Duplicate Pull Request Detection: When Time Matters

Qingye Wang
College of Computer Science and Technology, Zhejiang University
Hangzhou, China
Ningbo Research Institute, Zhejiang University
Ningbo, China
wqyy@zju.edu.cn

Bowen Xu
School of Information Systems, Singapore Management University
Singapore
bowenxu.2017@phdis.smu.edu.sg

Xin Xia
Faculty of Information Technology, Monash University
Melbourne, Australia
xin.xia@monash.edu

Ting Wang
College of Computer Science and Technology, Zhejiang University of Technology
Hangzhou, China
wangting@zjut.edu.cn

Shanping Li
College of Computer Science and Technology, Zhejiang University
Hangzhou, China
shan@zju.edu.cn

ABSTRACT

In open source communities (e.g., GitHub), developers frequently submit pull requests to fix bugs or add new features during development process. Since the process of pull request is uncoordinated and distributed, it causes massive duplication. Usually, only the first pull request qualified by reviewers can be merged to the main branch of the repository, and the others are regarded as duplication by maintainers. Since the duplication largely aggravates workloads of project reviewers and maintainers, the evolutionary process of open source repositories is delayed. To identify the duplicate pull requests automatically, Ren et al. proposed a state-of-the-art approach that models a pull request by nine features and determine whether a given request is duplicate with the other existing requests or not. Nevertheless, we notice that their approach overlooked the time factor which is a significant feature for the task. In this study, we investigate the influence of time factor and improve the pull request representation. We assume that two pull requests are more likely duplicate when their created time are close to each other. We verify the assumption based on 26 open source repositories from GitHub with over 100,000 pairs of pull requests. We integrate the time feature to the nine features proposed by Ren et al. and the experimental results show that it can substantially improve the performance of Ren et al.'s work by 14.36% and 11.93% in terms of F1-score@1 and F1-score@5, respectively.

CCS CONCEPTS

• **Software and its engineering** → Software maintenance.

KEYWORDS

GitHub, Duplicate Pull Request, Time Factor

ACM Reference Format:
Qingye Wang, Bowen Xu, Xin Xia, Ting Wang, and Shanping Li. 2019. Duplicate Pull Request Detection: When Time Matters. In *Internetware '19: Proceedings of the 11th Asia-Pacific Symposium on Internetware (Internetware '19), October 28–29, 2019, Fukuoka, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3361242.3361254

1 INTRODUCTION

Pull requests are commonly used by developers or teams to collaborate on their development work. Every contributor can fork or clone the repositories and make their changes independent with each other. Many open source projects on GitHub use pull requests to manage changes from contributors. They are quite useful for providing a mechanism to inform project maintainers about changes that contributors have made. Also, they could be used to initiate code review and general discussion on a series of changes before being merged into the main branch. Gousios et al. [11] have illustrated the popularity of pull request development model. They concluded that pull request is a significant mechanism for distributed software development. Many projects are growing fast with a large number of pull requests, e.g., *Rails*[1], which has received more than 23,000 pull requests.

Despite the benefit of pull request process, it in nature is an uncoordinated and distributed process. It is difficult to maintain an overview of what changes in individual pull request because contributors can make any necessary modifications [5, 8, 9, 37]. Developers could simultaneously submit pull requests implementing the same functionality. These pull

[1]https://github.com/rails/rails/pulls

requests are termed as *duplicate pull requests*. Much effort is wasted in open source projects due to duplicate development. It significantly increases the maintenance effort for project maintainers [8, 30]. Yu et al. [41] analyzed pull requests from 26 prevalent projects on GitHub. They found that on average 2.5 reviewers participated in review discussions of duplicate pull requests and 5.2 review comments were generated before the duplicate relation was identified. Moreover, duplicate development is common in the open source projects. Gousios et al. [11] summarized nine reasons why pull requests were rejected with 290 projects on GitHub, and found that 23% of pull requests were rejected because of duplicate development (including parallel development and superseding other pull requests). Wang et al. [38] conducted an empirical study to investigate why code changes were abandoned with four prevalent open source projects on Gerrit, which is commonly used for code review. They found that about 40% code changes were abandoned due to duplicate development. Thus, it is necessary to detect duplicate pull requests which will help project maintainers to decrease workload of processing duplicate pull requests.

To address the problem of duplicate pull requests, there are two threads of work as follows:

- **pull request retrieval:** given a new pull request, return other pull requests that are similar to it.
- **pull request classification:** given a new pull request, classify whether it is a duplicate pull request or not.

For the first thread, we refer to it as *pull request retrieval* problem as the task is similar to the retrieval of similar document from a corpus. Li et al. [22] proposed an approach to identify duplicate pull requests among history pull requests. They calculated the textual similarity between a pair of pull requests through title and description. At last, they got the top-K duplicate pull requests among existing ones by ranking them with arithmetic average of the two similarity value of title and description.

For the second thread, we refer to it as *pull request classification* problem as the task is to assign one out of the two labels (i.e., duplicate or not) to pull requests. Ren et al. [27] extracted nine features of pull requests where title and description are both included. For the nine features, they calculated their similarity respectively and adopted a machine learning algorithm to aggregate the nine features. At last, they got a specific result whether a pull request is duplicate or not. Ren et al. [27] compared their approach to Li et al.'s work [22], and achieved better results by 16-21% recall-rate@k. To our best knowledge, Ren et al.'s work is the state-of-the-art approach.

However, we observed two limitations of Ren et al.'s work. First, the evaluation methodology of Ren et al.'s work is extremely hard to follow which brings barriers to the following works. It takes huge manual effort to label for all of the pull requests for testing. To address this, we constructed a dataset that can be used to evaluate the approach automatically. Second, we observed that time factor plays a significant role to indicate whether two pull requests are duplicate or not. However, it is not considered in Ren et al's work.

In this paper, we aim to investigate how time affects duplicate pull request detection. For the pull requests in our training dataset, we extract ten features (i.e., the combination of time feature and the nine features proposed by Ren et al.). Then, we train a classifier with these features. Next, for new pull requests, we extract their features and take them as input into the classifier. The output is whether the pull requests are duplicate. We have experimented our approach based on Ren et al.'s work. It shows that our approach could outperform the study by Ren et al..

Our contributions are as follows:

(1) We proposed a new feature in terms of time for duplicate pull request detection, and found that it is a significant indicator to detect duplicate pull requests.
(2) We integrated the new feature into the existing features proposed by [27], and we found that the new feature boosted 14.36% and 11.93% improvements over the state-of-the-art approach in terms of F1-score@1 and F1-score@5, respectively.

The remainder of this paper is organized as follows. In Section 2, we introduce the background and motivation. In Section 3, we present our approach. In section 4, we describe our experiments. We present related work in Section 5. We discuss the implications and describe threats to validity in Section 6. We conclude our work and present future work in Section 7.

2 BACKGROUND AND MOTIVATION

2.1 Pull Request Process

Pull request as a distributed development model, and as implemented by GitHub in particular, has formed a new approach for collaboration in distributed software development. An outstanding characteristic of pull request development is that it allows developers to clone any public repository. The clone creates a repository that belongs to the developers. The developers can modify the repository without being part of the development team.

A typical pull request process on GitHub involves the following steps. First of all, contributors could find an attractive project by following some well-known developers and watching their projects. Secondly, by forking a repository, contributors add a new feature or fix a bug on their cloned repository. Thirdly, the contributors send the patches they modify from the forked repository to its source database by a pull request. Fourthly, all developers in the community have access to review the pull request. They can discuss and give some comments whether the pull request meets the standard or whether the project needs the feature implemented in the pull request or how to further improve the quality of the pull request and so on. Next, contributors would improve and update the pull request according to reviewers' suggestions by attaching new commits. Then, reviewers discuss the pull request again. Finally, a responsible manager of the core team takes all the view of reviewers into account, and then make a decision to merge or reject the pull request.

2.2 Ren et al.'s Study

To our best knowledge, the state-of-the-art approach is proposed by Ren et al. [27]. Ren et al. considered the same problem as ours and they proposed an approach to detect duplicate pull requests, which we used as baseline in this work. In their study, nine features included in five dimensions were proposed. The features include:

1) Change description: It is a summary of the pull requests written in natural language. It contains title and description. Ren et al. calculated the similarity for title and description respectively by Latent Semantic Indexing (LSI) [19] and cosine similarity techniques. They made the two similarity values as two features.

2) Patch content: It is the differences of text changes in every file by running 'git diff' command. Ren et al. extracted representative keywords from each pull request because they found developers often share keywords when defining variables and functions. Finally, they calculated the similarity by Vector Space Model (VSM) and cosine similarity, and made the similarity value as a feature.

However, when a pull request is duplicate with only a subset of another pull request, the similarity is small, which makes it difficult to identify duplicate pull requests. To solve the problem, they calculated the similarity of patch content only on overlapping files, and made the similarity value as another feature.

3) Changed files list: It contains all the changed files in a pull request. They operationalized the similarity of two lists of files into calculating the overlap between two sets of changed files by using Jaccard similarity coefficient. They made the similarity value as a feature.

Moreover, for changed files, in case that one pull request is much bigger than the other, which results in a small ratio of overlapping files, they made the number of overlapping files as another feature.

4) Location of code changes: It is a range of changed lines in the corresponding changed files. Ren et al. calculated the similarity of location by comparing the size of overlapping code blocks between a pair of pull requests. A pair of pull requests are more similar if they have more overlapping blocks. They defined the *Location similarity* as the length of overlapping blocks divided by length of all the blocks, and made the similarity value as a feature.

Besides, in order to catch duplicate pull requests between big and small size in file level, they calculated the similarity of pull request location for only overlapping files, and made it as another feature.

5) References to issue tracker: It is a common practice in which developers explicitly link a change to an existing issue or feature request in the issue tracker. In general, the pair of pull requests are likely to be duplicate if they reference the same issue. Hence, Ren et al. made the reference to issue tracker as a feature. They defined three possible values for this feature: if the references link to different issues, the value is -1; if they link to the same issue, the value is 1; otherwise it is 0.

With the nine features included in five dimensions as shown in Table 1 (the first nine features), Ren et al. used machine learning method to train a model. After conducting a preliminary experimental study among six popular machine learning algorithms, they used the AdaBoost [10] which could get the best results in the preliminary experimental study. Their results showed that they achieved 57-83% precision for detecting duplicate pull requests.

2.3 Motivation

Although Ren et al. extracted five dimensions of features, they missed an important factor that impacts the duplicate relation. Intuitively, if the created time of a pull request is close to the created time of another one, they are likely to be duplicate. For example, there are three pull requests in the same project. We define them as A, B, and C, which were created at May 1, 2012, June 1, 2018, and July 1, 2018, respectively. Intuitively, the pair of C and B have higher chance to be duplicate than the pair of C and A because the interval of created time between C and A is over six years where there may be many changes in the project. Figure 1 shows an example where there are two duplicate pull requests (i.e., #38571 and # 38587 pull requests in kubernetes/kubernetes on GitHub). The two pull requests achieved the same function according to their title and the code they added. The *# 38571 pull request*[2] was submitted on Dec 11, 2016, and was merged. The *#38587 pull request*[3] was submitted on Dec 12, 2016, and was closed because it was duplicated of # 38571 pull request which the developers said in the `Conversation` part.

To validate the assumption, we did statistic analysis for the interval of created time for each pair of pull requests in our duplicate dataset, which is shown in Figure 2. We found that the smaller the value of the interval of created time for two pull requests, the more the number of duplicate pull requests. That is, if the interval of created time of a pair of pull requests is small, they are more likely to be duplicate. Hence, in this paper, we aim to investigate how time factor affects duplicate pull request detection.

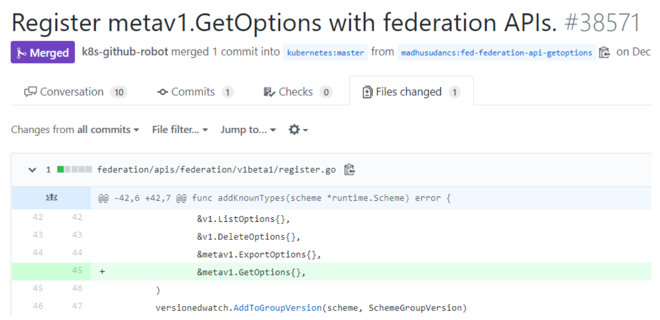
3 PROPOSED APPROACH

3.1 Overall Approach

Figure 3 presents our overall approach which includes two main phases: model building and prediction.

In the model building phase, our framework takes as input a set of training pull requests with known labels (duplicate or non-duplicate). The framework first extracts various features from the pull requests. The features are quantifiable characteristics of pull requests that could potentially differentiate duplicate pull requests from non-duplicate ones. In this paper, we consider features that are grouped into six dimensions: change description, path content, changed files list, location of code changes, reference to issue tracker and time. The first five dimensions of features are proposed by Ren et al.. We

[2]https://github.com/kubernetes/kubernetes/pull/38571/files
[3]https://github.com/kubernetes/kubernetes/pull/38587/files

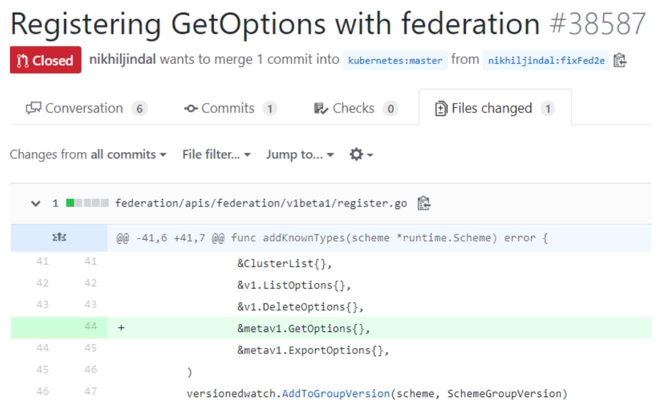


Register metav1.GetOptions with federation APIs. #38571

Merged k8s-github-robot merged 1 commit into kubernetes:master from madhusudancs:fed-federation-api-getoptions on Dec

```
1 federation/apis/federation/v1beta1/register.go
@@ -42,6 +42,7 @@ func addKnownTypes(scheme *runtime.Scheme) error {
                        &v1.ListOptions{},
                        &v1.DeleteOptions{},
                        &metav1.ExportOptions{},
+                       &metav1.GetOptions{},
                )
                versionedwatch.AddToGroupVersion(scheme, SchemeGroupVersion)
```

(a) #38571 pull request created at 11 December 2016 in kubernetes/kubernetes on GitHub



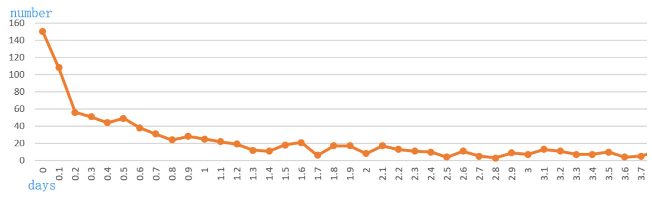
Registering GetOptions with federation #38587

Closed nikhiljindal wants to merge 1 commit into kubernetes:master from nikhiljindal:fixFed2e

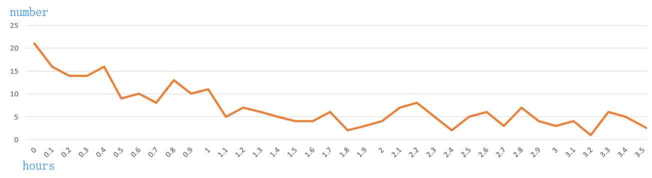
```
1 federation/apis/federation/v1beta1/register.go
@@ -41,6 +41,7 @@ func addKnownTypes(scheme *runtime.Scheme) error {
                        &ClusterList{},
                        &v1.ListOptions{},
                        &v1.DeleteOptions{},
+                       &metav1.GetOptions{},
                        &metav1.ExportOptions{},
                )
                versionedwatch.AddToGroupVersion(scheme, SchemeGroupVersion)
```

(b) #38587 pull request created at 12 December 2016 in kubernetes/kubernetes on GitHub

Figure 1: Two duplicate pull requests in kubernetes/kubernetes on GitHub



(a) The relation between interval of created time in days for two pull requests and the number of pull requests in duplicate dataset



(b) The relation between interval of created time in hours for two pull requests and the number of pull requests in duplicate dataset

Figure 2: The relation between interval of created time for two pull requests and the number of pull requests in duplicate dataset.

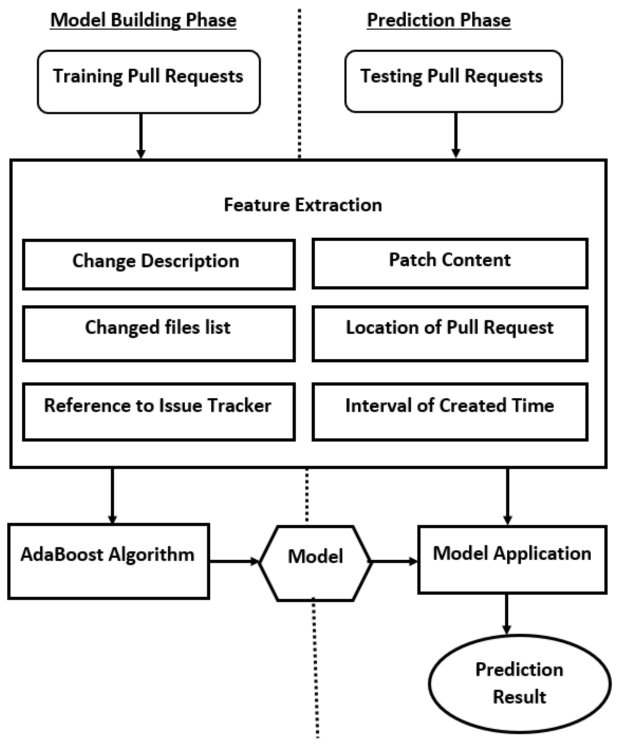


Figure 3: Overall Framework

proposed a new feature (i.e.,*time* feature). Then, based on these features and the labels of training pull requests, we use AdaBoost algorithm which is also used by Ren et al. to build a model. Finally, the framework produces a prediction model that output likelihood scores for the pull request to be duplicate with the ones in a given dataset.

In the prediction phase, the framework takes as input a set of testing pull requests whose labels are to be predicted. Features are first extracted from the pull requests. Then, model learned in the model building phase is applied to predict the labels of these pull requests by analyzing the extracted features. Finally, the framework ranks the pull requests in descending order according to the likelihood scores outputing by the model for each of them.

3.2 Feature Engineering

We want to capture the pertinent aspects of a pull request to decide if it is a duplicate pull request or not. Ren et al. [27] extracted nine features which are described in Section 2.2. However, there is an important feature (i.e., *time* feature) that Ren et al. did not take it into consideration.

Figure 1 shows an example of the relation between time and duplication which is described in Section 2.3. To further validate the influence of the interval of created time for two pull requests on their duplicate relation, we did statistic analysis in our duplicate dataset which included 2323 pairs of duplicate pull requests. We found that the time and duplicate relation are indeed related as shown in Figure 2. In Figure 2 (a), the X-axis represents the interval of created time between two pull requests in days, and Y-axis is the number of pull requests in our duplicate dataset. For Figure 2 (b), it is similar to Figure 2 (a), where the difference is the X-axis is the interval of created time between two pull requests in hours. As shown in the figure, the smaller the value in X-axis, the bigger the value in Y-axis. That is, the smaller the interval of created time for two pull requests, the more likely the two pull requests are duplicate.

Since the values of most features proposed by Ren et al. are between 0 and 1, we normalized the values of time feature by the method of min-max normalization to make it between 0 and 1 accordingly to eliminate the possible adverse effects caused by singular sample data. In total, we use ten features which are shown in Table 1.

Table 1: Dimensions and Corresponding Machine Learning Features

Dimension	Feature for Classifier	Value
Change description	Title_similarity	[0,1]
	Description_similarity	[0,1]
Patch content	Patch_content_similarity	[0,1]
	Patch_content_similarity_on _overlapping_changed_files	[0,1]
Changed files list	Changed_files_similarity	[0,1]
	Number_of_overlapping _changed_files	N
Location of code changes	Location_similarity	[0,1]
	Location_similarity_on _overlapping_changed_files	[0,1]
Reference to issue tracker	Reference_to_issue_tracker	{-1,0,1}
Time	Interval_of_created_time	[0,1]

3.3 Binary Classification

The goal of duplicate pull request detection in this paper is to classify a pair of pull requests as duplicate or not. In this work, we propose new feature to improve state-of-the-art approach [27] and use a machine learning algorithm to train a model. There are many machine learning algorithms, such as neural network, support vector machines, AdaBoost, k-Nearest Neighbor, random forest, logistic regressions and decision Trees. Ren et al. [27] assessed the performance of these machine learning algorithms for detecting duplicate pull requests, their experimental results showed that AdaBoost achieved the best result. Hence, in this paper, we use AdaBoost to train a model.

4 EXPERIMENTS

4.1 Experimental Data

We follow the experimental setup of the work by Ren et al. [27]. To evaluate our approach, ground truth is needed (i.e., a dataset where pull requests are labeled as duplicate). In this paper, we use a dataset which contains 2,323 pairs of duplicate pull requests from 26 prevalent repositories on GitHub [41] as shown in Table 2, which is used by Ren et al.. Half of the duplicate pull requests are picked which includes 1,174 pairs of duplicate pull requests of 12 repositories as the positive samples in the training dataset to calibrate classifier. The remaining 1,149 pairs of 14 repositories are used as testing dataset. This dataset only provides examples of duplicate pull requests, and does not provide non-duplicate pull requests which are more common in practice [11]. To solve this problem, some pairs of merged pull requests from the same repositories are randomly sampled according to the assumption that if two pull requests are both merged, they are most likely non-duplicate [27]. Overall, 100,000 pairs of negative samples are randomly sampled from the same repositories, 50,000 for training, and 50,000 for testing.

Table 2: Repositories and Their Duplicate Pull Request Pairs

Repository	Duplicate pairs	Language
`symfony/symfony`	216	PHP
`kubernetes/kubernetes`	213	Go
`twbs/bootstrap`	127	CSS
`rust-lang/rust`	107	Rust
`nodejs/node`	104	JavaScript
`symfony/symfony`	100	PHP
`scikit-learn/scikit-learn`	68	Python
`zendframework/zendframework`	53	PHP
`servo/servo`	52	Rust
`pandas-dev/pandas`	49	Python
`saltstack/salt`	47	Python
`mozilla-b2g`	38	JavaScript
`rails/rails`	199	Ruby
`joomla/joomla-cms`	152	PHP
`angular/angular.js`	112	JavaScript
`ceph/ceph`	104	C++
`ansible/ansible`	103	Python
`facebook/react`	74	JavaScript
`elastic/elasticsearch`	62	Java
`docker/docker`	61	Go
`cocos2d/cocos2d-x`	57	C++
`django/djang`	55	Python
`hashicorp/terraform`	52	Go
`emberjs/ember.js`	46	JavaScript
`JuliaLang/julia`	42	Julia
`dotnet/corefx`	30	C#

The upper half projects are used as training dataset, and the lower half projects are used as testing dataset.

4.2 Evaluation Metrics

We evaluate the performance of our approach and the baseline approach by using the following metrics:

Precision@k. Precision is the proportion of actually duplicate pull requests that are correctly classified as duplicate among all of the pull requests classified as duplicate. It is defined as:

$$Precision@k = \frac{ADPRs\ in\ top-k}{k} \tag{1}$$

In the above equation, $ADPRs$ refers to those actually duplicate pull requests. In this paper, we set k = 1, 2, 3, 4 and 5.

Recall@k. Recall is the proportion of actually duplicate pull requests that are correctly classified as duplicate among all of the actually duplicate pull requests in the repository. It is defined as:

$$Recall@k = \frac{ADPRs\ in\ top-k}{ADPRs\ in\ Repository} \tag{2}$$

It is impractical to check every pull request in repository to determine whether it is duplicate to the input pull request. Therefore, $ADPRs\ in\ Repository$ refers to the set of actually duplicate pull requests of the top 20 pull requests returned by Ren et al.'s approach for the input pull request. In this paper, we set k = 1, 2, 3, 4 and 5.

F1-score@k. F1-score is the harmonic mean of precision and recall, which can combine both of the two metrics above. It evaluates if an increase in precision (or recall) outweighs a reduction in recall (or precision) respectively.

$$F1-score@k = \frac{2 \times Precision@k \times Recall@k}{Precision@k + Recall@k} \tag{3}$$

Ren et al. used precision and recall to evaluate their approach. However, there is a limitation in their evaluation methodology. For calculating precision, every time they run the classifier, the output of the classifier is a pull request number (i.e., pull request ID) and the possibility score that the pull request is duplicate to a given one. They need many round of manually labelling the output data whether they are duplicate to calculate the precision, which is very time-consuming and takes huge human effort. In order to solve the problem, in this work, we built a fixed dataset to evaluate our approach. In test dataset, there are 1149 pairs of duplicate pull requests, and we randomly sampled 10% of the data (i.e., 115 pairs of duplicate pull requests) in test dataset because it takes huge human effort to label data. For each pair of the data, first, we compare the pull request numbers and find the one whose number is bigger which means the pull request is created later than another one. We aim to identify whether it is duplicate to the ones in history, so we need to tell whether the pull request created later is duplicate to the one created earlier. To our best knowledge, Ren et al.'s work is state-of-the-art. Thus, for the pull request whose number is bigger in the pair, we get top 20 most similar ones from all the history pull requests in this project whose pull request numbers are smaller than the given one with Ren et al.'s approach. Ren et al. trained a classification model with AdaBoost algorithm by nine features. The model can output the rank of the pull requests in descending order according to the likelihood scores. We get the top 20 pull requests assuming that all the potential duplicate pull requests to the given one exist in the top 20 pull requests. Then we label whether the pull request is duplicate with the 20 pull requests one by one. In total, we label 2,300 pairs (i.e., 115 * 20) of pull requests. Next, we use the labeled data to evaluate the approaches in this paper with Precision@k, Recall@k and F1-score@k.

4.3 Experimental Results

To investigate the impact of time feature on duplicate pull request detection, we conducted some experiments and studied the following three correlated research questions.

4.3.1 *RQ1: How does time affect duplicate pull request detection? How much improvement can it achieve over baseline if there is only time feature to train a classification model?*

Motivation. We aim to understand whether the interval of created time influences the detection of duplicate pull requests. The research question is to investigate how effective it is if we use the time feature to train a classification model and predict the result (i.e., time approach). Moreover, we want to know whether it can perform as well as, or better than the state-of-the-art approach (i.e., baseline approach). Answering this research question would shed light on whether *time* feature affects duplicate pull request detection.

Approach. To answer this research question, we compare the effect of time on duplicate pull request detection with the state-of-the-art baseline approach. First, we re-implement the experiment of baseline approach which includes nine features. Secondly, we conduct an experiment where there is just one time feature to train a classification model. We use our labeled data mentioned in Section 4.2 as test data. Given a pair of pull requests from the 115 pairs test data, we use the state-of-the-art model by Ren et al. (i.e., using nine features to train a model) to get the top 20 most similar pull requests to the one in the pair whose ID is bigger. We use the model which is trained by the only time feature to predict among the top 20 pull requests. Then we get the new list of top 20 most similar ones. This is our time approach which is the combination of just using time feature to train a model and testing the data which returns the top 20 pull requests with the approach by Ren et al. Finally, we evaluate the performance of the two experiments with Precision@k, Recall@k, F1-score@k, where the values of k range from 1 to 5.

Results. Table 3 shows the Precision@k, Recall@k and F1-score@k scores of baseline approach, time approach and the improvement of time approach over baseline.

As shown in the table, time approach could achieve a good result. The Precision@1, Recall@1, F1-score@1 are 56.52%, 51.46% and 53.87%, respectively. It means that the *time* indeed is an important feature to detect duplicate pull requests. In most evaluation metrics, i.e., Precision@k, Recall@k, F1-score@k where the values of k range from 2 to 5, the time approach outperforms the baseline approach. For example, the improvement of Precision@3, Recall@3, F1-score@3 are 11.34%, 8.90% and 10.64%, respectively. However, in terms of

Table 3: The result of baseline approach, time approach and improvement of time approach over baseline approach

Evaluation	k	Baseline	Time Approach	Improvement
Precision@k	1	65.22%	56.52%	**-13.33%**
	2	39.57%	40.87%	3.30%
	3	28.12%	31.30%	11.34%
	4	22.83%	24.78%	8.57%
	5	19.30%	21.22%	**9.91%**
Recall@k	1	56.97%	51.46%	**-9.67%**
	2	67.48%	70.83%	4.96%
	3	71.17%	77.51%	8.90%
	4	77.70%	82.72%	6.47%
	5	81.25%	85.93%	**5.76%**
F1-score@k	1	60.82%	53.87%	**-11.41%**
	2	49.88%	51.83%	3.90%
	3	40.31%	44.60%	10.64%
	4	35.29%	38.14%	8.09%
	5	31.20%	34.03%	**9.09%**

Precision@1, Recall@1, F1-score@1, time approach is worse than baseline approach. It means that time feature has a disadvantage of detecting the top 1 duplication.

In summary, time is an important factor that affects duplicate pull request detection, and time approach outperforms the state-of-the-art baseline method in terms of Precision@k, Recall@k, F1-score@k where the values of k range from 2 to 5.

4.3.2 *RQ2: How effective is our approach? How much improvement can it achieve over baseline approach?*

Motivation. To further investigate the significance of the influence of *time* on duplicate pull request detection, we want to see how effective it is if we combine time feature and the nine features in Ren et al.'s work (i.e., our approach). We also want to know whether the combination can perform as well as, or better than the state-of-the-art approach by Ren et al. (i.e., baseline approach). Answering this research question would stand out how significant the influence of time on duplicate pull request detection.

Approach. To answer this research question, We conduct two experiments. The first one is we re-implemented the baseline approach (i.e., using nine features to train a model) which returns the top 20 most similar pull requests to a given one. The second experiment is we combine the time feature and the nine features proposed by Ren et al. as input into a classifier. Then we use our labeled data mentioned in Section 4.2 as test data. The test process is the same to the one in RQ1. We evaluate the performance of the experiments with Precision@k, Recall@k, F1-score@k, where the values of k range from 1 to 5.

Results. Table 4 shows the Precision@k, Recall@k and F1-score@k scores of baseline approach, our approach and the improvement of our approach over baseline method.

Table 4: The result of baseline, our approach and improvement of our approach over baseline

Evaluation	k	Baseline	Our Approach	Improvement
Precision@k	1	65.22%	73.91%	**13.33%**
	2	39.57%	46.52%	17.58%
	3	28.12%	33.91%	20.62%
	4	22.83%	26.30%	15.24%
	5	19.30%	21.74%	**12.61%**
Recall@k	1	56.97%	65.67%	**15.26%**
	2	67.48%	78.87%	16.88%
	3	71.17%	84.25%	18.37%
	4	77.70%	86.86%	11.79%
	5	81.25%	88.67%	**9.13%**
F1-score@k	1	60.82%	69.55%	**14.36%**
	2	49.88%	58.52%	17.32%
	3	40.31%	48.36%	19.97%
	4	35.29%	40.38%	14.44%
	5	31.20%	34.92%	**11.93%**

As shown in the table, our approach could achieve a better result. The Precision@1, Recall@1, F1-score@1 are 73.91%, 65.67% and 69.55%, respectively. In all of the evaluation metrics in this paper, our approach outperforms the baseline approach. The biggest improvement is when k is 3 where Precision@3, Recall@3, F1-score@3 are 20.62%, 18.37% and 19.97%, respectively. The smallest improvement is when k is 5 where Precision@5, Recall@5, F1-score@5 are 12.61%, 9.13% and 11.93%, respectively. It means that *time* feature indeed is a significant feature for duplicate pull request detection.

To sum up, the approach of integrating the time feature into the exiting features proposed by Ren et al. significantly outperforms the state-of-the-art baseline method. The advantage of our approach is more evident for duplicate pull request detection.

4.3.3 *RQ3: How much improvement can our approach achieve over that using time feature to train a classification model?*

Motivation. We want to know whether the combination (i.e., our approach) of time feature and other nine features proposed by Ren et al. can perform better than the approach that only considers time feature to train a model (i.e., time approach). Answering this research question would shed light on when the time feature matters or does not matter.

Approach. To answer this research question, we compare the effect of time on duplicate pull request detection with the combination of time feature and other nine features. We conduct two experiments. The first one is using ten features to train a model, and we use our labeled data mentioned in Section 4.2 as test data. The test process is the same to the one in RQ1. The other experiment is just using time feature to train a model. Also, the test process is the same to the one in RQ1. We evaluate the performance of with Precision@k, Recall@k, F1-score@k, where the values of k range from 1 to 5.

Table 5: The result of time approach, our approach and improvement of our approach over time approach

Evaluation	k	Time	Our Approach	Improvement
Precision@k	1	56.52%	73.91%	**30.77%**
	2	40.87%	46.52%	13.83%
	3	31.30%	33.91%	8.33%
	4	24.78%	26.30%	6.14%
	5	21.22%	21.74%	**2.46%**
Recall@k	1	51.46%	65.67%	**27.60%**
	2	70.83%	78.87%	11.36%
	3	77.51%	84.25%	8.69%
	4	82.72%	86.86%	4.99%
	5	85.93%	88.67%	**3.19%**
F1-score@k	1	53.87%	69.55%	**29.09%**
	2	51.83%	58.52%	12.91%
	3	44.60%	48.36%	8.44%
	4	38.14%	40.38%	5.87%
	5	34.03%	34.92%	**2.60%**

Results. Table 5 shows the Precision@k, Recall@k and F1-score@k scores of our approach, time approach and the improvement of our approach over time approach.

As shown in the table, our approach performs better. In all of the evaluation metrics, i.e., Precision@k, Recall@k, F1-score@k where the values of k range from 1 to 5, our approach outperforms time approach. Especially, the improvement of Precision@1, Recall@1, F1-score@1 are 30.77%, 27.60% and 29.09%, respectively. It means that time feature has a disadvantage of detecting the top 1 duplication compared with the combination of ten features. As shown in the table, when the values of k range from 3 to 5, the approach of combining ten features does not significantly improve the result, which means that time feature matters in identifying the top k duplicate pull requests where the values of k range from 3 to 5.

In a word, the combination of time feature and existing features proposed by Ren et al. could achieve better result than only using time feature to train a classification model.

5 RELATED WORK

5.1 Duplicate Pull Request Detection

Yu et al. [41] created a dataset of duplicate pull requests (called DupPR) extracted from 26 popular open source projects on GitHub. Each pair of duplicate pull requests in DupPR were manually verified after an automatic identification process. The dataset constructed by Yu et al. which contains 2323 pairs of duplicate pull requests is used as part of our ground truth data in this paper.

Li et al. [22] proposed a method to detect duplicate pull requests. For a given pull request, they computed the textual similarity between it and other history pull requests, and then returned a candidate list of the most similar ones. Textual information consists of two parts: title and description. They investigated the detection performance of title similarity, description similarity and the combined similarity, respectively. Recall-rate is used to evaluate their approach. The evaluation result showed that about 55.3%-71.0% of the duplicate pull requests could be found when they used the combination of title similarity and description similarity.

Besides textual factor, Ren et al. [27] investigated more factors (e.g., source code information) on duplicate pull request detection. Firstly, they identified factors that indicated a pair of pull requests might be similar by manually checking 45 pairs of duplicate pull requests, and they summarized nine features which were included in five dimensions. Next, they calculated the similarity between pull requests for each feature. Finally, they used the list of similarities as input to train a classifier to predict whether a pair of pull requests were duplicate. Different from the above work, we explored whether time factor can be used to detect duplicate pull requests on GitHub.

5.2 Duplicate Bug Report Detection

Although we focus on duplicate pull request detection, there are many studies to detect other forms of duplicate submissions, such as bug report [12, 24, 28, 29, 39]. They are similar because the two kinds of submissions (i.e., pull request and bug report) are both in a distribute and parallel way. Moreover, the two kinds of duplicate detection are both comparing text information of submissions.

Although very few work studies on duplicate pull request detection, there are a number of studies on investigating how to identify duplicate bug reports. As proposed by Lin et al. [23] and reused by Kang [16], duplicate detection approach could be classified into three categories:

- Ranking approaches: [3, 6, 7, 13, 14, 21, 28, 31–34, 39, 42],they use a bug report as input and output a ranked list of duplicate candidates from a corpus of existing bug reports;
- Binary approach: [35], it takes a bug report as input and label it as either duplicate or not duplicate;
- Decision-making approaches: [1, 13, 18, 20], they consider a pair of bug reports and determine whether they are duplicate of each other.

Bug reports provide textual description that involves the natural language bug description reported by developers, i.e., title and description. The aforementioned unstructured information could be used by researchers to study the similarity between existing bug reports and an incoming bug report for classification purposes [17, 23, 26, 35, 36, 39]. Natural language information and information retrieval (IR) techniques are widely used to calculate the similarity scores between a given data and the retrieved data. In this context, Bettenburg et al. [4] conducted a study on the effort required for manually identifying duplicate issue report. Their classification model achieved an average precision and recall of 68% and 60%, respectively.

Researchers have studied the value of adding extra information other than the title and description to improve duplicate retrieval. The main sources of additional information are the bug report's creation data [7, 18, 20, 21, 28] and categorical features [1, 6, 13, 18, 20, 25, 28, 31, 35] such as the affected system version or component. Jalbert and Weimer [15] used surface features, textual semantics and graph clustering to predict duplicate status. Tian et al. [35] improved the work by [15]. They used maximum REP similarity, and added another category information. Also, they brought into relative similarity to determine the importance of the text similarity between bugs. Our work is different from the above studies since we focus on duplicate pull request detection rather than duplicate bug report detection.

6 DISCUSSION

6.1 Implications

We are interested to further investigate why the time factor plays a significant role for duplicate pull request detection. From Table 3, we found that the time approach achieved a better performance over baseline method in terms of Precision@k, Recall@k and F1-score@k where the values of k range from 2 to 5. The possible reasons are as follows: for each pair of pull requests in duplicate test dataset, we choose the one whose pull request number is bigger (i.e., the one who is created later) because we aim to decrease the workload of project maintainers, i.e., given a pull request, we identify whether it is duplicate with all the history pull requests. To avoid many round of manually labelling data for evaluation in Ren et al.'s approach, we build a fixed dataset. For the later created pull request of each pair in duplicate test dataset, we get the top 20 most similar pull requests to it by Ren et al. approach (i.e., return the top 20 most similar pull requests by the model trained by nine features). Moreover, the data statistics between the interval of created time and duplicate relation involved that time indeed affects duplicate relation, and it indicates that time feature could be a significant factor. Besides, from the Table 3, we could see that the Precision@1, Recall@1 and F1-score@1 of time approach are worse than the ones of baseline method. It means that time feature has disadvantage of detecting the top 1 duplicate pull request. The possible reason is that there are many new pull requests every day. It could't identify the top 1 duplicate pull request only using time feature because other features are also needed and important.

There have been a large amount of approaches proposed to detect other forms of duplicate submissions, including bug reports [12, 24, 28, 39] and Stack Overflow questions [2, 40]. They are similar to our work because the individuals are submitted in a distributed and parallel way. Thus leveraging time feature may be able to further improve those tasks.

6.2 Threats to validity

Threats to internal validity relate to errors in our code and personal bias in manual classification of labeling duplicate pull requests. To reduce errors in our code, we have double checked and fully tested our code, still there could be errors that we did not notice. To reduce personal bias on manual classification of pull requests, the first two authors classify them independently. We further reduce bias by using an external evaluator to help resolve disagreement. These steps increase our confidence in the manually created dataset.

Threats to external validity relate to the quantity and quality of our dataset and the generalizability of our results. To guarantee quantity and quality of our dataset, we use 26 open source projects and over 100,000 pairs of pull requests on GitHub. However, all the 26 projects are developed by open source communities, and it is still unclear whether our approach is generalizable when applying to projects in a company. Since open source communities are highly distributed and independent and developers are usually allowed to add new features or fix bugs, they are more likely to submit duplicate pull requests. In contrast, it is possible that developers in a company are assigned a task and usually they focus their all attention on their task rather than fixing bugs or adding new features that are not assigned to them. So in this paper, we only focus on the scope of open source projects. Besides, although 26 projects are analyzed, there are so many projects on GitHub. In the future, we plan to analyze more pull requests from additional software projects. Moreover, since it takes huge human effort to label all data in duplicate testing data which includes 1,149 pairs of pull requests, we randomly sampled 115 pairs. As for the 115 pairs of data, we picked up the top 20 most similar pull requests to it. Totally, we labeled 2,300 pull requests. In the future, we will label more pull requests.

7 CONCLUSION AND FUTURE WORK

Pull request is used commonly by teams or organizations for collaboration and the process of pull request is distributed and in parallel. Thus, there are often many duplicate pull requests being submitted that do the same thing (e.g., fix the same bug or add the same feature). To address this problem, duplicate pull request detection approaches are required. In this paper, we proposed a new feature in terms of time to detect duplicate pull requests. We have performed experiments with 26 projects on GitHub. We integrated the new feature into the existing features proposed by Ren et al. [27], and we found that the new feature boosted 14.36% and 11.93% improvements over the state-of-the-art approach in terms of F1-score@1 and F1-score@5.

In the future, we plan to investigate more features to improve the accuracy of duplicate pull request detection further. Moreover, we plan to release a tool that would help developers to identify duplicate pull requests early.

REFERENCES

[1] Karan Aggarwal, Finbarr Timbers, Tanner Rutgers, Abram Hindle, Eleni Stroulia, and Russell Greiner. 2017. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process* 29, 3 (2017), e1821.

[2] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. 2016. Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 402–412.

[3] Mehdi Amoui, Nilam Kaushik, Abraham Al-Dabbagh, Ladan Tahvildari, Shimin Li, and Weining Liu. 2013. Search-based duplicate defect detection: an industrial experience. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 173–182.

[4] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Duplicate bug reports considered harmfulâĂę really?. In *2008 IEEE International Conference on Software Maintenance*. IEEE, 337–345.

[5] Jürgen Bitzer and Philipp JH Schröder. 2006. The impact of entry and competition by open source software on innovation activity. In *The economics of open source software development*. Elsevier, 219–246.

[6] Vincent Boisselle and Bram Adams. 2015. The impact of cross-distribution bug duplicates, empirical study on Debian and Ubuntu. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 131–140.

[7] Markus Borg, Per Runeson, Jens Johansson, and Mika V Mäntylä. 2014. A replicated study on duplicate detection: Using Apache Lucene to search among Android defects. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 8.

[8] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An exploratory study of cloning in industrial software product lines. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 25–34.

[9] Neil A Ernst, Steve Easterbrook, and John Mylopoulos. 2010. Code forking in open-source software: a requirements perspective. *arXiv preprint arXiv:1004.2889* (2010).

[10] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[11] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 345–355.

[12] Lyndon Hiew. 2006. *Assisted detection of duplicate bug reports*. Ph.D. Dissertation. University of British Columbia.

[13] Abram Hindle, Anahita Alipour, and Eleni Stroulia. 2016. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering* 21, 2 (2016), 368–410.

[14] Abram Hindle and Curtis Onuczko. 2019. Preventing duplicate bug reports by continuously querying bug reports. *Empirical Software Engineering* 24, 2 (2019), 902–936.

[15] Nicholas Jalbert and Westley Weimer. 2008. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 52–61.

[16] Li Kang. 2017. Automated Duplicate Bug Reports Detection-An Experiment at Axis Communication AB.

[17] Nilam Kaushik and Ladan Tahvildari. 2012. A comparative study of the performance of IR models on duplicate bug detection. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 159–168.

[18] Nathan Klein, Christopher S Corley, and Nicholas A Kraft. 2014. New features for duplicate bug detection. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 324–327.

[19] Thomas K Landauer and Susan T Dumais. 1997. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review* 104, 2 (1997), 211.

[20] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 308–311.

[21] Johannes Lerch and Mira Mezini. 2013. Finding duplicates of your yet unwritten bug report. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 69–78.

[22] Zhixing Li, Gang Yin, Yue Yu, Tao Wang, and Huaimin Wang. 2017. Detecting duplicate pull-requests in github. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware*. ACM, 20.

[23] Meng-Jie Lin, Cheng-Zen Yang, Chao-Yuan Lee, and Chun-Chang Chen. 2016. Enhancements for duplication detection in bug reports with manifold correlation features. *Journal of Systems and Software* 121 (2016), 223–233.

[24] Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. 2003. Automated support for classifying software failure reports. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 465–475.

[25] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E Hassan. 2017. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *IEEE Transactions on Software Engineering* 44, 12 (2017), 1245–1268.

[26] Mohamed Sami Rakha, Weiyi Shang, and Ahmed E Hassan. 2016. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering* 21, 5 (2016), 1960–1989.

[27] Luyao Ren, Shurui Zhou, Christian Kästner, and Andrzej Wąsowski. 2019. Identifying Redundancies in Fork-based Development. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 230–241.

[28] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 499–510.

[29] Yoonki Song, Xiaoyin Wang, Tao Xie, Lu Zhang, and Hong Mei. 2010. JDF: detecting duplicate bug reports in Jazz. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 315–316.

[30] Ştefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. 2015. Forked and integrated variants in an open-source firmware project. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 151–160.

[31] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 253–262.

[32] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 45–54.

[33] Ashish Sureka and Pankaj Jalote. 2010. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific Software Engineering Conference*. IEEE, 366–374.

[34] Ferdian Thung, Pavneet Singh Kochhar, and David Lo. 2014. DupFinder: integrated tool support for duplicate bug report detection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 871–874.

[35] Yuan Tian, Chengnian Sun, and David Lo. 2012. Improved duplicate bug report identification. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 385–390.

[36] Akihiro Tsuruda, Yuki Manabe, and Masayoshi Aritsugi. 2015. Can we detect bug report duplication with unfinished bug reports?. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 151–158.

[37] Greg R Vetter. 2007. Open source licensing and scattering opportunism in software standards. *BCL Rev.* 48 (2007), 225.

[38] Qingye Wang, Xin Xia, David Lo, and Shanping Li. 2019. Why Is My Code Change Abandoned? *Information and Software Technology* 110 (02 2019). https://doi.org/10.1016/j.infsof.2019.02.007

[39] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*. ACM, 461–470.

[40] Bowen Xu, Amirreza Shirani, David Lo, and Mohammad Amin Alipour. 2018. Prediction of relatedness in stack overflow: deep learning vs. SVM: a reproducibility study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 21.

[41] Yue Yu, Zhixing Li, Gang Yin, Tao Wang, and Huaimin Wang. 2018. A dataset of duplicate pull-requests in github. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 22–25.

[42] Jian Zhou and Hongyu Zhang. 2012. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 852–861.