Automatically Answering API-Related Questions

Di Wu[1], Xiao-Yuan Jing[1], Haowen Chen[1], Xiaoke Zhu[1], Hongyu Zhang[2],
Mei Zuo[1], Lu Zi[1], Chen Zhu[1]
[1]State Key Laboratory of Software Engineering, School of Computer, Wuhan University, Wuhan, China
[2]School of Electrical Engineering and Computing, The University of Newcastle, Newcastle, Australia

ABSTRACT

Automatically recommending API-related tutorial fragments or Q&A pairs from Stack Overflow (SO) is very helpful for developers, especially when they need to use unfamiliar APIs to complete programming tasks. However, in practice developers are more likely to express the API-related questions using natural language when they do not know the exact name of an unfamiliar API. In this paper, we propose an approach, called SOTU, to automatically find answers for API-related natural language questions (NLQs) from tutorials and SO. We first identify relevant API-related tutorial fragments and extract API-related Q&A pairs from SO. We then construct an API-Answer corpus by combining these two sources of information. For an API-related NLQ given by the developer, we parse it into several potential APIs and then retrieve potential answers from the API-Answer corpus. Finally, we return a list of potential results ranked by their relevancy. Experiments on API-Answer corpus demonstrate the effectiveness of SOTU.

KEYWORDS

Application Programming Interface, Natural Language Question, Tutorials, Stack Overflow

1 INTRODUCTION

Application Programming Interface (API) is commonly used in software development [1, 2, 6]. When encountering unfamiliar APIs, developers tend to seek help from learning resources or discussion forums, such as API tutorials or Stack Overflow (SO).

API tutorial is an important API learning resource, which combines descriptive texts and code examples for the explanation of APIs. It explains the basic usage of different APIs (e.g., functionality, domain concepts or rationales of APIs) in the given programming task. However, the content of fragments segmented from tutorials is rather simple and may not be sufficient for developers who want to know more about the API for their specific programming tasks. In addition, it's difficult to find relevant fragments (if a tutorial fragment explains the usage of an API, they are relevant), because API tutorials are often lengthy and mixed with irrelevant information. Furthermore, all existing methods [3, 4, 7] focus on discovering relevant fragments based on the exact API names (such as org.joda.time.DateTime), rather than API-related NLQs.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194965

In this case, developers often turn to discussion forums such as SO. To get API-related information quickly, developers usually search similar questions in SO. Developers could reuse APIs and corresponding Q&A pairs related to those past questions to complete programming tasks. However, SO often explains how to accomplish a specific programming task with the API, and it lacks the basic API-related information. Thus, for developers, it may be insufficient to only use the Q&A pair to complete programming tasks.

The above observation shows that, considering only one API learning resource (tutorials or SO) may not be able to achieve satisfactory results because of the scarce information of API. In order to help developers find more API-related information, it is necessary to combine these two independent API learning resources. In practice, developers often do not know what APIs to use or simply forget the exact API names. They usually describe their API-related questions using natural language. A more practical and useful API-related information recommendation approach should be able to cater for NLQs, and the returned result should consist of the name of an API and the API-related information. Motivated by these observations, we propose SOTU to automatically find results for API-related NLQs from tutorials and SO. The major contributions of this paper are as follows:

(1) We are among the first to integrate API tutorials and SO together to help developers discover API-related information. We combine relevant API-fragment pairs from tutorials and relevant API-Q&A pairs from SO.

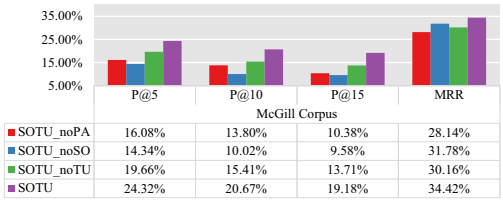
(2) We propose an approach, SOTU, which can automatically find useful answers from tutorials and SO based on API-related NLQs. In SOTU, we first construct API-Answer corpus by combining tutorials and SO, and then identify potential APIs from API-related NLQs. Finally, we return potential results from API-Answer corpus based on potential APIs.

2 THE PROPOSED APPROACH

2.1 The Construction of API-Answer Corpus

The Identification of API-fragment Pairs from Tutorials.

We first generate API-fragment pairs by following the same tutorial segmentation and API extraction strategy as used in [3]. After generating API-fragment pairs, since not all API-fragment pairs are relevant, we use the state-of-the-art method FRAPT [4] to identify whether an API-fragment pair is relevant or not. FRAPT is an unsupervised method, which intends to find relevant tutorial fragments. Specifically, FRAPT first introduces a fragment parser to identify APIs in tutorial fragments, replace ambiguous pronouns and variables with related ontologies and API names. FRAPT then designs a fragment filter, which is used to remove non-explanatory fragments according to non-explanatory detection rules. Next, FRAPT computes topic model score and PageRank score of each API-fragment



	P@5	P@10	P@15	MRR
	McGill Corpus			
SOTU_noPA	16.08%	13.80%	10.38%	28.14%
SOTU_noSO	14.34%	10.02%	9.58%	31.78%
SOTU_noTU	19.66%	15.41%	13.71%	30.16%
SOTU	24.32%	20.67%	19.18%	34.42%

Figure 1: Average Results Between SOTU and Other Compared Approaches

pair. Finally, FRAPT identifies relevant API-fragment pairs when their two scores are all larger than the threshold simultaneously. After identifying the relevance of API-fragment pairs, we choose relevant API-fragment pairs as part of API-Answer corpus.

The Extraction of API-Q&A Pairs from SO. In SO, the question describes the requirement of a programming task, and the APIs appeared in the corresponding accepted answer actually meet the requirement in the question. Thus, we assume that an API in the accepted answer is relevant to its corresponding Q&A pair. According to above observation, we first select questions tagged as the tutorial name to generate the subset of SO. For each subset of SO, we then collect the question and the corresponding accepted answer to construct the Q&A pair. To discover APIs, we extract content of the accepted answer, and split the content into tokens. We then match all API class names covered by the corresponding tutorial with each token. After that, we extract APIs to generate API-Q&A pairs.

Once we have obtained the relevant API-fragment pairs from tutorials and the API-Q&A pairs from SO, we construct an API-Answer corpus by combining these two sources of information.

2.2 The Retrieve of Corpus for Answering API-Related NLQs

Parsing API-Related NLQs. We first obtain all APIs associated with the tutorial, and collect the full API names and their corresponding descriptions from API specification (JavaDoc). Here, we treat the NLQs, the full API names and their descriptions as natural language documents. We perform pre-processing(tokenization, stop-word removal and stemming) for above documents. After that, we use the standard Vector Space Model (VSM) [5] to build the vector for each document. For a vector, each value corresponds to the weight of a word in a document. We then calculate the cosine similarity between the API description and the NLQ (s_{desc}) as well as the cosine similarity between the full API name and the NLQ (s_{name}). APIs with top-20 s_{desc} and s_{name} are selected to construct subsets, denoted as API_d and API_n. We split API_d and API_n into three subsets (subset1: APIs appearing in both API_d and API_n; subset2 or subset3: APIs only appearing in API_d or API_n). We denote the score of API_i as S_i. If API_i appears in subset1, then S_i equals to $s^i_{desc} + s^i_{name}$. If API_i appears in subset2 or subset3, then S_i equals to $(\min(s^i_{desc}, s^i_{name}) \times s^i_{desc})/(\max(s^i_{desc}, s^i_{name}) + \alpha)$, or $(\min(s^i_{desc}, s^i_{name}) \times s^i_{name})/(\max(s^i_{desc}, s^i_{name}) + \alpha)$. α is an adjustment factor to ensure that the score of subset1 is larger than other subsets, and we set α as 0.1. Finally, we return the top-20 APIs as potential relevant APIs.

Retrieval Answers Based on APIs. Once a potential API list is generated, we can choose potential results from API-Answer corpus by $C_i = corr^i_{api} \times corr^i_{ans1}$. C_i is overall similarity between question and the i^{th} potential result. $corr^i_{api}$ is the similarity between question and the potential API in the i^{th} potential results, which is computed by using S_i. $corr^i_{ans1}$ represents the textual similarity between API and the potential results. We compute $corr^i_{ans1}$ using VSM and cosine similarity.

3 EXPERIMENT

In our experiment, API-Answer corpus and Question set are taken to evaluate the performance of SOTU. *1) API-Answer Corpus.* API-Answer corpus combines tutorials and their corresponding SO. We employ 5 tutorials from McGill corpus[7], which consists of JodaTime, Math, Official Collections, Jenkov Collections and Smack APIs. For SO, we select 4661 API-Q&A pairs in total. We then construct an API-Answer corpus by combining these two sources of information. *2) Question Set.* We collect 30 real-world questions from SO and FAQ of API library for each tutorial. Finally, 150 questions are collected in total. In this paper, we employ P@k (Precision@k) and MRR to evaluate the performance of SOTU. We compare SOTU with **SOTU_noPA**, which deletes the identification potential APIs component. We also compare SOTU with **SOTU_noSO** and **SOTU_noTU**, which remove SO and tutorial fragments from the API-Answer corpus, respectively. Figure 1 shows average results between SOTU and other compared approaches on McGill corpus. From the Figure 1, SOTU can outperform other approaches on McGill corpus.

4 CONCLUSION

In this paper, we integrate the tutorials and SO to help developers find more API-related information. To our knowledge, we are the first to study the combination of API tutorials and SO. We then propose an approach, SOTU, to automatically find useful answers from tutorials and SO based on API-related NLQs. The experimental results confirm the usefulness of SOTU.

ACKNOWLEDGMENTS

This work was supported by the NSFC-Key Project of General Technology Fundamental Research United Fund under Grant No. U1736211, by the NSFC under Grant Nos. 61672208,U1504611,41571417.

REFERENCES

[1] Ekwa Duala-Ekoko and Martin P. Robillard. 2012. Asking and answering questions about unfamiliar APIs: An exploratory study. In *International Conference on Software Engineering*. 266–276.
[2] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 631–642.
[3] He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, and David Lo. 2016. A More Accurate Model for Finding Tutorial Segments Explaining APIs. In *International Conference on Software Analysis, Evolution, and Reengineering*. 157–167.
[4] He Jiang, Jingxuan Zhang, Zhilei Ren, and Tao Zhang. 2017. An unsupervised approach for discovering relevant tutorial fragments for APIs. In *International Conference on Software Engineering*. 38–48.
[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press.
[6] Tam The Nguyen, Hung Viet Pham, Phong Minh Vu, and Tung Thanh Nguyen. 2016. Learning API usages from bytecode: a statistical approach. In *International Conference on Software Engineering*. 416–427.
[7] Gayane Petrosyan, Martin P. Robillard, and Renato De Mori. 2015. Discovering Information Explaining API Types Using Text Classification. In *International Conference on Software Engineering*. 869–879.