SCIENCE CHINA
Information Sciences

• LETTER •

September 2019, Vol. 62 199105:1–199105:3
https://doi.org/10.1007/s11432-018-9822-1

A novel approach for recommending semantically linkable issues in GitHub projects

Yang ZHANG[1,2], Yiwen WU[1,2*], Tao WANG[1,2] & Huaimin WANG[1,2]

[1]*Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha* 410073*, China;*
[2]*College of Computer, National University of Defense Technology, Changsha* 410073*, China*

Received 12 July 2018/Revised 27 October 2018/Accepted 26 February 2019/Published online 29 July 2019

Citation Zhang Y, Wu Y W, Wang T, et al. A novel approach for recommending semantically linkable issues in GitHub projects. Sci China Inf Sci, 2019, 62(9): 199105, https://doi.org/10.1007/s11432-018-9822-1

Dear editor,

GitHub[1)] is a web-based project hosting platform which was launched in 2008 and has become one of the premier open-source development sites [1]. During the software development process of GitHub projects, issue reports, as an important development knowledge, are likely to be related as they contain relevant information. One of the manifestations is that many open issues in a project get linked to related issues by URL referencing. For a given issue, we refer to its related issues as linkable issues, or L-issues. Informing developers about L-issues can help them find similar bugs, useful resources, and critical information, which are helpful for quickly and efficiently resolving issues. This is especially true for newcomers who have just begun getting involved in the development process. Identifying L-issues can help them gain insights into the relationships between known issues, comprehend development requirements, and avoid duplicate work.

Unfortunately, information regrading L-issues are not readily available; developers often miss them during the issue resolution process. Especially for the large projects, developers might have to investigate a large number of issues, make the connection from the description of the issue(s) in the issue report to the L-issues. This manual linking process costs time and effort, depending on the experience and knowledge of the developers. Although GitHub provides an issue search engine, we initially tried to use it and found that the current search engine of GitHub does not allow one to conveniently find L-issues. Often, many of the top results are not the appropriate L-issues. Thus, providing an automated approach is necessary for locating L-issues to help developers save costs and let them focus on the knowledge related to a particular issue.

Our approach. Our approach works by generating and comparing distributed vectors of different dimensions for issue textual contents. First, we extracted the text information from all issue data. Then, for each issue report, we combined its title and description into a single document. Next, we preprocessed those issue documents via the following steps: (1) extracting all the words from each issue document; (2) removing stop words, numbers, punctuation marks, and other non-alphabetic characters; and (3) using Lancaster Stemmer method to transform the remaining words to their root forms, for reducing the feature dimensions and unifying similar words into a common representation.

Based on the preprocessed issue documents, we trained three vector representation models:

• Term frequency-inverse document frequency (TF-IDF) model. As the basic model, we wish to capture the relationship of word frequency between two issue documents. TF-IDF is one of the most popular information retrieval techniques. The main idea of TF-IDF is that if a term appears

* Corresponding author (email: wuyiwen14@nudt.edu.cn)
1) https://github.com/.

several times in one document and a few times in other documents, the term can be used to differentiate between the documents. Thus, the term has a high TF-IDF value. In our approach, we computed the word frequency similarity score (i.e., wf-score) between two issue documents using cosine similarity, which is a popular method that works well for TF-IDF vectors.

• Word embedding (WE) model. We aim to represent the relationship of words, contextually between issue documents. In this study, we used the skip-gram technique [2] to obtain the word embedding vectors. To learn a target word's representation, this model exploits the notion of context, where a context is defined as a fixed number of words surrounding the target word. To this end, for a given sequence of words, $\{w_1, w_2, \ldots, w_n\}$, the target word, w_i, whose representation must be learned, and for the length of the context window, k, the objective of the skip-gram model maximizes the log-likelihood: $\sum_{i=1}^{n} \log \Pr(w_{i-k}, \ldots, w_{i+k}|w_i)$. $w_{i-k}, \ldots, w_{i+k}$ are the context of the target word w_i. The probability $\Pr(w_{i-k}, \ldots, w_{i+k})$ is computed as $\prod_{-k \leqslant j \leqslant k, j \neq 0} \Pr(w_{i+j}|w_i)$. Here, the context words and target word are assumed to be independent. Furthermore, $\Pr(w_{i+j}|w_i)$ is defined as $\frac{\exp(\boldsymbol{w}_i \cdot \boldsymbol{w}'_{i+j})}{\sum_{w \in W} \exp(\boldsymbol{w}_i \cdot \boldsymbol{w})}$, where $\boldsymbol{w}$ and $\boldsymbol{w}'$ are the input and output vectors of word w and W is the vocabulary of all the words. By training a whole corpus, all words in the vocabulary of the corpus can be represented as a δ-dimensional vector. Then, each word can be transformed into a fixed-length vector. Each issue document can then be represented as a matrix, in which each row represents a word. Note that different issue documents can have different numbers of words, we transformed the document matrix into a vector by averaging all word vectors in the document. In particular, given a document matrix with n rows, we denote the i-th row of the matrix as $\boldsymbol{r}_i$ and the transformed document vector generated as $\frac{\sum_{i=1}^{n} \boldsymbol{r}_i}{n}$. Finally, given two word embedding vectors, we used cosine similarity to measure their word context similarity score (i.e., wc-score).

• Document embedding (DE) model. Our goal is to further compare the relationship between two issue documents by considering their document-level contextual information. In this study, we used paragraph vector-distributed bag of words [3], which is capable of learning representations of arbitrary length word sequences such as sentences, paragraphs, and large documents. Specifically, given a set of documents, $D = \{d_1, d_2, \ldots, d_N\}$ and a sequence of words, $W(d_i) = \{w_1, w_2, \ldots, w_k\}$, sampled from document $d_i \in D$, this model learns δ-dimensional embedding vectors of document d_i and each word, w_j, sampled from $W(d_i)$ ($\boldsymbol{v}_{d_i} \in R^{\delta}$, $\boldsymbol{w}_j \in R^{\delta}$). The model works by considering a word, $w_j \in W(d_i)$, to be occurring in document d_i and maximizes the log-likelihood, $\sum_{j=1}^{k} \log \Pr(w_j|d_i)$. The probability, $\Pr(w_j|d_i)$, is computed as $\frac{\exp(\boldsymbol{v}_{d_i} \cdot \boldsymbol{w}_j)}{\sum_{w_j \in W} \exp(\boldsymbol{v}_{d_i} \cdot \boldsymbol{w}_j)}$, where W is the vocabulary of all words across all documents in D. During model training, we added the issue submitter name to the vectors as a context label. Then, given two document embedding vectors, we calculated their similarity score of the document context (i.e., dc-score) using cosine similarity.

During our testing, we used the trained TF-IDF, WE, and DE models to calculate three similarity scores (i.e., wf-score, wc-score, and dc-score) for the query issue and each of the pending issues. Although all three scores can represent two issues' semantic similarity, they are complementary: (1) wf-score was generated by the TF-IDF model, which focuses more on issues' word frequencies across the whole corpus; (2) wc-score was generated by the WE model, which focuses on the contextual information of words; and (3) dc-score was generated by DE model, which focuses more on the latent relevance of document contexts. Finally, we added them up to produce a final score to better recommend linkable issues. Note that a higher score indicates that the semantic similarity of the corresponding pair of issues is greater.

Experiment setup. For our experiments, we collected more than 6000 actual linked issue pairs from two well-known GitHub projects: jquery[2] and request[3]. The baseline approach was obtained via GitHub's issue search engine[4]. We used the query, "search/issues?q=[Query]+type:issue+repo:[Repo]" and ranked the results by "relevance". Our ground truth was based on actual issue links posted in the issue comments. As with our previous approach [4], we extracted all link information using the "cross-referenced" GitHub API endpoint[5].

Regrading the evaluation metrics, we used top-k recall rate (Rr@k) to check whether a top-k recommendation was useful. If one of the actual

2) https://github.com/jquery/jquery.
3) https://github.com/request/request.
4) https://developer.github.com/v3/search/#search-issues.
5) https://developer.github.com/v3/issues/timeline/.

links of a given issue was within the top-k list, we counted it as a hit. Otherwise, we considered it a miss. In our experiments, in addition to Rr@1, we considered Rr@5 because many developers were willing to check top-5 results in bug localization tasks [5]. To consider the cases of an issue with multiple links, we also measured mean average precision (MAP) and mean reciprocal rank (MRR). These metrics are commonly used for recommendation system evaluations in software engineering tasks [6, 7].

In our experiments, we used an Intel(R) Core (TM) I5 2.6-GHz PC with 8-GB RAM running Windows 8 OS (64-bit). All actual linked issues in our dataset were regarded as query issues. We applied our models using the Python package, gensim[6)]. In this study, we mainly used the default values of gensim for all parameters. Specifically, for WE and DE models, we set the context window size, s, as 5. The initial learning rate, α, was set as 0.025, and the dimension of the vector, d, was set as 200.

Our results. Table 1 presents the experimental results showing the relative improvements our approach achieved over the baseline approach, i.e., GitHub's issue search engine. The Rr@1, Rr@5, MAP, and MRR values of our approach achieved 0.228/0.194, 0.435/0.375, 0.161/0.140, and 0.266/0.226, in the jquery project and the request project, respectively. The relative improvements achieved by our approach from GitHub's search engine were 165.1%/133.7%, 302.8%/200.0%, 163.9%/122.2%, and 192.3%/143.0% for Rr@1, Rr@5, MAP, and MRR, respectively. These results demonstrate that our approach is more effective than GitHub's issue search engine.

Table 1 Performance comparison results. GH's SE: GitHub's search engine; IPM: improvements

Project	Metric	Our approach	GH's SE	IPM (%)
Jquery	Rr@1	0.228	0.086	165.1
	Rr@5	0.435	0.108	302.8
	MAP	0.161	0.061	163.9
	MRR	0.266	0.091	192.3
Request	Rr@1	0.194	0.083	133.7
	Rr@5	0.375	0.125	200.0
	MAP	0.140	0.063	122.2
	MRR	0.226	0.093	143.0

The average Rr@1 value was only 0.211, and the average Rr@5 value was 0.405. In the initial analysis, we observed that many cases that failed when our approach was used involved issues whose essential information was presented as a picture or a code snippet. In future work, we plan to further investigate the failure reasons using qualitative analysis.

Furthermore, we only considered $k = 1, 5$ for the top-k recommendations. We expect to find even more linkable issues if we consider a larger k. We leave these investigations for future work.

Conclusion. This study proposed a novel approach for recommending semantically linkable issues (L-issues) in GitHub projects. Given a query issue, this approach considered issues' word correlation and contextual information, and output a ranked list of L-issues. The preliminary evaluation results demonstrated that this approach can achieve better performance compared to GitHub's default issue search engine. Generally, we considered the application and further development of modern and powerful automatic approaches for recommending linkable issues as an emerging and promising area. Our approach can serve as a foundation for more complicated and specific tasks, such as duplicate issue detection, issue lifetime prediction, concurrency issue categorization, issue classification, issue localization, and issue triaging.

Acknowledgements This work was supported by National Grand R&D Plan (Grant No. 2018YFB1003903) and National Natural Science Foundation of China (Grant No. 61432020).

References

1 Zhang Y, Wang H M, Yin G, et al. Social media in GitHub: the role of @-mention in assisting software development. Sci China Inf Sci, 2017, 60: 032102
2 Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality. In: Proceedings of Advances in Neural Information Processing Systems, 2013. 3111–3119
3 Le Q, Mikolov T. Distributed representations of sentences and documents. In: Proceedings of International Conference on Machine Learning, 2014. 1188–1196
4 Zhang Y, Yu Y, Wang H M, et al. Within-ecosystem issue linking: a large-scale study of rails. In: Proceedings of International Workshop on Software Mining, 2018. 12–19
5 Kochhar P S, Xia X, Lo D, et al. Practitioners' expectations on automated fault localization. In: Proceedings of International Symposium on Software Testing and Analysis, 2016. 165–176
6 Zhou J, Zhang H Y, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proceedings of International Conference on Software Engineering, 2012. 14–24
7 Rocha H, Valente M T, Marques-Neto H, et al. An empirical study on recommendations of similar bugs. In: Proceedings of International Conference on Software Analysis, Evolution, and Reengineering, 2016. 46–56

6) http://radimrehurek.com/gensim.