6-2014

SEWordSim: Software-Specific Word Similarity Database

Yuan TIAN
Singapore Management University, yuan.tian.2012@smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

Julia Lawall
INRIA, France

SEWordSim: Software-Specific Word Similarity Database

Yuan Tian[1], David Lo[1], and Julia Lawall[2]
[1]School of Information Systems, Singapore Management University, Singapore
[2]Inria/LIP6-Regal
{yuan.tian.2012,davidlo}@smu.edu.sg, julia.lawall@lip6.fr

ABSTRACT

Measuring the similarity of words is important in accurately representing and comparing documents, and thus improves the results of many natural language processing (NLP) tasks. The NLP community has proposed various measurements based on WordNet, a lexical database that contains relationships between many pairs of words. Recently, a number of techniques have been proposed to address software engineering issues such as code search and fault localization that require understanding natural language documents, and a measure of word similarity could improve their results. However, WordNet only contains information about words senses in general-purpose conversation, which often differ from word senses in a software-engineering context, and the software-specific word similarity resources that have been developed rely on data sources containing only a limited range of words and word uses.

In recent work, we have proposed a word similarity resource based on information collected automatically from StackOverflow. We have found that the results of this resource are given scores on a 3-point Likert scale that are over 50% higher than the results of a resource based on WordNet. In this demo paper, we review our data collection methodology and propose a Java API to make the resulting word similarity resource useful in practice.

The SEWordSim database and related information can be found at `http://goo.gl/BVEAs8`. Demo video is available at `http://goo.gl/dyNwyb`.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Measurement

Keywords

Word Similarity, Database, SEWordSim

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591071

1. INTRODUCTION

A fundamental problem in natural language processing (NLP) is how to measure the similarity of words [4]. Measuring word similarity is essential to the processing of more complex textual units, such as phrases, sentences, and complete documents. Thus, measuring the similarity of words has many applications in areas such as information retrieval or text categorization [2, 7]. A number of studies have proposed techniques to measure word similarity [1, 8, 14, 18]. Most of these techniques leverage WordNet [13], a well-known lexical database.

Relevance to Software Engineering. Recently, the software engineering community has proposed a number of tools that rely on textual software artifacts, leveraging information retrieval and document classification techniques [9, 15]. The success of the use of word similarity in the NLP community, suggests that the text-based tools of the software-engineering community could also benefit from this information. For example, word similarity information could be used to expand a query with additional similar words to improve the accuracy of an information retrieval based solutions (e.g., code search, bug localization, etc.), cf. [5]. However, most software-engineering tools that have been proposed do not use word similarity, and instead consider words that are not identical to be unrelated. Thus, there is a need for a word similarity resource that is usable by software-engineering tools. Constructing such a resource leveraging WordNet is one option. However, WordNet contains the relationships between words considering general-purpose texts, such as newspapers, that do not reflect the software-specific meanings of many words, such as Python and Eclipse. To address this issue, we have built a software-specific word similarity database.

Originality. Several previous works have attempted to construct a software-specific word similarity resource. The originality of our work lies in the kind of information sources considered. Specifically, we consider software information sites, such as StackOverflow,[1] forums, etc., in which developers and ordinary users discuss software related issues.[2] In previous work, Yang and Tan extract similar words from comments and identifiers in source code [20], while Howard et al. infer similar verbs from source code [6]. However,

[1]http://stackoverflow.com/
[2]The term software information site was first introduced in [19].

source code is not the only place to find words related to software development. Many of these words are also used in the mass of content posted in software information sites. Wang et al. [17] extract similar frequently used software tags, i.e., short text fragments used to bookmark contents, in the software information site FreeCode. However, tags represent only a small fraction of the words that appear in software information sites. In this work, we leverage the entire textual contents of the communications in a software information site, namely StackOverflow, to create a software specific word similarity database named SEWordSim, containing similarities of 5,636,534 pairs of words.

Complete details of our database construction process and the evaluation of the resulting database are available in our research paper [16]. The goal of this demo is to release our database and to promote its use by researchers, especially those that work on text analysis for software engineering.

The rest of this paper is organized as follows. Section 2 describes how the database is constructed and our preliminary evaluation. Section 3 presents our API for accessing our database. We discussion statistics and performance of the database and in Section 4. We describe related work in Section 5 and conclude in Section 6.

2. CONSTRUCTION APPROACH

We first describe how the similarity of words is computed. We then describe our database construction procedure.

2.1 Measuring Similarity

To measure the similarity of a pair of words, we use the concept of word co-occurrence. The co-occurrence of words w_1 and w_2 is the number of sliding windows of size n in a document corpus (i.e., a set of documents), where w_1 is at the center of the window, and w_2 appears in the window.

Based on these co-occurrences, we compute positive pointwise mutual information (PPMI) [10] between w_1 and w_2, denoted $PPMI(w_1, w_2)$, which measures the discrepancy between the *actual* and *expected* co-occurrence frequency of w_1 and w_2, assuming independence. If the software information sites from which we extract contents contain tags, as is the case of StackOverflow, we compute the following weighted $PPMI$ score to take these tags, which are often important terms, into account:

$$WPPMI(w_1, w_2) = W(j) \times PPMI(w_1, w_2) \quad (1)$$

$$\text{where } W(j) = \begin{cases} \alpha & \text{(if } j \text{ is a popular software tag)} \\ \beta & \text{(if } j \text{ is a nonpopular software tag)} \\ \gamma & \text{(otherwise)} \end{cases} \quad (2)$$

At this point, for each word w_i we have a vector of $PPMI$ scores (or $WPPMI$ scores, if tags are available) that characterize its co-occurrences with other words. To measure the similarity between w_i and another word w_j we take the cosine similarity of their corresponding vectors of $PPMI$ or $WPPMI$ scores.

2.2 Database Construction Procedure

To compute the similarity of pairs of words based on the similarity measure above, we perform following steps:

1. **Data Collection & Pre-Processing**: This step extracts content from software information sites. This content could be questions and answers in StackOverflow, posts in software forums, bug reports in bug tracking systems, and so on. We separate natural language from source code in this content. We first process the source code by removing programming language keywords and by splitting identifiers based on Camel casing and Pascal casing. The pre-processed source code and the natural language content are then subjected to standard text pre-processing techniques, i.e., tokenization, stop-word removal, and stemming. Some software information sites also allow users to attach short labels (i.e., tags) to content. We also extract these tags if they are available.

2. **Word Co-Occurrence Computation**: This step scans the document corpus and computes the co-occurrence of various pairs of words.

3. **Parameter Tuning**: This step optimizes the weights α, β, and γ that appear in Equation 2. Note that this step is only performed if tags exist. The parameter tuning step iteratively performs a greedy search based on a fitness function that it tries to optimize. The details of this step are available in our research paper [16].

4. **Similarity Computation**: This step computes similarity for various pairs of words based on the procedure described in the Section 2.1 and the weights learned in the parameter tuning step, if applicable. The result of this step is our database.

2.3 Preliminary Validation Study

To test the effectiveness of our database construction approach, we have used it to infer a word similarity database from 10,000 questions and answers posted in StackOverflow in January 2011. We then performed a user-assisted study to compare the accuracy of our database with a publicly available word similarity database constructed using WordNet [12]. Details about this study are found in our research paper [16]. Users gave the results produced using our database an average Likert score that is 50.2% higher than the results produced using WordNet-based database, resulting in an average discounted cumulative gain (DCG) that is 65.2% higher. Thus we conclude that our database produces results of substantially better quality.

3. METHODOLOGY FOR USERS

To make our results easily usable, we store word similarities in an SQLite database.[3] We have chosen SQLite because it implements a self-contained and server-less SQL database engine. The current version of our database contains similarities of 5,636,534 word pairs. We also release a dump of the database, which can easily be imported to other kinds of databases. Our database contains one table, named "Word_Similarity". To improve the efficiency of search in database, we create indexes on words and similarity values. Table 1 shows the structure of the table "Word_Similarity".

We also provide a simple Java API to allow users to extract similarities easily from an SQLite database. Our API consists of one class named `WordSimDBFacade`. The class diagram of `WordSimDBFacade` is shown in Table 2.

[3] `http://www.sqlite.org/`

Table 1: Structure of the table "Word_Similarity"

Field Name	Data Type(maximum length)	Index
term_1	Varchar(30)	Yes(Ascending)
term_2	Varchar(30)	Yes(Ascending
similarity	Double(20)	Yes(Descending)

Table 2: `WordSimFacade`'s class diagram

WordSimDBFacade
+ WordSimDBFacade(dbFile:String)
+ stemWord(word:String):String
+ isInDatabase(word:String):Boolean
+ computeSimilarity(word1:String,word2:String):double
+ findMostSimilarWord(word:String):String
+ findMostSimilarWords(word:String,minSim:double):List⟨String⟩
+ findTopNWords(word:String,n:int):List⟨String⟩
+ getAllWords():List⟨String⟩

The first method is a constructor that accepts one argument, which is the location of the SQLite database. The second method, `stemWord`, is a stemmer implementing the Porter Stemming algorithm. It takes a word as input and returns the root form of the word. This method should be called before calling the other methods, because our database only contains stemmed words. The third method, `isInDatabase`, takes a word as an argument and checks whether it exists in the database. This method allows a user to check if our database can be used for a word that the user is interested in. The fourth method `computeSimilarity` takes as input two words and returns the similarity between them. This similarity ranges from 0 to 1 if input words exist in our database, where 0 indicates that the words are unrelated and 1 indicates that the words are the same. If either word does not appear in the database, `computeSimilarity` returns -1.

The fifth method, `findMostSimilarWord`, returns the most similar word to an input word. The sixth method `findMostSimilarMethods`, returns all words whose similarity with the input word (the first argument) is greater than or equal to a minimum similarity threshold `minSim` (the second argument). The seventh method, `findTopNWords`, takes an input word (the first argument) and a number `n` (the second argument) and returns the top-n most similar words to the input word. These three methods return null if the input word does not exist in our database.

The last method, `getAllWords`, returns all words that appear in the database. With this method, a user can enumerate all the words and perform additional processing, e.g., compute clusters of similar words, etc., by repeatedly invoking the other methods in the class, e.g., `computeSimilarity`.

To help users understand our API, we document the API using Javadoc. The compressed documentation is available with the database. Figure 1 shows a piece of code that calls all the methods in our API.

```
public static void main(String args[]) {
 //read in database
 String inputFile="/folder/WordSim.db";
 WordSimDBFacade facade=new WordSimDBFacade(inputFile);

 //stem word
 String inputWord="folder";
 String stemmedInputWord=facade.stemWord(inputWord);
 String compareWord="directory";
 String stemmedCompareWord=facade.stemWord(compareWord);

 //set parameters
 double minSimilarityScore=0.3;
 int N=10;

 //invoke API methods
 System.out.print(
 facade.isInDatabase(stemmedInputWord)+"\n");
 System.out.print(
 facade.computeSimilarity(stemmedInputWord,
     stemmedComparedWord)+"\n");
 System.out.print(
 facade.findMostSimilarWord(stemmedInputWord)+"\n");
 System.out.print(
 facade.findMostSimilarWords(stemmedInputWord,
     minSimilarityScore)+"\n");
 System.out.print(
 facade.findTopNWords(stemmedInputWord,N)+"\n");
 System.out.print(
 facade.getAllWords());
 }
```

Figure 1: Sample code for API usage

4. DISCUSSION

4.1 Range of Similarity Scores

Some readers might be interested in the similarity score distribution of word pairs in our database. If all word pairs have very low similarity scores, then our database would not be useful. Figure 2 is a box plot showing the distribution of the maximum similarity score. To draw the box plot, for each word w in our database, we find the word pair (w,w') that has the maximum similarity score. The scores of these pairs are shown as the box plot. The box plot shows that 50% of the words in our database, which is represented by the box, have maximum similarity scores between 0.31 and 0.44. We also find that more than 13.5% of the words have maximum similarity scores greater than 0.5. Some of them have maximum similarity scores greater than 0.8.

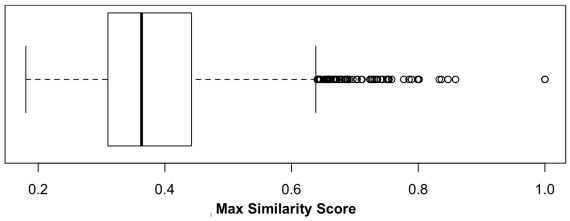

Figure 2: Distribution of maximum similarity score

4.2 Database Performance

To enable our database to be used in an interactive tool, the time it takes to process a query should be short enough. The March 2014 version of our SQLite database stores more than 5 million word pairs. To test the efficiency of this database, we randomly selected 1,000 words from a number of StackOverflow questions. Based on these words, we randomly generated 100 word pairs and queried our database for their similarity scores. We perform this experiment on an Intel Core i5 2.5GHz PC with 8GB of RAM running OS X 10.9.2. We use the latest SQLite version (i.e, Version 3.8.4.1) to process queries. Figure 3 shows the distribution of query time required to process each of the 100 word pairs. We observe that more than 75% of the queries could be completed in less than 0.02 seconds. The median query time is 0.012 seconds.

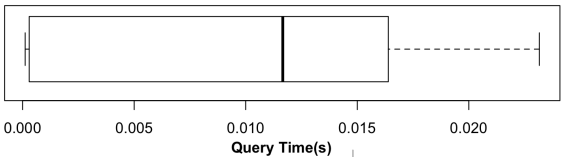

Figure 3: Distribution of query time

5. RELATED WORK

Generic Word Similarity. Many studies in the NLP propose techniques to measure the similarity of words, typically using WordNet [8, 14, 18]. Pedersen et al. have developed a tool based on WordNet to allow end users to compute similarities of words [11]. They have also pre-computed the similarities of many words and created a publicly accessible database containing these similarities [12]. Different from the above studies, we do not leverage WordNet; instead, we automatically construct a word similarity database from a software-specific corpus.

There are also a number of approaches that construct a thesaurus from a document corpus [3, 8]. These approaches also analyze the co-occurrence of words. Different from these studies, we leverage a software-specific corpus. We also leverage the phenomenon of social tagging where users assign tags to content in software information sites. These tags are often important software-specific terms. The automatic tuning step of our database construction approach assigns different weights for words that appear as popular tags, unpopular tags, and other words.

Software Specific Word Similarity. Yang and Tan [20] analyze comments and method signatures in software source code files by a sequence alignment and clustering based approach to infer semantically related words. Similarly, Howard et al. mine semantically similar verbs from comments and method signatures [6]. We propose a different approach that is tailored for content obtained from software information sites. Many software-related words are not in source code but in the mass of content posted by developers and users. Thus, our work complements their work.

Wang et al. analyze tags in FreeCode and infer relationships among these tags [17]. Our work generalizes Wang et al.'s work by considering not only tags but also the entire textual contents posted in StackOverflow. There are far fewer tags than words in the content posted to StackOverflow or other software information sites. Indeed, Wang et al. only measure similarities of 690 tags.

6. CONCLUSION AND FUTURE WORK

In this demo, we present our word similarity database for the software engineering community: SEWordSim. This database contains similarity information for 5,636,534 pairs of words. Different from a general-purpose word similarity database, such as one based on WordNet, our database is trained from software-specific documents. Thus, word similarity is considered in a software-specific context. Our approach takes as input software-specific documents, which are abundant and can be easily downloaded from various software information sites, e.g., StackOverflow, SourceForge, FreeCode, etc. Our preliminary evaluation shows that considering the software context allows SEWordSim to be more accurate than WordNet-based approaches on words related to software development. We have created a simple API to make it easy for researchers to use our database. Functionalities of our API can be composed to realize more complex functionalities, e.g., cluster similar words, etc.

In the future, we plan to incrementally release updated versions of our database periodically by extracting word similarities from an expanded software-specific document corpus.

7. REFERENCES

[1] D. Bollegala, Y. Matsuo, and M. Ishizuka. Measuring semantic similarity between words using web search engines. In *WWW*, 2007.

[2] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, 2008.

[3] L. Chen, P. Fankhauser, U. Thiel, and T. Kamps. Statistical relationship determination in automatic thesaurus construction. In *CIKM*, 2005.

[4] L. Dai, B. Liu, Y. Xia, and S. Wu. Measuring semantic similarity between words using HowNet. In *ICCSIT*, 2008.

[5] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *ICSE*, 2013.

[6] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. Automatically mining software-based, semantically-similar words from comment-code mappings. In *MSR*, 2013.

[7] S. F. Hussain and G. Bisson. Text categorization using word similarities based on higher order co-occurrences. In *SDM*, 2010.

[8] D. Lin. Automatic retrieval and clustering of similar words. In *COLING*, 1998.

[9] E. Linstead, S. K. Bajracharya, T. C. Ngo, P. Rigor, C. V. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Min. Knowl. Discov.*, 18(2), 2009.

[10] Y. Niwa and Y. Nitta. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *COLING*, 1994.

[11] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet:similarity-measuring the relatedness of concepts. In *AAAI*, 2004.

[12] T. Pederson. Wordnet::similarity. http://wn-similarity.sourceforge.net/.

[13] Princeton University. WordNet: A lexical database for English. http://wordnet.princeton.edu/.

[14] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

[15] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE*, 2007.

[16] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *CSMR-WCRE*, 2014.

[17] S. Wang, D. Lo, and L. Jiang. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In *ICSM*, 2012.

[18] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *ACL*, 1994.

[19] X. Xia, D. Lo, X. Wang, and B. Zhou. Tag recommendation in software information sites. In *MSR*, 2013.

[20] J. Yang and L. Tan. SWordNet: Inferring semantically related words from software context. *Empirical Software Engineering*, 2013.