SimilarAPI: Mining Analogical APIs for Library Migration

Chunyang Chen
Faculty of Information Technology, Monash University
Melbourne, Australia
Chunyang.Chen@monash.edu

ABSTRACT

Establishing API mappings between libraries is a prerequisite step for library migration tasks. Manually establishing API mappings is tedious due to the large number of APIs to be examined, and existing methods based on supervised learning requires unavailable already-ported or functionality similar applications. Therefore, we propose an unsupervised deep learning based approach to embed both API usage semantics and API description (name and document) semantics into vector space for inferring likely analogical API mappings between libraries. We implement a proof-of-concept website SimilarAPI (https://similarapi.appspot.com) which can recommend analogical APIs for 583,501 APIs of 111 pairs of analogical Java libraries with diverse functionalities. Video: https://youtu.be/EAwD6l24vLQ

KEYWORDS

Analogical API, Word embedding, Skip thoughts

ACM Reference Format:
Chunyang Chen. 2020. SimilarAPI: Mining Analogical APIs for Library Migration. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3382140

1 INTRODUCTION

Third-party libraries are an integral part of many software systems [27]. When a library that a software project currently uses is no longer under active development, or lacks certain desired features, or cannot satisfy performance requirements, developers of the project often need to migrate the project from the currently-used library to the others [26]. This migration activity is referred to as *library migration*. To complete a library migration task, the developer needs to first find a good analogical library that provides comparable features as the currently-used library. In this setting, we refer to the currently-used library as *source library*, and the analogical library as *target library*. Then, developers must establish the mappings of analogical APIs between libraries that implement the same or similar feature. Although analogical libraries can be effectively recommended [3], establishing analogical API mappings between libraries lack effective support. Without the knowledge of analogical API mappings, it is impossible to port the code from using the source library APIs to using the target library APIs.

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3382140

Some API mapping databases have been created by domain experts well versed in the source and target APIs, but only for a few source/target platform pairs, such as Android, iOS and Symbian Qt to Windows 7 [2]. For an arbitrary pair of source and target libraries, developers have to manually find analogical APIs between the two libraries by studying and comparing API semantics. Usually, there are two types of API semantics to investigate, i.e., source codes that demonstrate the usage of APIs (*API usage semantics*), and API description that explain the API functionalities (*API description semantics*). This investigation process is known to be tedious and error-prone [14], especially when developers are unfamiliar with the target library and the target library has tons of APIs [4, 6].

Having an automatic technique to create a database of likely API mappings between analogical source-target libraries can significantly ease the task. However, existing techniques have two fundamental limitations. First, existing techniques [17, 24, 29] require already ported or functionality similar applications for inferring likely API mappings between libraries. As a library often has several analogical libraries [3, 5, 8], it is unlikely to have already ported or functionality similar applications for an arbitrary pair of analogical source-target libraries. Second, some techniques [25] measure the textual similarity of the source and target API descriptions (names/documents) to discover likely API mappings. However, the descriptions of even the equivalent APIs written by different developers may share few words in common (i.e., have lexical gap). For example, two similar APIs may have very different descriptions such as *"Clear all entries in the MDC of the underlying implementation"* and *"Remove all values from the MDC"*. Traditional IR methods are not robust to recognize such API mappings when API names/documents have lexical gaps.

To overcome the above limitations in creating the database of likely API mappings for pairs of analogical source-target libraries, we present an approach that exploits unsupervised deep learning techniques to embed API usage semantics and API description semantics to infer likely API mappings between analogical source-target libraries. In this work, we focus on API mappings between a source API method and a target API method. Our key technical contributions are two-fold: 1) Embedding API usage semantics by unsupervised API and library embeddings. 2) Embedding API name/description semantics by unsupervised RNN embedded sentence vectors. Based on these three different kinds of embedding, we combine them for inferring analogical APIs across similar libraries.

As a proof of concept, we choose 97 Java libraries and their analogical Java libraries (in total 111 pairs of analogical source-target libraries and 583,501 API methods) from the knowledge base of analogical libraries [3]. These libraries support a diverse set of functionalities, including testing, data visualization, GUI development, information retrieval, etc. We discover likely analogical APIs

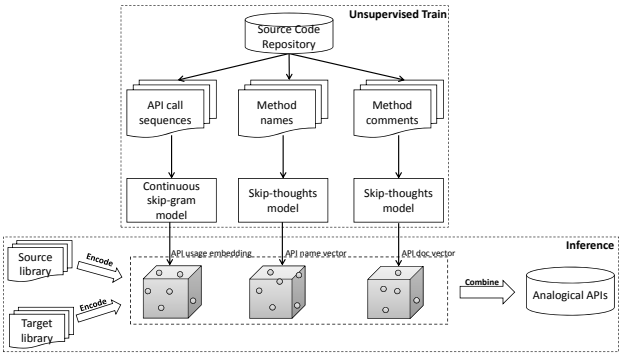

Figure 1: The overall framework of our approach

between these pairs of analogical libraries and build a web application SimilarAPI (https://similarapi.appspot.com[1]) for searching analogical APIs across these pairs of libraries.

By recording and analyzing the site visiting statistics via Google Analytics, it shows that more than 800 users have visited our site. From the visiting logs, we also observe some usage patterns of our website which can help address different information needs of developers in searching for analogical APIs. These quantitative and qualitative analysis demonstrate the usefulness of our tool.

Our contributions are listed as follows:

- We present the first reliable unsupervised approach to incorporate both API usage and description similarities to create the database of likely analogical API mappings.
- We build a large database of analogical API mappings for 111 pairs of analogical libraries and implement a web application for accessing the database.

2 APPROACH

Figure1 displays the overall approach including an unsupervised training phase and an inference phase. In the training phase, we crawl GitHub repositories and extract API call sequences, method names and comments from source code. We consider APIs as words and apply continuous skip-gram model to a corpus of API call sequences to learn API usage embeddings. Furthermore, based on the corpus of method names/comments, we train a name/comment skip-thoughts model [20] for obtaining the name/comment embedding. In the inference phase, our approach creates a database of likely API-method mappings between analogical libraries by combining API usage, name and document similarities. The detailed approach can be seen in our previous paper [9].

2.1 Encoding API Usage Similarity

Given the crawled GitHub projects, we extract API call sequences for each file. Note that we extract fully-qualified names from the crawled code using a partial program analysis (PPA) [13] tool for Java. We consider an API call sequence as a sentence, and each API method as a word. Studies [10, 12, 21] show that word embeddings are able to capture rich semantic and syntactic properties of software-specific words for measuring word similarity. Therefore, we develop a word-embedding based method [11, 19] to embed API usage semantics in a vector space based on the use of an API, together with other APIs (not necessarily from the same library), in API call sequences extracted from the code. Relational similarity between pairs of words is a metric for reasoning about analogical words [22, 23]. Chen et al [3] shows that library-language relational similarity (e.g., "NLTK:Python" and "OpenNLP:Java") performs better in inferring analogical libraries than directly using the cosine similarity of libraries, based on the tag embeddings learned from short and diverse question tag sequences. The obtained API vectors are then used to quantify API usage similarity between a source API method and a target API method.

We find that this observation also extends to analogical APIs between libraries. Therefore, in our approach, we use API-library relational similarity to measure the usage similarity between an API (a_s) in the source library (L_s) and an API (a_t) in the target library (L_t). This relational similarity can be computed by vector arithmetic as follows:

$$sim_{use} = \cos(uvec(a_s) - uvec(L_s), uvec(a_t) - uvec(L_t)) \quad (1)$$

where $uvec()$ (usage vector) is API embedding or library embedding, vector offset of API embedding and library embedding reflects the API-library relation (line) in the vector space, and relational similarity is computed as cosine similarity between the two vector offsets (i.e., the angle between the two API-library lines).

2.2 Encoding API Name & Description Similarity

We adopt skip thoughts [20] (an unsupervised RNN model) to encode API names (or documents) in a vector space. Skip-thoughts model is a recently proposed unsupervised RNN model that encodes the semantics of a sentence to vector by not only the words in a sentence, but also the surrounding sentences of the sentence in a document. It consists of an RNN encoder and two RNN decoders [7, 16, 28]. The encoder encodes words of a sentence to a sentence vector. Two decoders decode this sentence vector to the previous sentence and the next sentence in a document respectively.

As the name (or document) of an API describes the functionality of the API, the training code-related text should contain sentences serving the similar purpose. Therefore, we extract a method-name corpus and a method-comment corpus from the source code crawled from Github, which is used to train a method-name skip-thought model and a method-comment skip-thought model, respectively. According to the learning mechanism of skip-thoughts model, the training corpus should contain documents with a sequence of sentences. In this work, we construct a method-name (or method-comment) document for each code file, because a code file should group a set of closely-related methods. Inspired by the studies on the naturalness of source code [18], we order the names (or comment sentences) extracted from methods declared in a code file in the same order as method declarations. The intuition is that developers declare methods in an order according to certain natural relatedness among methods, and this natural relatedness is reflected in the order of the names (or comments) describing the methods' functionalities. That is, the surrounding names (or comments) of a name (or comment) provide the context to embed the semantics of the given name (or comment). After the training of skip-thought

[1]As this website is host on Google Cloud, you may not visit our site if you cannot access to Google.

model, we can obtain the name and document similarity of any two APIs across different libraries.

2.3 Building Analogical-API Knowledge Base

Given a source library, we search the knowledge base of analogical libraries [3] to find its analogical library i.e., target library. We determine the likelihood of an API method in the source library and an API method in the target library being analogical by combining three information similarities of the two APIs, i.e., the similarity of API names (sim_{name}), API documents (sim_{doc}), and API usage (sim_{usage}) into an overall similarity score:

$$Sim(a_s, a_t) = \alpha \times sim_{usage}(a_s, a_t) + \beta \times sim_{doc}(a_s, a_t) + \gamma \times sim_{name}(a_s, a_t) \quad (2)$$

where α, β, γ are the weight parameters in $(0, 1)$, and we make $\alpha + \beta + \gamma = 1$ so that the final similarity value is of the range $[-1, 1]$ where 1 is the most similar and -1 is the opposite. These weight parameters are tuned using a small evaluation set of analogical API mappings in the tool implementation.

For each API method in the source library, we rank the API methods in the target library by their overall similarity with the given source API. Then, we obtain a knowledge base of likely analogical API mappings for a pair of analogical source-target libraries.

3 TOOL SUPPORT

To train our skip-thought model and API embedding model, we downloaded 135,127 Java projects from Boa dataset [15]. In our work, we remove few-star (< 10) or deleted projects and obtain 135,127 Java projects as our dataset. These projects contain 2,058,240 source-code files. We extract API call sequences from the crawled code using a partial program analysis (PPA) [13] tool for Java. We collect a corpus of 10,554,900 API call sequences which includes 952,829 unique APIs for learning API embeddings. We use the Eclipse JDT compiler [1] to extract method names and comments (Javadoc format) from the crawled code. We obtain a corpus of 26,622,034 method names and a corpus of 2,798,837 comments for training the method-name and method-comment skip-thoughts models respectively. Note that method declarations always have a name, but they may not have comment and we also ignore those with irregular comments. Based on trained skip-thought and API embedding models, we then infer analogical APIs between 111 pairs[2] of analogical Java libraries like *(gson & jackson), (testng & junit), (pdfbox & itext).*

We implement our approach in an analogical-APIs searching web application[3] based on the knowledge base of likely analogical APIs between 111 pairs of analogical Java libraries. We host this web application on Google Cloud for easier access for developers. Figure 2 shows a screen shot of our web application. When searching an API, our tool will first return the documentation of this API, and also recommend a list of analogical APIs (with the similarity to the given library above the minimal similarity threshold) across the equivalent libraries in Java. In the current implementation, SimilarAPI presents up to 5 APIs with the highest similarity for each similar library, as developers are unlikely to look through a long list of recommendations. Note that listing up to 5 similar APIs is only an implementation decision, not a limitation of our approach.

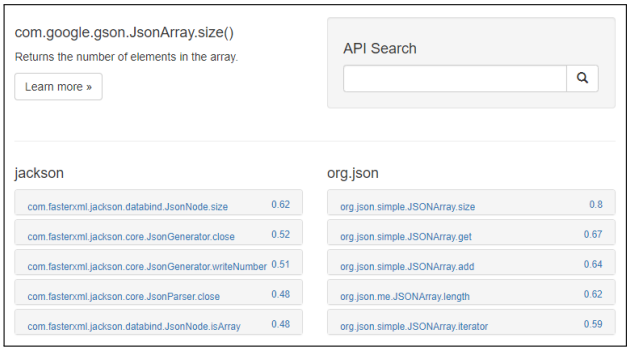

Figure 2: The screenshot of our site

4 EVALUATION

The accuracy and generality of extracted analogical API across different libraries are well documented in our previous paper [9]. In this work, we report the visiting logs of our site by Google Analytics, and demonstrate one typical usage scenario of our SimilarAPI.

4.1 Tool Usage

After implementing our site, we release it to the public. According to the Google Analytics about the site traffic, more than 800 users from 75 countries visit our site, from Jul 2018 to Nov 2019. These users visit 1,270 pages of our site with 1.44 pages in average for each session[4]. Note that most users visit our site from the Google Search with very specific query, so the average visit page in each session will be low as they can get what they want in the first several pages. 47 (5.8%) of 802 users are returning users, indicating users' consistent interest in our tool. By analyzing the detailed visiting log, we find that the top visited pages are about the alternatives of API org.apache.commons.io.FileUtils.writeStringToFile(), org.apache.log4j.MDC.put(), com.google.gson.JsonElement.getAsBigInteger(), org.apache.pdfbox.pdmodel.font.PDFont.getFontDescriptor(), etc. These real-world usage of our site demonstrate the usefulness of our work.

4.2 Usage Scenario

After quantitative analysis of the website visit data, we further qualitatively analyze user navigation patterns in our site. The primary goal of our tool is to help developers find analogical API in a specific library to that of another similar library. For example, one developer is going to migrate her code from using *gson* to *jackson* due to various reasons. When the developer is revising the code, she is searching alternatives to com.google.gson.JsonArrary.size() in our site. As seen in Figure 2, there are five most similar APIs listed with corresponding similarity score annotated. She can click the first recommendation for more documentations to see if the first one com.fasterxml.jackson.databind.JsonNode.size() may be the one that she needs. Once she is not sure, she can take the result from our recommendation as the seed to further search more information in that documents or refer to the search engine.

[2] All pairs can be seen in https://similarapi.appspot.com/allLibPair.html
[3] https://similarapi.appspot.com

[4] These visits are not triggered by the robots as the most crawlers cannot activate the Javascript code snippet embedded in the page for Google Analytics

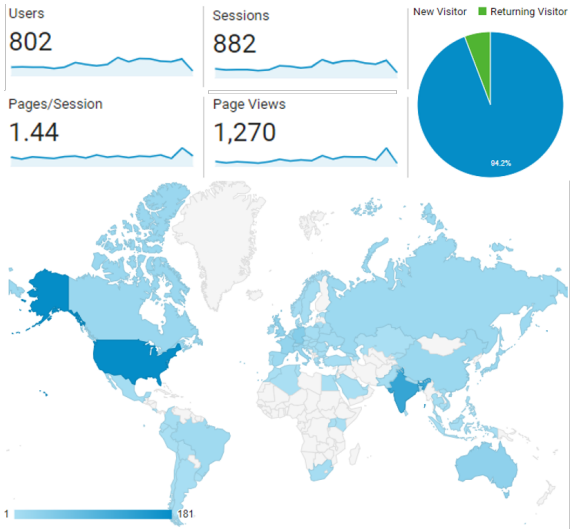

Figure 3: The visit history of our site from Google Analytics

Apart from *jackson*, we also provide alternative APIs in another similar library *org.json*. For example, the first recommended API org.json.simple.JSONArrary.size() may also satisfy developers' need. Note that although developers may not intend to migrate to that library, she may find that many APIs have exact mappings between *gson* and *org.json* in our sites which inspire her to reconsider which similar library for her project to migrate to.

5 CONCLUSION & FUTURE WORK

This paper presents a novel tool for recommending likely analogical API mappings between third-party libraries. Our approach incorporate three kinds of information modeling API usage, API name and documents for inferring API mappings. Our approach is the first attempt to incorporate all API usage, name and document similarities for inferring likely analogical APIs of third-party libraries. To evaluate our approach, we build the largest database of analogical APIs for 583,501 APIs of 111 pairs of analogical Java libraries.

In the future, we will extend our work to one-to-many and many-to-many API mappings which complements our current one-to-one mapping. Furthermore, we will extend our current tool to recommend analogical APIs between libraries written in different programming languages, by developing API-call-sequences and code-related-text extractors for different languages.

REFERENCES

[1] 2017. Eclipse JDT. http://www.eclipse.org/jdt/.

[2] 2017. Windows phone interoperability: Windows phone API mapping. http://windowsphone.interoperabilitybridges.com/porting.

[3] Chunyang Chen, Sa Gao, and Zhenchang Xing. 2016. Mining Analogical Libraries in Q&A Discussions–Incorporating Relational and Categorical Knowledge into Word Embedding. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 338–348.

[4] Chunyang Chen and Zhenchang Xing. 2016. Mining technology landscape from stack overflow. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 14.

[5] Chunyang Chen and Zhenchang Xing. 2016. SimilarTech: automatically recommend analogical libraries across different programming languages. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 834–839.

[6] Chunyang Chen, Zhenchang Xing, and Lei Han. 2016. Techland: Assisting technology landscape inquiries with insights from stack overflow. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 356–366.

[7] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2017. By the community & for the community: a deep learning approach to assist collaborative editing in q&a sites. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 32:1–32:21.

[8] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2018. What's Spain's Paris? Mining analogical libraries from Q&A discussions. *Empirical Software Engineering* (2018), 1–40.

[9] Chunyang Chen, Zhenchang Xing, Yang Liu, and Kent Long Xiong Ong. 2019. Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding. *IEEE Transactions on Software Engineering* (2019).

[10] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 450–461.

[11] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 744–755.

[12] Xiang Chen, Chunyang Chen, Dun Zhang, and Zhenchang Xing. 2019. SEthesaurus: WordNet in Software Engineering. *IEEE Transactions on Software Engineering* (2019).

[13] Barthélémy Dagenais and Laurie Hendren. 2008. Enabling static analysis for partial java programs. In *ACM Sigplan Notices*, Vol. 43. ACM, 313–328.

[14] Ekwa Duala-Ekoko and Martin P Robillard. 2012. Asking and answering questions about unfamiliar APIs: an exploratory study. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 266–276.

[15] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2015. Boa: Ultra-large-scale software repository and source-code mining. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 1 (2015), 7.

[16] Sa Gao, Chunyang Chen, Zhenchang Xing, Yukun Ma, Wen Song, and Shang-Wei Lin. 2019. A Neural Model for Method Name Generation from Functional Description. In *2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE.

[17] Amruta Gokhale, Vinod Ganapathy, and Yogesh Padmanaban. 2013. Inferring likely mappings between APIs. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 82–91.

[18] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 837–847.

[19] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell them apart: distilling technology differences from crowd-scale comparison discussions. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. ACM. 214–224.

[20] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in neural information processing systems*. 3294–3302.

[21] Suyu Ma, Zhenchang Xing, Chunyang Chen, Cheng Chen, Lizhen Qu, and Guoqiang Li. 2019. Easy-to-Deploy API Extraction by Multi-Level Feature Embedding and Transfer Learning. *IEEE Transactions on Software Engineering* (2019).

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[24] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N Nguyen. 2017. Exploring API embedding for API usages and applications. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 438–449.

[25] Rahul Pandita, Raoul Praful Jetley, Sithu D Sudarsan, and Laurie Williams. 2015. Discovering likely mappings between APIs using text mining. In *Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on*. IEEE, 231–240.

[26] Cédric Teyton, Jean-Rémy Falleri, and Xavier Blanc. 2012. Mining library migration graphs. In *2012 19th Working Conference on Reverse Engineering*. IEEE, 289–298.

[27] Ferdian Thung, LO David, and Julia Lawall. 2013. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering (WCRE 2013): Proceedings: Koblenz, Germany, 14-17 October 2013*. 182–191.

[28] Xu Wang, Chunyang Chen, and Zhenchang Xing. 2019. Domain-specific machine translation with recurrent neural network for software localization. *Empirical Software Engineering* 24, 6 (2019), 3514–3545.

[29] Hao Zhong, Suresh Thummalapenta, Tao Xie, Lu Zhang, and Qing Wang. 2010. Mining API mapping for language migration. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 195–204.