myjournal manuscript No.
(will be inserted by the editor)

What's Spain's Paris? Mining Analogical Libraries from Q&A Discussions

Chunyang Chen · Zhenchang Xing · Yang Liu

Received: Aug 31, 2017 / Accepted: Sep 5, 2018

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 for different programming languages or different mobile platforms. 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are limited by the community-curated list of libraries, blogs, or Q&A posts, which often contain overwhelming or out-of-date information.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-library questions by combining state-of-the-art word embedding technique and domain-specific relational and categorical knowledge mined from Stack Overflow. Given a library and a recommended analogical library, our approach further extracts questions and answer snippets in Stack Overflow about comparison of analogical libraries, which can potentially offer useful information scents for developers to further their investigation of the recommended analogical libraries. We implement our approach in a proof-of-concept web application and more than 34.8 thousands of users visited our website from November 2015 to August 2017. Our evaluation shows that our approach can make accurate recommendation of analogical libraries. We also demonstrate the usefulness of our analogical-library recommendations by using them to answer analogical-library questions in Stack Overflow. Google Analytics of our website traffic and analysis of the visitors' interaction with website contents provide the insights into the usage patterns and the system design of our web application.

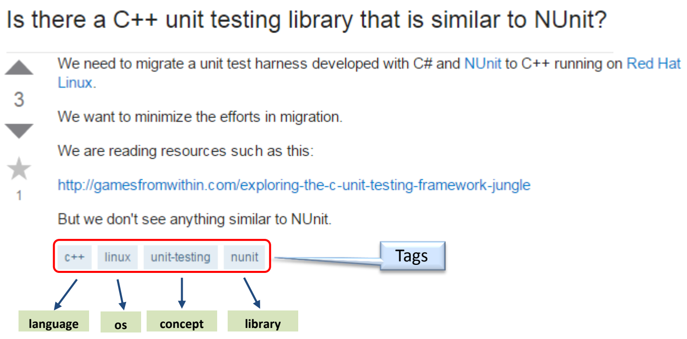
1 Introduction

Third-party libraries are an integral part of many software systems. Developers do not need to reimplement the wheels by using libraries which provide robust and efficient functionalities. 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 (Thung et al, 2013a). As a software project and the libraries used in the project inevitably evolve in their own direction, developers sometimes need to replace some currently-used libraries by another ones. For example, the library that a project currently uses is no longer under active development, or lacks certain desired features, or cannot satisfy performance requirements. In such cases, some developers want to find some good replacements (Teyton et al, 2012). In other cases, they switch to a new programming language, but they would like to "reuse"

Chunyang Chen
Faculty of Information Technology, Monash University, Australia
E-mail: chunyang.chen@monash.edu

Zhenchang Xing
College of Engineering & Computer Science, Australian National University, Australia
E-mail: zhenchang.xing@anu.edu.au

Yang Liu
School of Computer Science and Engineering, Nanyang Technological University, Singapore
E-mail: yangliu@ntu.edu.sg


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 example of analogical-library question in Stack Overflow.

their good experience with some libraries that they are familiar with (Zhong et al, 2010; Nguyen et al, 2014) (such as the example shown in Fig. 1). 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 Web for analogical libraries that can provide features comparable to the libraries they are already familiar with. They could find useful information in some community-curated list of libraries, such as *unit testing framework on Wikipedia*, and *Awesome PHP on Github*. These library lists are usually very comprehensive, but they often contain some obsolete or not-widely-adopted libraries (Wu et al, 2017) that may distract developers. For example, in the list of *unit testing framework on Wikipedia*, "Test manager" [1] developed in 1993 is still in the list, but it cannot even be searched up on the Internet. Developers may also find useful information in blogs (e.g., *"Beyond JUnit – Testing Frameworks alternatives"*) or forum posts (e.g., *"Alternatives to JUnit"*). Blogs and forum posts are usually more focused, but they are often opinion-based and many past posts also 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 (e.g., Fig. 1), but may not get the immediate answers.

Although many research works have been carried out for mining similar code snippets (Nguyen et al, 2013), functions (Teyton et al, 2013), or APIs (Zhong et al, 2010; Nguyen et al, 2014), finding analogical libraries for different programming languages or mobile platforms has been rarely investigated. A key challenge in analogical-library recommendation is that the program analysis (based on code) or information retrieval (based on text) methods cannot properly model the functionalities of libraries for reasoning their analogical relations.

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 (Barua et al, 2014), and that the main technologies or constructs that a question revolves around can usually be identified from question tags (Nasehi et al, 2012) (see Fig. 1). Instead of listing dozens of 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 can be periodically updated with new Stack Overflow questions, and thus 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 (NLP) applications (Mikolov et al, 2013b; Chen et al, 2014). Recently, neural network language models are demonstrated (Mikolov et al, 2013c; Turney, 2006) to be 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 an artificial tag sentence, and each tag as a word in the sentence. According to Stack Overflow policy[2], tags in a tag sentence are ordered by the frequency of a tag at the time a question is posted. As illustrated in Fig. 2,

[1] `https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#cite_note-496`
[2] https://meta.stackexchange.com/questions/77808/does-it-matter-the-order-you-tag-your-question

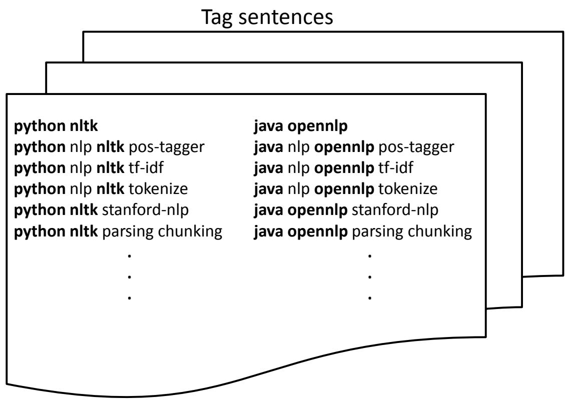

Tag sentences
python nltk
python nlp nltk pos-tagger
python nlp nltk tf-idf
python nlp nltk tokenize
python nltk stanford-nlp
python nltk parsing chunking
java opennlp
java nlp opennlp pos-tagger
java nlp opennlp tf-idf
java nlp opennlp tokenize
java opennlp stanford-nlp
java opennlp parsing chunking

Fig. 2 The similar context of Python's nltk and Java's opennlp in tag sentences. However, it is important to note that analogical libraries themselves rarely co-occur in the same tag sentences.

analogical libraries (such as Python's nltk and Java's opennlp) often 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 (Mikolov et al, 2013b) to learn the tag embedding of each tag using the surrounding tags of a tag in the corpus of tag sentences. Given a library (e.g., *python*'s *nltk*), we can then 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. It is important to note that analogical libraries are rarely used to tag the same Stack Overflow questions. Therefore, traditional association rule mining methods that mine co-occurring items in the same transactions cannot discover such analogical relationships.

However, directly applying neural network language models to the problem of learning tag embeddings in tag sentences, as opposed to learning word representations in natural language sentences, brings unique challenges. First, contrary to natural 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. There is no existing notion of linguistic rules equivalent to natural language domain. Second, Stack Overflow community relies on collaborative editing to curate post quality, including question tags (Chen et al, 2017a). However, considering the sheer amount of questions posted everyday, tags of some questions (for example newly posted questions that do not receive much attention) could still be noisy such that they cannot reflect the inherent relationship between tags and may 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 (Agrawal et al, 1994) to mine the correlation between pairs of 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 (Kazama and Torisawa, 2007) 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-language/platform pairs with analogical relationships.

Our previous works (Chen et al, 2016a; Chen and Xing, 2016a) implement the proposed approach for recommending analogical libraries for different programming languages. In this paper, we extend our previous work for analogical-libraries recommendation for different mobile platforms such as iOS, Android and Windows-Phone. Furthermore, after obtaining a list of recommended libraries, developers usually need to further investigate which recommended library is the most suitable one for their work. To that end, they need to search for information about the recommended libraries, such as active versions, runtime performance, reliability, documentation and available code examples. To assist developers in such further investigation, we develop a keyword-matching method to extract comparison questions and answer snippets in Stack Overflow discussions about a pair of analogical libraries. These comparison snippets may provide useful information scents about different aspects of recommended libraries, with which developers can further search more information about some recommended analogical libraries using search engines.

We implement our approach in a proof-of-concept web application (`https://graphofknowledge.appspot.com/similartech`) for programming-language based recommendation and (`https://graphofknowledge.appspot.com/similarmobiletech`) for mobile-platform based recommendation. The application takes as input a library name and recommends analogical libraries for different programming languages or different mobile platforms. We update the backend analogical library database using our approach with the latest Stack Overflow Data Dump related on Dec, 2017. We evaluate the analogical-libraries recommendations by our approach for randomly selected 140 libraries across different programming languages and mobile platforms. The results show that our approach can make accurate recommendation of analogical libraries for both different programming languages and mobile platforms. Furthermore, Google Analytics of our website traffic provides initial evidence of the potential usefulness of our web application for software developers. At the time of this submission (August 2017), more than 34.7 thousand users from 168 countries have visited our site for analogical libraries.

In addition to the contributions in our previous work (Chen et al, 2016a), this paper makes new contributions as follows:

- To demonstrate the generality of our approach, we extend our approach to analogical-libraries recommendation for different mobile platforms and evaluate the accuracy of mobile-platform-based recommendation. We also analyze the performance differences between programming-language based recommendation and mobile-platform based recommendation.
- To assist developers in their further investigation of the recommended analogical libraries, we develop a keyword-matching method to extract comparison questions and answer snippets in Stack Overflow that may provide useful information scents when developers discuss and compare certain pair of analogical libraries. We conduct a user study to evaluate whether the extracted comparison snippets contain useful information that developers may be interested in when comparing analogical libraries.
- We comparatively study the effectiveness of two popular word-embeddings techniques (continuous skip-gram model and continuous bag-of-words model) for learning tag embeddings. Our results show that continuous skip-gram model outperforms continuous bag-of-words model.
- We evaluate the usefulness of our analogical-libraries recommendations by comparing our recommendations with user-provided answers to 70 analogical-libraries-related questions in Stack Overflow. Our results show that most libraries in answers are covered by recommended libraries using our approach.
- Our web application receives steady visits (on average 1.1K per month) since its launch in November 2015. We log the user visit behavior and analyze the users' logs for insights.

2 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 an artificial 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 pair mining (Agrawal et al, 1994) to mine the correlation among tags, especially the correlation between a library and its associated programming language (Section 2.2), and use word embedding techniques (e.g., continuous skip-gram model (Mikolov et al, 2013b) or continuous bag-of-words model) to learn tag embeddings (Section 2.4). 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 2.3). Tag embeddings and relational and categorical knowledge of tags are incorporated to build the knowledge base of analogical libraries for different programming languages or mobile platforms (Section 2.5). Given a query library, our approach returns analogical libraries based on this knowledge base. Furthermore, to assist developers in their further investigation of the recommended libraries with some useful information scents, our approach formulates queries for a

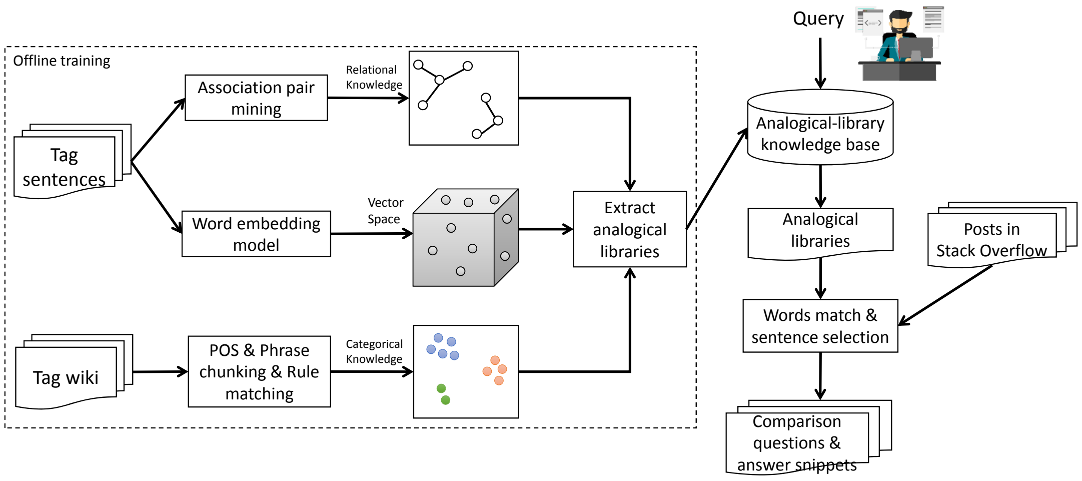


Fig. 3 The overview of our approach

given pair of analogical libraries and search Stack Overflow posts for questions and answer snippets that likely compare the two analogical libraries (Section 2.6).

2.1 Approach Input

We use Stack Overflow question tags as the input to our approach for the following reasons:

– First, question tags are the metadata of a question. Nasehi et al (2012) show that they represent the main technologies or concepts that a question revolves around. That is, tags provide a compact and meaningful representation of software engineering concepts and technologies that developers are concerned with.
– Second, Stack Overflow has strict policy regarding tag usage and the creation and approval of new tags. The community also constantly merges tag synonyms. Users must use a set of community-approved tags, rather than using any arbitrary words to tag their questions. This practice ensures the consistent use of the same tag to represent the same concept or technology. In contrast, question titles and post contents are free text which is inevitably much more noisy. For example, users may write "JavaScript" in many different forms, such as "js", "java-script", or "javascrpt". Such abbreviations, synonyms and misspellings are difficult to unify and they will negatively influence the learning of tag embeddings due to the word sparsity issue.
– Third, question tags can be directly processed as software entities. However, it is a challenging task to discriminate mentions of software entities (e.g., programming language or libraries) in normal text. For example, given a sentence "you can use underscore in javascript", we will need robust named entity recognition and entity linking techniques (Ye et al, 2016a) to recognize that "underscore" in the sentence actually refers to the software library "underscore.js".

In this work, we are concerned with programming-language and library tags (see Section 2.3 for our method for determining tag categories). Programming languages and libraries can have different versions. But when searching for analogical libraries, developers have to first know which analogical library they may use. And then they can search more information and learn more about the analogical libraries, for example, to determine which specific library version fits their needs the best. Our approach aims to recommend analogical libraries, rather than specific library versions. To that end, we normalize Stack Overflow tags with version numbers, such as "sql-server-2007", "asp.net-mvc-4", "c++11". Specifically, we remove the detailed number (also the hyphen before number if applicable) but retain the core parts of the tag. After normalizing tags with version numbers, we remove possible duplicates within the tag sentences, for example, an original tag sentence "sql-server, visual-studio, sql-server-2008" will be reduced into "sql-server, visual-studio". Normalizing tags with version numbers increases the tagging frequencies of a library. This also helps to alleviate the word sparsity issue for learning tag embeddings."

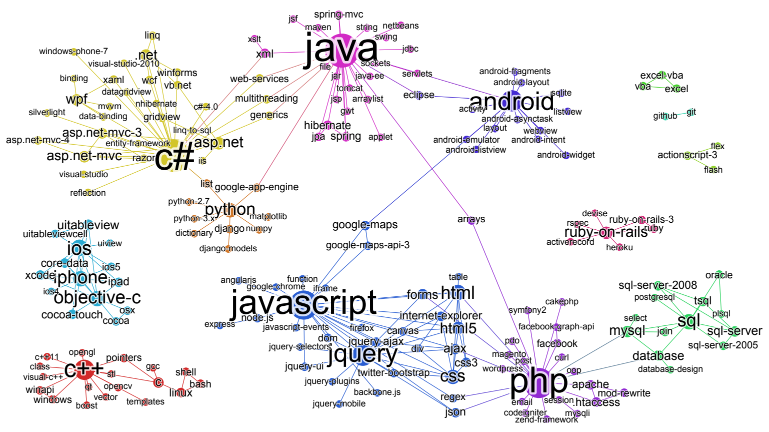


Fig. 4 An example of tag correlation graph

2.2 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 (Nasehi et al, 2012) (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 (Agrawal et al, 1994) 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:

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

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

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 pair $t_1 \Rightarrow t_2$. Note that our association rules involve only single item in antecedent and consequent. Given the mined association pairs 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 pairs, 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 3). For better observation, we apply community detection methods (Blondel et al, 2008) 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. This resulting tag correlation graph can be attributed to the common practice in Stack Overflow that users tag their questions with relevant programming language and libraries. Therefore, the associations between pairs of related library and programming language have generally higher support and confidence than the associations between pairs of other categories of tags, forming the communities of tags centred on a relevant programming language (or platform).

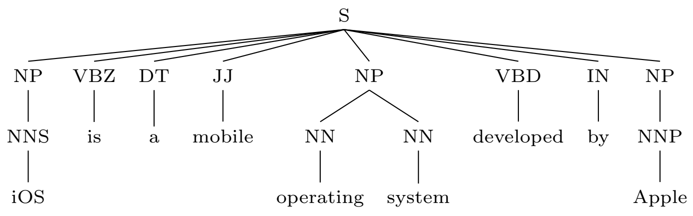


Fig. 5 POS tagging and phrase chunking results of the definition sentence of the tag *iOS*

2.3 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 sequence 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*. For the generality of our approach, we define *noun sequence* in this work as a sequence consisting of consecutive nouns between other POS tags, including at least one noun.

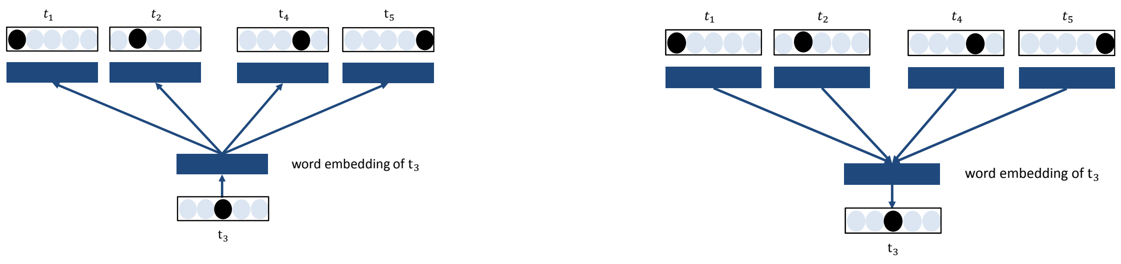
Based on the above observation of tag definitions, we use the NLP methods (similar to the methods used in (Kazama and Torisawa, 2007) for named entity recognition) to extract such noun sequence from the tag definition sentence as the category of a tag. Given the tagWiki of a tag in Stack Overflow, we 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 noun, verb, adjective. Phrase chunking is the process of segmenting a sentence into its subconstituents, such as noun phrases, verb phrases. Different tools usually agree on the POS tags of nouns, and we find that POS tagger in NLTK (Bird, 2006) is especially suitable for our task. In NLTK, the noun is annotated by different POS tags[3] including NN (Noun, singular or mass), NNS (Noun, plural), NNP (Proper noun, singular), NNPS (Proper noun, plural). Then we use the phrase chunking i.e., regular expression to recognize consecutive nouns (at least one noun) as noun sequence by their POS tags. 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 sequence (*operating system* in this example) after the be verb (*is* in this example). We use this noun sequence as the category of the tag. That is, the category of *iOS* is *operating system*.

With this method, we obtain 318 categories for the 23,658 tags that have TagWiki. We manually normalize these 318 categories labels, such as merging *operating system* and *os* as *os*, *libraries* and *lib* as *library*, and normalizing uppercase and lowercase (e.g., *API* and *api*). As a result, we obtain 167 categories. Furthermore, we manually categorize these 167 categories into four general categories: programming language, platform, library, and concept/standard. These four general categories are defined in our previous work for named entity recognition (Ye et al, 2016a). This generalization step is necessary, especially for the library tags that broadly refer to the tags whose fine-grained categories can be library, framework, api, toolkit, wrapper, and so on[4]. This is because the meaning of these fine-grained 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.

Although the above method obtains the tag category for the majority of the tags, the first sentence of the TagWiki of some tags is not formatted in the standard "tag be noun sequence" form. For example, the first sentence of the TagWiki of the tag *itext* is "Library to create and

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

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



(a) Continuous skip-gram model (b) Continuous bag-of-words model

Fig. 6 The architecture of the two word embeddings models. The continuous skip-gram model predicts surrounding words given the central word, and the CBOW model predicts the central word based on the context words. Note the differences in arrow direction between the two models.

manipulate PDF documents in Java", or for *markermanager*, the tag definition sentence is "A Google Maps tool", or for *ghc-pkg*, the tag definition sentence is "The command ghc-pkg can be used to handle GHC packages". As there is no *be* verb in this sentence, the above NLP method cannot return a noun sequence as the tag category. According to our observation, for most of such cases, the category of the tag is still present in the sentence, but often in many different ways. It is very likely that the category word appears as the first noun sequence that match the existing category words in the definition sentence. Therefore, 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 9,648 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 exclude these uncategorized tags as potential candidates.

2.4 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 (2013c,b) demonstrate that the word embeddings encode similarities between pairs of words, for example, the *gender* relation exhibited by the pairs "man:woman", "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, 2006). 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, given a corpus of tag sentences, we use word embedding methods (Mikolov et al, 2013a) 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 or mobile platforms 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. In the same way, we can infer analogical libraries for different mobile platforms like $afnetworking - ios + android \approx volley$.

In this work, we use association rule mining to discover the explicit correlation of a library and its associated programming language (or platform), because Stack Overflow users commonly tag their questions with the relevant programming language (or platform) and library. However, association rule mining cannot support the discovery of analogical libraries, because association rules can capture the correlations between tags only if they co-occur frequently enough in the same transactions. However, analogical libraries rarely co-occur together in the same questions in Stack Overflow. For example, "python-imaging-library" and "aforge" are two libraries specifically for computer vision in Python and C# respectively, but they never occur together in Stack Overflow. Even for the same programming languages, analogical libraries are also rarely mentioned in the same questions. This is because Stack Overflow questions are usually about the use of a particular library rather than many libraries. As a result, analogical relationships cannot be extracted by association rules. Different from association rule mining, word embedding learns the semantics of a centre word from the words often appearing in the context of the centre word. Therefore, it can infer analogical tags from the similar contexts even the two tags never co-occur in the same questions.

There are two kinds of widely-used word embedding methods (Mikolov et al, 2013a), the continuous skip-gram model (Mikolov et al, 2013b) and the continuous bag-of-words (CBOW) 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 (Fig. 6(a)), while the CBOW is the opposite, that is, predicting the center word by the context words (Fig. 6(b)). Note that word order within the context window is not important for learning word embeddings.

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:

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

while the objective of the CBOW model is:

$$L = \frac{1}{K} \sum_{k=1}^{K} \log p(t_k | (t_{k-N}, t_{k-N+1}, ..., t_{k+N})) \tag{2}$$

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. In our application of the word embedding, a tag sentence is a training text stream, and each tag is a word. As one tag sentence is short (has at most 5 tags), we set N at 4 in our approach so that the context of one tag is all other tags in the current sentences. That is, the context window contains all other tags as the surrounding words for a given tag. Therefore, tag order does not matter in this work for learning tag embeddings.

The probability $p(t_{k+j}|t_k)$ in Eq. 1 or $p(t_k|(t_{k-j}, t_{k-j+1}, ..., t_{k+j}))$ in Eq. 2 can be formulated as a log-linear softmax function which can be efficiently solved by the negative sampling method (Mikolov et al, 2013b). 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.

To determine which word-embedding model performs better in our analogical library reasoning task , we carry out a comparison experiment, and the details are discussed in Section 4.3.2.

2.5 Building Analogical-Libraries Knowledge Base

We build a knowledge base of analogical libraries for different programming languages (or mobile platforms) by incorporating tag embeddings and categorical and relational knowledge of tags. First, we obtain the initial analogical-library candidates for different programming languages (or mobile platforms) based on tag embeddings and tag categories. Then, we refine the initial candidates according to the correlations of libraries and programming languages (or mobile platforms).

2.5.1 Is the Word Embedding Enough for Recommendation?

One of the most important and famous characteristics of word embeddings is that they can capture analogy relationships among words like $man - woman \approx king - queen$. In this work, we apply this characteristic to software-specific data to infer analogy libraries like $python{-}nltk \approx java{-}opennlp$. However, no techniques alone are perfect, especially when a technique is adapted to an application context (software tags in this work) different from its original application context (general English text). For example, words in general text do not have categories. They are just words. But software tags can be of different categories, such as languages, libraries, or general concepts. For example, for an analogy query $python{-}nltk \approx java{-}?$, in addition to libraries such as *opennlp* and *stanford-nlp*, tag-embedding based reasoning alone will also find relevant concept such as *pos-tagger* and *word-sense-disambiguation*. Although relevant, these concepts are out of scope of analogical library recommendation. Furthermore, different software tags also exhibit different types of correlations. For example, libraries and their implementation languages have strong correlations as they usually appear together when Stack Overflow users tag the question. In contrast, a general concept (e.g., named-entity-recognition) can be related to many libraries and languages. Based on such domain-specific characteristics, we combine tag-embedding based analogical reasoning with tag category information and library-language/platform association to improve the results of analogical library recommendation.

2.5.2 Obtain Analogical-library Candidates

Given a library tag t_1, we first examine its correlated tags to determine its base programming language (or mobile platform), denoted as $base_1$. Let $base_2$ be a programming-language (or mobile-platform) tag which can be the same as $base_1$ or be different from $base_1$. Let $vec(x)$ be the tag embedding of the tag x. To find the analogical libraries t_2 for the $base_2$ as the library t_1 for the $base_1$, 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 $vec(t_1) - vec(base_1) + vec(base_2)$, i.e.,

$$\underset{t_2 \in T}{\operatorname{argmax}} \cos(vec(t_2), vec(t_1) - vec(base_1) + vec(base_2)) \tag{3}$$

where T is the set of library tags excluding t_1, and $cos(u, v)$ is the cosine similarity of the two vectors.

Note that tags whose tag embedding is similar to the vector $vec(t_1) - vec(base_1) + vec(base_2)$ 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 NLP 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 as candidates.

In practice, there could be several analogical libraries t_2 for the $base_2$ as the library t_1 for the $base_1$. Thus, we select library tags t_2 with the cosine similarity in Eq. 3 above a threshold $Thresh$. 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*.

We empirically develop a guideline to set the candidate selection threshold. First, our experiment shows that a fixed threshold for all libraries often lead to an overall unsatisfactory recommendation. For example, low threshold results in too many irrelevant candidates, while high threshold results in too few relevant candidates. This could be attributed to the word embeddings technique

Source lib	**Target Language**	**Top-5 recommendations from word embedding**
jackson	python	simplejson, ~~flexjson~~, ~~pickle~~, ~~jsonserializer~~, ~~attributeerror~~
jfreechart	c#	~~candlestick-chart~~, mschart, ~~shieldui~~, ~~syncfusion~~, ~~radar-chart~~
nltk	java	~~nlp~~, ~~named-entity-recognition~~, opennlp, gate, ~~language-model~~
itextsharp	javascript	~~acrofields~~, ~~itextpdf~~, ~~weasyprint~~, jspdf, ~~html2pdf~~
phpunit	c++	cppunit, cxxtest, cpputest, ~~heap-corruption~~, ~~msvc12~~

Table 1 Examples of filtering results by relational knowledge (in blue) and categorical knowledge (in red)

that learns more accurate tag representations for frequent tags than for less frequent tags, because frequent tags have more training data (Mikolov et al, 2013a). As such, we find that threshold should be adjusted based on the usage frequency of tags. For less frequent tags, higher threshold should be used in order to filter out irrelevant candidates due to less accurate tag embeddings. In contrast, lower threshold could be used for frequent tags so that as many relevant candidates as possible will be selected. In practice, we can determine concrete thresholds for a dataset by sampling library tags with different usage frequencies and manually examine the recommendation results for the sampled libraries following the procedure in Section 4.3. The objective is to achieve a good balance of accuracy and coverage in the overall recommendation. Note that we can estimate the coverage by collecting and examining the top-rank recommended libraries for a group of libraries with similar functionalities.

2.5.3 Refine Initial Results

The initial analogical-library candidates sometimes include libraries that are not for the given programming language (or mobile platform) $base_2$. For example, *itextsharp* is a *c#* library for PDF generation and manipulation. To find analogical libraries for *javascript* as the library *itextsharp* for *c#*, by Eq. 3 we would obtain some libraries, such as *itextpdf* (a library for java and c#), *weasyprint* (a library for python), and *html2pdf* (a library for php). Although these libraries support similar features (e.g., PDF generation and manipulation) to the *itextsharp*, they are not libraries for *javascript*. In our approach, we rely on the correlation between a library and a programming language (or mobile platform) (i.e., relational knowledge) to select the libraries for a given programming language. Specifically, we consider a programming language (or mobile platform) that a library has the strongest association with as the programming language (or mobile platform) that the library is implemented in. Using this relational knowledge, we can exclude libraries that are not for the given programming language (or mobile platform). More examples for filtering irrelevant tags by relational and categorical knowledge can be seen in Table 1. The filtering by relational knowledge is in blue and the filtering by categorical knowledge is in red.

2.6 Assisting Analogical-Libraries Comparison

After knowing some candidate analogical libraries, developers are likely interested in the comparison between the current library and an analogical library, such as runtime performance, reliability, active versions, documentation, community size, to determine whether the recommended analogical library meets their needs. To migrate from the current library to an analogical library, developers also need to understand the process of migration and the efforts needed, for example, how to use the analogical library by referring to the experience with the current library. That is, developers have to collect and study more information about some recommended analogical libraries than just knowing the name of these libraries. We would like to provide developers some information scents to assist their further search and investigation of the recommended analogical libraries. To that end, we develop a keyword-based matching method to extract sentences in Stack Overflow questions and answers that likely assist developers in comparing analogical libraries. It is important to note that this feature is not to replace other effective information retrieval methods (e.g., search engines), but to provide hints about some aspects of analogical libraries when developers discuss and compare them in Stack Overflow questions and answers.

lib pairs	Extracted comparison questions
opennlp, stanford-nlp	Part of speech tagging in OpenNLP vs StanfordNLP
awt, swing	What is the difference between Swing and AWT?
bokeh, matplotlib	Plot topics with bokeh or matplotlib
junit, testng	Drawbacks of TestNG compared to jUnit?
log4net, nlog	log4net vs. Nlog
django-haystack, sphinx	django haystack or sphinx for simple search?

Table 2 Example questions about the comparison of the two analogical libraries in Stack Overflow

lib pairs	related answer snippets
opennlp, stanford-nlp	i liked the stanford parser better than opennlp , again just looking at documents, mostly news articles
awt, swing	swing and awt both provide user interface components , however swing is built on top of awt
innodb,myisam	there are some features that are only available using myisam, like full text search, but unless you need these, i would go with innodb.
junit, testng	testng strives to be much more configurable than junit , but in the end they both work equally well.
d3.js, dc.js	moving onto dc.js ... steps : you need to load the following libraries and css files...
log4net, nlog	nlog seems to be better maintained : an incompatibility of log4net with .net4 remained unresolved in log4net for quite a long time ...
m2crypto, pyopenssl	the analog of command ... in m2crypto is: ... you can use m2crypto instead of pyopenssl with twisted
beautifulsoup, jsoup	jsoup is the java version of beautiful soup

Table 3 Example sentences in Stack Overflow answers about the comparison of the two analogical libraries

2.6.1 Extract Comparison Questions

Given a library and one of its analogical libraries in the analogical-library knowledge base, we try to find Stack Overflow questions whose question title mentions the two libraries. To check if a question title mentions a library, We first lowercase both the question title and the library tag. If the library tag contains hyphen, we consider that a question mentions the library if the question contains any of the original form of the library tag, or no-hyphen form, or hyphen-to-space form (e.g., *stanford-nlp, stanford nlp, stanfordnlp*).

Furthermore, we develop several heuristic rules according to our observation of the extracted questions to remove the false positives that unlikely compare the two libraries:

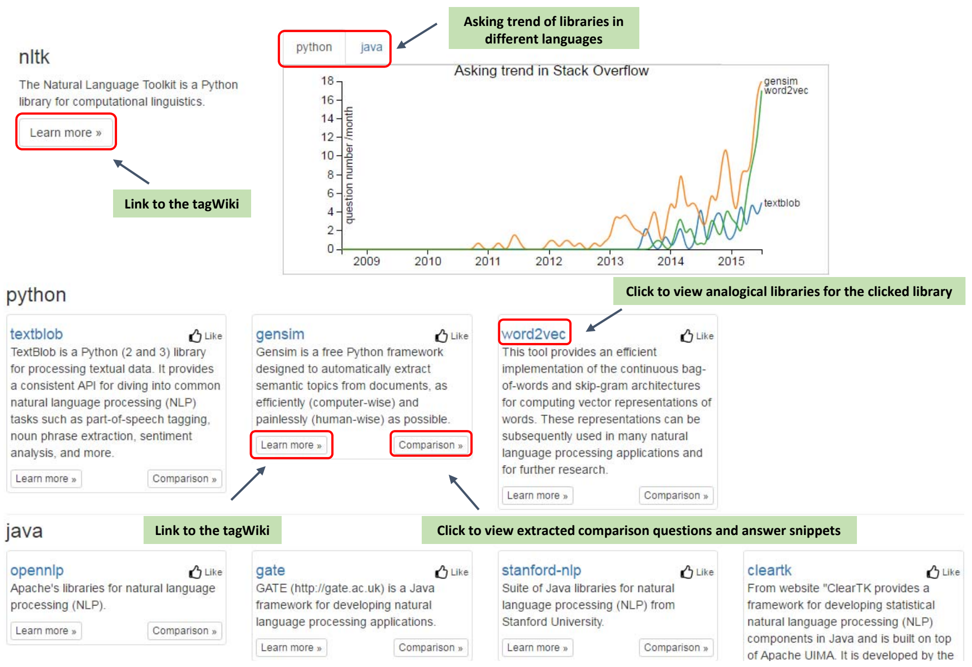
– Question title should not mention the two libraries like “lib1/lib2” format because this format usually means that the two libraries are the same or interchangeable in the question;
– Question should not be asked as “how” statements because such questions are usually about how to use the mentioned libraries;
– There are not quoted elements in the question tile because such questions usually are only about some specific elements of a library.

Table 2 lists some example questions about the comparison of analogical libraries extracted using the proposed keyword-matching method. Such questions provide developers some hints for comparing analogical libraries and selecting suitable ones for their tasks.

2.6.2 Extract Comparison Answer Snippets

A Stack Overflow question may have several answers and/or comments which often comprise a long discussion thread to read. Sometimes developers may want more direct hints, for example, several sentences mentioning and comparing the two libraries Huang et al (2018), instead of the whole discussion thread. Furthermore, some statements about the comparison of the two libraries may appear in the answers to some questions which are not originally asked about the comparison of the mentioned libraries. Therefore, in addition to extracting comparison questions for analogical libraries, we also try to extract comparison sentences in question answers that mentions analogical libraries. Note that we do not extract comparison sentences from question body because the question asker may not understand the two libraries very well and their statements may be misleading or wrong.

We first find all posts containing two libraries, same as the procedures for find comparison questions in last section. To guarantee the quality of the extracted sentences, we only extract sentences from answers with vote larger than 1 ($vote = upvote - downvote$). After extracting the candidate answers that mention the two libraries, we first remove all code snippets (enclosed by HTML tag $< code >$) and then split the answer texts into sentences by punctuation such as ". ", "!", ";", etc. We select sentences that mention the two libraries at the same time. Sometimes, there may be many sentences mentioning the two libraries. To give developers a concise overview of comparison


Asking trend of libraries in different languages
python
java
Asking trend in Stack Overflow
question number /month
18
16
14
12
10
8
6
4
2
0
2009
2010
2011
2012
2013
2014
2015
gensim
word2vec
textblob
Link to the tagWiki
Click to view analogical libraries for the clicked library
Link to the tagWiki
Click to view extracted comparison questions and answer snippets
Like
Learn more »
Comparison »

nltk

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

python

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

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

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

cleartk
From website "ClearTK provides a framework for developing statistical natural language processing (NLP) components in Java and is built on top of Apache UIMA. It is developed by the

Fig. 7 The scrrenshot of our website *SimilarTech* and the important elements in the website

snippets, we take only the two sentences appearing first in the answer as the representative snippets of the answer. After finding the two sentences, we show them in the same order as they appear in the original answer. Finally we rank the candidate answers by their vote to display to the developers.

Some example comparison answer snippets can be seen in the Table 3. We can see that these answer snippets provide more direct hints about different aspects, usage differences and migration steps of the two libraries.

3 Tool Support

This section describes the proof-of-concept implementation of our approach and the practice of search engine optimization so that our website can be indexed by search engines.

3.1 Tool Description

We develop a web application with two parts. One is called *SimilarTech* (`https://graphofknowledge.appspot.com/similartech`) and the other is called *SimilarMobileTech* (`https://graphofknowledge.appspot.com/similartech/mobile.html`). Given a library name, *SimilarTech* automatically recommends its analogical libraries for different programming languages and *SimilarMobileTech* recommends analogical libraries across different mobile platforms. The current backend of *SimilarTech* and *SimilarMobileTech* is an analogical-libraries knowledge base built with the Stack Overflow data dump that contains Stack Overflow post data from July 31st, 2008 to Dec 3rd, 2017. The backend knowledge base can be updated periodically as the new data dump is released.

The data dump we use in the current implementation contains 14,995,834 questions and 41,856 different tags. As some infrequent or emerging tags do not have corresponding TagWiki, we collect in total 36,088 tags that have TagWiki for mining relational and categorical knowledge of tags and for learning tag embeddings. Among 36,088 tags in our dataset, 8,211 tags are categorized as library tags. We use the implementation of continuous skip-gram algorithm (Mikolov et al, 2013b) in Word2Vec[5] to learn tag embeddings. We set tag embedding dimension at 200. By small-scale

[5] `https://code.google.com/p/word2vec/`

pilot study, we empirically set the *support* threshold at 2.3×10^{-6}, and the $confidence$ threshold at 0.15.

In the current implementation, *SimilarTech* recommends analogical libraries for the top-six most frequently-asked programming languages in Stack Overflow, i.e., *java*, *javascript*, *c#*, *php*, *python* and *c++*. *SimilarMobileTech* recommends analogical libraries for the top-three most frequently-asked mobile platforms, i.e., *ios*, *android* and *windows-phone*.

Fig. 7 shows a screenshot of our *SimilarTech*. *SimilarMobileTech* has the same user interface design. Given a library, *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 to the *weka*. In such cases, *SimilarTech* recommends no libraries for that particular programming language. In the same vein, *SimilarMobileTech* may not recommend analogical libraries for a particular mobile platform.

For each recommended analogical library, both *SimilarTech* and *SimilarMobileTech* show a brief definition extracted from the corresponding TagWiki. They 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. This information could provide hints about community size of library users and availability of online learning resources, and offer some indicators of library popularity (Chen and Xing, 2016b).

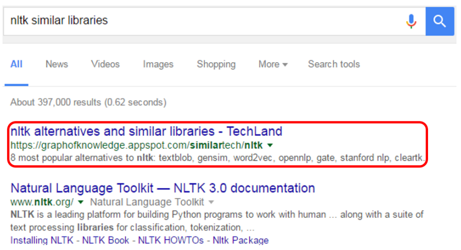
Clicking the recommended library navigates to the analogical-library page of the clicked library. Clicking the button "Comparison" for a recommended library navigates to the comparison page between the searched library and the clicked library. The comparison page presents comparison questions and answer snippets extracted from Stack Overflow discussions, which could aid users to compare commonalities and differences of the two libraries and understand the potential migration issues.

3.2 Search Engine Optimization

Our website provides a portal to the mined analogical-library knowledge base. However, developers will not benefits from our knowledge base unless they are aware of the presence of our website and use the information in our website when they search analogical libraries. Therefore, we carry out search engine optimization (SEO) for our website so that it can be indexed and recommended by search engines. SEO is the process of maximizing the number of visitors to a particular website by ensuring that the site appears high in the search results returned by a search engine for certain types of queries[6].

To achieve the SEO, we take three steps for our website. First, we submit all pages inside our website to several main-stream search engines such as Google, Bing and Yandex so that these search engines will crawl our site and index its content inside their database. Each page in our website contains information about a library and its analogical libraries. Second, based on the recommended analogical libraries for a library, we automatically generate a brief description for each page in our website and add this description as the title and metadata for each page. Search engines use the provided webpage title and metadata to present the webpage in the search results. Examples of webpage title and metadata in the search results can be seen in Fig. 8. The shown webpage title and metadata can lead web searchers to our website for further information. They can also be used as extended suggestions which may help web searchers refine their queries even without

[6] `https://en.wikipedia.org/wiki/Search_engine_optimization`



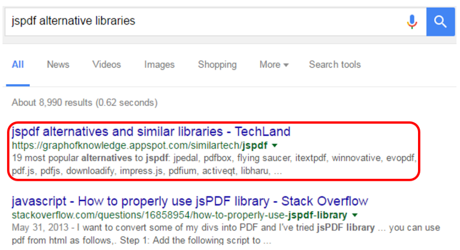
nltk similar libraries

About 397,000 results (0.62 seconds)

nltk alternatives and similar libraries - TechLand
https://graphofknowledge.appspot.com/**similar**tech/**nltk**
8 most popular alternatives to **nltk**: textblob, gensim, word2vec, opennlp, gate, stanford nlp, cleartk,

Natural Language Toolkit — NLTK 3.0 documentation
www.**nltk**.org/ Natural Language Toolkit
NLTK is a leading platform for building Python programs to work with human ... along with a suite of text processing **libraries** for classification, tokenization, ...
Installing NLTK - NLTK Book - NLTK HOWTOs - Nltk Package

(a) nltk similar libraries



jspdf alternative libraries

About 8,990 results (0.62 seconds)

jspdf alternatives and similar libraries - TechLand
https://graphofknowledge.appspot.com/similartech/**jspdf**
19 most popular **alternatives** to **jspdf**: jpedal, pdfbox, flying saucer, itextpdf, winnovative, evopdf, pdf.js, pdfjs, downloadify, impress.js, pdfium, activeqt, libharu, ...

javascript - How to properly use jsPDF library - Stack Overflow
stackoverflow.com/questions/16858954/how-to-properly-use-**jspdf-library**
May 31, 2013 - I want to convert some of my divs into PDF and I've tried **jsPDF library** ... you can use pdf from html as follows,. Step 1: Add the following script to ...

(b) jspdf alternative libraries

Fig. 8 Two example search in Google and our page ranks the first in the search results

visiting our website. Third, we make our website design mobile-friendly, as such optimization can rank our site higher in search engine[7].

After SEO, for some queries (e.g., xxx similar libraries or xxx alternative libraries), our webpages can rank quite high in the search results especially by Google search engine. Two examples can be seen in Fig. 8. The two queries were issued on November 17, 2016 in Singapore and Australia. Our "nltk" and "jspdf" page rank the first for the respective query. As Google adjusts the page ranking not only based on page content and links, but also click rate[8], the high-ranking of our webpages in the search results means that many users truly click our webpages from the search results. According to the Google Analytics for our website, in the past 6 months, on average more than 2000 users around the world visit our site from their Google search results in each month. Our website usage data is limited but promising. It demonstrates the popularity and usefulness of our knowledge base and website. The usage data of our site will be further analyzed in Section 6.

4 Accuracy 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.

4.1 The Accuracy of Tag Categorization

From 33,306 tag with tag category extracted by out method, we randomly sample 1000 tags whose categories are determined using the NLP method, and the other 1000 tags whose categories are determined by the dictionary look-up method (see Section 2.3).

5 final-year Computer Science undergraduate students are recruited to manually examine the category of these 2000 tags by reading their corresponding TagWiki. As it is clear that whether a tag represents a programming language, a library, or a general computing concept and whether the extracted category corresponds to what a tag represents, we assign different participants with different sets of tags, so that we can examine as many results as possible with limited number of participants. In this experiment, each participant is assigned 200 tags from the NLP method, and 200 tags from the dictionary look-up method for checking. Among these 1000 sampled tag categories from NLP methods, 838 (83.8%) tags are correctly extracted by our proposed methods. For the 1000 sampled tags from the word match, 788 (78.8%) of them are accurate. 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 *rpy2* states that "RPy is a very simple, yet robust, Python interface to the R Programming Language". The default POS tagging recognizes *simple* as the noun which is then regarded as the category by our rule. Second, the dictionary look-up method sometimes makes mistakes, as the matched category may not be the real category. For example, the TagWiki of the tag *honeypot*

[7] `https://webmasters.googleblog.com/2016/03/continuing-to-make-web-more-mobile.html`
[8] `https://goo.gl/nb5czF`

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 2000 sampled tags, there are 487 tags whose category can be regarded as library. 401 out of these 487 tags (82.3%) are correctly categorized. The accuracy of our tag categorization provides a solid basis for the analogical-libraries reasoning tasks.

Note that the selection of detailed POS tagging tool (We use *NLTK* in this work) may influence the accuracy of extracting tag category. Therefore, we have also tried POS tagger in Standford-NLP (Manning et al, 2014), and the tag categories of 22,808 (96.8%) out of all 23,658 tags identified by Stanford-NLP POS tagger are the same as the that of NLTK POS tagger.

4.2 The Semantic Distance of Tag Correlations

To evaluate the mined relational knowledge of correlated tags, we adopt the metric similar to “Google distance” (Cilibrasi and Vitanyi, 2007; Gligorov et al, 2007). 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, 2015) 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 non-popular queries[9].

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 could also result 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 semantic distance 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.

4.3 The Accuracy of Analogical-Libraries Recommendation

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

4.3.1 Evaluation procedure and metrics

We randomly sampled 100 libraries as the test cases from our analogical-library knowledge base for programming languages. As analogical-library knowledge base for mobile platforms is smaller, we randomly sampled 40 libraries as the test cases for mobile platforms. These test-case libraries support a diverse set of functionalities, such as visualization, networking, machine learning, searching, testing, and so on. 7 students are recruited (6 PhD student and 1 master student) to evaluate the accuracy of our analogical-library recommendations. All the participants are majored in computer

[9] The detailed threshold to discriminate popular or unpopular queries is a commercial secret of Google.

science and have at least 4-year programming experience. Each of the participants is randomly assigned 20 test-case libraries and they are asked independently judge the accuracy of the recommended analogical libraries for the assigned 20 test-case libraries.

To evaluate the impact of different kinds of knowledge for analogical-library recommendation, we ask the participants to evaluate the accuracy of four different methods: vector offset of tag embeddings and relational and categorical knowledge of tags (denoted as $w2v{+}rc_{kg}$ in the following discussion), vector offset of tag embeddings and relational knowledge of tags ($w2v + r_{kg}$),vector offset of tag embeddings and categorical knowledge of tags ($w2v + c_{kg}$), and vector offset of tag embeddings alone ($w2v$).

As there is no ground truth of analogical libraries, the participants have to manually check each recommended library for a given test-case library. They examine information from library's official website, TagWiki, wikipedia, and other available online information. If the recommended library can provide comparable features as the given test-case library, the recommendation is considered 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.

Our approach is inspired by the use of word embeddings to solve analogy questions of word pairs (Mikolov et al, 2013c). 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 (Manning et al, 2008; Wang et al, 2014) 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.

For a given test-case library, let's assume that *SimilarTech* recommends at least one library for n $(1 \le n \le 6)$ 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 $(1 \le i \le n)$. The $Precision_i@k$ ($k = 1, 2, 3, 4, 5$ in this evaluation) for the programming language PL_i is $correct_i@k/k$. We compute the $Precision@k$ for the given test library 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 library for the given test library.

In the analysis of the recommendation accuracy, we use boxplot to visually compare $Precision@k$ for all the 100 test-case libraries. In addition, we perform t-test (Student, 1908; Vasilescu et al, 2014) statistical comparison of distribution of $Precision@k$ for all the 100 test-case libraries in the four different recommendation settings. As we are concerned with the comparison between our model and other baselines, we carry out t-test between w2v+rckg and each baseline method (w2v+r_kg, w2v+c_kg, w2v). The same evaluate metric applies for the platform-based recommendation by *SimilarMobileTech*.

We use a small k value because there are usually only a small number of analogical libraries for a given library. According to our observation, most of libraries have 0 to 4 analogical libraries. In our current tool, we display up-to four analogical libraries for a programming language in a row (see Fig. 7). Each recommended library has a short description of the library features. In this presentation design, the exact ranking of a library in the recommendation list is not critical, because users can quickly read through the entire recommendation list. Therefore, we do not use rank-based metrics such as mean reciprocal rank in this experiment.

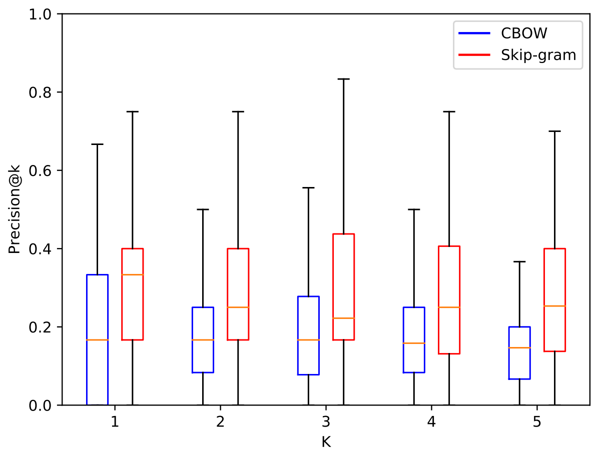
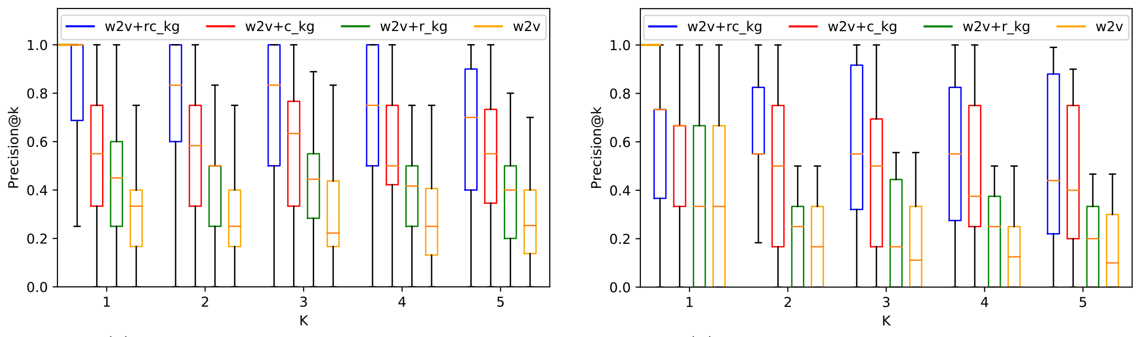


Fig. 9 The comparison between the CBOW and continuous skip-gram model



(a) language-based recommendation (b) platform-based recommendation

Fig. 10 Recommendation accuracy using the combination of different kinds of knowledge

4.3.2 Comparison between Continuous Skip-gram and CBOW

We manually check the accuracy of analogical library recommendations by the two different word embedding models respectively. The result can be seen in Fig 9. It shows that the continuous skip-gram model can learn higher-quality tag embedding than the CBOW model, leading to higher precision of the recommendations with K ranging from 1 to 5. Similar results have also been reported in (Mikolov et al, 2013a) that shows the skip-gram model owns better accuracy, especially for rare words. Therefore, we adopt the skip-gram model as the default tag embedding model in this work, and all the following experiments are based on the skip-gram model.

4.3.3 Accuracy results

Fig. 10 illustrates the Precision@k of the four recommendation methods based on different kinds of knowledge. We can see that the tag-embeddings-only (w2v only) recommendation performs poorly. The median recommendation accuracy by tag embedding alone is about 0.3 in the top-1 recommendation, and even lower as 0.2 in the top-5 recommendation. This is because tag sentences are short and 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. The median of Precision@1 is 1.0 and the Precision@5 is still reasonably high at 0.7. Categorical knowledge of tags can boost the accuracy more than relational knowledge of tags. Incorporating both knowledge yields the best accuracy. Our results suggest that incorporating domain-specific categorical and relational knowledge with tag embeddings can enhance analogical reasoning tasks in software engineering context. We adopt the T∼procedures (Vasilescu et al, 2014) to evaluate the significance of our results compared with other baseline settings. The $p - value$ (< 0.05) between w2v+rckg and other settings shows that our results are significant.

Our results show that, given a test-case library, the top-1 library that our approach 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. The relatively lower precision for platform-based recommendation could be attributed to the nature of mobile platforms. Unlike general programming languages which can be used for many common tasks, each mobile platform usually has many unique features that differentiate one platform from another. For example, *viewdeck* is a library for iOS apps which provides new features and enchanted view controllers. *mfi* is the Apple program that hardware developers must join to be able to manufacture and brand products as being made for iOS devices. The libraries that are highly unique for a specific mobile platform usually have no alternatives for the other mobile platforms. As such, although the recommended libraries across mobile platforms may have some similarities to a given library, they are often not analogical libraries, because some features are unique to one mobile platform and do not actually have counterparts on the other mobile platform. This results in the lower precision for platform-based recommendation.

To verify if the association rules can discover analogical libraries, we conduct a simple but effective experiment. Among all 100 language-based test libraries manually checked in Section 4.3.1, we have collected 790 pairs of analogical libraries which are marked as correct. We then count the frequency of the co-occurrence of pairs of analogical libraries in the tag sentences in the whole dataset. The results show that only 142 (18%) of these pairs of analogical libraries co-occur in the same tag sentences more than 10 times. But compared with totally 14,995,834 tag sentences, 10 times is too small to be extracted by association rules unless the support threshold is set extremely low ($10/14,995,834 = 6.710^{-7}$). But this extremely low support threshold will produce too many false positives of association pairs. Furthermore, even the support value is set at this extremely low threshold, association rules can only get at most 18% analogical library pairs but also a lot of false positive pairs. That is why it is impossible to use the association rules mining alone to obtain our knowledge base of analogical libraries.

4.3.4 Analysis of inaccurate recommendations

Although our approach can make accurate analogical-libraries recommendations in most cases, as the first work of this kind that combines word embeddings technique 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 or mobile platform.

For the first case, an example is *ruby-on-rails* (a web application framework for *Ruby*). We expect that our approach 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 *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. As illustrated in the *viewdeck* and *mfi* examples above, unique libraries for mobile platforms often result in poor analogical-libraries recommendation across mobile platforms.

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

4.3.5 Recommendation changes due to Stack Overflow data evolution

For the 100 language-based test libraries, we manually compare the recommendation results from the latest data with results in our previous work (Chen et al, 2016a) which adopt the data from July 2008 to Aug 2015. most of the recommendation results from two datasets are the same because most core technologies are still the same in just a few years. But meanwhile, we also observe three frequent change patterns. First, some old technologies in the previous recommendations disappear in the current recommendation. For example, *cleartk*[10] (a framework for developing NLP components in Java) is no longer in the top list of analogical libraries to *NLTK*, as *cleartk* stopped its update two year ago. Second, some new technologies begin to appear in the current recommendations. Let's take the *theano* (a numerical computation library for Python, widely used in deep learning) as an example, its analogical libraries now include the latest deep learning libraries such as *Tensorflow* and *PyTorch* which are released in the recent years. Third, some emerging libraries are replacing old libraries. For instance, *PIL* (Python-Image-Library) is a python library for manipulating images. But its development appears to be discontinued with the last commit to the PIL repository in 2011. Consequently, a successor project called *Pillow* has forked the PIL repository and added more functionality and Python 3.x support. More users begin to use *pillow* rather than *PIL*. Therefore, *pillow* is surpassing *PIL* for some recommendation to image libraries like *opencv*.

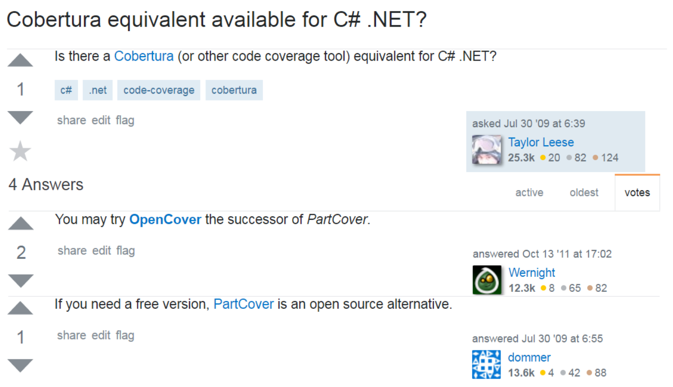
4.4 The Relevance of Comparison Questions & Answer Snippets

As our approach extracts comparison snippets about the two analogical libraries, we randomly sample 70 pairs of analogical libraries for this experiment. Each of the 7 participants (same as the last section) are randomly assigned 10 pairs of libraries. They are asked to evaluate whether or not an extracted comparison question or answer snippet is about the comparison of some aspects of a given pair of libraries. Before the real experiment, we conduct a pilot study to evaluate the potential discrepancies that could be made by different annotators. The pilot study involves the first two authors examining the extracted comparison snippets for another five randomly sampled pairs of analogical libraries. We find that the assessment is very straightforward and the two annotators do not have discrepancies regarding whether an extracted comparison question or answer snippet is about the comparison of a given pair of libraries. Furthermore, we would like to examine as many results as possible with only limited number of participants. Therefore, we assign different participants with different sets of library pairs in the real experiment.

For each pair of libraries, the participants evaluate the top-5 extracted comparison questions and answer snippets. Note that some library pairs may have less than 5 extracted questions or answer snippets. For this experiment, the participants evaluate in total 315 extracted questions and 336 answer snippets for the sampled 70 pairs of analogical libraries.

203 (64.4%) of questions and 253 (75.3%) are marked as related to the comparison about the two analogical libraries. We further check why some questions and answer snippets are marked as irrelevant of comparison. We find that some questions are about the migration from one library to another library such as *"StructureMap to Ninject conversion"*. Although such questions or answer snippets are not related to the comparison, it could still be useful information for developers who look for analogical libraries, as they could help developers understand the migration process and avoid some potential mistakes by learning others' experience. Some comparison-irrelevant questions/answer snippets are about how the two libraries can complement each other. For example, one answer snippet of *nltk* and *stanford-nlp* is *"always refer to ... for the latest instruction on how*

[10] `https://github.com/ClearTK/cleartk`


Cobertura equivalent available for C# .NET?

Is there a Cobertura (or other code coverage tool) equivalent for C# .NET?

c# .net code-coverage cobertura

asked Jul 30 '09 at 6:39 Taylor Leese 25.3k ●20 ●82 ●124

4 Answers

You may try **OpenCover** the successor of *PartCover*.

answered Oct 13 '11 at 17:02 Wernight 12.3k ●8 ●65 ●82

If you need a free version, PartCover is an open source alternative.

answered Jul 30 '09 at 6:55 dommer 13.6k ●4 ●42 ●88

Fig. 11 An example analogical question and its answers in Stack Overflow. Relevant libraries are often hyperlinked in the answers.

to interface stanford nlp tools using nltk ...". It tells how users can interface *stanford-nlp* with *nltk*. This could be useful, because *stanford-nlp* may have better performance in some aspects but developers may still want to use *nltk* due to Python's convenience.

5 Usefulness Evaluation

To demonstrate the usefulness of the proposed approach for analogical-library recommendation, we sample some questions about analogical-library recommendation in Stack Overflow, and investigate how well our recommendation can answer such questions, compared with answers provided by Stack Overflow users.

5.1 Experimental Setup

In Stack Overflow, there are many analogical questions such as "*Is there a C++ unit testing library that is similar to NUnit?*" (in Fig. 1) and "*Cobertura equivalent available for C# .NET?*" (in Fig. 11). We define several heuristic rules (e.g, question title contains "similar library", "alternative", "equivalent libraries") to collect a set of candidate analogical questions in Stack Overflow. Then we manually filter out some inappropriate questions which is not about library recommendation, and finally sample 50 questions with more than one answers as the language-based analogical questions and 20 questions with more than one answers as mobile-based analogical questions[11].

After that, we recruit 7 students (same to the Section 4.3.1) to extract all the recommended libraries from the answers. We find that for these analogical questions, answerers often add a hyperlink to the recommended library so that readers can access the relevant resource for more details about the library (see Fig. 11). Based on this observation, we ask the participants to pay more attention to such elements. To make the extracted library mentions consistent with the format of tags in Stack Overflow, we lowercase all of them and replace the space with "-". After building the ground truth sets for the sampled analogical questions, we check how many of the ground-truth answers provided by Stack Overflow users are covered by our recommendation results for each analogical question.

5.2 Results

10 sample questions and answers can be seen in Table 4. For the 50 language-based analogical questions, on average, 71.3% libraries in answers provided by Stack Overflow users are covered by the recommended libraries using our approach. The average coverage rate is 62.3% for the 20 mobile-based analogy questions. It means that the majority of the libraries mentioned in the Stack Overflow answers can be automatically recommended by our approach.

[11] The list of sampled questions can be found at `https://graphofknowledge.appspot.com/questions`

Question	**Stack Overflow answers**	**Our recommendation**
Cobertura equivalent available for C# .NET?	ncover, opencover, partcover	opencover,visual-studio-test-runner,partcover
Alternative for PHP GD library in python	python-imaging-library	python-imaging-library, pillow, scikit-image
Modern alternative to Java XStream library?	jaxb, xmlbeans, jibx	simple-framework, castor, xmlunit, jibx
Alternative to Java3D	jogl, jMonkeyEngine	jMonkeyEngine, jogl, jzy3d, worldwind
Open source Enthought Python alternative	anaconda, pythonxy	healpy, miniconda, pythonxy, canopy
PIL ImageTk equivalent in Python 3.x	pillow	pillow, pypng, scikit-image,pythonmagick
Crypto++ equivalent in C	polarssl, openssl	botan, polarssl, cryptoapi
Java equivalent for Python NLTK	opennlp	gate, opnenlp, stanford-nlp, cleartk
iAd alternative for ios ad display?	adwhirl, adsense	adwhirl, chartboost, openfeint
ASIHTTPRequest equivalent for Android?	android-async-http	android-async-http, multipartentity

Table 4 Example analogical questions and their answers from Stack Overflow and analogical libraries recommended by our method

We further explore the libraries in Stack Overflow answers but are not covered by our recommendation. We conclude two reasons for the missing. First, some libraries in Stack Overflow answers are indeed in our recommendation list (i.e., above the candidate selection threshold, such as *ncover* for the first question in Table 4). But in this experiment, we consider only the top 4 recommended libraries as they are what our tool currently presents, and exclude the lower-ranked recommendations. Second, some libraries in Stack Overflow answers may be helpful for the analogical library mentioned in the answers, but these libraries themselves are not analogical libraries. For example, *pandas* can help the Pythons' visualization tool *matplotlib* conveniently visualize the data, but *pandas* itself can not be regarded as an alternative to the Javascript's visualization library *d3.js*. Such auxiliary libraries will not be covered in our recommendation which results in the decrease of the coverage.

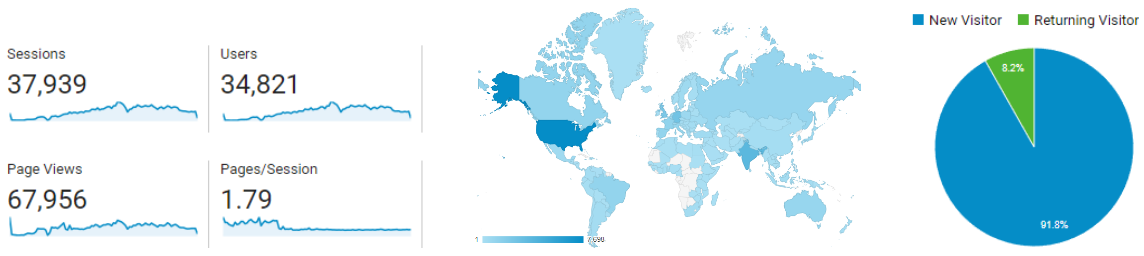
Apart from the covered libraries, our recommendation also contains some extra libraries that are not in Stack Overflow answers. According to our observation, an important reason for such extra libraries in our recommendation is the emerging new libraries that are not available at the time when the questions were asked and answered. For example, the second question in Table 4 was asked five years ago, but the new analogical libraries *pillow* and *scikit-image* that our approach recommends appear only about three years ago. Old posts in Stack Overflow are rarely updated with such new development in the field, which is a key issue in finding analogical libraries using these online posts. Our approach provides an alternative to recommend more up-to-date information.

6 Field Study

We release our website *SimilarTech* to the public in November 2015 and post this news on several programming-related websites (e.g., `http://stackapps.com/questions/6667`, `http://stackapps.com/questions/6924`). Google Analytics[12] is embedded into our site to monitor our site traffic. Furthermore, we record the detailed page visit history in our backend server, i.e., which pages users visit, when users visit our website, and the IP address of users. We also log the users' interaction with the webpage content when they click recommended libraries, library TagWiki links, comparison button, and asking trend tabs. These detailed user behavior data allows us to gain insights into our approach and tool support which may benefit research of similar recommendation systems. As we release the website *SimilarMobileTech* recently, we do not collect enough usage data at the time of this submission. Therefore, we only analyze the usage data of our *SimilarTech* website. In addition to the general site traffic statistics, we further investigate four research questions regarding who visits our website and what the users are interested in in our website:

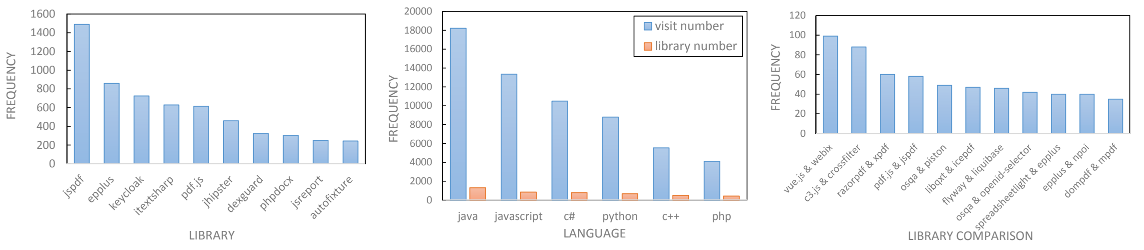
- RQ1: For which libraries and libraries comparisons are the users most interested in seeking analogical libraries?
- RQ2: Do the users like to find analogical libraries within the same programming language or across different languages?
- RQ3: Do the users explore the TagWiki and asking trend that our website provides?
- RQ4: Do professional developers from major IT companies visit our site?

[12] `https://analytics.google.com/`



(a) Visiting statistics (b) Geo map (c) Returing users

Fig. 12 The traffic of our website from Google Analytics



(a) Top-10 libraries (b) Visit frequency by language (c) Top-10 comparisons

Fig. 13 (a) The top-10 most-frequently visited libraries in our website. (b) The number of libraries in each language that have been visited and the total number of visits. (c)The top-10 most-frequently visited library comparisons

6.1 Site Traffic Statistics

According to the results from Google Analytics, more than 34,821 users from 168 countries visited our site, from November 11, 2015 to August 29, 2017. As shown in Fig. 12(a), these users on average browse 1.79 pages in each session and they browse in total more than 67,956 pages in 37,940 sessions[13]. The top 4 countries are the US (20.3%), India (9.7%), Germany (6.9%), and the UK (5.7%) (Fig. 12(b)) which account for 42.6% visits. 8.2% users visit our site more than once (Fig. 12(c)) which shows their recurring interests in our tool. The usage data of our website, albeit limited, demonstrates both the needs and the interests in analogical-libraries recommendation that our approach supports.

We note that the per-session page visit is not very high for our website. This can be attributed to the design rationale of our website. The users of our website are expected to have on mind specific information needs for certain analogical libraries when they visit our website. Our website is designed to provide the users with a concise summary of the information they may need to find analogical libraries. Then, the users may click library TagWiki or extracted comparison questions and answer snippets to obtain more information about the recommended library in Stack Overflow. Or they may leave our website to search the recommended library for more information. As a result, we do not expect a high per-session page visit. In fact, through search-engine optimization, the users may obtain the key information they need (i.e., the name of the analogical libraries) from the webpage metadata displayed in the search results. They may then search the library names directly, without visiting our website. Even in this scenario, our website still fulfills its design goal to assist developers' analogical-library search.

6.2 Most Interested Libraries, Languages and Comparisons (RQ1)

According to the logs of webpages visited, 3,929 libraries in the *SimilarTech* website have been visited which account for 50.5% of all the libraries in our knowledge base. The top-10 most frequently visited libraries can be seen in Fig. 13(a) and all of them have been visited more than 200

[13] As most search engine robots do not activate javascript, robot traffic is not counted in Google Analytics (Google, 2016).

times. As mentioned in Section 2.2, the relational knowledge of tags can tell us which programming language each library is primarily implemented in. For the six programming languages that our website currently supports, we count the number of libraries in each language that have been visited and the total number of visits. Fig 13(b) shows that libraries in Java are most frequently visited (1,311 libraries and 18,314 times) and libraries in PHP is the least visited (maybe due to the relatively fewer third-party libraries in PHP). Fig 13(c) depicts the top-10 most frequently visited library comparisons. There are totally 9,395 visits for 6,572 different library comparisons, and such statistics demonstrate the interest of users in further comparisons between similar libraries. Note that the comparison service is launched in our website in 2017, so we do not collect as much data as similar library visits.

6.3 Analogical Libraries Within or Across Languages (RQ2)

Our approach recommends analogical libraries for the same programming language as the given library and across different languages. When the users search a library or view information of a library in the *SimilarTech* website, we consider that the users are interested in the library. We attempt to estimate the developers' interests in analogical libraries within the same langauge or across different languages by analyzing the logs of how users search libraries and view library information in our website.

In our website, users can enter a library name in the search box, and the website shows the analogical-library page for the searched library. On an analogical-library webpage, users can click a recommended library which brings the users to the analogical-library page for the clicked library. Users can also click "Learn more" to view the TagWiki of a recommended library. We collect the sequence of the user's actions (excluding the visit of the home page) in our website during a visit session.

As we want to analyze the users' interests in analogical libraries within or across languages, sequences with only one action are ignored. From the 11465 sequences that contain at least two actions, we extract pairs of consecutive actions that involve a library searched followed by an analogical library searched/viewed. We obtain 15039 such pairs. For each pair of libraries, we check the language of the libraries. Among 15039 pairs, 12721 pairs of libraries are in the same language, and the rest 2318 pairs of libraries are across different languages.

The users' search and browsing behavior in our website seems to suggest that the users are more interested in analogical libraries within the same language than across different languages. Two reasons may account for such results. First, we place the same-language analogical libraries in the first row in the web page. This may attract more clicks. Second, many developers are most familiar with one programming language. Thus, they may prefer analogical libraries in the same language over using some libraries in another language they are not good at.

We further analyze the potential migration pattern of developers when they are interested in analogical libraries across different languages, i.e., developers want to find analogical libraries from which language to the other. Table 6.3 displays the results of the language migration matrix. We can see that java and *c#* developers more likely search for analogical libraries in other languages, compared with developers of other languages. On the other hand, *python* seems to attract developers from other languages to change to use libraries in *python*. This could be because *python* has many libraries that are good alternatives for libraries in other languages and also maybe due to its popularity in deep learning. For *php*, it seems that PHP developers do not often change to use libraries in other languages, and developers of other languages do not often change to use PHP libraries either.

It is important to note that this analysis could be biased for two factors. First, users may simply view the information on an analogical-library page without clicking any recommended libraries and/or their TagWikis. In fact, we have about 71.1% action sequences containing only one action. In such cases, users may still have interests in some information in the webpage, for example, view asking trend, highlight some words in the TagWiki snippets while reading them, copy some keywords for their further search. However, we do not have clear signals about what they may be interested in. Second, it is very likely that after finding some hints for analogical

source \ **target**	java	javascript	python	c#	c++	php	**sum**
java		117	226	234	94	38	709
javascript	63		94	87	41	52	337
python	107	44		84	81	26	342
c#	150	89	101		110	26	476
c++	52	23	148	59		11	293
php	24	33	57	35	12		161
sum	396	306	626	499	338	153	

Table 5 Language migration statistics

libraries in our website, users leave our website to google the libraries for more details. We cannot collect any behavior data (like libraries searched and webpages read) outside our website.

6.4 Usefulness of TagWiki and Asking Trend (RQ3)

Our web application does not just provide the names of analogical libraries, but also provides a brief description of the recommended library and a summary of asking trends of the recommended libraries. We collect the users' interaction with the provided information in order to understand whether they are useful or not. The behavior tracking component was deployed in July 3, 2016. The analysis below is based on the data collected from July 3, 2016 to Aug 29, 2017.

In the last 14 months, apart from the homepage, 42,127 library pages in our websites are visited. Among these web pages, users access the TagWiki of the searched library, asking trend tabs, the TagWiki for the recommended libraries, and/or the library comparison in 15,730 pages (37.3%). Users access the TagWikis of some recommended libraries in 2,936 pages. Users click asking trend tabs for different languages in 2,224 pages. It indicates that some users are not only interested in knowing the name of analogical libraries, but also want to know more details about our recommendations. Surprisingly, users click the TagWiki of the searched library in 11,314 pages. This could be because the design of our website which may mislead the users to click the link of a recommended library when they only want to read the TagWiki of the library. Within 1,522 pages, the user clicked the comparison button to compare the searched library with the recommended libraries. In the current design, clicking the link of a recommended library brings the users to the analogical-library page for the clicked library, while the uses must click "Learn more" for a library to access its TagWiki. When the users find that clicking the link of the recommended library does not lead them to the TagWiki, they may then click "Learn more" for the clicked library on its analogical-library page to access its TagWiki. We will further improve our website design to make it more user-friendly.

Although many users do not explicitly click any elements in the web pages they visit, it does not mean that they do not get the information they need. It is likely that the users find the information they need from the TagWiki snippets and the asking trends that is already presented in the webpage. For example, the web page displays the asking trends of the analogical libraries in the same programming language as the searched library by default. If the users are only interested in analogical libraries in the same programming language as the searched library (according to the analysis in Section 6.3, this is likely the case), they do not need to click any asking trend tabs to get the trend information they need. Furthermore, it is impossible to track how the users use the information from our website once they leave our website. Therefore, our analysis provides a conservative estimate of how useful the information our webpage provides could be.

6.5 Real Developers' Visits (RQ4)

For each visitor of our website, we record their IP address. Given the IP address, we can find their Internet Service Provider using the ipinfo.io service[14]. From the ISP, we can know the organization to which the IP address belongs. Table 6.5 lists several IT companies whose IP addresses visit our website frequently. As the visits to our website only last several web pages, we rule out the

[14] `http://ipinfo.io/`

Company	**#IPs**
Google Inc.	218
Amazon.com, Inc.	207
Microsoft Corporation	98
Alibaba (China) Technology Co., Ltd.	73
Cisco Systems, Inc.	42
Oracle Corporation	41
Apple Inc.	37
Hewlett-Packard Company	35
Intel Corporation	32
Facebook, Inc.	19

Table 6 The number of visits from big companies

Metrics	**Company**	**Rest**
returning visitors	22.1%	8.2%
pages per session	1.47	1.79

Table 7 The comparison of visits between developers from big companies and the rest

possibility of the web crawler. In addition, as our website is related to programming, we assume the visits from these companies are from their developers.

We further explore the visit logs of the users from the listed IT companies, and compare their behaviors with that of other users. We have in total 801 users from the listed IT companies, and these users browse 1440 pages in 979 sessions. Table 6.5 shows the behavior difference between the IT company users and the other users. It is more likely that the company users will revisit our website (22.1%) than the other users (8.2%). But the company users visit fewer pages (1.47) in each session than the other users (1.79). This could be because these company users are more experienced and have specific information needs on mind. Therefore, they do not need to explore the recommended analogical libraries in our website.

We then analyze the click behaviors of these company visitors to conclude their migration patterns. Among all 221 sessions with more than one query, we find that most of users (90%) are interested in the analogical libraries within the same language, and the most popular languages are *Java* and *Javascript* which are widely used in the industry. For the cross-language clicks, the most frequent migration is from C++ to *Python*, *Java* to *Python*, and *Java* to *Javascript* which indicates the interests migrating from objected-oriented programming language to scripting languages. Note that as the log size is small and sparse, we do not find very frequent library migrations. In addition, this observation may not represent the real trends of the industry.

7 Related Work

7.1 Recommendation system in Software Engineering

Recommendation systems are widely utilized in Software Engineering context. Many applications have been proposed to recommend code snippets for developers, such as Jungloid (Xu et al, 2005), ParseWeb (Thummalapenta and Xie, 2007), MAPO (Zhong et al, 2009). Chen et al (Chen et al, 2016b,a) recommend technology landscape for developers to have an overview of certain high-level technologies. Some works (Chan et al, 2012; Thung et al, 2013b) 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[15]. Many researchers have proposed methods to assist code migration, such as code mapping (Nguyen et al,

[15] `http://stackoverflow.com/questions/212151/`

2013), function mapping (Teyton et al, 2013), and API migration (Zhong et al, 2010; Nguyen et al, 2014). In contrast to these code-level migration, our approach supports library-level migration.

Thung et al analyze the library co-occurrence patterns in software projects to recommend relevant libraries for a software project(Thung et al, 2013a). Teyton et al analyze the evolution of projects' dependencies (Teyton et al, 2012, 2014) 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.

7.2 Word Embedding in Software Engineering

Our approach is motivated by the recent success of using word embedings techniques to solve semantic and syntactic analogy tasks in NLP applications (Mikolov et al, 2013b; Turney, 2006). 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 works (Xu et al, 2014; Zhou et al, 2015), we propose to incorporate relational and categorical knowledge of tags into tag embeddings to improve the accuracy of analogical-libraries reasoning tasks. Different from their works (Xu et al, 2014; Zhou et al, 2015) in which relational and categorical knowledge are provided by human experts, our approach mines domain-specific knowledge automatically from Q&A discussions and community wikis on Stack Overflow.

Word embedding has recently been adopted in analyzing software engineering data. Some researchers (Vu et al, 2015, 2016) mine user opinions in the mobile application reviews by extracting semantically related words by word embedding. Ye et al (2016b) improve the information retrieval in software engineering text by using word embeddings to measure document similarities. There are also works (Van Nguyen et al, 2016; Nguyen et al, 2016, 2017) encoding API elements with word vector representation and then map word vectors for code migration. Chen et al (2017b) combine word embedding and lexical rules to extract abbreviations and synonyms of software-specific terms.

Word embeddings can also be incorporated into deep learning methods like Convolutional Neural Network for different tasks in software engineering, such as predicting semantically linkable knowledge (Xu et al, 2016), cross-lingual information retrieval (Chen et al, 2016c), and duplicate bug retrieval (Deshmukh et al, 2017). Different from these works, we not only encode tags with word embedding techniques, but also exploit relational and categorical knowledge of tags to reason about analogical libraries.

Our approach assumes that Stack Overflow questions have high-quality tags through its collaborative editing mechanism. This assumption is based on an empirical study of the trade-offs of introducing collaborative editing model to Stack Overflow (Li et al, 2015; Chen et al, 2017a, 2018) in the social computing community. Several studies (Xia et al, 2013; Wang et al, 2014) in software engineering community also investigate the quality of Stack Overflow tags and they propose machine learning algorithms for recommending missing tags based on the correlation analysis of question content and tags. These tag improving techniques can be integrated into the collaborative editing process to improve the efficiency of collaborative editing, which may subsequently increase the quality of question tags taken as input in our method.

Association rule mining (Agrawal et al, 1993) is a rule-based machine learning method for discovering interesting relations between variables in large databases. To filter out insignificant rules, lift and fisher exact test (Webb, 2006) has been adopted, besides the min-support and min-confidence. However, among all mined association pairs, we are only concerned with pairs of a library and a programming language in this work. As it is a common practice in Stack Overflow that users tag their questions with relevant programming language and libraries, the associations between pairs of related library and programming language have generally higher support and confidence than the associations between pairs of other categories of tags. Therefore, based on the tags' categorical knowledge and a relatively high min-support and min-confidence threshold, association pairs of library and program language can be well extracted, without the need to resort to more complex lift with Fisher exact test method.

7.3 Practical Tools in Software Recommendations

It is worth mentioning some related non-academic projects. SimilarWeb[16] is a website that provides both users engagement statistics and similar competitors for websites and mobile applications. AlternativeTo[17] 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.

8 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. 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 or different mobile platforms.

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, and the relevance of comparison questions and answer snippets. Our approach achieves very promising results for analogical-libraries recommendation. We also implement our approach in a web application[18] and release the web application for public use and evaluation. Our analysis of the visit data of our web application reveals the needs for future research on API-level migration support across frequently-seen library migration. The usage data also shows the importance of the wiki-style description of technologies and the technology trend analysis for recommendation tasks.

In the future, we will 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

[16] `www.similarweb.com/`
[17] `http://alternativeto.net/`
[18] `https://graphofknowledge.appspot.com/similartech`

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.

9 Acknowledgement

We'd like to appreciate the valuable review from reviewers. This work is partially supported by the seed grant from Monash University.

References

Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: Acm sigmod record, ACM, vol 22, pp 207–216

Agrawal R, Srikant R, et al (1994) Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB, vol 1215, pp 487–499

Barua A, Thomas SW, Hassan AE (2014) What are developers talking about? an analysis of topics and trends in stack overflow. Empirical Software Engineering 19(3):619–654

Bird S (2006) Nltk: the natural language toolkit. In: Proceedings of the COLING/ACL on Interactive presentation sessions, Association for Computational Linguistics, pp 69–72

Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10):P10,008

Chan WK, Cheng H, Lo D (2012) Searching connected api subgraph via text phrases. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, p 10

Chen C, Xing Z (2016a) Similartech: automatically recommend analogical libraries across different programming languages. In: Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on, IEEE, pp 834–839

Chen C, Xing Z (2016b) Towards correlating search on google and asking on stack overflow. In: The 40th IEEE Computer Society International Conference on Computers, Software & Applications, IEEE, pp 83–92

Chen C, Gao S, Xing Z (2016a) 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), IEEE, vol 1, pp 338–348

Chen C, Xing Z, Han L (2016b) Techland: Assisting technology landscape inquiries with insights from stack overflow. In: Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on, IEEE, pp 356–366

Chen C, Xing Z, Liu Y (2017a) By the community & for the community: A deep learning approach to assist collaborative editing in q&a sites. PACMHCI 1(CSCW):32:1–32:21

Chen C, Xing Z, Wang X (2017b) Unsupervised software-specific morphological forms inference from informal discussions. In: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, pp 450–461

Chen C, Chen X, Sun J, Xing Z, Li G (2018) Data-driven proactive policy assurance of post quality in community q&a sites. vol 2, pp 33:1–32:22

Chen G, Chen C, Xing Z, Xu B (2016c) Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 744–755

Chen W, Zhang Y, Zhang M (2014) Feature embedding for dependency parsing. In: Proceedings of the International Conference on Computational Linguistics

Cilibrasi RL, Vitanyi P (2007) The google similarity distance. Knowledge and Data Engineering, IEEE Transactions on 19(3):370–383

Deshmukh J, Podder S, Sengupta S, Dubash N, et al (2017) Towards accurate duplicate bug retrieval using deep learning techniques. In: Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on, IEEE, pp 115–124

Gligorov R, ten Kate W, Aleksovski Z, Van Harmelen F (2007) Using google distance to weight approximate ontology matches. In: Proceedings of the 16th international conference on World Wide Web, ACM, pp 767–776

Google (2015) Google trends. `https://www.google.com.sg/trends/`

Google (2016) Google analytics policy. `https://support.google.com/analytics/answer/1315708?hl=en`

Huang Y, Chen C, Xing Z, Lin T, Liu Y (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, pp 214–224

Kazama J, Torisawa K (2007) 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), pp 698–707

Li G, Zhu H, Lu T, Ding X, Gu N (2015) Is it good to be like wikipedia?: Exploring the trade-offs of introducing collaborative editing model to q&a sites. In: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, ACM, pp 1080–1091

Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S, McClosky D (2014) The stanford corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations, pp 55–60

Manning CD, Raghavan P, Schütze H, et al (2008) Introduction to information retrieval, vol 1. Cambridge university press Cambridge

Mikolov T, Chen K, Corrado G, Dean J (2013a) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013b) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119

Mikolov T, Yih Wt, Zweig G (2013c) Linguistic regularities in continuous space word representations. In: HLT-NAACL, pp 746–751

Nasehi SM, Sillito J, Maurer F, Burns C (2012) 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, pp 25–34

Nguyen AT, Nguyen HA, Nguyen TT, Nguyen TN (2014) 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, pp 457–468

Nguyen TD, Nguyen AT, Nguyen TN (2016) Mapping api elements for code migration with vector representations. In: Proceedings of the 38th International Conference on Software Engineering Companion, ACM, pp 756–758

Nguyen TD, Nguyen AT, Phan HD, Nguyen TN (2017) Exploring api embedding for api usages and applications. In: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, pp 438–449

Nguyen TT, Nguyen AT, Nguyen HA, Nguyen TN (2013) A statistical semantic language model for source code. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM, pp 532–542

Student (1908) The probable error of a mean. Biometrika pp 1–25

Teyton C, Falleri JR, Blanc X (2012) Mining library migration graphs. In: Reverse Engineering (WCRE), 2012 19th Working Conference on, IEEE, pp 289–298

Teyton C, Falleri JR, Blanc X (2013) Automatic discovery of function mappings between similar libraries. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, IEEE, pp 192–201

Teyton C, Falleri JR, Palyart M, Blanc X (2014) A study of library migrations in java. Journal of Software: Evolution and Process 26(11):1030–1052

Thummalapenta S, Xie T (2007) 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, pp 204–213

Thung F, Lo D, Lawall J (2013a) Automated library recommendation. In: Reverse Engineering (WCRE), 2013 20th Working Conference on, IEEE, pp 182–191

Thung F, Wang S, Lo D, Lawall J (2013b) Automatic recommendation of api methods from feature requests. In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, IEEE, pp 290–300

Turney PD (2006) Similarity of semantic relations. Computational Linguistics 32(3):379–416

Van Nguyen T, Nguyen AT, Nguyen TN (2016) Characterizing api elements in software documentation with vector representation. In: Proceedings of the 38th International Conference on Software Engineering Companion, ACM, pp 749–751

Vasilescu B, Serebrenik A, Goeminne M, Mens T (2014) On the variation and specialisation of workload—a case study of the gnome ecosystem community. Empirical Software Engineering 19(4):955–1008

Vu PM, Nguyen TT, Pham HV, Nguyen TT (2015) Mining user opinions in mobile app reviews: A keyword-based approach (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE, pp 749–759

Vu PM, Pham HV, Nguyen TT, et al (2016) Phrase-based extraction of user opinions in mobile app reviews. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 726–731

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

Wang S, Lo D, Vasilescu B, Serebrenik A (2017) Entagrec++: An enhanced tag recommendation system for software information sites. Empirical Software Engineering DOI 10.1007/s10664-017-9533-1

Webb GI (2006) Discovering significant rules. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 434–443

Wu Y, Wang N, Kropczynski J, Carroll JM (2017) The appropriation of github for curation. PeerJ Preprints 5:e2952v1

Xia X, Lo D, Wang X, Zhou B (2013) Tag recommendation in software information sites. In: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, IEEE, pp 287–296

Xu B, Ye D, Xing Z, Xia X, Chen G, Li S (2016) Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, pp 51–62

Xu C, Bai Y, Bian J, Gao B, Wang G, Liu X, Liu TY (2014) 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, pp 1219–1228

Xu DML, Bodık R, Kimelman D (2005) Jungloid mining: Helping to navigate the api jungle. In: POPL

Ye D, Xing Z, Li J, Kapre N (2016a) Software-specific part-of-speech tagging: an experimental study on stack overflow. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM, pp 1378–1385

Ye X, Shen H, Ma X, Bunescu R, Liu C (2016b) From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 404–415

Zhong H, Xie T, Zhang L, Pei J, Mei H (2009) Mapo: Mining and recommending api usage patterns. In: ECOOP 2009–Object-Oriented Programming, Springer, pp 318–343

Zhong H, Thummalapenta S, Xie T, Zhang L, Wang Q (2010) Mining api mapping for language migration. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, ACM, pp 195–204

Zhou G, He T, Zhao J, Hu P (2015) 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, pp 250–259