Southern University of Science and Technology

Master's Thesis Proposal

Title:

An Empirical Study of the Selection Benchmarks for Multiple Pull Requests

Department _____

Discipline _____

Supervisor _____

Student Name _____

Student Number _____

Date of Proposal Report _____

Graduate school

TABLE OF CONTENTS

CHAPTER 1 Background

1.1 Inspiration

The pull-based development model [1] is widely used in the development work of distributed software teams. Taking Figure 1-1 as an example, in this model, the software contributor does not need permission to access the project code base, but he can clone the project code base directly to the local by fork operation. After cloning the code, the software contributor modifies the local project code and submits a pull request to the reviewer. If the modification is approved by the reviewer, the modification will be merged into the main branch of the project code base. If the reviewer and contributor fail to reach a consensus, one of them will close the pull request.

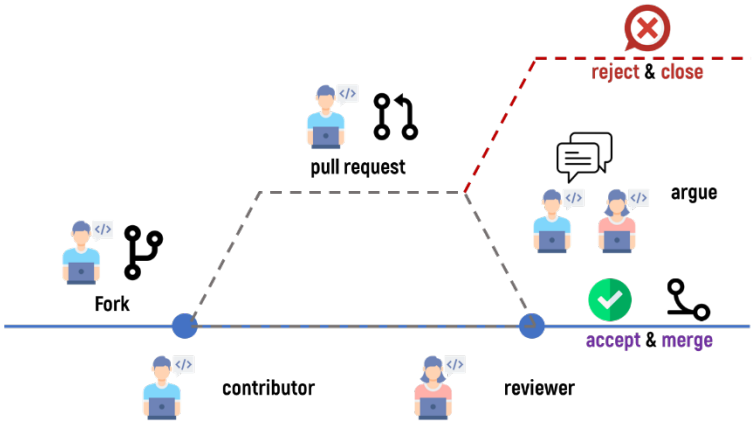
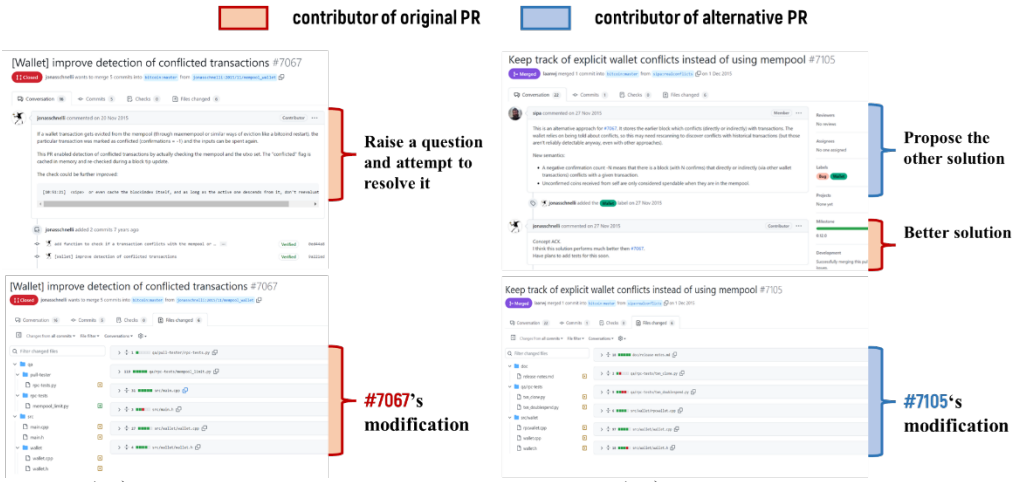


Figure 1-1 The pull request development model

There are many purposes for pull requests, such as patch submission, code review, new feature discussion, and so on [2]. However, in real work scenarios, such as the case where two software contributors propose different solutions to solve the same issue, the members of a project core team will face multiple pull requests at the same time. We call these multiple pull requests duplicate pull requests (DPR) [3]. In the case of duplicate pull requests, we call the earliest pull request the original contribution pull request (PR_{org}), and the subsequent pull requests with the same purpose alternative contribution pull requests (PR_{alt}). It is worth mentioning that there may be more than one alternative pull request.



（a）original pull request　　（b）alternative pull request

Figure 1-2　A group of duplicated pull requests

Figure 1-3　Alternative pull request is accepted

Figure 1-2 shows a group of duplicate pull requests. In Figure 1-2 (a), contributor, jonasschnelli, finds an issue on November 20, 2015. Subsequently, he uploaded a modification and upload a pull request titled, “[Wallet] improve detection of conflicted transactions #7067.” In Figure 1-2 (b), the other contributor, sipa, created a second pull request titled, “Keep track of explicit wallet conflicts instead of using mempool #7105.” In the pull request, contributor, sipa, stated that the approach proposed in the pull request was an alternative approach to pull request #7067. After comparing the solutions, contributor, jonasschnelli, felt that the solution mentioned in the alternative pull request was better than theirs. As shown in Figure 1-3, after subsequent scrutiny, the alternative pull request of the original pull request, #7067, was accepted by the project core team.

Having realized the existence of duplicate pull requests, we were inspired to incorporate comparative factors derived from the selection benchmarks among multiple pull requests into the evaluation model of contributing merge requests.

Therefore, we decided to conduct the research, “An Empirical Study of the Selection Benchmarks For Multiple Pull Requests.”

1.2 Domestic and Abroad Research Situation

In the area of the prediction model of pull requests, many researchers have put forward their research models. Gousios et al. [4] studied the factors affecting the merger decision from three aspects: pull request characteristics, project characteristics, and developer. And they proposed a random forest prediction model based on 15 influencing factors such as the number of submissions, the number of changed rows, and the number of tests. Yu et al. [5] studied the factors affecting the merger decision from three aspects: social and technique, process, and continuous integration. Then, they proposed a linear regression model based on 19 influencing factors, such as the success rate of contributors, the number of followers, the complexity of description, the availability of integrators, the Friday effect, and the results of continuous integration. Jiang et al. [6] studied the factors that affect pull requests from four aspects: code feature, text feature, contributor feature, and project feature. They proposed an extreme gradient boosting model, XGBoos, based on 17 influencing factors such as whether the text has a body, whether the text has a link and the similarity between the pull requests accepted in the last three months. Their experiments showed that the prediction results of XGBoost were better than the model of Gousios’ model, CTCPPre [4].

Hitherto, many researchers have discovered the existence of multiple pull requests in open-source software platforms such as GitHub, and therefore they studied the phenomenon from different views. Zhang et al. [8] proposed the terminology, competing pull requests—a group of contributing pull requests that modify the same code. They also pointed out that among the 60 repositories they studied, there were up to three-quarters of repositories where the ratio between the number of competing pull requests and the number of pull requests is more than 31% [8]. Tsay et al. [9] pointed out that in some cases, although the contributor’s contribution was rejected, the core team would achieve the contributor's technical goal in another way. Ma P et al. [10] used the cosine algorithm [11] and the doc2vec algorithm [12] to depict the structural similarity and semantic similarity of competing pull requests, respectively. They also found that in the pull-

based development model, there were indeed pull requests with potential competition, and the similarity between pull requests would increase the difficulty of the reviewer's decision-making process [10]. Wang et al. [13] pointed out that there were two research directions on duplicated pull requests. One is that given a specific pull request, we give a similar pull request, which we call pull request retrieval. The other one is that given a specific pull request, we determine if it is a duplicate pull request, which we call pull request classification. Additionally, they studied 10 factors from six aspects including change description, patch content, changed files list, location of code changes, reference to the issue tracker, and time. At last, they apply the machine learning method, AdaBoost [14], to the problem, pull request classification. Based on their former work [16] where they only considered textual similarity between pull requests, Li et al. [15] proposed a new method combining textural similarity and change similarity to enhance the performance of the pull request retrieval model. Their experiments showed that compared with the old method which could distinguish about 54.8% of duplicate pull requests, the new method could distinguish 83.4% of duplicate pull requests on average.

1.3 Purpose and Significance

1.3.1 Purpose

Through the study, we will achieve the following objectives: (1) Study the phenomenon of multiple pull requests, and understand its basic situation of some open source software platform projects; (2) Make the dataset of duplicate pull requests; (3) Summarize the selection benchmarks between the original pull requests and alternative pull requests from the dataset; (4) Considering summarized factors derived from the selection benchmarks and factors influencing the merging decision of single pull request, we propose an evaluation model for multiple pull requests.

1.3.2 Significance

Through the study, we will have a deeper understanding of the phenomenon of multiple pull requests and the decision-making factors between the original

contribution merge request and the alternative contribution merge request. At the same time, compared with the evaluation model that only considers a single pull request, the model proposed in this study incorporates the interaction factors between multiple pull requests into the evaluation model to fully depict the process of pull request decisions. Combined with machine learning, the evaluation model proposed in this paper will better serve the reviewer's decision-making process by valuing pull requests in the pull-based development model.

CHAPTER 2 Research Content and Proposal

2.1 Research Content

We will conduct our study from these five aspects: distribution study of multiple pull requests, selection benchmark exploration among multiple pull requests, dataset collection, model construction for multiple pull requests, and model comparison.

2.2 Research Proposal

According to what we said in chapter 2.1, we decide to divide the whole study process into five parts including multiple pull requests collection, selection benchmarks exploration, dataset collection, model construction, and model comparison.

2.2.1 Multiple Pull Requests Collection

In this session, we will select some open-source projects in GitHub and use automatic crawl and manual filters simultaneously to collect many groups of original pull requests and alternative pull requests.

- *"dup of #xxxx"*
- *"Closed by https://github.com/rails/rails/pull/13867"*
- *"This has been addressed in #27768."*

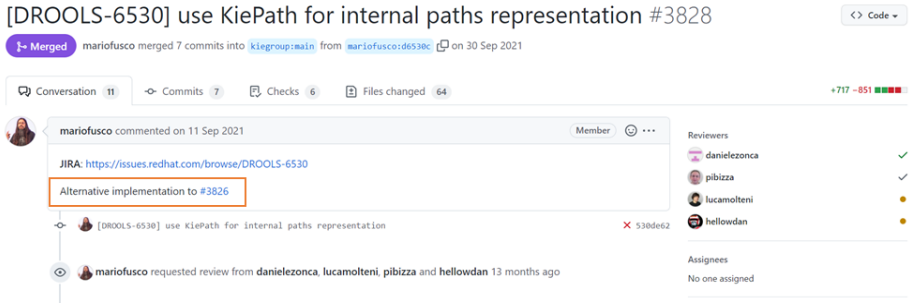
Figure 2-1 The link directing to the other pull request

Li et al.[16] mentioned, "When reviewers are commenting on a pull request, they mention other related pull requests by leaving a link to them." And he gave the example shown in Figure 2-1. However, via our observation, we found a case where the reviewer also left a link directing to another pull request to express, "The pull request is the alternative pull request of a pull request." As shown in Figure 2-2 (a), contributor, mariofusco, stated that the pull request was the alternative implementation to pull request #3826. In Figure 2-2 (b), contributor, anthonyvdotbe, said, "This is alternative (and IMHO) better approach than #1024." From the phenomenon, we deemed that whatever the people attaching the link is contributor or reviewer, the pull request containing a link directing to another

pull request is more likely to be an alternative pull request, and in turn, the directed pull request is more likely to be the original pull request.

Based on our observation, in order to collect more multiple pull requests as many as possible and lessen our workload, we will filter pull requests with the crawl principle whether there is a link directing to another pull request. shows the workflow of our crawler.

Considering that a pull request directing to another pull request may not be the one we desire, after using the crawler to get data, we need to screen out illegal pull requests manually. Then, we obtain a collection of duplicated pull requests. By analyzing the collection of pull requests, we can obtain some information, such as the proportion of duplicated pull requests for each project and the average proportion of duplicated pull requests per language.



（a）kiegroup/drools/pull/3828

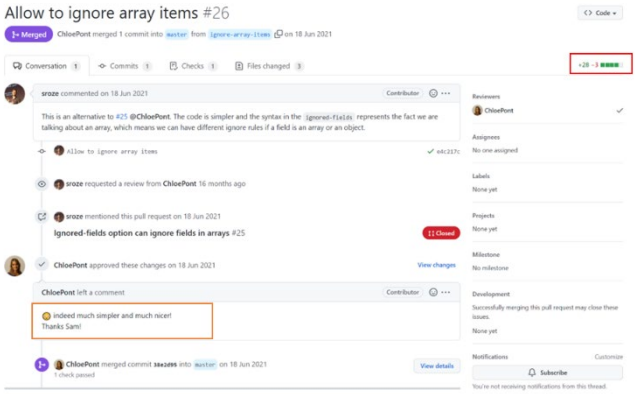
（b）neo4j/neo4j-java-driver/pull/1029

Figure 2-2 A reviewer left a link directing to another pull request

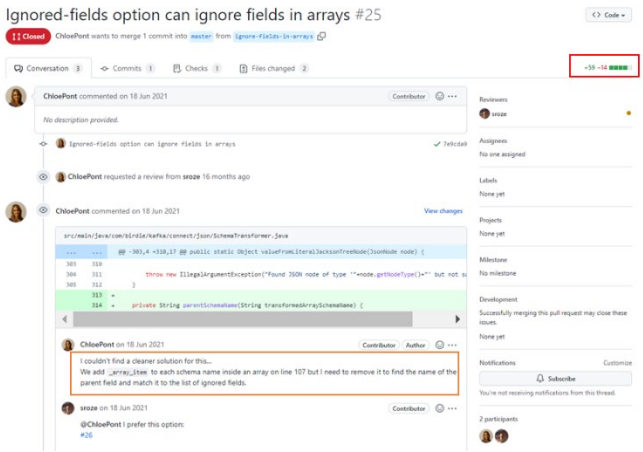
2.2.2 Selection Benchmarks Exploration

In this section, we will explore the selection benchmarks among multiple

pull requests. Based on the definition of multiple pull requests, we know that facing multiple pull requests, for different reasons, such as better readability, successful testing, simplification, and so on, reviewers will decide to merge only one pull request.



（a）birdiecare/connect-smts/pull/26



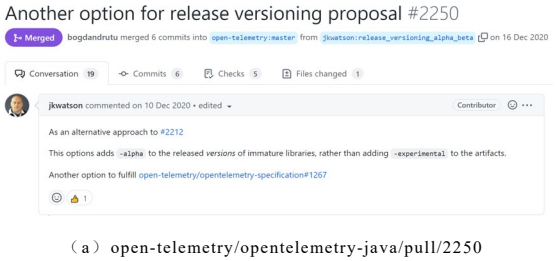
（b）birdiecare/connect-smts/pull/25

Figure 2-3　Simplification selection benchmark

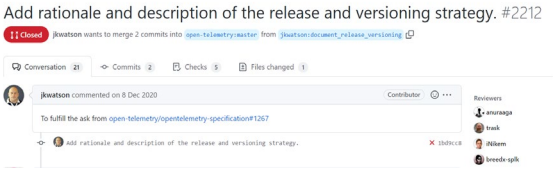
Figure 2-3 shows the case whose selection benchmark is simplification. In Figure 2-3 (b), contributor, ChloePont, submitted a pull request, #25, where she modified 73 lines (53 added lines and 14 deleted lines) mentioning that she had not found any cleaner solution. In Figure 2-3 (a), contributor, sroze, submitted a pull request, #26, where he modified 26 lines (28 added lines and 3 deleted lines) mentioning that the alternative pull request was simpler. Latter, ChloePont felt that the solution was so much simpler and nicer. Finally, ChloePont decided to accept pull request, #26, and close pull request #25.

Figure 2-4 shows the case whose selection benchmark is readability. Figure

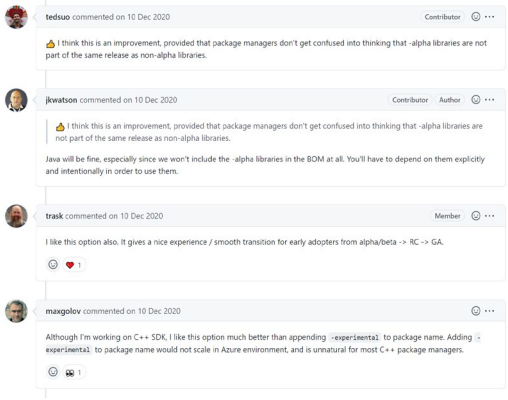
2-4 (a) and (c) shows original pull request #2250 and alternative pull request #2212 respectively. In alternative pull request #2250, contributor, jkwatson, deemed that instead of adding "-edxperimental" to the artifacts, they should added "-alpha". In Figure 2-4, members in project core team approved jkwatson's idea and they thought that the idea was helpful for others' understanding.



（a）open-telemetry/opentelemetry-java/pull/2250



（c）open-telemetry/opentelemetry-java/pull/2212



（b）comments in #2250

Figure 2-4　Readability selection benchmark

2.2.3 Data Collection

Having determined the selection benchmarks among multiple pull requests, we will pick up a set of features $\boldsymbol{F_1}$ by depicting the selection benchmarks. Simultaneously, we will also pick up the other set of features $\boldsymbol{F_2}$ by depicting the decision process of a single pull request. Taking the union set of $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$, we will get the feature set $\boldsymbol{F}$ for multiple pull requests.

Using the links of multiple pull requests, we use a crawler to get raw information used for calculating features in the feature set $\boldsymbol{F}$. Then, based on the transformed pattern of each feature in set $\boldsymbol{F}$, we get features for depicting the process of multiple pull request decisions.

Take the selection benchmark, simplification, as an example. Simplification means that the code of the solution provided by the pull request is simpler than others. In order to reflect the selection benchmark, we decide to characterize it from two angles: the number of code modifications and code similarity. Because the number of multiple pull requests might be more than two, we designate the original pull request as the compared standard. We assume that we have a pair of multiple pull requests, PR_1 and PR_2. PR_1 is the original pull request while

PR_2 is the alternative pull request. From the angle of the number of code modifications, we put forward the code modification ratio (CMR). Here are the calculation formulae:

$$CMR_i = \frac{m_i}{m_{\text{org}}} \tag{2-1}$$

$$m_i = add_i + del_i \tag{2-2}$$

$$m_{org} = add_{org} + del_{org} \tag{2-3}$$

, where CMR_i is the code modification ratio between i-th alternative pull request and original pull request, and m is the number of modified code lines. add is the number of added lines, and del is the number of deleted lines.

From the angle of code similarity, based on the work of Li et.al. [14] for change similarity, we use following method to depict the code similarity between the original pull request and alternative pull request. Here is the formula:

$$CS_i = 0.5 \times S_{file}(PR_{org}, PR_i) + 0.5 \times S_{code}(PR_{org}, PR_i) \tag{2-4}$$

, where CS_i is the code similarity between the original pull request and i-th alternative pull request, and the function $S_{file}(\cdot)$ returns the similarity between two sets of modified files, and the function $S_{code}(\cdot)$ returns the average similarity of added code modification.

As you can see, the calculation of CMR is divided into two parts – one is the similarity between two sets of modified files, and the other one is the average similarity of added code modification. Figure 2-5 and Figure 2-6 shows the pseudocode of the function $Sfile(\cdot)$ and $Scode(\cdot)$ respectively. It is worth mentioning that we modified the pseudocodes of Li et.al. [14] to make them suitable for our study.

In $Sfile(\cdot)$, the input includes original pull request PR_{org} and alternative pull request PR_{alt}, and the output is the similarity between two sets of modified files. In the pseudocode, we extract all sets of modified files, F_{org} and F_{alt}, from PR_{org} and PR_{alt}. Then, we travel through all pairs of modified files (f_i, f_j) iteratively, and we use the function $FilePathSim(\cdot)$ to calculate the similarity f_{sim} between two file paths, and then we store the tuple (f_i, f_j, f_{sim}) into the list $fpairs\_sim$. We sort the list $fpairs\_sim$ in the descending order of f_sim, and we get to the top tuple top_tuple from sorted list $fpairs\_sim$ and put

$top\_tuple[2]$ into the list top_sim one by one. It is worth mentioning that every time we get the tuple with highest f_sim, we should delete tuples that meet the condition $f_i == top\_tuple[0] \mid f_j == top\_tuple[1]$. Additionally, we should find top tuple $\min\left(Len(F_{org}),\ Len(F_{alt})\right)$ times. Finally, we return the average value of top_sim.

Algorithm 1 Calculating Similarity Between Two Sets Of Modified Files

```
1: procedure SFILE(PR_org, PR_alt)
2:     initialize two sets of paths of modified files F_org, F_alt with PR_org and
   PR_alt respectively
3:     len_org ← the number of elements in F_org
4:     len_alt ← the number of elements in F_alt
5:     initialize fpairs_sim to an empty list
6:     for all f_i ∈ F_org do
7:         for all f_j ∈ F_alt do
8:             fsim ← FilePathSim(f_i, f_j)
9:             Add tuple (f_i, f_j, fsim) to the list fpairs_sim
10:        end for
11:    end for
12:    Sort fpair_sim in the desending order of the property fsim
13:    initialize top_sim to an empty list
14:    top_n ← min (len_org, len_alt)
15:    while top_n > 0 do
16:        get the tuple top_tuple with highest fsim from list top_sim
17:        Add top_tuple[2] to the list top_sim
18:        Delete all tuples satisfying the condition (f_i == top_tuple[0] or f_j ==
   top_tuple[1]) in fpairs_sim
19:        top_n − −
20:    end while
21:    sum ← the sum of top_sim
22:    return sum/top_num
23: end procedure

24: procedure FILEPATHSIM(f_i, f_j)
25:    p_i ← the f_i's path relative to the root path of the project
26:    p_j ← the f_j's path relative to the root path of the project
27:    split p_i by file separator and store elements in elem_i
28:    split p by file separator and store elements in elem_j
29:    pos ← 0
30:    while pos < Len(elem_i) or pos < Len(elem_j) do
31:        if elem_i[pos] ≠ elem_j[pos] then
32:            break
33:        end if
34:        pos++
35:    end while
36:    return pos/ max (Len(elem_i), Len(elem_j))
37: end procedure
```

Figure 2-5 Pseudocode of function $Sfile(\cdot)$

In $Scode(\cdot)$, the input includes original pull request PR_{org} and alternative pull request PR_{alt}, and the output is the average similarity of added code modification. First, we extract all sets of modified files, F_{org} and F_{alt}, from PR_{org} and PR_{alt}. Then, we get the intersection F_{int} of F_{org} and F_{alt}. n_{int} is the number of elements in F_{int}. Traveling through F_{int}, we use function $AddedLines(\cdot)$ to extract added lines of file f_i of a given pull request. Next, we travel through all pairs of add_lines_{org} and add_lines_{alt}. For a pair of add lines, $(line_i, line_j)$, we use function $GetToken(\cdot)$ to split $line_i$ and $line_j$ into the set of symbol set $token_i$ and $token_j$, and we calculate the similarity between two tokens, tkn_sim, and then add the tuple $(token_i, token_j, tkn\_sim)$ to the list

tkn_pairs_sim. We sort tkn_pairs_sim in the descending order of tkn_sim, and we get top tuple top_tuple from sorted list $fpairs\_sim$ and put $top\_tuple[2]$ into the list top_sim one by one. It is worth mentioning that every time we get the tuple with highest tkn_sim, we should delete tuples that meet the condition $token_i == top\_tuple[0] \mid token_j == top\_tuple[1]$. Then, we add the average value of top_sim to list $code\_sim$. Finally, at the end of the iteration, we return the average value of $code\_sim$.

Algorithm 2 Calcualting Average Similarity Of Added Code Modification

```
1: procedure SCODE(PR_org, PR_alt)
2:     initialize two sets of paths of modified files F_org, F_alt with PR_org and
   PR_alt respectively
3:     F_int ← F_org ∩ F_alt
4:     n_int ← the number of elements in F_int
5:     initialize code_sim to an empty list
6:     for all f_i ∈ F_int do
7:         add_lines_org ← AddedLines(PR_org, f_i)
8:         add_lines_alt ← AddedLines(PR_alt, f_i)
9:         n_org ← the number of elements in add_lines_org
10:        n_alt ← the number of elements in add_lines_alt
11:        initialize tkn_pairs_sim to an empty list
12:        for all line_i ∈ add_lines_org do
13:            for all line_j ∈ add_lines_alt do
14:                token_i ← GetTokens(line_i)
15:                token_j ← GetTokens(line_j)
16:                tkn_sim ← Len(token_i ∩ token_j)/Len(token_i ∪ token_j)
17:                Add tuple (token_i, token_j, tkn_sim) to tkn_pairs_sim
18:            end for
19:        end for
20:        Sort tkn_pairs_sim by tkn_sim
21:        top_n ← min (n_org, n_alt)
22:        tmp ← top_n
23:        initialize top_sim to an empty list
24:        while top_n > 0 do
25:            get the tuple top_tuple with highest tkn_sim from list
   tkn_pairs_sim
26:            Add top_tuple[2] to the list top_sim
27:            Delete all tuples satisfying the condition (token_i == top_tuple[0]
   or token_j == top_tuple[1]) in tkn_pairs_sim
28:            top_n − −
29:        end while
30:        sum_topn ← the sum of top_sim
31:        Add sum_topn/tmp to the list code_sim
32:    end for
33:    sum ← the sum of code_sim
34:    return sum/n_int
35: end procedure

36: procedure ADDEDLINE(PR, f)
37:     return a list of all added lines of file f in pull request PR
38: end procedure

39: procedure GETTOKEN(line)
40:     replace each punctuation in string line with whitespace symbol
41:     split line by whitespace and store elements in set token
42:     return token
43: end procedure
```

2

Figure 2-6 Pseudocode of function $Scode(\cdot)$

2.2.4 Model Construction

(1) Correlation analysis

In the section, due to the fact that the types of features vary from one to another, we will use different methods to analyze the correlation between each feature and the status of the merge decision:

(a) If the feature is a continuous variable

For continuous variable, such as the number of modifications, we will use point-biserial correlation coefficient [17]. We assume that we have continuous variable X and dichotomous variable Y. Here are formulae for calculating point-biserial correlation coefficient r_{XY} between X and Y:

$$r_{XY} = \frac{M_1 - M_0}{\sigma}\sqrt{\frac{n_1 n_0}{n}} \tag{2-5}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2} \tag{2-6}$$

, where M_0 is the average value on the continuous variable X for all data points in group 0 whose data point's Y value is equal to 0, and M_1 is the average value on the continuous variable X for all data point in group 1 where data point's Y value is equal to 1, and n_0 and n_0 are the number of data points in group 1 and group 2 respectively, and n is the total number of all data points, and σ is the standard deviation of X.

(b) If the feature is a dichotomous variable

For dichotomous variables, such as the condition if there is testing, we will use the odds ratio [18]. We assume that we have a dichotomous variable X and a dichotomous variable Y. In order to clearly demonstrate the odds ratio we use the 2×2 Table 2-1 shows the distribution of X and Y.

Table 2-1 The distribution of X and Y (sample 1)

X \ Y	0	1
0	n_{00}	n_{01}
1	n_{10}	n_{11}

Here are formulae for calculating the odds ratio t r_{XY} between X and Y:

$$r_{XY} = \frac{n_{00} \times n_{01}}{n_{10} \times n_{11}} \tag{2-7}$$

, where n_{ab} is the number of data points satisfying the condition that X is equal to a and Y is equal to b.

(c) If the feature is an unordered categorical variable

For unordered categorical variables, such as the identity of the developer,

we will use Pearson's chi-square test [20] and Cramér's V [21] to obtain analyze the statistical significance of the correlation and the strength of the correlation respectively. So as to demonstrate the calculation clearly, we use the $r \times c$ Table 2-2 shows the distribution of X and Y.

Table 2-2 The distribution of X and Y (sample 2)

X \ Y	y_1	…	y_c
x_1	v_{11}	…	v_{1c}
…	…	…	…
x_r	v_{r1}	…	v_{rc}

Here are formulae about Pearson's chi-square test:

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{\left(v_{ij} - E_{ij}\right)^2}{E_{ij}} \tag{2-8}$$

$$E_{ij} = \frac{\left(\sum_{a=1}^{c} v_{ia}\right) \times \left(\sum_{b=1}^{r} v_{bj}\right)}{N} \tag{2-9}$$

$$N = \sum_{i=1}^{r}\sum_{j=1}^{c} v_{ij} \tag{2-10}$$

$$df = (r-1) \times (c-1) \tag{2-11}$$

, where χ^2 is Pearson's chi-square statistic, E_{ij} is the theoretical number under an independent assumption, and N is the grand total of observations, and df is the degree of freedom. After obtaining Pearson's chi-square statistic X^2 and the degree of freedom, we consult the chi-square probability table, part of which is shown in Figure 2-7, to determine how the statistical significance of the correlation between two variables.

	Area in the Right Tail											
	0.999	0.995	0.990	0.975	0.950	0.900	0.100	0.050	0.025	0.010	0.005	0.001
Degrees of Freedom												
1	0.000	0.000	0.000	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879	10.828
2	0.002	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597	13.816
3	0.024	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838	16.266
4	0.091	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860	18.467
5	0.210	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750	20.515
6	0.381	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548	22.458
7	0.598	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278	24.322
8	0.857	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955	26.124

Figure 2-7 Partial chi-square probability table

Although we can know the significance of the correlation between two variables, we still don't know how strong the correlation is. Therefore, we need Cramér's V to show strength. Here is formula about Cramér's V.

$$V = \sqrt{\frac{\chi^2/N}{\min(r-1, c-1)}} \tag{2-12}$$

, where χ^2 is Pearson's chi-square statistic, N is the grand total of observations, r is the number of columns, and c is the number of rows.

(2) Method selection, training, and evaluation

After eliminating redundant features with low correlation, in this section, we adopt four machine learning methods to our study. They are support vector machine [21], naïve Bayes [22], random forest [23], and extreme gradient boosting [24]. Additionally, accuracy (ACC), precision and recall, and F1-score will be used to measure the performance of our model. Table 2-3 shows the distribution of predicted results of a model. Here are formulae about these indicators.

Table 2-3 The distribution of predicted results

Ture class / Hypothesized class	POSITIVE	NEGATIVE
YES	n_{TP}	n_{FP}
NO	n_{FN}	n_{TN}

Here are some formulae used for calculating accuracy, precision and recall, and F1-score.

$$ACC = \frac{(n_{TP} + n_{TN})}{n_{TP} + n_{FP} + n_{FN} + n_{TN}} \quad (2\text{-}13)$$

$$Precision = \frac{n_{TP}}{n_{TP} + n_{FP}} \quad (2\text{-}14)$$

$$Recall = \frac{n_{TP}}{n_{TP} + n_{FN}} \quad (2\text{-}15)$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (2\text{-}16)$$

2.2.5 Model comparison

To evaluate the performance of our evaluation model of multiple pull requests, we choose the random forest model RFPre proposed by Gousios et al. [4], the linear regression model LRPre proposed by Yu et al. [5], and the XGBoost model CTCPPre proposed by Jiang et al. [6]. We value the performance of each model with indicators mentioned in 2.2.4 Model Construction.

CHAPTER 3 Prospective Objective

3.1 Prospective Objective

We have three prospective objectives: (1) Analyze the situation of multiple pull requests and prove the prevalence of multiple pull requests; (2) Determine the selection benchmarks of multiple pull requests and summarize descriptive factors of multiple pull requests; (3) Obtain the best evaluation model for multiple pull requests.

CHAPTER 4 Current Work and Schedule

4.1 Current Work

Hitherto, we have completed the following tasks: (1) Through reading papers whose topics are related to multiple pull requests, we have formed an initial understanding of our study and drafted a detailed plan to carry out our study; (2) Based on crawler technology and manual screening, we have obtained about 378 groups of multiple pull requests, and by scrutinizing all multiple requests, we have summarized some selection benchmarks.

4.1.1 Schedule

Table 4-1 shows the schedule of our study.

Table 4-1 The schedule of our study

Period	Work Content
October 2022 – November 2022	Collect multiple pull requests
November 2022 – December 2022	Scrutinize multiple pull requests and summarize selection benchmarks
December 2022 – January 2023	Depict selection benchmarks and get factors of the machine learning model
January 2023 – February 2023	Make a dataset for multiple pull requests
February 2023 – March 2023	Write codes and evaluate models
March 2023 – April 2023	Tidy experimental data and write papers

CHAPTER 5 Requirement and Funding

5.1 Requirement

In order to finish our study successfully, we need a server for running our programs.

5.1.1 Funding

We do not need extra funding.

CHAPTER 6 Difficulty and Solution

6.1 Difficulty

We estimate that we may come across the following three difficulties: (1) A large amount of sterile manual screening tasks; (2) Unreadable pull requests; (3) an underperforming model.

6.1.1 Solution

For difficulty (1), we should read more papers about multiple pull requests detection and enhance the selection accuracy of our crawler to lower our manual burden.

Because individual experience and depth of understanding are different from person to person. For difficulty (2), to get a better understanding of a pull request, we can communicate with others about it and reach a consensus.

Many aspects, such as bad feature design, overfitting, underfitting, unsuitable model, etc., can influence the overall performance of our model. For difficulty (3), we should redesign features, or find a more suitable learning method.

REFERENCES

[1] Barr E T, Bird C, Rigby P C, et al. Cohesive and isolated development with branches[C]//International Conference on Fundamental Approaches to Software Engineering. Springer, Berlin, Heidelberg, 2012: 316-331.

[2] Gousios G, Zaidman A, Storey M A, et al. Work practices and challenges in pull-based development: The integrator's perspective[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 358-368.

[3] Li Z, Yu Y, Zhou M, et al. Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects[J]. IEEE Transactions on Software Engineering, 2020.

[4] Gousios G, Pinzger M, Deursen A. An exploratory study of the pull-based software development model[C]//Proceedings of the 36th international conference on software engineering. 2014: 345-355.

[5] Yu Y, Wang H, Filkov V, et al. Wait for it: Determinants of pull request evaluation latency on github[C]//2015 IEEE/ACM 12th working conference on mining software repositories. IEEE, 2015: 367-371.

[6] Jiang J, Zheng J, Yang Y, et al. CTCPPre: A prediction method for accepted pull requests in GitHub[J]. Journal of Central South University, 2020, 27(2): 449-468.

[7] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[8] Zhang X, Chen Y, Gu Y, et al. How do multiple pull requests change the same code: A study of competing pull requests in github[C]//2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2018: 228-239.

[9] Tsay J, Dabbish L, Herbsleb J. Let's talk about it: evaluating contributions through discussion in GitHub[C]//Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering. 2014: 144-154.

[10] Ma P, Xu D, Zhang X, et al. Changes Are Similar: Measuring Similarity of Pull Requests That Change the Same Code in GitHub[M]//Software Engineering and Methodology for Emerging Domains. Springer, Singapore, 2017: 115-128.

[11] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 14-24.

[12] Le Q, Mikolov T. Distributed representations of sentences and documents[C]//International conference on machine learning. PMLR, 2014: 1188-1196.

[13] Wang Q, Xu B, Xia X, et al. Duplicate pull request detection: When time matters[C]//Proceedings of the 11th Asia-Pacific Symposium on Internetware. 2019: 1-10.

[14] Freund Y, Schapire R E. A decision-theoretic generalization of on-line learning and an application to boosting[J]. Journal of computer and system sciences, 1997, 55(1): 119-139.

[15] Li Z X, Yu Y, Wang T, et al. Detecting duplicate contributions in pull-based model combining textual and change similarities[J]. Journal of Computer Science and Technology, 2021, 36(1): 191-206.

[16] Li Z, Yin G, Yu Y, et al. Detecting duplicate pull-requests in github[C]//Proceedings of the 9th Asia-Pacific Symposium on Internetware. 2017: 1-6.

[17] Tate R F. Correlation between a discrete and a continuous variable. Point-biserial correlation[J]. The Annals of mathematical statistics, 1954, 25(3): 603-607.

[18] Bland J M, Altman D G. The odds ratio[J]. Bmj, 2000, 320(7247): 1468.

[19] Plackett R L. Karl Pearson and the chi-squared test[J]. International statistical review/revue internationale de statistique, 1983: 59-72.

[20] Cramér H. Mathematical methods of statistics[M]. Princeton university press, 1999.

[21] Suthaharan S. Support vector machine[M]//Machine learning models and algorithms for big data classification. Springer, Boston, MA, 2016: 207-235.

[22] Webb G I, Keogh E, Miikkulainen R. Naïve Bayes[J]. Encyclopedia of machine learning, 2010, 15: 713-714.

[23] Rigatti S J. Random forest[J]. Journal of Insurance Medicine, 2017, 47(1): 31-39.

[24] Chen T, He T, Benesty M, et al. Xgboost: extreme gradient boosting[J]. R package version 0.4-2, 2015, 1(4): 1-4.