset, that we had at our disposal [5]. The outcomes of the analysis illustrate the correlations between the time, needed for a pull request to be merged, with other descriptive features from pull request, projects and users,(see table 1). Taking into account the above information, we construct classifiers to predict the merging time. We succeed to predict accurately the time for pull request, with accuracy up to 87%, using classifiers such as Naive Bayes and Random Forest [11].

The rest of this paper is organized as follows: section 2 illustrates briefly the related work, section 3 defines the problem that we try to solve, section 4 presents the background of the supporting technologies. Our approach is illustrated in Section 5. Finally, Section 6 summarizes the paper with remarks on future work.

2. RELATED WORK

Many attempts try to analyze or to find any correlation between pull requests and other project characteristics. Gousios et al, [5], created a dataset that contains information about pull requests. The research question that sprung from this paper is which traits are the most suitable to characterize a pull request and if there is any relation between the pull request characteristics, project's characteristics and users' characteristics, in order to conduct a basic exploration of the pull-based distributed development model. The contribution of this work was the analysis of pull-based system and the data set with the pull requests.

Another approach, which is the most similar with ours, tried to predict if a pull request is going to merged or not [4]. They give answer to a set of research questions like, if the pull based development model is popular, what are the life-cycle characteristics of a pull-request and what factors affect the decision and the time required to merge a pull request. To do that, they create a dataset with projects with sufficient history of pull-requests.

They extract any available information from the data set and the do cross-correlation to find out the subset of traits with the most descriptive power. Afterwards they deploy machine learning techniques to predict if a pull request is going to merge or not. They conclude that a small number of factors affect both the decision to merge a pull request and also they discover that, technical reasons contribute a small fraction in the rejection of a pull request. The difference with our attempt, is that we aim to predict the time period each pull request needs before being merged, instead of the merge decision itself.

Other studies, [2, 3], which tried to predict the estimated time of response in Stack Overflow [1], mentioned the difficulties of predicting the exact time. Hence they proposed a way to categorize the time into bins. For instance, they create bind for a day, a week and a month. In this paper we will use similar approach to predict time. The difference with their approach is that we are going to split the time into several time bins, (e.g 10 bins) and we are going to predict the probability for a pull request to be in each of these bins. Another difference is that we aim to determine the time bin a pull request needs to be merged, instead of the lifetime of the merge.

3. PROBLEM

Working on a project for a contributor might be very time consuming, hence this user should organize their time and work on projects based on the time it takes for a suggestion they made on a project to be approved and merged, or rejected with a reason why it is not approved. In the meanwhile the programmer can keep working on other things without having to worry about project A for which they submitted a pull request and having to jump from project A to project B. The job of a programmer is usually very demanding and requires a lot of time, tests, debugging, thinking and code writing and sometimes twenty-four hours for such a person might not be enough for everything they have to do. And here the prediction of the time for a pull request from state “Open” to state “Merged” comes in very useful as time management is crucial.

The time for a pull request to be merged or rejected might be different for projects, written in different programming languages, also if one project is more popular than another or if a project is related to the user’s primary job or just for improvements of an open-source application or feature. We are going to research and confirm or reject these assumptions based on the data we have from Github.

Having a fairly accurate time prediction on this issue, is likely to help users in their every-day job and will make their work way more efficient. To understand if that would be useful, we conducted a small study, under the form of a questionnaire, among programmers who use Github in their full-time or part-time programming job.

3.1 Survey

We decided to conduct a small research among programmers who use Git in their everyday full or part-time job. The survey was under the form of a questionnaire. As programmers are busy people, in order to make them respond to it, we had to ask as less questions as possible. Thus, our questionnaire consists of only four questions, one of which is for their name. The questions we asked are as follows:

- Q1: Name
- Q2: Which programming language do you write on?
- Q3: Would it be useful to know how long it will take for your pull request to be merged?
- Q4: Please give us more details on your answer of question 3

Of these questions, only one was required and that was question 3, related to our main paper topic. Question 4 was a request for developers to elaborate on their answer on question 3, so that we can get a better idea of their opinion. As part of our initial analysis, was a research on number of bugs for projects, written in different programming languages, so we wanted to keep a question for the programming language in the questionnaire and have different kinds of developers answer it.

Since we did not have so many options, we shared the questionnaire in social media such as Facebook and in the Q&A website Quora and asked people for who we know for sure that they work as developers, to answer our questions and to also share the survey with their colleagues. We got 16 answers in total, which is not enough for such a study, however this could be part of a future empirical study on the matter. The results of our survey are as follows:

- **50%** answered that it **will be useful** to be able to predict the pull request merge time
- **18.8%** responded that this **won’t be a useful** feature for them
- **18.8% do not use** Github
- **12.5%** have **no opinion** on this topic

We also got an interesting comment, supporting the usefulness of the merge-request time prediction which states: *”If my pull request takes around a year or longer to get merged, it shows to me that the project is either understaffed or no longer active. That will hold me back from making*

[1] `http://stackoverflow.com/`

more pull requests. The other way around, if it is likely to be merged pretty quickly (say, within a couple of months for a big project), that will ***encourage*** *me to* ***contribute more*** *since I can see that my changes are* ***taken seriously*** *AND are* ***worth*** *something."*

4. BACKGROUND

This section describes two supporting technologies/concepts for our work: Pattern Recognition [9], GitHub [6].

4.1 Pattern Recognition

"Pattern recognition is the scientific discipline whose goal is the classification of objects into a number of categories or classes" [9]. Depending on the application, the classes could be either images or signals or feature vectors etc. In order to tackle the classification problem, i.e, to find the classifier which returns the minimum classification error, several approaches exist that provide a remedy [9].

These approaches are divided in two main categories: the supervise learning algorithm and the unsupervised learning algorithms. In the first category, the desired output of the object is already known and therefore, the classifier is trained by exploiting this knowledge. On the contrary, in the second category there is no such knowledge and therefore the classification task becomes even harder to solve. In this paper, we will use naive bayes and random forest algorithms to predict the merge time of a pull request [11].

4.2 GitHub

GitHub is one of the most important source of information that is oriented in software development. With 10 millions of repositories, GitHub is, without a doubt, one of the most prominent web-based Git repository hosting service. Many research was made to understand the characteristics of the repositories in GitHub and the behaviors of the users/collaborators [6]. GitHub helped making open source much more decentralized. It is now less about the project and more about the individuals. [8]

The Git repositories are managed by one person or a group of people [10], but the majority of developers can not directly change the code on that project. They need first to fork the repository locally on their machines, to edit the code and then to submit a pull request where someone reviews the changes and merges the request with the main code or rejects the request if it unsatisfactory in any sense. The advantage of knowing the estimated time for such a request to be approved or rejected, could give developers an opportunity for better time management.

In this papers we will exploit this rich source of information [5], to find out the merging time of a pull request. GitHub provides us, with useful information about all the objects (i.e. users, projects, commits, comments etc), and the interactions between these objects.

5. APPROACH

The main focus of this research is to understand how useful a prediction of pull request merging time will be and how can we predict it. To achieve this, we have two main research questions:

RQ1: Which features contribute most to the prediction of the time for merging?
RQ2: How can we predict the merging time of a pull request?

To start answering the questions, we read a number of related papers, explaining which algorithms are the most suitable for our purpose and we decided to focus on a representative and big dataset, to select a set of candidate features on which we use cross-correlation analysis to obtain the best of them that contribute most to the prediction of pull-request merging time. Thus, we answer RQ1.

To answer RQ2, we start by selecting the merged_at feature and label the request as "merge" if it is not NA. Then, after filtering out the un-merged requests data, we select the features to use to derive the time in minutes before each request is merged. For the advanced-level mining, we run the following classification algorithms, known to output good results on large datasets [7] such as ours: Random Forests, Naive Bayes and Support Vector Machines [9].

Given a pull request that was made from a programmer, our methodology works in 6 steps, 1)Find a representative dataset which enables us to make valid statements [4], 2)Proceed with initial analysis of the dataset, 3)Find the correlations of the predictors in the dataset, 4)Create 10 bins , which illustrate the merge time (e.g 1 min, 2 min, 30min, 1h etc), 5)Feature selection from the initial variables and 6)Classify and predict the time bin of a pull request.

To predict the merge duration of a pull request, first we need a dataset. We chose MSR 2014 Challenge Dataset [4]. This dataset is exclusively for pull requests. That is to say, it contains only information about the pull requests. The features of this dataset are illustrated in table 1.

Since the time in our data is a numeric value and is not easy to predict exactly [2, 3], we divided the merge time into 10 bins. The bins are illustrated in figure 3. As we can see from the figure 3, the majority of the pull requests, is distributed uniformly between the bins of 1 hour and 2 days. That is to say, the majority of pull request is merged between 1 hour and 2 days after the user submits their pull request. Many interesting facts, are sprung from this figure. That is, the request which were merged in 5 minutes and 15 minutes are relatively lower with the requests that are merged in less of minute after the request. Also, a significant amount of request is merged after 1 week period, considering the fact that huge projects have many contributors and it takes longer for all the merge requests to be revised and merged.

Thus, our goal, is to predict the correct time bin for a pull request. As we are concentrated on the merge time, we neglect all the un-merged pull requests and keep and work only with the merged pull requests. First, we conduct an initial analysis to discover the distributions of the numeric variables in this dataset. Moreover we calculate the pairwise Pearson correlation, [1], in order to find any hidden correlation (positive, negative or neutral) between the numerical variables. This information, will give us an intuition about the relatedness between the features in the dataset. The results of

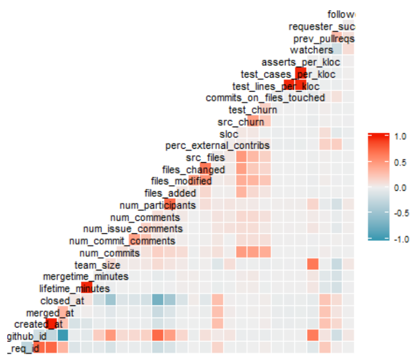

Figure 1: Correlation for the entire dataset

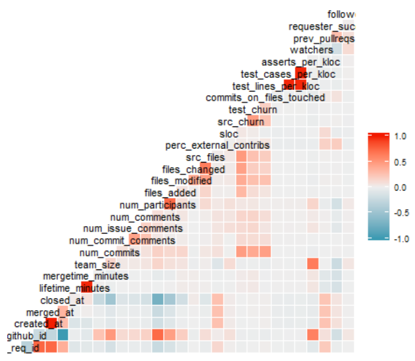

Figure 2: Correlation only for merged pull requests

the correlations for all (merged/unmerged) pull request are shown in figure 1 and for only the merge pull request at figure 2. There is a slightly difference between the two figures on the correlations of the variables. An interesting finding, is that, the merging time is slightly negatively correlated with the number of watchers and number of previous pull requests. The strong correlations are the dark red squares. They are intersections between two features.

Furthermore, we extract the distributions of the predictors and we discover, as we expected, that the majority of them are extremely skewed. We select the variables, which are more correlated with the merge time. Their distributions are illustrated in figure 4. As we want to see if it's possible to reach normal distribution on our variables, we scale and visualize (see figure 5) these distributions using the logarithmic scaling (log).

Afterwards, we follow similar approach with previous study, [4], in order to predict the merge decision with different features and classifiers, like the Support Vector Machine [12]. Finally, we conduct an advance analysis, using naive Bayes and Random Forest to train our data, using 10 fold cross validation, in order to answer our research questions.

6. EVALUATION

All the experiments in this study use the same dataset [4]. This data contains all the information about pull requests. For the first experiment, we follow similar approach with a previous study [4], with the exception that we used different features and extra classifiers. We used the features

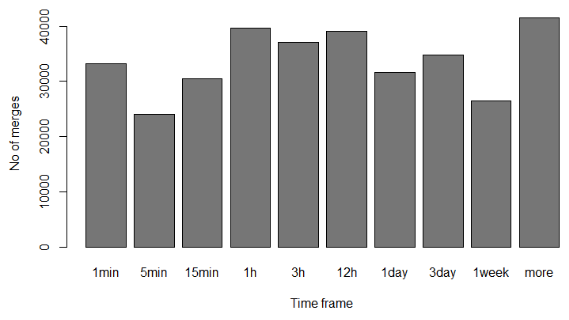

Figure 3: Merge estimation time, divided in time frames

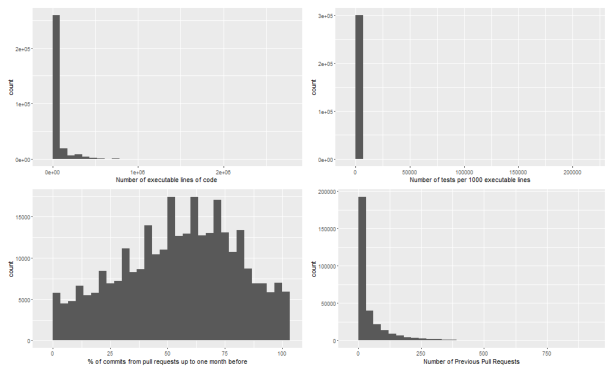

Figure 4: Distributions of variables

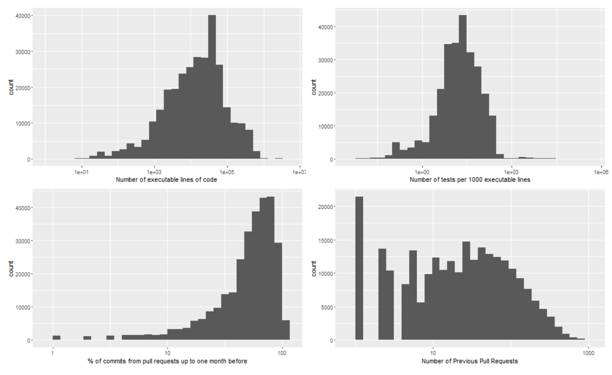

Figure 5: Logarithmic Distributions of variables

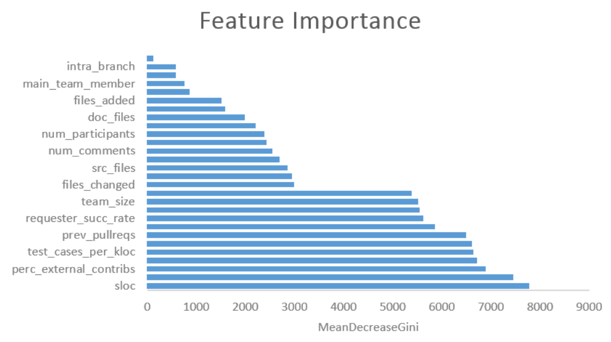

Figure 6: Feature Importance

requester_succ_rate, perc_external_contribs and main_team_member, see table 1.

For the second experiment, we attempt to predict the time bin of the pull requests. We use all the features, that we have at our disposal. And we try to predict the time bin, using support vector machine and random forest algorithms. Moreover, we find the features that contribute the most to our model.

Classifier	Accuracy	Precision	Recall	F1
Naive Bayes	0.8	0.83	0.94	0.88
SVM	0.81	0.81	0.8	0.82
RandomForest	0.79	0.80	0.85	0.82

Table 2: Merging Decision Prediction

For the first experiment we manage to predict the merging decision with up to 83 percent of precision and 94 percent of recall. The results from all the classifiers are presented in the table 2. By evaluating the accuracy, precision, Recall and F1 of the three algorithms, we can see that accuracy of Naive Bayes is as high as the accuracy of the Support Vector Machines' while the rest of the criteria is higher than the criteria of the other two algorithms. Hence we chose Naive Bayes (naivebayes) as the most applicable one for our data.

For the second experiment we train two classifiers, Naive Bayes and Random Forest. For the first class (1min) we manage recall of 77%, precision of 74% with Naive Bayes. For random forest we manage to predict the recall of 82% and 93% precision. For the rest classes we did not manage to get more accurate results. Overall, random forest returns better results for the category: "More"(Recall: 81%, Precision: 89%). Furthermore, we found the importance of each feature, using random forest, the results from the analysis are illustrated in figure 6.

6.1 Results

Base on the experiments, we succeed to predict the time bin for a given pull request. The results in table 4, indicate that our approach can predict the time a pull request is needed, to be merged. The difference of the outcomes from the two classifiers is reasonable. That is to say, Naive Bayes assume that the predictors are statistically independent, also Naive Bayes is a linear classifier and therefore some of the results in table 3, are significant low.

Class	Recall	Precision	Accuracy
1min	0.77633	0.74476	0.76055
5min	1.339e-03	9.976e-01 6	4.995e-01
15min	0.41699	0.67049	0.54374
1h	0.12402	0.87381	0.49892
3h	0.0054209	0.9944655	0.4999432
12h	0.13204	0.90699	0.51952
1day	9.480e-04	9.985e-01	4.997e-01
3day	0.14634	0.92118	0.53376
1week	0.0103774	0.9962237	0.5033005
more	0.12300	0.98259	0.55279

Table 3: Merging Time Prediction Naive Bayes

Class	Recall	Precision	Accuracy
1min	0.82374	0.93375	0.87874
5min	0.45397	0.99071	0.72234
15min	0.44373	0.98891	0.71632
1h	0.55490	0.94077	0.74783
3h	0.49346	0.96419	0.72883
12h	0.57531	0.91280	0.74406
1day	0.47859	0.96469	0.72164
3day	0.50402	0.95100	0.72751
1week	0.44906	0.98123	0.71514
more	0.8129	0.8970	0.8550

Table 4: Merging Time Prediction Random Forest

On the other hand, random forest is a non-linear approach, which creates multiple decision trees in the training phase and returns the class with the highest likelihood. The results from the random forest, outperform the results from Naive Bayes. Random Forest, can predict with 93% precision and 82% recall if a pull request will be merged in the next minute, with 91% precision and 57% recall if it will be merged in the next 12 hours and with 89% precision and 81% recall if it will merged in more that one week. This is the difference with previous studies [4], we predict the merge time using multiple time bins to categorize the time and to predict the correct time bin for a pull request.

Furthermore, we find the factors which contribute most in the prediction using random forest. We conclude that the four most important factors are the number of tests per 1000 executable lines, the number of executable lines of code in the main project repository, the percentage of commits from pull requests up to one month before and the number of previous pull requests (see figure 6.

7. CONCLUSION

GitHub is one of the most popular hosting services for source code management. Millions of codes modifications are conducted on, which provides rich data features and space for analysis. This paper did an analysis on the feature of pull request. Users can modify the code of a project on GitHub and submit a pull request, owner of the project can choose whether to merge this pull request with his source code or not. In this paper, we did a research on pull request merging time prediction.

First, we split the pull request dataset in two parts, depending on if the pull requests are merged or not. Then, we conduct an analysis to find any correlation between the available features in our dataset. We split the merging time into 9 time bins and divide our dataset in these 9 bins. We find out that, the majority of the pull request are merged after the period of 1 hour. Using random forest and Naive Bayes, we achieved to predict, for a given pull request, with accuracy up to 87%, recall up to 82% and precision 93% if it will me merge in the next 1 minute after the submission of the pull request, with 74% accuracy if it will merged in the next 12 hours and with 85% accuracy if it will be merged in more that 1 week.

A user can take advantage of this information to improve his/her performance, by making more pull request and to create an efficient schedule to increase their productivity. A future work based on this concept would be to utilize techniques for feature extraction, like Principal Component Analysis [9] (PCA), to devise extra factors that would be more correlated with the merging time, in order to achieve even more accurate results. Another future work might be to remove one of the time bins: for less than 1 minute and rerun the classifiers, again to get even more accurate results.

8. REFERENCES

[1] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[2] S. Ercan, Q. Stokkink, and A. Bacchelli. Automatic assessments of code explanations.

[3] J. Goderie, B. M. Georgsson, B. van Graafeiland, and A. Bacchelli. Eta: Estimated time of answer predicting response time in stack overflow.

[4] 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*, pages 345–355. ACM, 2014.

[5] G. Gousios and A. Zaidman. A dataset for pull request research. *Submitted to MSR.*

[6] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101. ACM, 2014.

[7] B. Liu, E. Blasch, Y. Chen, D. Shen, and G. Chen. Scalable sentiment classification for big data analysis using na x00ef;ve bayes classifier. In *Big Data, 2013 IEEE International Conference on*, pages 99–104, Oct 2013.

[8] M. Rogers. The github revolution: Why we're all in open source now. 2013.

[9] S. Theodoridis and K. Koutroumbas. *Pattern Recognition, Fourth Edition.* Academic Press, 4th edition, 2008.

[10] Various. Managing multiple people working on a project with git. 2012.

[11] X. Wu and V. Kumar. *The top ten algorithms in data mining.* CRC Press, 2009.

[12] Z. Xuegong. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica*, 26(1):32–42, 2000.

Feature	Description
pull_req_id	The database id for the pull request
project_name	The name of the project (same for all lines)
github_id	The Github id for the pull request.
created_at	The epoch timestamp of the creation date of the pull request
merged_at	The epoch timestamp of the merge date of the pull request
closed_at	The epoch timestamp of the closing date of the pull request
lifetime_minutes	Number of minutes between the creation and the close of the pull request
mergetime_minutes	Number of minutes between the creation and the merge of the pull request
merged_using	The heuristic used to identify the merge action. The field can have the following values
github	The merge button was used for merging
commits_in_master	One of the pull request commits appears in the project's master branch
fixes_in_commit	The PR was closed by a commit and the commit SHA is in the project's master
commit_sha_in_comments	The PR's discussion includes a commit SHA
forward_links	Boolean, true if the pull request comments include a link to a newer pull request
team_size	The number of people that had committed to the repository directly
num_commits	Number of commits included in the pull request
num_commit_comments	Number of code review comments
num_issue_comments	Number of discussion comments
num_comments	Total number of comments (num_commit_comments + num_issue_comments)
num_participants	Number of people participating in pull request discussions
files_added	Files added by the pull request
files_deleted	Files deleted by the pull request
files_modified	Files modified by the pull request
files_changed	Total number of files changed (added, modified, deleted) by the pull request
src_files	Number of src files touched by the pull request
doc_files	Number of documentation files touched by the pull request
other_files	Number of other (non src/doc) files touched by the pull request
perc_external_contribs	% of commits from pull requests up to one month before.
total_commits_last_month	Number of commits
main_team_commits_last_month	Number of commits to the repository during the last month
sloc	Number of executable lines of code in the main project repo
src_churn	Number of src code lines changed by the pull request
test_churn	Number of test lines changed by the pull request
commits_on_files_touched	Number of commits on the files touch by the pull request during the last month
test_lines_per_kloc	Number of test (executable) lines per 1000 executable lines
test_cases_per_kloc	Number of tests per 1000 executable lines
asserts_per_kloc	Number of assert statements per 1000 executable lines
watchers	Number of watchers (stars) to the repo at the time the pull request was done.
requester	The developer that performed the pull request
prev_pullreqs	Number of pull requests by developer up to the specific pull request
requester_succ_rate	% of merged vs unmerged pull requests for developer
followers	Number of followers to the pull requester at the time the pull request was done
intra_branch	Whether the pull request is among branches of the same repository
main_team_member	Boolean, true if the pull requester is part of the project's main team .

Table 1: Features of Pull Requests