Samuel Coburn, Jessica Kim
COMS 6156
Final Project Proposal

The goal of our final project is to examine the efficacy of code embeddings [1] in identifying the similarity of code snippets in comparison to existing tools with the same functionality.

The project relates to the midterm papers authored by Coburn and Kim in which the topic of code completion technologies were discussed. In code completion software, it is vital for the technology to recommend adequate and timely recommendations to the user based on the current context of their “in-progress” program. To do so effectively, code completion technologies cluster methods of similar functionality in order to provide accurate recommendations with respect to the code snippet being written by the developer [2]. The problem of clustering methods is viewed as determining the similarity between code snippets. Over the years, many tools have been developed with the goal of determining the similarity between code snippets using a variety of factors, specifically “metrics-based, text-based, token-based, tree-based, and graph-based” tools [3]. A thorough experiment was conducted in 2018 in order to compare the performance of code similarity analyzers which utilize the aforementioned factors in order to make their similarity judgements; the experiment, described in detail in “A comparison of code similarity analysers”, examines the performance of 30 code similarity analyzers in total [3]. To adequately gauge the capabilities of the observed code similarity analyzers, the authors subjected each technology to “five experimental scenarios”, all of which touched on a separate quality of their definition of code similarity [3]. While the project began with the “jumping-off point” of code completion technologies, code similarity analyzers can be utilized to solve a variety of scenarios, including determining code plagiarism in a classroom setting or software copyright infringement [3].

One year after “A comparison of code similarity analysers” was published (2019), the paper “code2vec: Learning Distributed Representations of Code” was also published. The paper introduces the concept of code embeddings, a code representation that builds on the foundational work of word2vec [4]. Code embeddings are described in the paper as a method for “representing snippets of code as continuous distributed vectors, which can be used to predict semantic properties of the snippet” [1]. The authors of the article note that a potential application of code2vec’s code embeddings could be used in order to determine the semantic similarity between code snippets, something they remark would be “difficult to find without [their] approach” [1]. While this statement expresses the idea that determining the semantic similarity between code snippets is difficult without the use of code embeddings, other methods to achieve this task exist, namely the aforementioned tree-based and graph-based similarity analyzers are capable of capturing the semantic relationships between code fragments (albeit with drawbacks, like O(N^3) runtime) [3]. The goal of our final project is to empirically test the validity of this remark with respect to the experimental process laid out in the paper “A comparison of code similarity analysers”, and to compare the results of our experiment with the results obtained in the original experiment.

Starting our experimental process, we will begin by converting the provided Java code from the code analyzer paper into AST representations, which can be viewed and interpreted by the code embedding model. This can be done by using the preprocessing program provided by the authors of the code2vec article, which takes in Java files and processes them in order to be correctly used in the code2vec model. Following this, we will train code embeddings using the code embedding model provided on GitHub by the authors of code2vec [5]. The readme file located in the authors' GitHub repository provides step-by-step guidance in order to accurately perform these aforementioned steps in our experiment. Using our code embeddings, we can determine the similarity between two code snippets by performing the cosine similarity between their respective code embeddings. In line with the experimental process laid out in the code similarity analyzers paper, these comparisons will be performed for all groupings of the 100 observed code snippets, making for 10,000 comparisons in total. Using the results of our experiment, we will be able to compare our analysis of the efficacy of code embeddings in accomplishing the code similarity task to the results of the original experiment, with special reference to code similarity analyzers capable of capturing semantic similarity (like tree-based and graph-based tools).

In order to conduct our experiment, we will utilize the open-source data made available by the authors of both “A comparison of code similarity analysers” and “code2vec”. The former article provides the sources of the code snippets utilized for their experiments, namely 100 classes written in the Java programming language [6, 7, 8]. Additionally, they also provide a detailed description of their experimental process, which will be followed in order to gauge the capabilities of code embeddings in the same scenarios. The latter article provides the code necessary (specifically the GitHub repository of the code2vec research project) to train our own code embeddings necessary for the experiment [5]. Because we are following the experimental process laid out in the code similarity analyzer article, we need only train code embeddings for the 100 Java classes used in the original research article. As such, we are confident that we are in possession of the necessary hardware and software in order to perform this experiment within the timeframe allotted to us.

Citations:
[1] Alon, U., Zilberstein, M., Levy, O. and Yahav, E., 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, *3*(POPL), pp.1-29. Available at: https://arxiv.org/abs/1803.09473 (Accessed: 24 March 2022).
[2] Wen, F., Aghajani, E., Nagy, C., Lanza, M., and Bavota, G., 2021. Siri, Write the Next Method, *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 138-149,Available at: https://ieeexplore-ieee-org.ezproxy.cul.columbia.edu/abstract/document/9402018/references#references (Accessed: 24 March 2022).
[3] Ragkhitwetsagul, C., Krinke, J., and Clark, D., 2018. A comparison of code similarity analysers. *Empirical Software Engineering*, pp.2464-2519. Available at: https://link.springer.com/article/10.1007/s10664-017-9564-7 (Accessed: 24 March 2022).

[4] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. Available at: https://arxiv.org/abs/1301.3781 (Accessed: 24 March 2022).
[5] Alon, U., Zilberstein, M., Levy, O. and Yahav, E. *tech-srl/code2vec*, viewed 24 March 2022, <https://github.com/tech-srl/code2vec>.
[6] Marino, D., *COP 3503 Lecture Notes*, viewed 24 March 2022, <http://www.cs.ucf.edu/~dmarino/ucf/cop3503/lectures/>.
[7] *Java - Basics, Advanced Programming With Free Source Code Downloads*, viewed 24 March 2022, <http://www.softwareandfinance.com/Java/>.
[8] Ragkhitwetsagul, C., Krinke, J., and Clark, D., *OCD Framework: A Comparison of Code Similarity Analysers,*
viewed 24 March 2022, <https://ucl-crest.github.io/OCD/>.