Changes are Similar: Measuring Similarity of Pull Requests that Change the Same Code in GitHub

Ping Ma, Danni Xu, Xin Zhang, and Jifeng Xuan*

School of Computer Science, Wuhan University, Wuhan 430072, China
jxuan@whu.edu.cn

Abstract. Pull-based development is widely used in globally collaborative platforms, such as GitHub and BitBucket. A pull request is a set of changes to existing source code in a project. A developer submits a pull request and tends to update the source code. Due to the parallel mechanism, several developers may submit multiple pull requests to change the same lines of code. This fact results in the conflict between changes, which makes the project manager difficult to decide which pull request should be merged. In this paper, we conducted a preliminary study on measuring the similarity of pull requests that aim to change the same code in GitHub. We proposed two methods, i.e., the *cosine* and the *doc2vec*, to quantify the structural similarity and the semantic similarity between pull requests and evaluated the similarity on four widely-studied open source Java projects. Our study shows that there indeed exists high similarity between competing pull requests and the similarity among projects diversifies. This complicates the merging decision by project managers.

Keywords: Pull requests · GitHub · Similarity · Empirical study · Code changes.

1 Introduction

GitHub is widely-used in collaborative software development. A developer who is engaged in open-source projects tend to use GitHub to support their collaboration. According to the official website, up to June 2018, GitHub has over 28 million users and 57 million repositories, making it the largest host of source code in the world [2]. GitHub achieves collaborative development via the mechanism of pull requests. Once a developer wants to update a project, he can submit a pull request which consists of one or more code changes to the target project. A submitted pull request waits to be merged into the repository or discarded by the manager of the target project.

Due to the parallel mechanism of GitHub, it often happens that developers make different changes to the same lines of code during the same time period. This makes pull requests potentially compete with each other, i.e., competing

* Corresponding author: Jifeng Xuan

pull requests [20]. Such pull requests may contain changes that have the same or different goals and may cause the conflicts on the structure or the semantics of code. It is complicated and time-consuming for a project manager to decide which pull request should be merged in high priority [7], [23], [8]. The existence of conflicts between changes by these pull requests exacerbates the difficulty for the merging decision. We speculate that the similarity between these pull requests may have a great impact on the difficulty that project managers have to face when they make decisions for merging pull requests.

In this paper, we conducted a preliminary study on measuring the similarity between pull requests that aim to change the same code in GitHub. We used the *cosine* and the *doc2vec* to measure the similarity of code structure and semantics, respectively. We evaluated the similarity on four open source Java projects with the most forks in GitHub. Our study contain 6,469 pairs of pull requests, each pair of which has two pull requests that contain changes on overlapped code. We explored the similarities between these pull requests via three research questions, including the similarities between pull requests, the similarity distribution, and the correlations between two measurement methods in use.

Our study shows that there indeed exists high similarity between pull requests that change the same code. In the four Java projects we studied, the average similarity between each pair of pull requests is over 0.9; the highest average similarity among these four projects is up to 0.9976. In the pull requests we measured, over 75 percent of the similarity between each pair of pull requests is above 0.8 and half of the similarity is 0.95 or higher. Our study shows that there is high correlation between the two measurement methods; this indicates that both the structural similarity and the semantic similarity exist between the pull requests that aim to change the same code.

This paper makes the following contributions:

- We conducted a study of measuring the similarity between pull requests that aim to change the same code. We proposed two methods to measure the similarity of code structure and semantics on four open source Java projects with the most forks in GitHub.
- We answered three research questions and found that there indeed exists high similarity between competing pull requests. This provides the foundational result for analyzing the conflicts between pull requests.

The remaining of this paper is organized as follows. Section 2 presents the background and the motivation. Section 3 describes two measurement methods of similarity. Section 4 presents the experimental setup, which includes data preparation, two measurement methods, and three research questions. Section 5 details the results of our experiment. Section 6 explains the threats to the validity. Section 7 lists the related work and Section 8 concludes this paper.

2 Background and Motivation

We introduce the background of merging pull requests and the motivation of exploring the similarity between pull requests.

2.1 Background

Collaborative platforms, such as GitHub and BitBucket, have provided high interaction between developers and code projects via pull requests. A project manager can deploy the codebase in GitHub and allows other developers to *fork* (i.e., clone) the project into their own account as a copy. Developers could freely make code changes to the project that has been forked in their account. Once several changes are made, a pull request that includes the changes can be submitted to the target project. A submitted pull request waits for the decision by the project manager, i.e., deciding merging this pull request into the project or discarding it.

Pull requests are the key artifacts that make developers accomplish collaborative development in GitHub. Technically, a pull request consists of one or more commits, each of which contains edits to the original source code at a particular time. Once these changes are accepted by the project manager, the pull request is merged into the original repository. Conversely, the pull request would be closed if the manager chooses not to accept these code changes. GitHub provides a free and flexible platform for developers to submit pull requests; hence, the project manager may receive multiple pull requests during a time period. In this case of multiple pull requests, it is time-consuming for the manager to decide which pull request should be merged.

2.2 Motivation

Due to the parallel mechanism of GitHub, several developers may submit different pull requests to change the same lines of code, which cause the competing pull requests [20]. This fact results in the conflict between changes because the code changes that competing pull requests have made focus on the same target. A project manager can choose one or zero among these competing pull requests and merge it into the original project.

As a manager of an open-source project, he/she has to manually check all competing pull requests to ensure the contribution of pull requests and to decide merging which pull request. The existence of competing pull requests makes the merging decision difficult. Do competing pull requests behave similar? – An intuitive speculation is that competing pull requests are similar since they aim to update the same lines of code. Supposing the similarity between competing pull requests is low, we can surmise that directly distinguish different competing pull requests is possible; supposing the similarity is high, a semantical or further detailed analysis could help for the merging decisions. Motivated by the exploration on the similarity of pull requests, we conducted a preliminary study on the similarity between pull requests that tend to change overlapped pieces of code.

3 Measurement Methods of Similarity

There is no reliable and effective way to measure the similarity of code. In this paper, we used two measurement methods to check the structural similarity and the semantic similarity between pull requests. For the structural similarity, we employ the *cosine* that is designed to measure the textual similarity between sentences; for the semantic similarity, we employ the *doc2vec* that is developed to extract the semantics of paragraphs.

3.1 The *cosine* Method

The *cosine* similarity is usually used to measure the textual similarity of sentences [21]. The key idea of the *cosine* is to count the number of co-appearance of words to reveal potential semantics [17].

We used the *cosine* method to measure the similarity between pull requests. The *cosine* similarity uses the *cosine* value of angles of two vectors in the vector space. Given two n-dimension vectors X and Y, the *cosine* similarity of two vectors is defined as follows,

$$Sim_{\cos}(X,Y) = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}}$$

where x_i and y_i denote the element value of the ith dimension in X and Y, respectively. The closer the *cosine* is to 1, the more similar two vectors are.

Given two pull requests, we treat them as two pieces of code. To utilize the *cosine* to measure the similarity between pull requests, we convert the changed code into textual sentences based on the following two steps. First, we filter out the punctuation from the code and separate the code into tokens. For instance, giving the code `assertThat(processDefinitions.getBody().getContent().hasSize(4));`, we transfer the code into a sequence of `assertThat`, `processDefinition`, `getBody`, `getContent`, `hasSize`, and `4`.

Second, we collect the frequency of tokens in two pull requests and obtain the word-frequency vectors. Then given two vectors, we leverage the *cosine* to calculate the similarity.

3.2 The *doc2vec* Method

In addition to measuring the structural similarity between pull requests with the *cosine*, we employ the *doc2vec* to reveal semantic similarity. The *doc2vec*, or the *paragraph2vec*, is an unsupervised algorithm that can extract the vector expression of sentences or documents [11]. This algorithm is an extension of a widely-used method *word2vec* [12].

In the *doc2vec*, a large corpus is used to train a model by maximizing the conditional likelihood between words and sentences with the hierarchical softmax and negative sampling. Different from the *cosine*, the *doc2vec* generates word vectors via an unsupervised learning process. Learned vector by the *doc2vec* can be used to calculate the similarity between sentences or documents. The implementation of the *doc2vec* directly outputs the distances between two sentences and can be converted into the similarity.

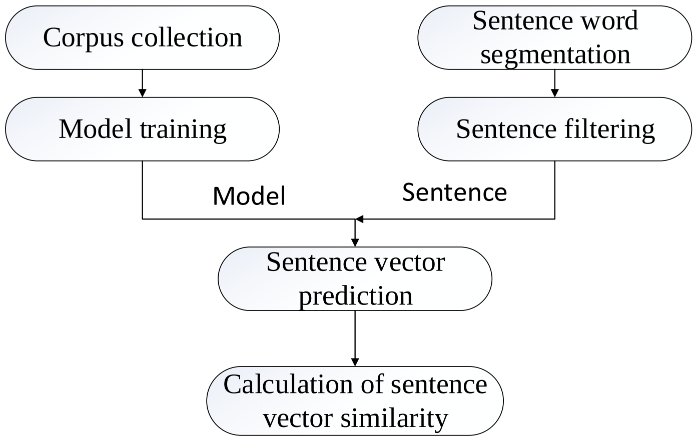
Corpus collection
Sentence word segmentation
Model training
Sentence filtering
Model
Sentence
Sentence vector prediction
Calculation of sentence vector similarity

Fig. 1. Framework of using *doc2vec* to measure the similarity between pull requests.

Fig. 1 presents the framework of using the *doc2vec* to obtain the similarity. To apply the *doc2vec* to calculate the semantic similarity of pull requests, we train the model to get the approximated likelihood of words and sentences. In each project, we extract the source code of a previous version before the submitted timestamps of pull requests. During the training, we filter out the punctuations in the source code and use all tokens in these processed source code as a training corpus. Then we employ the *doc2vec* to build the model with our training corpus. According to the implementation of the *doc2vec*, sentences that only contain tokens inside the corpus can be measured by the learned model. In each project, we use the learned model to infer the dependency of tokens in the input pull requests and use these learned vectors to calculate the similarity.

Note that the *doc2vec* is not the only way to measure the semantic similarity of two sentences or paragraphs. The Latent Semantic Analysis (LSA) by Deerwester et al. [6] and the Latent Dirichlet Allocation (LDA) by Blei et al. [5] are also widely used to detect the similarity in natural language processing. Our work does not aim to find out an optimal similarity measurement; instead, the goal of our work is to show the existence of similarity between pull requests.

4 Experimental setup

We present the steps of data preparation and the design of three research questions.

Table 1. Data collection of four Java projects with the most forks in GitHub

Project	# of forks	# of pairs of pull requests
spring-projects/spring-framework	14.7K	4799
spring-projects/spring-boot	20.0K	1178
apache/incubator-dubbo	14.4K	401
elastic/elasticsearch	11.5K	91
Total	60.6K	6469

4.1 Data Preparation

We conducted a preliminary study on four pairs of open-source Java projects with the most forks in GitHub. Table 1 presents the data of four projects in our study. There are 6,469 pairs of pull requests in total. We describe the data preparation as follows.

First, we selected the top-5 Java projects with the most forks in GitHub. We followed Zhang et al. [20] to collect all pull requests that are submitted from January 1st to December 31st, 2017. Then we extracted all pairs of pull requests, which contain changes on overlapped code. Among the five projects, we found that one project iluwatar/java-design-patterns contains only four pairs of pull requests that change the same code, then we kept all other four projects in our study. These four are spring-projects/spring-framework,[1] spring-projects/spring-boot,[2] apache/incubator-dubbo,[3] and elastic/elasticsearch,[4] respectively.

Second, we identified the pull requests that change the same code as follows. We considered that a group of competing pull requests as all pull requests that change the same lines during an overlapping time period. For instance, if Pull request `PrA` and Pull request B `PrB` edit at-least one same line of code. We consider that `PrA` and `PrB` belong to one pair of pull requests. In this paper, we focus on the similarity between pull requests. Thus, we only identify all pairs of pull requests that change the same code, rather than groups of competing pull requests as shown in [20].

Third, we merged all changes in one pull request and collected the changed code from the original codebase. The changed code can be viewed as a combination of the added code and the deleted code. To evaluate the similarity of pull requests, we only reserved the added lines of code as the new code in a pull request. If a pull request has no added lines, we directly discarded this pull request. After data preparation, we collected 6,469 pairs of pull requests from four projects.

[1] Project spring-framework, http://github.com/spring-projects/spring-framework/.
[2] Project spring-boot, http://github.com/spring-projects/spring-boot/.
[3] Project incubator-dubbo, http://github.com/apache/incubator-dubbo/.
[4] Project elasticsearch, http://github.com/elastic/elasticsearch/.

4.2 Research Questions

The aim of this paper is to explore the similarity between competing pull requests and to understand whether the high similarity between competing pull requests that complicates the merging decision by project managers. We designed three Research Questions (RQs) and conduct a preliminary study to find out the answers.

RQ1. How does the similarity between competing pull requests perform?

In our work, we tend to understand the structural similarity based on the *cosine* and the semantical similarity based on the *doc2vec*. We give a numerical result on the similarity between each pair of pull requests via evaluating the two measurement methods in RQ1.

RQ2. What is the distribution of the similarity between competing pull requests?

Besides the statistical values, we further study the similarity distributions on each project. In RQ2, we show that the similarity between pull requests diversifies and we could obtain a distribution of the similarity between competing pull requests.

RQ3. Is there any correlation between the two measurement methods of similarity?

Our study showed the results of two measurement methods of similarity. Thus, in RQ3, we evaluate the Pearson correlation coefficient to detect the correlation between these methods. We also leveraged the Wilcoxon signed rank test to figure out whether the two measurement behave different.

5 Experimental Results

We empirically examined the results of three RQs and present the existence of similarity between pull requests.

5.1 RQ1. How does the similarity between competing pull requests perform?

In RQ1, we present the numerical result of the study that measures the similarity between pull requests that contain changes on overlapped code of four Java projects in GitHub.

Table 2 shows the results of similarity between pull requests by measuring the structural similarity with the *cosine*. The similarity between pull requests measured by the *cosine* is high with the maximum value of 1.0. In three out of four, the average similarity is over 0.9; the value of the exceptional project is 0.8963, which could be considered as 0.9.

Table 3 shows the results of similarity between pull requests by measuring the semantic similarity with the *doc2vec*. In four Java projects we studied, the average similarity between pull requests of each project exceeded 0.9.

Table 2. Structural similarity between pull requests measured by the *cosine* on four projects

Project	Min	Median	Max	Average	Std
spring-projects/spring-framework	0.2887	0.9591	1.0000	0.8963	0.1552
spring-projects/spring-boot	0.3780	0.9881	1.0000	0.9251	0.1294
apache/incubator-dubbo	0.5164	1.0000	1.0000	0.9904	0.0370
elastic/elasticsearch	0.0000	1.0000	1.0000	0.9450	0.1190

Table 3. Semantic similarity between pull requests measured by the *doc2vec* on four projects

Project	Min	Median	Max	Average	Std
spring-projects/spring-framework	0.6893	0.9952	1.0000	0.9613	0.0735
spring-projects/spring-boot	0.7471	0.9989	1.0000	0.9744	0.0520
apache/incubator-dubbo	0.8855	1.0000	1.0000	0.9976	0.0091
elastic/elasticsearch	0.3899	1.0000	1.0000	0.9855	0.0472

Project apache/incubator-dubbo has the highest similarity, whose average value is 0.9976.

As shown in Tables 2 and 3, the maximum value of the similarity is 1.0 regardless of any project or method. This shows that there indeed exists high similarity between pull requests that aim to change the same code. Meanwhile, the average value behave similar. For instance, the project with the maximum value of average similarity by the *cosine* and by the *doc2vec* is the same while the project with the minimum value is the same. This leads to the guess on the correlation between two measurement methods. We will further examine this correlation in Section 5.3.

We notice that among four projects, the minimum similarity between pull requests of several projects is relatively low, such as 0.0 in Project elastic/elasticsearch when measured with the *cosine* and 0.3899 when measured with the *doc2vec*.

Based on the above findings, we can conclude that the similarity between pull requests is generally high, but there is still low similarity between several pull requests. This results in the diversity of similarities. On the one hand, RQ1 shows the evidence that it is difficult for project managers to decide which pull request should be merged into the codebase because of the high degree of similarity between pull requests. On the other hand, project managers need to spend much time in figuring out the semantics of these pull requests because of the diversity of similarity. This exacerbates the difficulty of the merging decision.

5.2 RQ2. What is the distribution of the similarity between competing pull requests?

In RQ2, we show the distribution of similarity between pull requests of four Java projects. The similarity values is calculated by the *cosine* and the *doc2vec*.

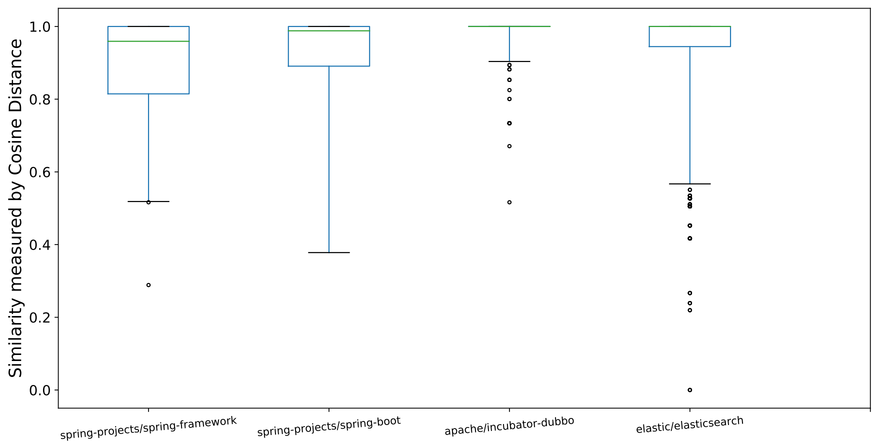

Fig. 2. Box-plots of similarity between pull requests measured by the *cosine* on four projects

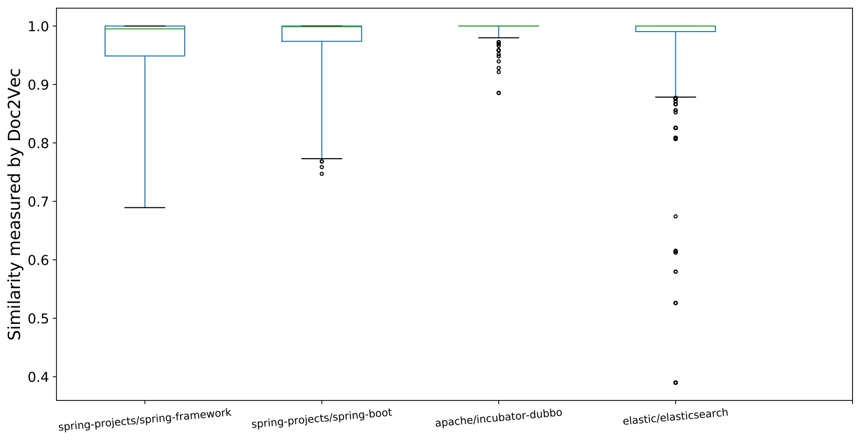

Fig. 3. Box-plots of similarity between pull requests measured by the *doc2vec* on four projects.

Figure 2 presents the box-plots of similarity measured with the *cosine* between pull requests from Table 2; Figure 3 presents the box-plots of similarity measured with the *doc2vec* between pull requests from Table 3. For each of the four projects, no matter which method is used for measurement, three-quarters of pairs of pull requests have a similarity of 0.8 or higher; half of pairs have a similarity of 0.95 or higher. These results indicate that between pull requests that contain changes on the overlapped code, both structural similarity and semantic similarity exist.

Table 4. Pearson correlation coefficient and the Wilcoxon signed rank test between the structural similarity and the semantic similarity

Project	Pearson correlation coefficient	The *p*-value
spring-projects/spring-framework	0.8179	9.6490e-11
spring-projects/spring-boot	0.8626	3.1635e-29
apache/incubator-dubbo	0.8347	7.4276e-37
elastic/elasticsearch	0.7683	0.0000

To sum up, we find that the similarity between the majority of pull requests is high. This fact undoubtedly adds the difficulty to project managers for the merging decision in GitHub.

5.3 RQ3. Is there any correlation between the two measurement methods of similarity?

In RQ3, we show the correlation betwen the similarity by the two measurement methods. To conduct the evaluation, we used the Pearson correlation coefficient to detect the probability of correlation [14].

Table 4 shows the Pearson correlation coefficient and the *p*-value based on the Wilcoxon signed rank test. The absolute value of the Pearson correlation coefficient indicates the correlation. An absolute value over 0.7 could be considered as high correlation; We use the Wilcoxon signed rank test to explore how different are two measurement methods [14]. If the *p*-value is less than 0.05, we consider that there exists statistical significance between the similarities by the *cosine* and the *doc2vec*.

As shown in Table 4, the result explores the effect of two measurement methods on the similarity values. The values of Pearson correlation coefficient show that the two measurement methods reach high correlation with the maximum value of 0.86. This indicates that the structural similarity by the *cosine* and the semantic similarity by the *doc2vec* share many options on the similarity. Although the *cosine* and the *doc2vec* estimate the similarity in different ways, it is possible that many pieces of structural similarity are also contain the semantic similarity.

All the *p*-values are less than 0.05. This fact shows that the two measurement methods behave statistically significant differences on the pairs of pull requests in all the four projects.

We conclude the answers to RQ3 as follows. The two measurement methods, the *cosine* and the *doc2vec*, show statistically significant results. However, the result by these two methods is different but similar. This fact may indicate the overlap between the structural similarity and the semantic similarity.

6 Threats to Validity

We list the threats to the validity of our work in three categories.

Construct validity. In our study, we only used pull requests of four open-source Java projects with the most forks in GitHub, which contains 6,469 pairs of pull requests. A large study on more pull requests may reveal more findings about the similarity between pull requests. Our experiment contains two measurement methods for the calculation of structural similarity and semantic similarity between pull requests; our result shows there exists diversity among the similarity. Our study has not covered many typical methods of measuring the similarity, such as the LDA method of topic models. The study in our work can be viewed as a preliminary result for counting the similarity between pull requests. The goal of our study is to motivate the exploration of solving merging conflicts.

Internal validity. We used *doc2vec* to calculate the similarity between competing pull requests via learning a model from a corpus of previous source code. However, a learned model is usually limited by the scale of the corpus. In general, a corpus in natural language processing is much larger than the source code used in our work. Thus, it is possible that the corpus in use has already hurt the measurement in our work. A straightforward resolution is to involve more projects to form a corpus.

External validity. Our study has only explored the similarity between pull requests in Java; meanwhile, the measurement of similarity is also conducted on Java source code. There exists a threat that the Java code is naturally more similar than code in other languages. In addition, is the detected similarity by the *cosine* or the *doc2vec* really similar? This is a bias between automatic measurement and program comprehension. A study on the opinions of developers could help understand the bias.

7 Related Work

We presented the related work in two parts, change merging and text similarity measurement.

7.1 Change Merging

In collaborative development, the modification of source code is implemented via merging changes. The main methods for change merging can be divided into three categories: unstructured merge, structured merge, and semi-structured merge.

The unstructured merge mixes different versions by using the largest common subsequences matching of textual lines. This method is fast, but can result in the disorder among changes [13]. The *diff* tool [1] is a typical technique using unstructured merge. The structured merge transfers the code into abstract syntax trees and combines the code among these trees. The process of combination is limited by the grammar of programming languages. A tool of structured merge is JDime, proposed by Apel et al. [3]. The semi-structured merge represents programs as program trees and uses an abstraction of the structure of the document to provide information on how the commits are merged, such as the tool FSTMerge by Apel et al. [4].

The process of merging changes into the codebase may be delayed by various factors. Yu et al. [18], [19] have explored the factors of evaluation latency for pull requests and recommended reviewers for pull requests. Jiang et al. [10] have conducted a study on the inactive yet available assignees in GitHub. Xuan et al. [15] split test cases into small changes and refactored test cases to assist program repair. Zhu et al. [22] have studied the patterns of using folders to understand the project popularity. Xuan et al. [16] proposed a sampling strategy to learn the configuration from changes. Jiang et al. [9] studied the content and reasons of forking behaviors in GitHub.

7.2 Text Similarity Measurement

The measurement of text similarity has been widely studies. A typical approach is the vector-based similarity method. These methods, such as the *cosine* in this paper, transfer the original text objects that associates with a weight of importance into term vectors and then use a function to measure these vectors to calculate the final output similarity. The vector-based method is efficient, but cannot recognize the semantics of the text. Many methods are proposed to solve this problem. Blei et al. [5] proposed Latent Dirichlet Allocation (LDA), a typical method using generative topic models. This method samples words of two textual paragraphs and infers the similarity based on probability distributions of hidden topics. Latent Semantic Analysis (LSA) by Deerwester et al. [6] is a method based on linear projection, which identifies term-vectors from a low-dimensional space of a learned matrix.

A *word2vec* method is a family of modeling algorithms for generating word embeddings [12]. The word2vec model uses two-layer neural networks to train from linguistic contexts of words. The *doc2vec* [11] used in this paper is an extension version of word2vec and *doc2vec* supports the inference of document embeddings.

In this paper, we leverage both the structural similarity (i.e., the *cosine*), and the semantic similarity (i.e., the *doc2vec*) to identify the similarity between pull requests. We aim to examine the similarity and then understand the collaborative development behaviors.

8 Conclusion

In this paper, we conducted a study to find out whether there exist high similarity between pull requests that contain changes on the same code. We employed two measurement methods to calculate the similarity between pull requests and evaluated these methods on four Java projects with the most forks in GitHub. Our study has covered 6,469 of pull requests. We explored the similarity via answers to three research questions. Experimental results show that there indeed exist high similarity between pull requests, which may result in the difficulty of merging pull requests to project managers. The results also indicate that there exists shared opinions by two measurement methods, the *cosine* and the *doc2vec*.

Our future work is to conduct a large study on the similarity between code changes with the same target. We plan to examine and understand the reasons behind the similarity in this study.

Acknowledgments. The work is supported by the National Key R&D Program of China under Grant No. 2018YFB1003901, the National Natural Science Foundation of China under Grant Nos. 61502345 and 61872273, the Young Elite Scientists Sponsorship Program by CAST under Grant No. 2015QNRC001, and the Technological Innovation Projects of Hubei Province under Grant No. 2017AAA125.

References

1. Comparing and merging files. http://www.gnu.org/software/diffutils/manual/ (2016)
2. GitHub Repository Search. https://github.com/search?q=+&type= (2018)
3. Apel, S., Leßenich, O., Lengauer, C.: Structured merge with auto-tuning: balancing precision and performance. In: IEEE/ACM International Conference on Automated Software Engineering, ASE '12, Essen, Germany, September 3-7. pp. 120–129 (2012)
4. Apel, S., Liebig, J., Brandl, B., Lengauer, C., Kästner, C.: Semistructured merge: rethinking merge in revision control systems. In: 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) and 13th European Software Engineering Conference (ESEC), Szeged, Hungary, September 5-9, 2011. pp. 190–200 (2011)
5. Blei, D.M., Ng, A.Y., Jordan, M.I., Lafferty, J.: Latent dirichlet allocation. Journal of Machine Learning Research **3**, 993–1022 (2003)
6. Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science, **41**(6), 391–401 (1990)
7. Gousios, G., Pinzger, M., van Deursen, A.: An exploratory study of the pull-based software development model. In: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07. pp. 345–355 (2014)
8. Gu, Y., Xuan, J., Zhang, H., Zhang, L., Fan, Q., Xie, X., Qian, T.: Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence. Journal of Systems and Software **148**, 88–104 (2019)
9. Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P.S., Zhang, L.: Why and how developers fork what from whom in github. Empirical Software Engineering **22**(1), 547–578 (Feb 2017)
10. Jiang, J., Lo, D., Ma, X., Feng, F., Zhang, L.: Understanding inactive yet available assignees in github. Information & Software Technology **91**, 44–55 (2017)
11. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. pp. 1188–1196 (2014)

12. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013), http://arxiv.org/abs/1301.3781
13. Perry, D.E., Siy, H.P., Votta, L.G.: Parallel changes in large-scale software development: an observational case study. ACM Trans. Softw. Eng. Methodol. **10**(3), 308–337 (2001)
14. Ross, S.M.: Introduction to probability and statistics for engineers and scientists (2. ed.). Academic Press (2000)
15. Xuan, J., Cornu, B., Martinez, M., Baudry, B., Seinturier, L., Monperrus, M.: B-refactoring: Automatic test code refactoring to improve dynamic analysis. Information & Software Technology **76**, 65–80 (2016)
16. Xuan, J., Gu, Y., Ren, Z., Jia, X., Fan, Q.: Genetic configuration sampling: learning a sampling strategy for fault detection of configurable systems. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19. pp. 1624–1631 (2018)
17. Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., Wu, X.: Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. **27**(1), 264–280 (2015)
18. Yu, Y., Wang, H., Filkov, V., Devanbu, P.T., Vasilescu, B.: Wait for it: Determinants of pull request evaluation latency on github. In: 12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17. pp. 367–371 (2015)
19. Yu, Y., Wang, H., Yin, G., Wang, T.: Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? Information and Software Technology (2016)
20. Zhang, X., Chen, Y., Gu, Y., Zou, W., Xie, X., Jia, X., Xuan, J.: How do multiple pull requests change the same code: A study of competing pull requests in github. In: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018. pp. 228–239 (2018)
21. Zhou, J., Zhang, H., Lo, D.: Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, Zurich, Switzerland. pp. 14–24 (2012)
22. Zhu, J., Zhou, M., Mockus, A.: Patterns of folder use and project popularity: a case study of github repositories. In: 2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19. pp. 30:1–30:4 (2014)
23. Zhu, J., Zhou, M., Mockus, A.: Effectiveness of code contribution: from patch-based to pull-request-based tools. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE '16, Seattle, WA, USA, November 13-18. pp. 871–882 (2016)