Predicting Github's Pull-Requests using Feature Selection Algorithms

Arjun Bharadwaj
University of California, Davis
abharadwaj@ucdavis.edu

Christopher Lock
University of California, Davis
cclock@ucdavis.edu

ABSTRACT

In the past decades, Github has become a place where multiple developers can collaborate on various projects. In order to actively colloborate and contribute to projects, developers make certain kinds of changes and submit a *pull-request* for the change to be approved and merged. Under this scenario, the core team member(s) approve or deny the request and take action based on the change. There can be many factors which may affect the acceptance of the pull-request and examining these various features can lead to helpful insights. Many authors have relied on extracting features based on heuristics such as the team-size, bugs found, source files changed etc. However, in this paper, we focus on feature extraction using statistical methods such as univariate selection, recursive feature elimination, and tree-based selection methods. We present our findings and the performance of these various algorithms and the predictive models in this paper. In addition, we also summarize some interesting findings about the results and the data itself.

1. BACKGROUND

1.1 Introduction

Within the past few years, there has been a lot of interest on open source projects. These projects are hosted on various websites such as Github and SourceForge. Under this context, numerous developers contribute their unique ideas on various projects. There is a certain kind of methodology behind the collaboration model and the one that Github uses is called the *pull* request model. In this model, the developer submits a change to a given project, which is later approved by a core team member(s). It is often interesting to study pull requests and the various features associated with them. When a developer submits a pull request, the request is considered to be "opened" and when the core team member decides to take an action, it is then "closed". Ultimately, a pull request is either accepted or rejected and it can be intriguing to analyze which factors might affected such a result.

In this paper, we primarily replicate the study performed by other authors who have studied and performed several statistical analyses on the pull requests model. However, we focus our analyses on the feature selection part because choosing important features can ultimately affect the final results.

Analyzing the Github data can be quite complex since there are many features which affect whether a pull request was merged into the main branch or not. Therefore, we perform a feature extraction on the data using three methods: Univariate Feature Selection, Trees-Based selection, and Recursive Feature Elimination. The idea is we apply each of these algorithms, extract the important features and train our predictive model using these features.

After choosing the appropriate features, we then train a classifier under a supervised learning setting to predict whether a pull request will be accepted or not. Ultimately, the idea is that if we can determine which factors affect the acceptance of a pull request, developers can tweak their pull requests in order to make it more likely acceptable. For example, if we see that pull requests associated with bigger code sizes are rejected, developers can then take steps to make sure the pull requests are smaller in size. The other purpose of conducting this experiment is of-course to compare and contrast the findings of other authors as mentioned previously.

1.2 Related Work

The first significant paper that we will referenced for this project is the [4] Yue Yu et.al paper, "Wait For It: Determinants of Pull Request Evaluation Latency on GitHub". In this paper, the authors analyze the pull-request latency and the factors which might affect them. In the paper, the authors often use good heuristics features to predict the outcome of latency. This addresses and solves the problem, but it would be more beneficial to strengthen the initial feature extraction phase by validating it with machine learning and statistical algorithms. This is what we hope to achieve in this experiment.

Another related work is depicted in Rahman's paper [3], "An Insight into the Pull Requests of GitHub". In this paper, the authors have focused primarily on studying the pull request discussion texts in order to predict the success of a pull request. This paper sheds some light on how we can examine pull requests discussion texts to gather insightful data and perform analysis. Analyzing string documents can be quite challenging often and this paper outlines some ways in which these texts could be used in order to predict the success rate of a particular pull request. However, due to

This report was created for the Software Engineering Project for the class ECS 260 of UC Davis. If any questions or comments arise, feel free to email the above email addresses.

UC DAVIS University of California, Davis - USA

DOI: UCDavis

time constraints and the scope of the project, we did not end up performing any kind of text analyses.

The last line of related work is [2] Gousious and Zaidman's paper, "A Dataset for Pull Request Research". In this paper too, they define the feature selection based on popular heuristics such as patch submission and acceptance, code reviewing, and bug triaging. The clarity of this paper is very beneficial because it clearly outlines the features and delineates the separate categories: Pull request characteristics, Project characteristics, and developer characteristics, which might influence pull requests. So it will indeed be interesting to compare their features with the ones we choose for our findings.

2. HYPOTHESIS

We propose the following research questions and convey our hypothesis on each one.

- **RQ1: Do pull request acceptance have a trend over time? If so, what is it in terms of minutes or days? In terms of months?**

  As we will explain in our data gathering section, we gathered all open source projects which are relatively large in terms of team members in the year 2016. The nature of these projects are such that they have a large number of core team members and contributors. This presence can be helpful since the community can approve and make changes to the project efficiently and instantaneously. Therefore, since bigger open source projects more likely have higher core members, we think that the pull requests should be accepted relatively quickly. We think that it will most likely be three or four days. We also think that number of pull requests that get accepted each month stays the same or increases since there should not be any really impact of the month on the acceptance rate.

- **RQ2: What are some features a developer can control in order for his/her pull request to likely get accepted?**

  We plan to bring value to a developer who plans to submit a pull request. From his or her perspective, we want to examine the various factors that are controllable such as the churn rate, title and description complexity, which can be carefully designed or modified in order to make the pull-request likely acceptable. We hypothesize that shorter pull requests have higher acceptance rates since they are short and make less code changes and therefore are easier to review and approve for the core members and are beneficial in terms of the scope of the project.

- **RQ3: What is the importance of tests in pull-requests acceptance? (i.e are testing variables powerful indicators for the acceptance of pull requests?)**

  There is a lot of emphasis on test-driven development in the industry as well as open source projects. Due to this phenomenon, it will definitely be interesting to examine the impact of test variables in the acceptance of pull-requests. We think that the test variables will probably be one of the top five features. This is most likely because in open source projects, testing is given a lot of prominence. Therefore, we will be examining a few test variables in our dataset and be evaluating their importance using the feature selection algorithms.

3. DATA GATHERING

We gathered data from primarily two sources: *GHTorrent* and *TravisTorrent*. We first used GHTorrent's data to obtain all the open source projects on Github which had atleast more than 10 pull requests in the year 2016 over a period of seven months (January to July- since that was all the data available in the dump). We examined a total of 47 Open Source Projects including several famous ones such as *scipy, ipython, pandas* etc. Having obtained the main features such as the pull request number, project name, etc, we then included the other important variables such as the churn rate, length of the pull request, changed files etc. However, we came to a consensus that there were several other important variables such as `gh_test_cases_per_kloc` and `gh_test_lines_per_kloc` which were important to evaluate. However, this data was present in TravisTorrent's sql dump. Therefore, we joined the GHTorrent's Data with the TravisTorrent's data using the pull request id. In the very end, we had a total of $36,776$ pull requests to train our model and the data comprised of 43 columns.

3.1 Preprocessing

Even after the preprocessing stage of combining the *GHTorrent* and *TravisTorrent* data, we had to remove some redundant features such as the id numbers, SHA-1 Hash Id, etc. since we were only concerned with feeding important numerical data into the machine learning models later on. This resulted in 36 features. We then identified the target variable, `merged` which indicated whether a pull request was merged or not. We chose all the other numerical columns as the features in our model. Therefore, the goal was to build a binary classifier under a supervised setting using a subset of the 36 features. The subset of these features was determined based on certain feature selection algorithms which will be highlighted in detail in the following section.

4. RESULTS

In order to gain more insights and understand the behavioral pattern for the pull requests, we first decided to examine the correlation between the time it took for the pull requests to be merged with the number of pull requests across all projects. The results were surprising as we found that the pull-request acceptance rate was even quicker than we expected (in hours as opposed to days that we had proposed). The graph in the appendix outlines our results [1].

Several interesting observations can be made here. The first being that distribution follows a power law, similar to many findings in empirical software engineering. In addition, we also made the observation that almost half of the pull requests (53%) get accepted under 8 hours, which seems quite intriguing. This insight refutes our first research question that it takes days to process and approve pull requests. In addition, we also graphed the number of pull requests that were accepted for each month. Surprisingly, we found that the number of pull requests that were accepted each month decreased.[1] We hypothesize that the reason for this is that the complexity of the project increases as the year progresses, so adding more features and accepting more pull

requests has a higher chance of breaking the existing code and therefore more pull requests are rejected. However, with the given data, this hypothesize is extremely difficult to verify and therefore the result is inconclusive . Therefore, the finding for the first research question is as follows:

> For large open source projects, it is likely that the pull request is accepted in the **order of hours** rather than **days or weeks**. Furthermore, the pull request acceptance follows a trend in the sense that **the number of pull requests accepted decreased over a period of time**.

In the second phase of the project, we applied three different feature selection algorithms and evaluated each one. The first feature selection algorithm we applied was the *tree-based selection algorithm.* We used the scikit's learn `ExtraTreesClassifier` to rank the relative importance of each feature. [1]

In the figure, all the values add up to 1 and hence each feature's importance is normalized. The number of comments is the most discriminative feature in the dataset based on the result of $1,000$ trees. When the second method, *Univariate selection* using the Chi test was applied on the dataset, the top 10 features that were obtained were *churn rate, description length, additions, network count, sloc, changed files, deletions, commits , watchers count, and open issues count.* Furthermore, applying the Recursive Feature Elimination method gave us a different set of features. The important point to understand here is why each feature selection gave us a different set of results. We hypothesize that this could be due to two reasons.

The first reason is that there could be strong correlation between the features and each feature selection algorithm takes its own approach of dealing with these types of conflict. Therefore, in order to identify whether there were correlations between the features, we computed a pairwise similarity matrix and produced a heatmap based on its values.

We found that most of the variables were barely correlated with each other. However, there was still some strong correlation between the test variables such as `gh_test_cases_per_kloc` and `gh_assert_cases_per_kloc`. To resolve this and avoid possible problems in the future, we removed strongly correlated features before feeding them into the machine learning models.

The second reason as to why the feature selection could give us a different set of results is the distribution of data itself. To examine and study this further, we plotted the histograms for each feature. [1]. We immediately saw that the values for most of the variables were extremely skewed. To resolve this, we logged those particular features. [1] However, even after logging, we found that there were still around 6 or 7 features whose values were skewed. Therefore, we think that due to the distribution of data itself, we might get some inaccuracies when we run the models.

In order to gain more insight on the data, we also performed the Principal Component Analysis (PCA). We mainly plotted two graphs: the spree plot and the biplot. For a detailed discussion on the plots, we encourage the reader to look at the appendix.The finding of the distribution of variance among features using PCA strongly reinforces the variance findings of the paired-variance heat map discussed

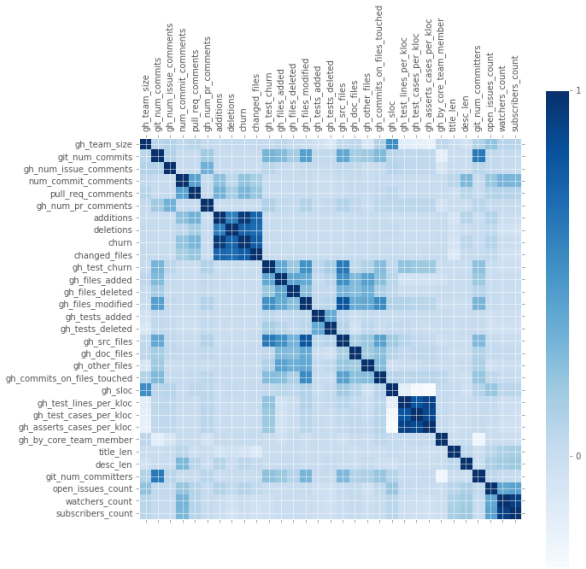

Figure 1: Heatmap representing pairwise correlation between features

previously: Most of the features with high contributions to the variances of PC1, PC2, PC3 all have strong correlations according to the heat map when one of those features is compared pair-wise with the other features, contributing to its corresponding principal component. This suggests that the two methods implied similar conclusions regarding variance. The variables contributing to the variance of P2 are again an outlier as they do not exhibit strong correlations in the heat map. PCA did not significantly reduce the dimensional data. Reducing 37 features to 30 did not allow us to prune as many features as all of the other models, so we determined it would not be useful to use PCA a predictive method. In this case though, it did return useful findings regarding variance.

4.1 Evaluation of Models

We then picked the top 10 overlapping features of the feature selection results and proceeded to the modelling phase. In this phase, we made sure to **perform a time-dependent training and test split.** This is because we wanted to achieve an accurate prediction setting we could use the historical data to predict the future data. We decided to evaluate the accuracy of the models based on *three months of data* to predict the result of the rest of the months. Having done so, we identified the target variable, *merged or not merged*, and chose the remaining 10 columns as our input variables to the models.

The models that we chose to predict the acceptance were logistic regression, kNN, Random Forests, Adaboost, and SVM. When we trained the model and evaluated them on the test sets, we obtained the following accuracies.

The results were surprising because this indicated that the models performed slightly worse or no better than predicting at random. In other words, predicting at random (similar to flipping a coin), gave us 69.87%. To further understand and evaluate the models, we also plotted the *AUC* curve for each model.

Similar to the accuracies listed above, it turned out that

Table 1: Accuracies of the Machine Learning Models

Logistic Regression	67.07%
kNN	67.55%
Random Forests	71.81%
Adaboost	71.98%
SVM count	66.36%

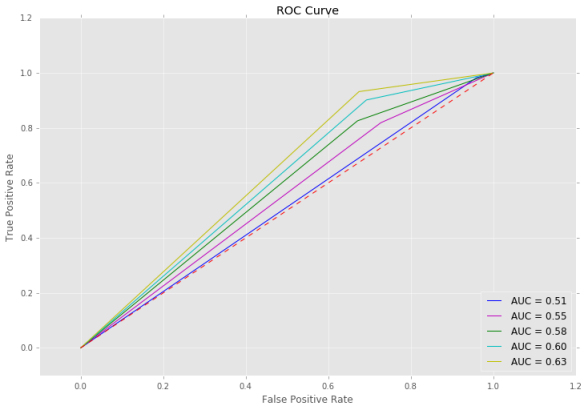

Figure 2: AUC Curves for Models

Adaboost was the best model giving us an AUC of 0.63. Yet, this is classification model performs quite poorly. We suspect the reason is that the prediction setting itself is quite challenging. In other words, pull requests and their features are quite unpredictable and the data itself is skewed in some ways, making prediction quite difficult. Therefore, the most accurate value that our classification model gives us is around 72% as seen in table 1.

After we evaluated the models, we then preceded to provide answers to the next research question. We made certain hypothesis about features such as description length, churn rate, number of commits etc. As an example, our null hypothesis was that the average description length of the pull requests that was merged were less than the ones for which the pull requests were rejected. In other words, we asked the question, can a developer make changes such as reduce the description length, churn rate, etc. such that his/her pull request will likely get accepted? We performed Wilcox tests on three or four features and examined the scenarios where the pull requests were accepted and rejected. We found that there was a significant difference and indeed supported our hypothesis. For instance, the box plot for merged vs non-merged description lengths show that the average description length for the pull requests which are merged are less than the average description length which are not merged.[1] When the Wilcox test was performed, this also confirmed our result. As a step further, we also decided to evaluate the effect size of variables and doing so gave us 0.23 for the description length which meant that the effect was small.[1] Doing so validated and answered our second question:

> A developer can indeed control features and reduce important features such as the **description length, churn rate, number of commits, and title length** such that the pull request will *likely* get accepted.

The last research question was answered by the feature selection and the machine learning phase that we performed. We found that adding the test variables such as `gh_asserts_cases_per_kloc`, `gh_test_lines_per_kloc` had little or no impact on the accuracy of the model. This was done through evaluation of features in the machine learning phase and well as analyzing the feature importance in the preprocessing phase. Therefore, we can summarize our finding for the last research question as follows:

> We found that project and pull request characteristics **are more powerful indicators** than the variables which represent the information about tests.

5. THREADS TO VALIDITY

We believe that the largest threat is the external validity of the study because of the sample size and the skewness of the data itself. We noticed that even after logging the data, some of the features's distribution seem to be extremely skewed. Furthermore, if we have a larger dataset, we might have better accuracies. Another threat to validity is "overfitting" which is an inherent issue in modeling methods: a large amount of dimensional data creates the risk that random "noise" is actually being modeled instead of an authentic relathionsip. A possible threat to the internal validity of our study is differentiating between correlation and causation. Our algorithms could highlight a true aspect of a feature but cause the reader to make incorrect assumptions. One must be careful to ensure the relationship implied by a modeled feature(s) is a causation relationship as opposed to coincidental correlation, or that even when a relationship exists, that it should be a correct and inferred relationship.

6. CONCLUSION

The main finding from this entire experiment was that pull-request prediction is a challenging task. This is due to the handling of the numerous features and the correlation of the features themselves. Furthermore, we studied the acceptance of pull-requests over a period of time and found that most of them got accepted in the order of hours and the acceptance rate constantly decreased over the period of months. We also proposed a way to show that developers can change some features such as the description length, churn, and the number of commits such that the request might be accepted. Finally, the feature extraction and modelling phase showed that the test variables were not as powerful indicators when compared to the project and pull request characteristics.

7. FUTURE WORK

We merely performed all our analyses using numerical data. It would be very interesting and beneficial if some textual analyses could be performed by reading the pull-request comments and the description of the request and find out whether it could improve the accuracy rate of the models. In addition, we only collected data over a period of 7 months since that was all that was available in the dump. Having a bigger dataset could perhaps improve the accuracy of the models.

8. REFERENCES

[1] A. Bharadwaj and C. Lock. Project Appendix. https://github.com/prudentprogrammer/Pull-Requests-Determinants, 2016. [Project Appendix].

[2] G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 368–371, New York, NY, USA, 2014. ACM.

[3] 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*, MSR 2014, pages 364–367, New York, NY, USA, 2014. ACM.

[4] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. Wait for it: Determinants of pull request evaluation latency on github. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 367–371, Piscataway, NJ, USA, 2015. IEEE Press.