2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Combining Word2Vec with Revised Vector Space Model for Better Code Retrieval

Thanh Van Nguyen, Anh Tuan Nguyen, Hung Dang Phan, Trong Duc Nguyen
Electrical and Computer Engineering Department
Iowa State University
Email: {thanhng,anhnt,hungphd,trong}@iastate.edu

Tien N. Nguyen
Computer Science Department
University of Texas at Dallas
Email: tien.n.nguyen@utdallas.edu

***Abstract*—API example code search is an important application in software engineering. Traditional approaches to API code search are based on information retrieval. Recent advance in Word2Vec has been applied to support the retrieval of API examples. In this work, we perform a preliminary study that combining traditional IR with Word2Vec achieves better retrieval accuracy. More experiments need to be done to study different types of combination among two lines of approaches.**

***Keywords*-Word2Vec; Information Retrieval; Code Search**

I. Research Problem and Background

Libraries and frameworks become very popular in software development. Developers use Application Programming Interfaces (APIs) provided as part of the libraries and frameworks to extend the functionality of their applications. In software development, not all API usages are well-explained in library documentation. In current practice, developers often have to use code search engines to find the desired API code examples.

Traditional code search/retrieval uses Information Retrieval (IR) techniques with text matching [1], [2], [4]. Other group of IR-based approaches considers the relations among API elements [5], [11], [12], [15]. McMillan *et al.* [8] first locate the set of APIs that are textually similar to the query and then finds code examples cover most of them. Exemplar [5] takes a textual query and uses IR and program analysis to retrieve relevant applications. Portfolio [7] considers also the context of call graphs when taking the given texts as queries. Chan *et al.* [3] model API calls as a graph to find connected subgraphs with nodes having high textual similarity to the query.

This line of solution with IR is not always ideal since in a program, developers could use names that may be different from English queries. For example, one may use the query *"insert an element into an array at a given position"* to find the usage of `List.add`, which shares no common terms with this query. This is referred to as the *lexical mismatch* problem [14].

Recently, deep learning with Word2Vec has been leveraged to address the lexical mismatch in code search/retrieval [10], [14]. Word2Vec [9] was run on software documentation and tutorials to build the vectors for the English terms and API code elements including classes and methods. Since the documentation and tutorials contain both texts and API elements, Word2Vec is able to project English terms and code elements in a shared vector space. Semantic similarity between a textual query and an API example is measured based on the distances of the vectors of terms in the query and code elements in the example.

While Word2Vec has been shown to address the lexical mismatch, the question remains unanswered is that whether the two approaches, IR and Word2Vec, could be combined to achieve better code retrieval accuracy.

II. Retrieval Approaches

A. Word2Vec in Ye et al. [14]

In this study, for the representative Word2Vec-based approach, we chose the one that was used in Ye *et al.* [14] and ran it on documentation to build the vectors for API elements and English texts. The relevance between a query Q and a code example C is decomposed into the relevance of every code element in C with respect to Q, which is computed via the distance between the vectors for each code element and each English term in Q. All examples are then ranked with respect to their scores to find the most relevant ones to the query.

B. Revised Vector Space Model (rVSM) [16]

For IR approaches, we chose an advanced one that were shown to perform well in SE data, rVSM [16]. It was used to locate the potential buggy files based on bug reports' contents. Bug reports and source files are considered as documents and represented as vectors of words. The similarity between a bug report and a source file is measured by the cosine distance between two respective vectors. To process source code, rVSM splits the identifiers into separate words. Then, it collects all the words in bug reports and source files to build a dictionary. The dictionary size is the number of the dimensions of the vectors for the documents. rVSM computes the weight for a word based on a new term frequency-inverse document frequency (*tf-idf*) formula and a new scoring scheme among the vectors that takes documents' lengths into account [16].

C. Combined Technique (rVSM+Word2Vec)

While rVSM relies on lexical matching between query and code, Word2Vec can reveal the relationship among them even when they do not necessarily have textual matching. In some sense, we expect that they could complement each other to enhance the retrieval performance. We hence combine the two

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.90

TABLE I
KODEJAVA RETRIEVAL DATASET

Number of API code examples	437
Number of unique words	644
Number of unique APIs	986
Word occurrence frequency	6.7
API occurrence frequency	4.3
Average length of query	9.8
Average length of code example	11.9

TABLE II
API CODE EXAMPLE RETRIEVAL ACCURACY

	top-1	top-2	top-3	top-4	top-5
Word2Vec	11.7	18.5	22.7	26.3	29.5
rVSM	35.0	46.0	50.8	54.9	56.5
rVSM+Word2Vec	**37.5**	**48.7**	**54.7**	**58.1**	**60.9**

techniques with a new score that is a linear combination of the two scores from Word2Vec and rVSM:

$$\alpha \times norm\_rVSM(Q,C) + (1-\alpha) \times norm\_Word2Vec(Q,C)$$

where $norm\_rVSM(Q,C)$ and $norm\_Word2Vec(Q,C)$ are the rVSM and Word2Vec scores normalized over all code examples, and α is trade-off that controls the importance of the two similarity scores. We adjust the trade-off parameter α based on the Jaccard similarity between the query Q and the code example C. Given two sets of tokens Q and C, Jaccard index is computed as follows:

$$Jaccard\,(Q,C) = \frac{|Q \cap C|}{|Q| + |C| - |Q \cap C|}$$

In our experiment, we chose α empirically:

$$\alpha = \begin{cases} 0 & \text{if } norm\_Jaccard(Q,C) \leq 0.65 \\ 1 & \text{otherwise} \end{cases}$$

The normalized textual similarity between query Q and code C controls the linear combination such that the Word2Vec score is used if $\alpha \leq 0.65$ and the rVSM score is used otherwise. More sophisticated combination, *e.g.,* from adaptive learning as in [13] can also be explored in future work.

III. EXPERIMENT

Data Collection. For the training dataset, we collected JDK documentation with a total of 5,712 unique API elements and 5,679 words. The occurrence frequencies for an API and a word are 3.1 and 22.2 respectively.

For the testing dataset, we collected from KodeJava [6], a tutorial website for Java. A page of tutorials contains a title in English describing a task and fragments of Java code. We picked the pages with a single code fragment. The titles and the corresponding code fragments are used as a ground truth. We have a dataset of 437 pairs of titles (queries) and corresponding examples. (see Table I). For Word2Vec, we set the context window size $C = 5$ and the dimensionality $N = 200$ to learn the vectors. If the correct API example appears in the top-k list of the retrieved API examples, we count it as a correct

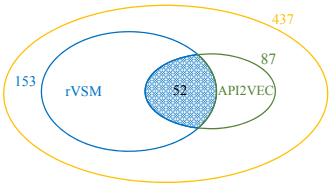

Fig. 1. Comparison on Top-1 Retrieved Examples

TABLE III
TOP-RANKED EXAMPLES OF API EXAMPLE CODE SEARCH

Q	Breaks a text or sentence into words?
R	Locale.US BreakIterator.getWordInstance BreakIterator.setText
Q	Insert an element in array at a given position?
R	List.List Arrays.asList List.addAll List.add List.toArray Arrays.toString
Q	Get the maximum number of concurrent connections?
R	Connection.getMetaData DatabaseMetaData.getMaxConnections
Q	Launch user-default web browser?
R	URI.create Desktop.Desktop Desktop.getDesktop Desktop.browse

case; otherwise, it is a miss. The top-k accuracy is the ratio between the number of hits over the total number of cases.

Results. Table II shows the result. As seen, rVSM performs better than Word2Vec from 15–17%. The combined approach performs the best. From Word2Vec's and rVSM's results, we observed how well they perform with one suggestion (Fig. 1). Word2Vec fails in the cases that it does not seen enough data to observe semantic similarity. It correctly suggests 87 examples, while rVSM achieves 153. Of these examples, 35 cases were retrieved by Word2Vec but not by rVSM. We further studied on textual queries and corresponding code examples and found that rVSM fails in cases of low textual similarities between query and code example. For instance, for the query *"insert an element in array at a given position"*, Word2Vec retrieves the correct code example at top 1 while rVSM does at top 22 due to the mismatch of *"insert"* and List.add, List.addAll. For this reason, the combined score function improves rVSM by 11 out of these 35 cases and the accuracy accordingly increases from 2.5–4% or 5–8% relative to rVSM. That is, the resulting search engine is able to complement well between Word2Vec and state-of-the-art rVSM in the cases with low textual similarities. Table III shows a few top-ranked examples in which the queries have lexical mismatch with the code examples.

IV. CONCLUSION

API code search is a useful application in software engineering. Traditional approaches to API code search are based on information retrieval. Recent advance in Word2Vec has been applied to support the retrieval of API examples. In this paper, we showed a preliminary study that combining traditional IR with Word2Vec in fact achieves better retrieval accuracy.

V. ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) grants CCF-1723215, CCF-1723432, TWC-1723198, CCF-1518897, and CNS-1513263.

REFERENCES

[1] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: A search engine for open source code supporting structure-based search. In OOPSLA '06, pages 681–682. ACM, 2006.

[2] Black Duck Open Hub. http://code.openhub.net/.

[3] W.-K. Chan, H. Cheng, and D. Lo. Searching Connected API Subgraph via Text Phrases. In FSE '12, pages 10:1–10:11. ACM, 2012.

[4] Codase. http://www.codase.com/.

[5] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. A search engine for finding highly relevant applications. In ICSE '10, pages 475–484. ACM, 2010.

[6] Kode java. https://kodejava.org/.

[7] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: Finding relevant functions and their usage. In ICSE'11, pp 111–120. ACM, 2011.

[8] C. McMillan, D. Poshyvanyk, and M. Grechanik. Recommending source code examples via api call usages and documentation. In RSSE '10, pages 21–25. ACM, 2010.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In NIPS'13, pages 3111–3119, 2013.

[10] T. V. Nguyen, A. T. Nguyen, and T. N. Nguyen. Characterizing api elements via textual descriptions in software documentation with vector representation. In ICSE'16 Poster. ACM, 2016.

[11] D. Puppin and F. Silvestri. The social network of java classes. In *SAC'06*, pages 1409–1413. ACM, 2006.

[12] Z. M. Saul, V. Filkov, P. Devanbu, and C. Bird. Recommending random walks. In ESEC-FSE '07, pages 15–24. ACM, 2007.

[13] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In ASE '11, pages 253–262. IEEE, 2011.

[14] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In ICSE '16, pages 404–415. ACM, 2016.

[15] W. Zheng, Q. Zhang, and M. Lyu. Cross-library api recommendation using web search engines. In ESEC/FSE '11, pages 480–483. ACM, 2011.

[16] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In ICSE '12, pages 14–24. IEEE Press, 2012.