2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering

Mining Analogical Libraries in Q&A Discussions

— Incorporating Relational and Categorical Knowledge into Word Embedding

Chunyang Chen, Sa Gao, Zhenchang Xing
School of Computer Engineering, Nanyang Technological University, Singapore
chen0966@e.ntu.edu.sg; gaos0011@e.ntu.edu.sg; zcxing@ntu.edu.sg

***Abstract*—Third-party libraries are an integral part of many software projects. It often happens that developers need to find analogical libraries that can provide comparable features to the libraries they are already familiar with. Existing methods to find analogical libraries are limited by the community-curated list of libraries, blogs, or Q&A posts, which often contain overwhelming or out-of-date information. In this paper, we present a new approach to recommend analogical libraries based on a knowledge base of analogical libraries mined from tags of millions of Stack Overflow questions. The novelty of our approach is to solve analogical-libraries questions by combining state-of-the-art word embedding technique and domain-specific relational and categorical knowledge mined from Stack Overflow. We implement our approach in a proof-of-concept web application (https://graphofknowledge.appspot.com/similartech). The evaluation results show that our approach can make accurate recommendation of analogical libraries (Precision@1=0.81 and Precision@5=0.67). Google Analytics of the website traffic provides initial evidence of the potential usefulness of our web application for software developers.**

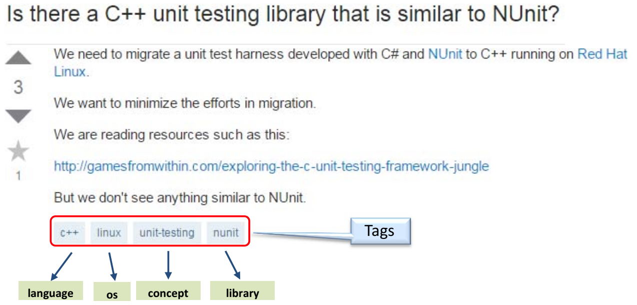
***Keywords*-Analogical libraries; Word embedding; Knowledge graph; Relational knowledge; Categorical knowledge;**

I. Introduction

Third-party libraries are an integral part of many software systems. Thung et al. [1] show that among 1,008 projects in GitHub they investigate, 93.3% of which use third-party libraries, at an average of 28 third-party libraries per project. It often happens that a developer needs some analogical libraries that can provide features comparable to the libraries he is already familiar with. Fig. 1 presents an example of such information need.

Sometimes, the library that a developer currently uses is no longer under active development, or lacks certain desired features, or cannot satisfy performance requirements. In such cases, the developer wants to find some good replacements [2]. In other cases, the developer switches to a new programming language, but he would like to "reuse" his good experience with some libraries that he is familiar with [3], [4] (such as the example shown in Fig. 1). Even though some libraries provide interfaces to multiple programming languages, most libraries are implemented in only one language and work the best with that language. It would be desirable to find analogical libraries that are best suited for the new programming language that the developer switches to.

Developers can search the Internet for analogical libraries. They could find useful information in some community-curated list of libraries, such as *unit testing framework on Wikipedia*[1], and *Awesome PHP on Github*[2]. These library lists are usually very comprehensive, but they often contain many more or less crappy libraries. Developers may also find useful information in blogs (e.g., *"Beyond JUnit - Testing Frameworks Alternatives"*[3]) or forum posts (e.g., *"Alternatives to JUnit"*[4]). Blogs and forum posts are usually more focused, but they are often opinion-based and contain out-of-date information. When developers cannot find satisfactory information on the Internet or want to confirm their search findings, they may ask on Q&A web sites like Stack Overflow (such as the example shown in Fig. 1), but may not get the immediate answers.



Is there a C++ unit testing library that is similar to NUnit?

We need to migrate a unit test harness developed with C# and NUnit to C++ running on Red Hat Linux.

We want to minimize the efforts in migration.

We are reading resources such as this:

http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle

But we don't see anything similar to NUnit.

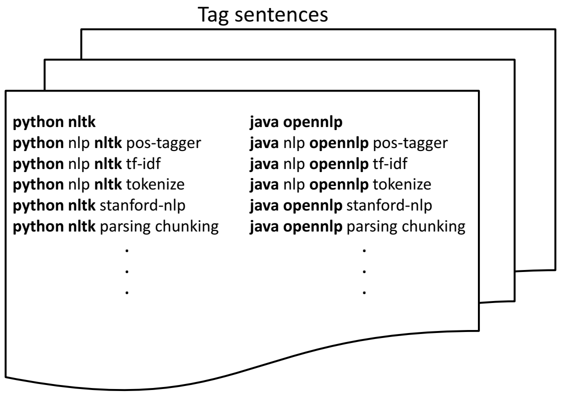
Fig. 1. An exmaple of analogical-library question on Stack Overflow.

In this paper, we present a new approach to find analogical libraries. Our approach is based on the empirical findings showing that taken in aggregate posts on Stack Overflow act as a knowledge repository of developers' practices and thoughts [5], and that the main technologies or constructs that a question revolves around can usually be identified from question tags [6] (see Fig. 1). Instead of listing dozens of crappy libraries or relying on blogs or Q&A posts, our approach recommends analogical libraries based on a knowledge base of analogical libraries mined from tags of millions of Stack Overflow questions. This knowledge base is like forever evolving blog posts about good analogical libraries to the libraries that one is familiar with.

Our approach is motivated by the recent success of neural network language models in Natural Language Processing

[1] https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
[2] https://github.com/ziadoz/awesome-php
[3] http://www.javacodegeeks.com/2012/04/beyond-junit-testing-frameworks.html
[4] http://www.coderanch.com/t/95225/Testing/Alternatives-JUnit

978-1-5090-1855-0/16 $31.00 © 2016 IEEE
DOI 10.1109/SANER.2016.21



python nltk | **java opennlp**
python nlp **nltk** pos-tagger | **java** nlp **opennlp** pos-tagger
python nlp **nltk** tf-idf | **java** nlp **opennlp** tf-idf
python nlp **nltk** tokenize | **java** nlp **opennlp** tokenize
python nltk stanford-nlp | **java opennlp** stanford-nlp
python nltk parsing chunking | **java opennlp** parsing chunking
. | .
. | .
. | .

Fig. 2. The similar context of Python's nltk and Java's opennlp in tag sentences

(NLP) applications [7], [8]. Recently, Mikolov et al. [9] and Turney [10] demonstrate that neural network language models are able to learn word representations (or word embeddings) that can be exploited to solve analogy questions of the form "a is to A as ? is to B", for example, "Paris is to France as ? is to Spain". The unknown word "?" can be inferred from the words (e.g., Madrid) whose word embedding is most similar to the resulting vector of vector arithmetic $a - A + B$ (e.g., $Paris - France + Spain$)

In our approach, we consider tags of a Stack Overflow question as a tag sentence, and each tag as a word in the sentence. As illustrated in Fig. 2, analogical libraries (such as Python's nltk and Java's opennlp) would share similar context in tag sentences, such as common concepts and techniques. Given a corpus of tag sentences derived from Stack Overflow questions, we use continuous skip-gram learning algorithm [7] to learn the word representation of each tag using the surrounding context of the tag in the corpus of tag sentences. Given a library (e.g., *python*'s *nltk*), we reduce the problem of finding analogical libraries for a programming language (e.g., *java*) as *nltk* for *python* to a K-nearest-neighbor search for the tags (e.g., *opennlp*) whose word representation is the most similar to the vector $nltk - python + java$ in the resulting word embedding space.

However, application of neural network language models to the problem of learning tag embeddings in tag sentences, as opposed to learning word representations in everyday NLP problems, brings unique challenges. First, contrary to everyday language where linguistic rules and notions of words and sentences are clearly defined, question tags on Stack Overflow are composed of only up-to five terms where there is no existing notion of the surrounding context equivalent to natural language domain. Second, question tags could be noisy or biased such that they cannot reflect the inherent relationship between tags and further mislead the learning process.

To address these challenges, we incorporate domain-specific relational and categorical knowledge into tag embeddings in order to produce better mappings of analogical libraries. In our approach, relational knowledge encodes the correlation between tags. We use association rule mining [11] to mine the correlation between tags from tag co-occurrence patterns in millions of Stack Overflow questions. Categorical knowledge encodes the category of tags (e.g., library, framework, concept, platform, database, and so on). We use Part-of-Speech tagging and phrase chunking methods [12] to analyze the TagWiki description of each tag to determine the category of the tags. Both relational and categorical knowledge can serve as valuable external information to help differentiate library-program-language pairs with analogy relationships, even if there is little context information or biased/noisy context information in tag sentences.

We implement our approach in a proof-of-concept web application (https://graphofknowledge.appspot.com/similartech). The application takes as input a library name and recommends analogical libraries for different programming languages. The backend analogical-libraries knowledge base is built using the latest Stack Overflow data dump. We evaluate the analogical-libraries recommendations for randomly selected 100 libraries using our approach. The results show that our approach can make accurate recommendation of analogical libraries (Precision@1=0.81 and Precision@5=0.67). Furthermore, Google Analytics of the website traffic provides initial evidence of the potential usefulness of our web application for software developers.

We make the following contributions in this work:

- We formulate analogical-libraries recommendation as a NLP analogy task and adapt the cutting-edge word embedding technique to infer analogical libraries.
- We mine domain-specific relational and categorical knowledge from Stack Overflow and use the mined knowledge to enhance analogical-libraries reasoning.
- We implement our approach in a working web application and evaluate the quality of analogical-libraries recommendation using our approach.

II. Related Work

Recommendation systems are widely utilized in Software Engineering context. Many applications have been proposed to recommend code snippets for developers, such as Jungloid [13], ParseWeb [14], MAPO [15]. Chan et al. [16] and Thung et al. [17] recommend API methods according to natural-language queries. In industry, code search engines have been developed (such as Google code, OpenHub) for developers to search code on the Internet. Compared with these code-level recommendation systems, our approach works at a different level of granularity i.e., library-level, and recommends analogical third-party libraries to the developers.

Language migration is a common phenomenon for developers as they may have to switch from one programming language to another according to the task requirements. The biggest challenge is usually the code and library migration, rather than learning a new language itself[5]. Many researchers

[5] http://stackoverflow.com/questions/212151/

have proposed methods to overcome the code migration challenge, such as code mapping [18], function mapping [19], and API migration [3], [4]. In contrast to these code-level migration approaches, our approach supports library-level migration.

Thung et al. [1] analyze the library co-occurrence patterns in software projects to recommend relevant libraries for a software project. Teyton et al. [2] analyze the evolution of projects' dependencies on third-party libraries to recommend libraries that can replace an existing library in a software project. Different from these approaches, our approach does not rely on the information about the projects' dependencies on third-party libraries. Instead, we mine analogical libraries from the crowdsourced knowledge in domain-specific Q&A sites (such as Stack Overflow). Furthermore, existing approaches are limited to recommend libraries for the same programming language, while our system can recommend alternative, comparable libraries across different programming languages.

Our approach is motivated by the recent success of neural network language models for solving semantic and syntactic analogy tasks in NLP applications [7], [10]. We propose to learn tag embeddings from a corpus of tag sentences derived from Stack Overflow questions, and solve the problem of finding analogical libraries using the vector arithmetic of the resulting tag embeddings. Different from English text in common NLP problems, our tag sentences are short and lack of linguistic rules and notions. Inspired by the recent work by Xu et al. [20] and Zhou et al. [21], we propose to incorporate relational and categorical knowledge of tags into tag embeddings to improve the accuracy of analogical-libraries reasoning tasks. Different from [20], [21] in which semantic knowledge are provided by human experts, our approach mines domain-specific knowledge automatically from Q&A discussions and community wikis on Stack Overflow.

Finally, it is worth mentioning some related non-academic projects. SimilarWeb[6] is a website that provides both users engagement statistics and similar competitors for websites and mobile applications. AlternativeTo[7] is a social software recommendation website in which users can find alternatives to a given software based on user recommendations. These websites can help regular web users to find similar or alternative websites or software applications. But their content is not useful for domain-specific information needs of software developers, for example, to find analogical libraries for different programming languages. In contrast, our web application is built on software-engineering data and is specifically designed for software developers.

III. The Approach

Our approach takes as input the tags of each question in Stack Overflow and the TagWiki of each tag, and produces as output a knowledge base of analogical libraries (Fig. 3). Our approach considers the tags of a Stack Overflow question as a tag sentence, and each tag of the question as a word in the tag sentence. Given a set of Stack Overflow questions, we build a corpus of tag sentences, one tag sentence per question. Given the corpus of tag sentences, we use association rule mining [11] to mine the correlation between tags (Section III-A), and use continuous skip-gram model [7] to learn tag embeddings (Section III-C). We develop POS tagging and phrase chunking methods to analyze the tag definition in the TagWiki of each tag to determine the tag category (Section III-B). Finally, we incorporate tag embeddings and relational and categorical knowledge of tags to build the knowledge base of analogical libraries (Section III-D).

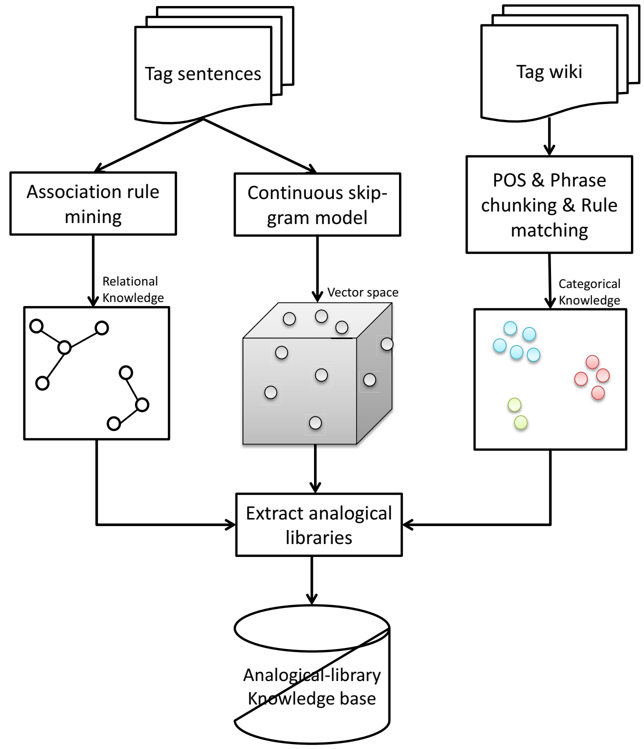

Tag sentences
Tag wiki
Association rule mining
Continuous skip-gram model
POS & Phrase chunking & Rule matching
Relational Knowledge
Vector space
Categorical Knowledge
Extract analogical libraries
Analogical-library Knowledge base

Fig. 3. The overview of our approach

A. Mining Relational Knowledge

In Stack Overflow, each question has up to 5 tags. These tags usually identify the main technologies and constructs that the question revolves around [6] (see Fig. 1 for an example). As Stack Overflow manages question tags as a set of terms, the correlation between tags are implicit. We use association rule mining [11] to discover important correlation between tags.

In our application of association rule mining, we regard each tag sentence as a transaction, and each tag in the sentence as an item in the transaction. There are two parameters in association rule mining:

$$support(t_i, t_j) = \frac{\#tagSent\ containing\ (t_i\ and\ t_j)}{\#tagSent}$$

$$confidence(t_i \Rightarrow t_j) = \frac{\#tagSent\ containing\ (t_i\ and\ t_j)}{\#tagSent\ containing\ t_i}$$

[6] www.similarweb.com/

[7] http://alternativeto.net/

where t_i and t_j are two different tags, and $tagSent$ is a tag sentence. The $support$ value measures how frequent the two tags co-occur in all the tag sentences. The $confidence$ value measures the proportion of the tag sentences containing both t_i and t_j compared with all the tag sentences containing t_i.

If the support value and confidence value of a tag pair $\{t_1, t_2\}$ are above the respective threshold t_{sup} and t_{conf}, we obtain an association rule $t_1 \Rightarrow t_2$. Given the mined association rules between tags, we construct a tag correlation graph. The tag correlation graph is an undirected graph $G(V, E)$, where the node set V contains the tags appearing in the association rules, and the edge set E contains edges $< t_1, t_2 >$ if the two tags has the association rule $t_1 \Rightarrow t_2$, $t_2 \Rightarrow t_1$ or both.

The tag correlation graph captures important relational knowledge between relevant technologies. Fig. 4 shows an example of tag correlation graph. Note that this graph is only a very small portion of the entire tag correlation graph that is mined from Stack Overflow data dump (see Section IV). For better observation, we apply community detection methods [22] to the graph so that tags in one community are in the same color. We can see that each tag community has at least one center tag which is usually a programming language or platform, and each community contains tags (e.g., libraries, frameworks, tools, concepts, databases) that are highly correlated with that programming language or platform.

B. Mining Categorical Knowledge

In Fig. 4, we can see that the tags can be of different categories, such as programming language, library, framework, tool, IDE, operating systems, etc. To determine the category of a tag, we resort to the tag definition in the TagWiki of the tag. The TagWiki of a tag is collaboratively edited by the Stack Overflow community. Although there are no strict formatting rules in Stack Overflow, the TagWiki description usually starts with a short sentence to define the tag. For example, the tagWiki of the tag *iOS* starts with the sentence "iOS is a mobile operating system developed by Apple". Typically, the first noun phrase just after the *be* verb defines the category of the tag. For example, from the tag definition of *iOS*, we can learn that the category of *iOS* is *operating system*.

Based on this heuristic, we use the NLP methods (similar to the methods used in [12] for named entity recognition) to extract such noun phrase from the tag definition sentence as the category of a tag. Given the tagWiki of a tag in Stack Overflow, we extract the first sentence of the TagWiki description, and clean up the sentence by removing hyperlinks and brackets such as "{}", "()". Then, we apply Part of Speech (POS) tagging and phrase chunking to the extracted sentence. POS tagging is the process of marking up a word in a text as corresponding to a particular part of speech, such as common noun, verb, adjective. Phrase chunking is the process of segmenting a sentence into its subconstituents, such as noun phrases, verb phrases. We use the Python NLTK library[8] for POS tagging[9] and phrase chunking [23]. Fig. 5 shows the results for the tag definition sentence of *iOS*. Based on the POS tagging and phrase chunking results, we extract the first noun phrase (NP) (*operating system* in this example) after the be verb (*is* in this example). We use this noun phrase as the category of the tag. That is, the category of *iOS* is *operating system*.

With this method, we obtain 318 categories for the 19,573 tags (about 54% of all the tags that have TagWiki). We manually normalize these 318 categories labels, such as merging *operating system* and *os* as *os*, normalizing uppercase and lowercase (e.g., *API* and *api*). As a result, we obtain 167 categories.

Although the above method obtains the tag category for the majority of the tags, the first sentence of the TagWiki of many tags is not formatted as "tag be noun phrase" form. For example, the first sentence of the TagWiki of the tag *itext* is "Library to create and manipulate PDF documents in Java". As there is no *be* verb in this sentence, the above NLP method cannot return a noun phrase for the tag category. We use a dictionary look-up method to determine the category of such tags. Specially, we use the 167 categories obtained using the above NLP method as a dictionary to recognize the category of the tags that have not been categorized using the NLP method. Given an uncategorized tag, we scan the first sentence of the tag's TagWiki from the beginning, and search for the first match of a category label in the sentence. If a match is found, the tag is categorized as the matched category. For example, the tag *itext* is categorized as *library* using this dictionary look-up method. Using the dictionary look-up method, we obtain the category for 11,059 more tags.

Note that we cannot categorize some (less than 15%) of the tags using the above NLP method and the dictionary look-up method. This is because these tags do not have a clear tag definition sentence, for example, the TagWiki of the tag *richtextbox* states that "The RichTextBox control enables you to display or edit RTF content". This sentence is not a clear definition of what *richtextbox* is. Or no category match can be found in the tag definition sentence of some tags. For example, the TagWiki of the tag *carousel* states that "A rotating display of content that can house a variety of content". Unfortunately, we do not have the category "display" in the 167 categories we collect using the NLP method. When building analogical-libraries knowledge base, we ignore these uncategorized tags as potential candidates.

C. Learning Tag Embeddings

Word embeddings are low-dimensional vector representations of words that are built on the assumption that words with similar meanings tend to present in similar contexts. Recently, Mikolov et al. [9], [7] demonstrate that the word embeddings encode similarities between pairs of words, for example, the $gender$ relation exhibited by the pairs "man:woman",

[8] http://www.nltk.org/_modules/nltk/tag.html

[9] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

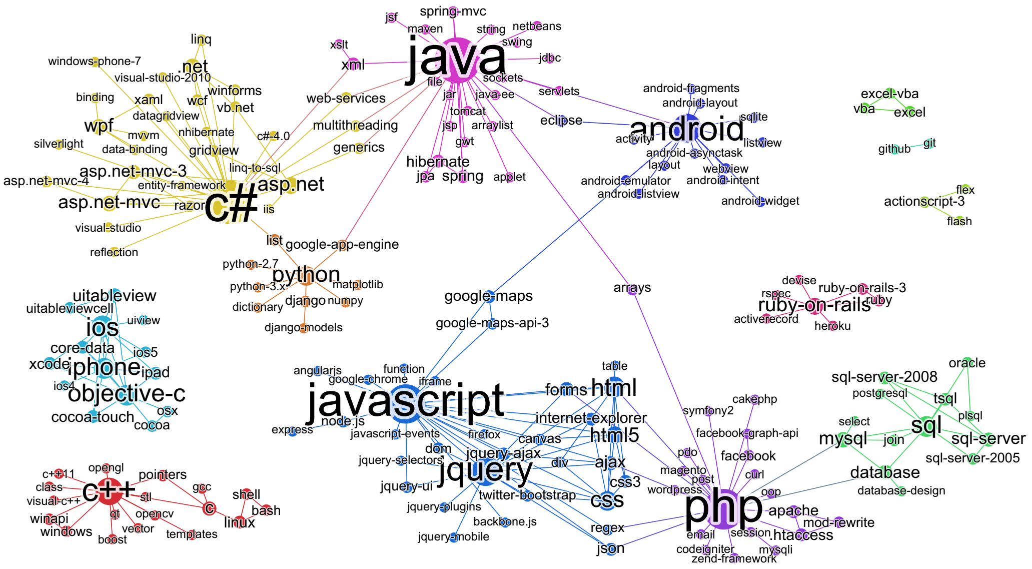


Fig. 4. An example of tag correlation graph

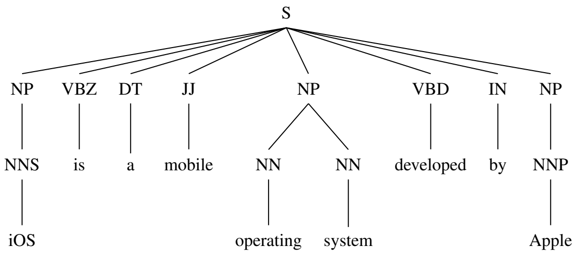


Fig. 5. POS tagging and phrase chunking results of the definition sentence of the tag *iOS*

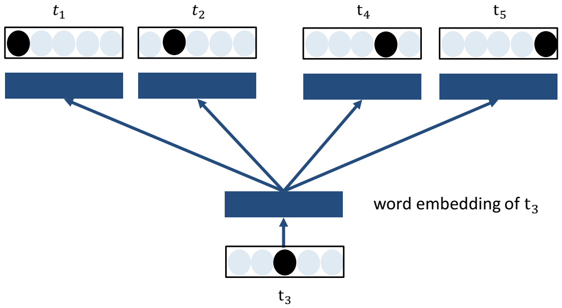


Fig. 6. The continuous skip-gram model. It predicts the surrounding words given the center word.

“king:queen”, the $capital - of$ relation in “Paris:France”, “Madrid:Spain”. Such similarities are referred to as *linguistic regularities* by Mikolov et al. and as *relational similarities* by Turney [10]. Remarkably, Mikolov et al. show that such relations are reflected in vector offsets between word pairs (e.g., $man - woman \approx king - queen$, $Paris - France \approx Madrid - Spain$), and that by using simple vector arithmetic one could apply the relation and solve analogy questions of the form “a is to A as ? is to B” in which the nature of the relation is hidden. That is, the identity of the unknown word “?” can be inferred from the words whose word embedding is most similar (e.g., by cosine similarity) to the vector $a - A + B$, for example, $king$ for $man - woman + queen$, $Madrid$ for $Paris - France + Spain$).

In our approach, give a corpus of tag sentences, we use continuous skip-gram learning algorithm [7] to learn the word representation of each tag using the surrounding context of the tag in the corpus of tag sentences. In the resulting word embedding space, the vector offsets between analogical libraries and their corresponding programming languages would exhibit *relational similarity*, for example, $nltk - python \approx opennlp - java$. Thus, given a library (e.g., *python*’s *nltk*), we can infer analogical libraries for a programming language (e.g., *java*) as *nltk* for *python* by a K-nearest-neighbor search for the tags (e.g., *opennlp*) whose word representation is the most similar to the vector $nltk - python + java$ in the resulting word embedding space.

Continuous skip-gram model [7] is a recently proposed algorithm for learning word representations using a neural network model. As illustrated in Fig. 6, the objective of the continuous skip-gram model is to learn the word representation of each word that is good at predicting the co-occurring words in the same sentence. Specifically, given a sequence of training text stream $t_1, t_2, ..., t_k$, the objective of the continuous skip-gram model is to maximize the following average log probability:

$$L = \frac{1}{K} \sum_{k=1}^{K} \sum_{-N \preceq j \preceq N, j \neq 0} \log p(t_{k+j} | t_k) \qquad (1)$$

where t_k is the central word, t_{k+j} is its surrounding word with the distance j, and N indicates the context window size to be $2N+1$. In our application of the continuous skip-gram model, a tag sentence is a training text stream, and each tag is a word. As tag sentence is short (has at most 5 tags), we set N at 2 in our approach. That is, the context window contains all other tags as the surrounding words for a given tag.

The probability $p(t_{k+j}|t_k)$ in Eq. 1 can be formulated as a log-linear softmax function which can be efficiently solved by the negative sampling method [7]. After the iterative feed-forward and back propagation, the training process finally converges, and each tag obtains a low-dimension vector as its word representation (i.e., tag embedding) in the resulting vector space.

D. Building Analogical-Libraries Knowledge Base

Given the relational and categorical knowledge of the tags and the tag embeddings of the tags, we build a knowledge base of analogical libraries as follows.

In our approach, the library tags broadly refer to the tags whose categories are library, framework, api, toolkit, wrapper, and so on[10]. This is because the meaning of these categories is often overlapping, and there is no consistent rule for the usage of these terms in the TagWiki. For example, in Stack Overflow's TagWiki, *junit* is defined as a framework, *google-visualization* is defined as an API, and *wxpython* is defined as a wrapper. All these tags are referred to as library tags in our approach.

Given a library tag t_1, we first examine its correlated tags to determine the programming language tag PL_1. Let PL_2 be a programming language tag which can be the same as PL_1 or be different from PL_1. Let $vec(x)$ be the tag embedding of the tag x. To find the analogical libraries t_2 for the programming language PL_2 as the library t_1 for the programming language PL_1, we find the library tags t_2 whose tag embedding $vec(t_2)$ is most similar (by cosine similarity in this work) to the vector $vec(t_1) - vec(PL_1) + vec(PL_2)$, i.e.,

$$\underset{t_2 \in T}{\operatorname{argmax}} \cos(vec(t_2), vec(t_1) - vec(PL_1) + vec(PL_2)) \quad (2)$$

where T is the set of library tags excluding t_1, and $cos(u, v)$ is the cosine similarity of the two vectors. In practice, there could be several analogical libraries t_2 for the programming language PL_2 as the library t_1 for the programming language PL_1. Thus, we select library tags t_2 with the cosine similarity in Eq. 2 above a threshold t_{al}.

Take the library *nltk* (a NLP library in python) as an example. As shown in the Fig. 7, for *python*, our approach returns the analogical libraries such as *textblob* and *gensim*; for *java*, our approach returns the analogical libraries such as *stanford-nlp*, *opennlp*, and *gate*.

Note that tags whose tag embedding is similar to the vector $vec(t_1) - vec(PL_1) + vec(PL_2)$ may not always be library tags. For example, tag embeddings of the tags *nlp*, *named-entity-recognition* and *language-model* are similar to the vector $vec(nltk) - vec(python) + vec(java)$. These tags are relevant to the *nltk* library as they refer to some NLP concepts and tasks, but they are not analogical libraries to the *nltk*. In our approach, we rely on the category of tags (i.e., categorical knowledge) to return only library tags.

The returned library tags sometimes include libraries that are not for the given programming language PL_2. For example, *beautifualsoup* is a *python* library for html parsing and web scraping. To find analogical libraries for *java* as the library *beautifualsoup* for *python*, by Eq. 2 we would obtain some libraries, such as *scraperwiki* (a library for ruby, python and php), *nokogiri* (a library for ruby), and *lxml* (a library for python). Although these libraries support similar features (e.g., html parsing, web scraping) to the *beautifulsoup*, they are not libraries for *java*. In our approach, we rely on the correlation between a library and the programming language(s) (i.e., relational knowledge) to select the libraries for a given programming language.

IV. Tool Support

This section describes the proof-of-concept implementation of our approach and provides the initial usage data of the tool by public users.

A. Tool description

We develop a web application called *SimilarTech* (https://graphofknowledge.appspot.com/similartech). It takes as input a library name, and recommends libraries analogical to the given one for different programming languages. The backend of *SimilarTech* is an analogical-libraries knowledge base built with the latest Stack Overflow data dump that contains Stack Overflow post data from July 31st, 2008 to Aug 16th, 2015. This knowledge base can be automatically updated periodically as the new data dump is released.

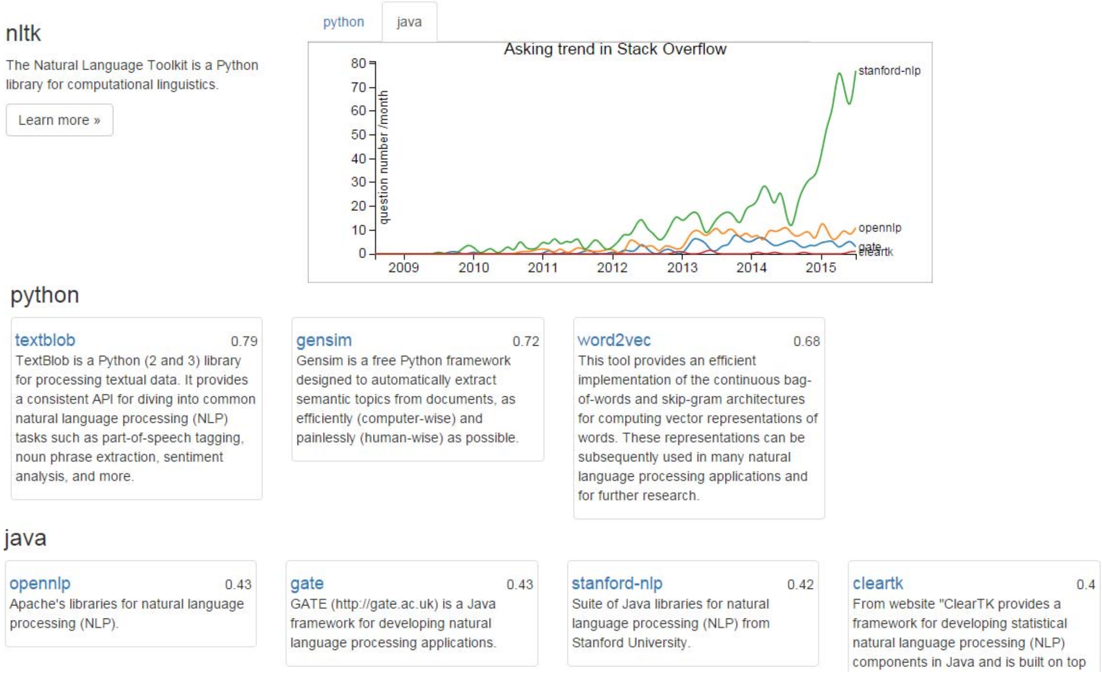
The data dump contains 9,970,064 questions and 41,856 different tags. As some infrequent or emerging tags do not have corresponding TagWiki, we collect in total 36,197 tags that have TagWiki in our dataset for mining relational and categorical knowledge of tags and for learning tag embeddings. Among 36,197 tags in our dataset, 7,783 tags are categorized as library tags. We use the famous implementation of continuous skip-gram algorithm [7] in Word2Vec[11] to learn tag embeddings. We set tag embedding dimension at 200. In the current implementation, our web application recommend analogical libraries for the top-six most frequently-asked programming languages in Stack Overflow, i.e., *java*, *javascript*, *c#*, *php*, *python* and *c++*.

Fig. 7 shows a screen shot of our web application. Given a library, our tool attempts to recommend analogical libraries (with the similarity to the given library above the minimal similarity threshold $t_{al} = 0.4$[12]) for the six programming

[10] A complete list can be found at https://graphofknowledge.appspot.com/libCategory

[11] https://code.google.com/p/word2vec/

[12] We experimentally select this value as lower threshold leads to more errors while higher one results in fewer results.



nltk

The Natural Language Toolkit is a Python library for computational linguistics.

python

textblob 0.79
TextBlob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and more.

gensim 0.72
Gensim is a free Python framework designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible.

word2vec 0.68
This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing applications and for further research.

java

opennlp 0.43
Apache's libraries for natural language processing (NLP).

gate 0.43
GATE (http://gate.ac.uk) is a Java framework for developing natural language processing applications.

stanford-nlp 0.42
Suite of Java libraries for natural language processing (NLP) from Stanford University.

cleartk 0.4
From website "ClearTK provides a framework for developing statistical natural language processing (NLP) components in Java and is built on top

Fig. 7. The scrrenshot of our website *SimilarTech*

languages. In the current implementation, *SimilarTech* presents up to four libraries with the highest similarity for each programming language. The rationale is that developers would be unlikely to look through a long list of recommendations and there are usually just a few most popular libraries for each programming language. Note that listing up to four libraries is only an implementation decision, not a limitation of our approach.

Different programming languages may have different numbers of recommended analogical libraries. This is natural because some programming languages have more alternatives for a particular task, while others have less. In some cases, a programming language may not have any analogical libraries for the given library. For example, developers rarely use *javascript* for machine learning tasks. Thus, there are no well-known machine learning libraries written in *javascript*. For the machine learning library *weka*, none of the *javascript* libraries is similar enough (i.e., above the minimal similarity threshold $t_{al} = 0.4$) to the *weka*. In such cases, *SimilarTech* recommends no libraries for that particular programming language.

For each recommended analogical library, *SimilarTech* shows a brief definition extracted from the corresponding TagWiki. Clicking a library name navigates to the analogical-libraries page for the clicked library. This allows the user to interactively explore the underlying analogical-libraries knowledge base. *SimilarTech* also summarizes the number of questions tagged with a library per month, and plots the metrics over time in a so-called asking trend. The asking trends of analogical libraries allow the user to easily compare the amount of the questions for each library on Stack Overflow.

B. Tool feedback

We release our website to the public and post this news on several programming-related websites (e.g., http://stackapps.com/questions/6667/). According to the Google Analytics of the website traffic, more than 300 users from 50 countries visit our site, from Nov 11, 2015 to Nov 13, 2015 (i.e., during the time we write this paper). These users on average browse 4.8 pages in each session and they browse 1,594 pages in total[13]. Analysis of the site logs shows that users in total browse 302 libraries for analogical-libraries recommendation. The top-ten most-frequently visited libraries are listed in Fig. 8. The usage data of our website, albeit very limited, demonstrates both the needs and the interests in such analogical-libraries recommendation that our approach supports. As more usage data is accumulated in the future, we will further investigate usage patterns and recommendation usefulness from user click-through data.

We would like to point out that, although fully-functional, *SimilarTech* is a proof-of-concept prototype that we use to facilitate data analysis and experiment with the approach. This paper focuses on solving the data extraction and modeling challenge underlying analogical libraries recommendation. Although the usage data provides some initial evidence of the potential usefulness of our web application for finding analogical libraries, we do not claim that *SimilarTech* could be put in operation without additional engineering efforts.

[13] As most search engine robots do not activate Javascript, robot traffic is not counted in Google Analytics [24].

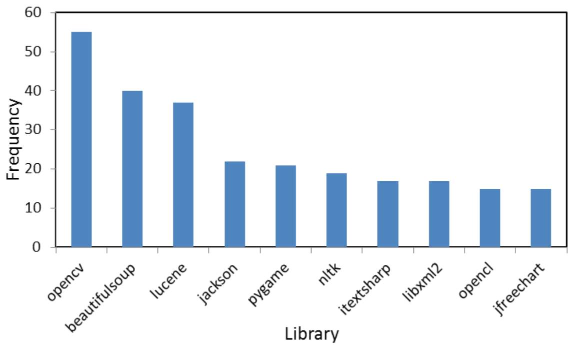


Fig. 8. The top-ten most-frequently visited libraries on *SimilarTech*

V. Evaluation

In this section, we evaluate the mined relational and categorical knowledge of tags and the accuracy of analogical-libraries recommendations. Then we zoom-into specific cases in which our approach makes poor recommendation to understand the limitations of our approach.

A. The Accuracy of Tag Categorization

We randomly sample 500 tags from 30,632 tags whose categories can be determined using either the NLP method or the dictionary look-up method (see Section III-B). We manually examine the category of these 500 tags by reading their corresponding TagWiki. Among these 500 sampled tags, 407 (81.4%) tags are correctly labeled by our proposed methods. According to our observation, two reasons lead to the erroneous categorization. First, some tag definition sentences are complex which can lead to erroneous POS tagging results. For example, the tagWiki of the tag *pyml* states that ”PyML is an interactive object oriented framework for machine learning written in Python”. Our method recognizes *object* as the category because it is the first noun after the *be* verb. Second, the dictionary look-up method sometimes makes mistakes. For example, the TagWiki of the tag *honeypot* states “A trap set to detect or deflect attempts to hack a site or system”. Our approach matches the *system* as the category of the *honeypot*.

As this work focuses on tags whose categories can be regarded as library, such as *library, framework, api, toolkit, wrapper, etc.*, we further check the correctness of these library tags in the sampled tags. Among the 500 sampled tags, there are 96 tags whose category can be regarded as library. 81 out of these 96 tags (84.4%) are correctly categorized. The accuracy of our tag categorization provides a solid basis for the analogical-libraries reasoning tasks.

B. The Semantic Distance of Tag Correlations

To evaluate the mined relational knowledge of correlated tags, we adopt the metric called “Google distance” [25], [26]. Google distance is a crowd-scale method to measure the semantic distance between a set of words by analyzing search engine data. The assumption is that the co-occurrence of a set of words in the same queries is a good indicator of the semantic distance between the words.

In this work, we use Google Trends [27] to evaluate the semantic distance of the correlated tags in the mined tag correlation graph. Google Trends is a public web service that shows how frequent a particular search-term is searched compared with the total search-volume in Google search. Given a pair of correlated tags (e.g., <java, swing>) in the tag correlation graph, we query the Google Trends with the two tags as a search term (i.e., “java swing”). Google Trends will provide the trend statistics for popular queries, and report “no enough data” for less popular queries.

We randomly sample 1,000 pairs of tags (i.e., tag relations) in our tag correlation graph. A small percentage of tag relations (13.1%) are not present in Google Trends (i.e., no enough data”). That is, these pairs of tags are not popular queries according to Google Trends. However, a pair of tags not present in Google Trend does not necessarily indicate wrong tag relations. First, some tags of emerging techniques (e.g., *apiary.io*) may not accumulate enough search volume on Google. Second, the difference between tagging behavior and search behavior also results in a small percentage of tag pairs not present in Google Trend. For example, Stack Overflow users always use *javascript* and *video.js* together to tag questions, while web users search Google with *video.js* only without *javascript*.

Among the 1,000 sampled tag relations, 137 are correlations between a programming language and a library. We further check the accuracy of these 137 library-programming-language correlations. The results show that 88.3% of these 137 correlations appear in Google Trends. Overall, the mined relational knowledge of tags can accurately represent the semantic relationships between software-specific entities, including programming languages and libraries.

C. The Quality of Analogical-Libraries Recommendation

We first describe the method and metric to evaluate the accuracy of our analogical-libraries recommendation. Then, we present the evaluation results.

1) Evaluation Procedure and Metrics: We randomly sampled 100 libraries in our analogical-libraries knowledge base as the test cases. These test-case libraries support a diverse set of functionalities, such as visualization, NLP, machine learning, searching, testing, and so on.

Our approach is inspired by the use of word embeddings to solve analogy questions of word pairs [9]. The original word-pair analogy tasks includes two sets: semantic analogies such as $Paris - France \approx ? - Spain$ and syntactic analogies such as $quickly - quick \approx ? - slow$. In these work-pair analogy tasks, there is only one correct answer, for example $Madrid$ for $Paris - France \approx ? - Spain$, and $slowly$ for $quickly - quick \approx ? - slow$.

In contrast, our analogical-libraries task may return several analogical libraries for a given library, as there is rarely only one solution in software engineering context. For example, for the NLP library *nltk* for *Python*, there are several comparable libraries for *Java*, such as *standford-nlp*, *opennlp*, *gate*. Therefore, we use the Precision@k metric [28], [29] to evaluate the accuracy of analogical-libraries recommendation. Note that as the set of all analogical libraries is literally unknown, it is impossible to evaluate Recall@k.

As there is no ground truth of analogical libraries, we have to manually check each recommended library for a given test-case library. We examine information from library official website, TagWiki, wikipedia, and other available online information. If the recommended library can provide comparable features as the given test-case library, we consider the recommendation as correct. Note that we do not consider relevant libraries as correct recommendations. For example, *SimilarTech* recommends the *powermock* and *mockito* for the library *junit*. *powermock* and *mockito* are mocking framework for testing. Although *powermock* and *mockito* are relevant to the library *junit*, we do not consider them as analogical library to *junit*, because they do not provide comparable features as the given library.

For a given test-case library, let's assume that *SimilarTech* recommends at least one library for n ($1 \leq n \leq 6$) programming languages. Let $correct_i@k$ be the number of correct recommendations in the top-k recommended libraries for a particular programming language PL_i ($1 \leq i \leq n$). The $Precision_i@k$ ($k = 1, 2, 3, 4, 5$ in this evaluation[14]) for the programming language PL_i is $correct_i@k/k$. We compute the Precision@k of the overall analogical-libraries recommendation as:

$$\sum_{i=1}^{n} \frac{Precision_i@k}{n}$$

i.e., the average of the $Precision_i@k$ metrics of all the programming languages with at least one recommended libraries.

2) Accuracy: In our experiments on analogical-libraries recommendation, we compare the pure tag embedding based recommendation (i.e., the baseline method $w2v$ (word to vector)) with the relational-knowledge powered recommendation ($w2v + r\_kg$), the categorical-knowledge powered recommendation ($w2c + c\_kg$), and the relational- and categorical-knowledge powered recommendation ($w2c + rc\_kg$).

Fig. 9 illustrates the Precision@k of the four recommendation methods. We can see that the pure tag embedding based recommendation performs poorly. The recommendation accuracy by tag embedding alone is less than 30%. This is because tag sentences have much noisy context information, compared with natural language sentences. Incorporating relational and categorical knowledge of tags into analogical-libraries recommendation can significantly improve the accuracy of the recommendation. Categorical knowledge of tags can boost the accuracy more than relational knowledge of tags. Incorporating both knowledge yields the best accuracy.

When incorporating both knowledge, the Precision@1 is 81%, and the Precision@5 is still reasonably high at 67%. That is, given a test-case library, the top-1 library that *SimilarTech* recommends for each programming language is high likely an analogical library, and the majority of the top-5 recommended libraries for each programming language are analogical libraries. Our results suggest that integrating domain-specific knowledge with word embeddings can enhance analogical reasoning tasks in software engineering context.

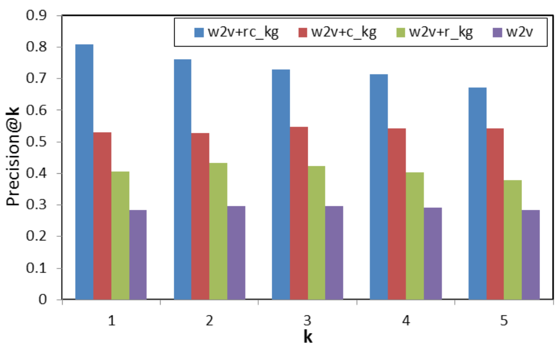


Fig. 9. Performance of using relational knowledge and categorical knowledge on analogical-libraries recommendation based on our proposed approach

D. Limitations of Our Approach

Although our approach can make accurate analogical-libraries recommendations in most cases, as the first work of this kind that combines word embeddings with domain-specific knowledge for analogical reasoning tasks, we would like to further investigate in which cases our approach cannot make good recommendations. This will help us, as well as other researchers and designers of similar systems, understand the limitations of our approach and address them in the future.

To that end, we investigate the test cases for which the Precision@5 metrics are below 0.2, i.e., almost all the recommended libraries for a given test-case library are incorrect. We find that such test-case libraries fall into two categories: either a full-stack framework that supports a wide range of features or a library that provides a very specific feature or support some unique features for a particular language.

For the first case, an example is *ruby-on-rails* (a web application framework for *Ruby*). We expect that the system can recommend analogical framework such as *node.js* for *JavaScript*, *django* for *Python*, and *codeigniter* for *PHP*. But the recommendations by *SimilarTech* do not include any such web application frameworks. The fundamental reason for such poor recommendations is that neural network language models assume that similar words share similar context such that word embeddings can be learned from the surrounding context. However, these full-stack frameworks can be used in very diverse context, which leads to very diverse tag sentences. As a result, in the resulting word embedding space, these frameworks and their respective programming languages do not exhibit relational similarity (or linguistic regularity) which is necessary for analogical reasoning. Thus, our approach fails to recommend analogical web application frameworks for the

[14] we use a small k value because developers are unlikely to look through a long recommendation list

ruby-on-rails.

For the second case, examples include *jnotify* and *mako*. *jnotify* is a Java library that allow Java application to listen to file system events. *mako* is a template library providing non-XML syntax which compiles into Python modules and conceptually can be considered as an embedded Python language. Due to their very specific or language-dependent features, it is unlikely that other programming languages have comparable libraries.

To sum up, our approach is not suitable for finding analogical libraries for feature-rich, full-stack frameworks or language-dependent, unique libraries.

VI. Conclusion and Future Work

Third-party libraries assist developers in finishing software engineering tasks more efficiently without the need to reinvent the wheels. However, due to many reasons such as lack of active maintenance of the libraries being used or language migration, developers often need to find some alternative and comparable libraries to replace the libraries they are already familiar with. Although developers can find useful information in community-curated list, blogs and Q&A posts on the Web, the information is likely to require tedious and time-consuming browsing and aggregation, or is likely to be out of date to mislead developers especially the novice.

In this paper, we propose an automated technique to recommend analogical libraries across different programming languages. We adopt the cutting-edge deep learning method in NLP applications (also known as word embeddings) to the software engineering data. We further enhance the original word embedding technique with software-engineering domain knowledge to better answer analogy questions in software engineering context. Given a library, our approach can recommend several most salient analogical libraries for different programming languages.

We evaluate all the components of our approach, including the quality of the mined relational and categorical knowledge, and the quality of the analogical-libraries recommendation. Our approach achieves very promising results for analogical-libraries recommendation (Precision@1=0.81 and Precision@5=0.67). We also implement our approach in a web application[15] and release the web application for public use and evaluation.

In the future, we will improve our web application and analyze the website traffic and user behaviors in our website to enhance the accuracy of analogical-libraries recommendation. Furthermore, we are very interested in extending our approach to fine-grained level of analogy relationships, for example, mining analogical APIs across different libraries or programming languages in Q&A discussions or other online resources (e.g., Github). Tens of thousands of API analogy questions can be found on Stack Overflow, which indicates the urgent needs for the automatic tool support at the API level. We believe the ability to easily find analogical APIs and their usage can boost developers' productivity and efficiency when they migrate from one programming language to another unfamiliar language.

Exiting studies of Stack Overflow data focus on mining discussion topics or recovering traceability between software project data and Stack Overflow posts. In contrast, our work demonstrates the feasibility of turning software engineering social content on Stack Overflow into a knowledge base of software-specific entities and their relationships to improve developers' life on the Internet. Apart from analogical-libraries recommendation, another contribution of this work is an initial knowledge graph that captures the domain-specific relational and categorical knowledge of tens of thousands of software-specific entities. In the future, we will extend this initial knowledge graph with more software-specific entities (e.g., APIs) and richer set of relationships between entities. We are interested in entity-centric search systems that can exploit this knowledge graph for not only displaying additional facts and direct information about the central entity in a query, but also to provide extended suggestions for users who would like to browse. This work can be considered as the very first step towards our long-term goal.

Acknowledgment

The authors would like to thank Han Lei for the UI design of our website. This work was partially supported by MOE AcRF Tier1 Grant M4011165.020.

References

[1] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 182–191.

[2] C. Teyton, J.-R. Falleri, and X. Blanc, "Mining library migration graphs," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 2012, pp. 289–298.

[3] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining api mapping for language migration," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 195–204.

[4] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Statistical learning approach for mining api usage mappings for code migration," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 2014, pp. 457–468.

[5] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[6] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 25–34.

[7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[8] W. Chen, Y. Zhang, and M. Zhang, "Feature embedding for dependency parsing," in *Proceedings of the International Conference on Computational Linguistics*, 2014.

[9] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations." in *HLT-NAACL*, 2013, pp. 746–751.

[10] P. D. Turney, "Similarity of semantic relations," *Computational Linguistics*, vol. 32, no. 3, pp. 379–416, 2006.

[11] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[15] https://graphofknowledge.appspot.com/similartech

[12] J. Kazama and K. Torisawa, "Exploiting wikipedia as external knowledge for named entity recognition," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 698–707.

[13] D. M. L. Xu, R. Bodık, and D. Kimelman, "Jungloid mining: Helping to navigate the api jungle," in *POPL*, 2005.

[14] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 204–213.

[15] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "Mapo: Mining and recommending api usage patterns," in *ECOOP 2009–Object-Oriented Programming*. Springer, 2009, pp. 318–343.

[16] W.-K. Chan, H. Cheng, and D. Lo, "Searching connected api subgraph via text phrases," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 10.

[17] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of api methods from feature requests," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 290–300.

[18] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 532–542.

[19] C. Teyton, J.-R. Falleri, and X. Blanc, "Automatic discovery of function mappings between similar libraries," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE, 2013, pp. 192–201.

[20] C. Xu, Y. Bai, J. Bian, B. Gao, G. Wang, X. Liu, and T.-Y. Liu, "Rc-net: A general framework for incorporating knowledge into word representations," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 1219–1228.

[21] G. Zhou, T. He, J. Zhao, and P. Hu, "Learning continuous word embedding with metadata for question retrieval in community question answering," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics Beijing, China, 2015, pp. 250–259.

[22] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[23] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[24] "Google analytics policy," https://support.google.com/analytics/answer/1315708?hl=en.

[25] R. L. Cilibrasi and P. Vitanyi, "The google similarity distance," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 3, pp. 370–383, 2007.

[26] R. Gligorov, W. ten Kate, Z. Aleksovski, and F. Van Harmelen, "Using google distance to weight approximate ontology matches," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 767–776.

[27] "Google trends," https://www.google.com.sg/trends/.

[28] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.

[29] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec: an enhanced tag recommendation system for software information sites," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 291–300.