Code similarity: A comparison of methods

Mahmoud Zeydabadinezhad
Department of Computer Science
Emory University
Atlanta, USA
mzeydab@emory.edu

***Abstract*—Exploiting the similarity between codes has multiple applications from plagiarism detection in schools to code patching in software engineering. In this work, I compared some of the metrics and a machine learning based method previously proposed in the literature for measuring the code similarity**

Keywords—code similarity,

I. Introduction

Although copying and pasting is not encouraged in school coding assignments and could cause serious consequences, it is a common practice in software engineering. Aside from plagiarism detection, finding the similar pieces of codes is crucial during debugging and software patching. It also has application in ontology alignment (1). This is not always a trivial task as the codes may be modified for various purposes. String metrics-based, token sequence pattern-matching, abstract syntax tree (AST) and machine learning (knowledge) based methods are some of the approaches proposed for code similarity detection (2).

II. Materials and Methods

A. Data

I selected 3 python codes that were modified by three students in BMI500 course to denitrify the patients' age from a text file.

B. String metric based methods

The basic idea when using the metric-based methods is to compare the codes by treating them as strings and finding the similarity between them. Codes can be compared word by word or can be partitioned into chunks or corpus. A string metric is a metric that measures similarity or dissimilarity (distance) between two text strings for approximate string matching or comparison Words can be similar in two ways, lexically and semantically. Words are similar lexically if they have a similar character sequence (3). In this work I used Levenshtein (4), Jaro (5), and real minimal edit distance as my method of choice for character-based similarity measure. Levenshtein defines distance between two strings by counting the minimum number of operations needed to transform one string into the other, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters (4). Jaro is based on the number and order of the common characters between two strings; it takes into account typical spelling deviations and mainly used in the area of record linkage (5).

C. Sentence embedding

I used the code template publicly available (6) to implement Universal Sentence Encoder (USE) (7) as a tool to compute sentence embeddings and inferring sentence level meaning similarity. USE is a knowledge-based similarity method that measures the semantic similarity. It is demonstrated that sentence level embeddings surpass the performance of transfer learning using word level embeddings alone (7).

III. Results

Figure 1. shows the heatmap plot of Levenshtein measure applied to the python codes.

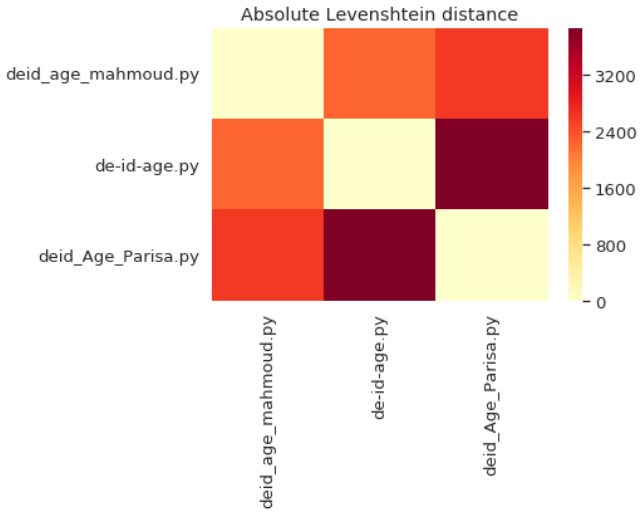

Fig 1. Levenshtein measure for 3 python codes.

Figure 2. shows the heatmap plot of Jaro string similarity measure applied to the python codes.

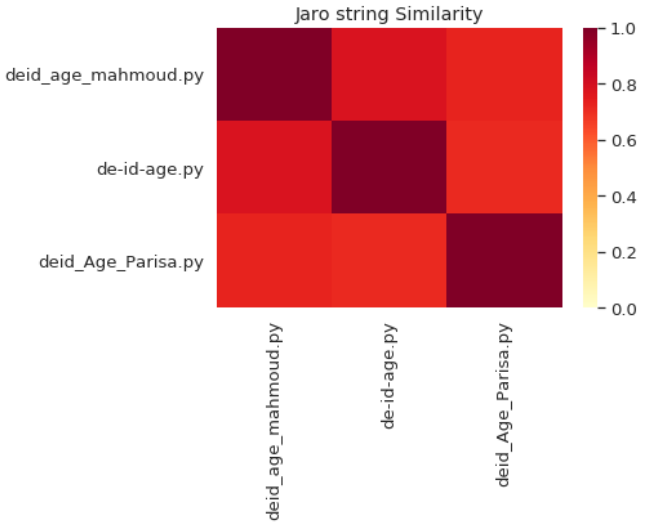

Fig 2. Jaro measure for 3 python code.

Figure 3. shows the heatmap plot of ratio similarity measure applied to the python codes.

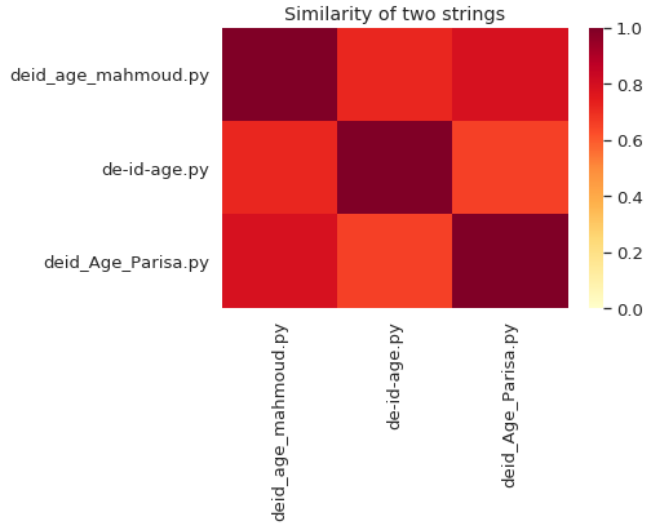

Fig 3. Ratio of similarity between 3 python codes.

Figure 4. shows the heatmap plot of USE applied to the python codes.

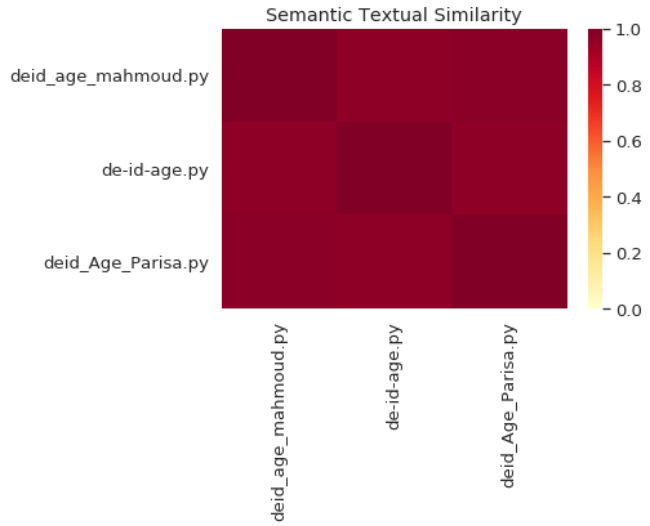

Fig 4. Semantic textual similarity for 3 python codes.

IV. CONCLUSION

I compared three string metric-based methods to investigate the similarity between 3 python codes to perform word matching in a lexical manner. I also used Universal Sentence Encoding as a knowledge-based method to calculate the code similarities in a word semantic manner. The absolute Levenshtien distance is 0 for two identical string and increases as the difference between them increases. Based on this measure "de-id-age.py" and "deid_Age_Parisa.py" are the pair with highest difference. The Jaro string similarity is 1 for two identical string and approaches 0 as the difference increases. Based on this measure "de-id-age.py" and "deid_Age_Parisa.py" are the pair with highest difference. The ratio similarity is 1 for two identical string and approaches 0 as the difference increases. Based on this measure "de-id-age.py" and "deid_Age_Parisa.py" are the pair with highest difference. The semantic textual similarity computed by USE is one for two identical sentence and approaches to 0 as the sentences differ in meaning. Based on this measure, all the 3 codes are very similar to each other. In fact, the heatmap associated with this measure has the least contrast compared to the other string-based methods. Having the prior knowledge that each student wrote her/his own code by modifying another master code, it is reasonable to expect that the 3 codes to be very similar to each other and in this way the USE results seem to be the most reasonable output.

In this work our goal was to compare the codes. To this end, comparing the codes with string-based methods is not ideal as any change in variable names and syntax configuration or even converting loops to a vectorized piece of codes counts as a difference while the fact is that they follow the same logic. The USE method may suffer to a less degree from the above-mentioned points but as its main purpose is to find the semantic similarity between sentences it might not be ideal either. I believe a framework that coverts the codes to a somehow standard space and then comparing them would be more beneficial. Abstract syntax trees (AST) could be a possible way to build this framework.

REFERENCES

[1] Cheatham, Michelle, and Pascal Hitzler. "String similarity metrics for ontology alignment." *International Semantic Web Conference*. Springer, Berlin, Heidelberg, 2013.

[2] Chilowicz, Michel, Etienne Duris, and Gilles Roussel. "Syntax tree fingerprinting for source code similarity detection." *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 2009.

[3] Gomaa, Wael H., and Aly A. Fahmy. "A survey of text similarity approaches." *International Journal of Computer Applications* 68.13 (2013): 13-18.

[4] Hall, Patrick AV, and Geoff R. Dowling. "Approximate string matching." *ACM computing surveys (CSUR)* 12.4 (1980): 381-402..

[5] Jaro, Matthew A. "Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida." *Journal of the American Statistical Association* 84.406 (1989): 414-420.

[6] https://research.google.com/seedbank/seed/5664902681198592

[7] Cer, Daniel, et al. "Universal sentence encoder." *arXiv preprint arXiv:1803.11175* (2018).