Code Similarity via Natural Language Descriptions

Eran Yahav Meital Ben Sinai
Technion - Israel Institute of Technology
yahave,mbs@cs.technion.ac.il

Abstract

Code similarity is a central challenge in many programming related applications, such as code search, automatic translation, and plagiarism detection. In this work, we reduce the problem of semantic relatedness between code fragments into a problem of semantic relatedness of textual descriptions. Our main idea is that we can use the relationship between code and its textual descriptions as established in question-answering sites such as STACKOVERFLOW. Consequently, we can determine semantic relatedness and similarity, of code fragments across different programming languages, a task considered extremely difficult using traditional approaches. We have implemented our approach, and used crowed-sourced labeling of similarity to evaluate it over 1500 pairs of code fragments. Results show that we gain around 80% precision and 75% recall, and demonstrate the promise of this approach.

1. Introduction

Consider the two code fragments of Fig. 1. The code fragment in Fig. 1(a) is written in Java and the code fragment in Fig. 1(b) is written in PHP. Despite their considerable syntactic difference, and the fact that they are written in two completely different programming languages, we would like to establish the two as similar, as they have the same functionality. The challenge is to establish such semantic similarity (more generally - semantic relatedness) automatically. The main idea of this paper is to consider the problem of semantic relatedness between code fragments, by considering the semantic relatedness of their corresponding textual descriptions. Following this idea requires: (i) establishing a relationship between a code fragment and its textual description(s). (ii) measuring semantic similarity between textual descriptions related to code. To address the first challenge, we can rely on relationships between code and its description as created in common question-answering sites, such as STACKOVERFLOW (SOF), on documentation, blog posts, etc. Fig. 2 provides an overview of our approach. The approach works by first extracting information from sources that correlate code to its description (as mentioned above) and then constructing a database of these relationships. After this database is constructed, we can link a given code fragment to its related textual description. Consequently, instead of only using the code fragments themselves, we also use their descriptions and measure the descriptions' semantic similarity, while discovering latent information.

```
Date d1 = new Date();
Date d2 = new Date();
d2.setTime(d1.getTime() + 1*24*60*60*1000);
```

(a)

```
define(DATETIME_FORMAT, 'y-m-d H:i');
$time = date(DATETIME_FORMAT,
    strtotime("+1 day", $time));
```

(b)

Figure 1: Semantically related fragments for addition of one day to a given date. (a) is written in Java and (b) in PHP.

2. Similarity between Code Fragments

Given two code fragments p_1 and p_2, we would like to define a *quantitative* similarity metric $sim_{Code}(p_1, p_2) \rightarrow [0, 1]$ between p_1 and p_2, such that it measures the *semantic similarity* of the code fragments. That is, $sim_{Code}(p_1, p_2)$ should measure similarity of the functionality of p_1 and p_2, regardless of their syntactic differences. Further, we would like this measure to work across code using different libraries, and even code written in different programming languages. While there has been a lot of classic work on notions of *semantic equivalence* and *semantic differencing*, measuring similarity across different libraries or languages is considered to be extremely difficult.

The main idea of this work is do define $sim_{Code}(p_1, p_2)$ through *semantic similarity of textual documents describing each code fragment*.

3. Semantic Similarity of Descriptions

The question of semantic similarity between textual documents is a central question in many NLP tasks, and is studied heavily in the NLP community.

We assume the existence of a *description oracle*, which takes a code fragment and returns a set of natural-language documents describing the functionality of the code fragment. We explain how to construct such description oracle in Section 4. We refer to the set of natural-language documents related to a code fragment as its *semantic description*.

We measure the distance between two *semantic descriptions* using standard text similarity methods. First, We take the *semantic description* and process it by stemming (changing each word to its base form), removing stop words and punctuation signs. Then, we use Latent Semantic Analysis

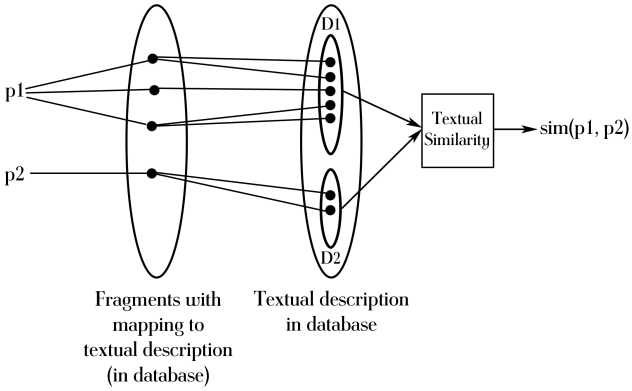

Figure 2: An overview of our approach

(LSA) on top of Term Frequency - Inverse Document Frequency (tf.idf), to build a vector out of the processed text which is compared to other vectors using Cosine Similarity.

4. Building a Description Oracle

We chose to construct our *description oracle* based on SOF, due to its volume and coverage. SOF includes many *"how to..?"* questions that are answered with a code fragment. Our investigation shows that in most cases the question can play a major role in the *semantic description* of a piece of code, and use SOF to establish a mapping between code fragments and their textual descriptions. To build the *semantic description* of a code fragment, we use the text all answers and questions that contain the code fragment. Given a code fragment and a corresponding SOF question, we use this reverse index to map a code back to the text of all of the SOF questions and answers in which it appears.

From Code Fragment to Descriptions Given two arbitrary code fragments p_1 and p_2, and a description oracle containing a massive number of fragments and their description, we would like to be able to compute the semantic similarity $sim_{code}(p_1, p_2)$ even when p_1 and p_2 are not present in the description oracle. Towards that end, we use standard techniques for *syntactic similarity* of code fragments. Specifically, we can find similar fragments in the oracle's database even when they differ in variable names, parameter values.

5. Code Fragments Type Analysis

While keeping in mind the problem of similarity across languages, we wanted to utilize the code itself to support the indirect text-based similarity function. Towards that end, we use the type signatures of code fragments as another signal in measuring their similarity. We implemented our analysis for Java programming language and Python. Using this information, compare code fragments written in different programming languages based on their type signatures.

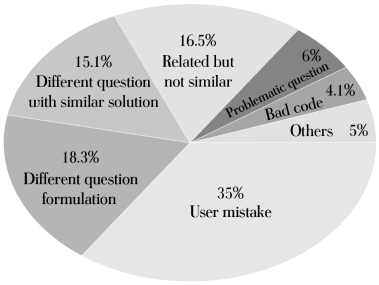

Figure 3: Analysis of more than 200 incorrect classifications

6. Preliminary Results

Prototype Implementation We have implemented our approach, and evaluated it. One of the main challenges is obtaining labeled data, to which we can compare our results. Towards that end, we have implemented a system that allows us to crowd-source this task. The system, LTDOW, is available online here: `http://ltdow.eu01.aws.af.cm/`.

Similarity Classifier We compared our results with the results obtained from LTDOW and computed the precision, recall and accuracy of similar code fragments. In our case, we denote precision as the fraction of code fragment pairs which are labeled as similar, that are indeed similar, and recall is denoted as the fraction of similar code fragment pairs that are actually labeled as similar. The experimental database contains more than 1500 pairs of code fragments. The preliminary results show that more than 85% of our labels are consistent with the users' labels, while the precision is higher than 80% and the recall is above 75%. To improve the results, we performed a deep examination of the mismatches between our system and the manually labeled information. We found out that almost 20% of the wrong classification are originating from differently formulated descriptions (shown in Fig. 3).

Link Code to its Description We checked two sets of code fragments from multiple programming languages: (i) randomly picked 25 fragments from our database (control group), and (ii) randomly picked 25 fragments from our database after applying mutations. We used each of the fragments as an input query and evaluated our ability to retrieve the correct fragment. Fragments from the first group correctly returned as the first match in all cases. Fragments from the second group retrieved as the first result in $16/25$ of the cases, second in $8/25$ and not at all only once.

7. Conclusion and Future Work

We presented a novel approach for measuring semantic similarity between code fragments based on their corresponding textual descriptions. We combined text similarity techniques with lightweight analysis of types and showed that it leads to promising results. As future work, we plan to enrich the analysis of the code itself and use ESA as another textual similarity measure.