Samuel Coburn, Jessica Kim
Team Name: The Code Completers
COMS 6156
Final Project Progress Report

As an overview, the goal of our final project is to examine the efficacy of code embeddings in identifying the similarity of code snippets in comparison to existing tools with the same functionality. We will be extending an experiment performed in “A comparison of code similarity analyzers” by including different code embeddings from the ones used in the experiment. We will be creating the code embeddings using a method provided by "code2vec: Learning Distributed Representations of Code,” as the authors have made it easy to generate our own from their pre-trained models. The former paper provides 100 classes written in Java, which we will use to generate the code embeddings.

There are three datasets we are using to conduct our experiments on code embeddings. The first is a dataset of 100 Java files which were created by performing various pervasive modifications on 10 source files. The two other datasets normalize the files from the first dataset by passing them through a decompiler. The two datasets correspond to the two different decompilers used to generate the code, specifically Krakatau and Procyon. In total, there are 300 Java source files that we will use to create code embeddings. We will create code embeddings using two different pre-trained models provided on the code2vec GitHub page. The first model is the default trained model provided on their GitHub page, which was utilized in the code2vec paper. The second model was trained on a much larger dataset and is also available on the code2vec GitHub page. We will create embeddings for each pre-trained model separately, and we will perform the required analysis on them separately, in line with the analysis of code similarity analyzers paper. In total, we will have created 600 code embeddings for this project.

The experiment performed in “A comparison of code similarity analyzers” lists four scenarios that cover different aspects and characteristics for code similarity. Scenario 1 describes tool performance against pervasive modifications. Scenario 2 is about the analysis of tool performance against an available data set which contains fragments of boiler-plate code that are reused with or without modifications. Scenario 3 refers to the effects of normalization through the use of decompilers. Scenario 4 uses alternative metrics, like precision-at-n, rather than the F1 scores of the code similarity tools in order to measure performance. Lastly, Scenario 5 determines the performance of the code similarity tools in the presence of both pervasive modifications and boiler-plate code. We have access to the dataset used for both Scenario 1 and Scenario 3. Additionally, we will also be able to perform the alternative analysis as described in Scenario 4. We will not be able to perform Scenario 2 and Scenario 5 for two major reasons. Firstly, the dataset used for these scenarios, the SOCO dataset, is encrypted on GitHub without a decryption key. Because of this, we do not have access to the data in the dataset. Additionally, the authors of the code similarity analysis paper mention modifications on the code within the SOCO dataset they made before completing their analysis. If we did have access to the SOCO dataset, we would not be able to accurately replicate the modifications made to the source code in the dataset. As such, we have decided to not perform Scenarios 2 and 5 of the code similarity analysis paper. Because of this change, our project, more specifically, examines the capabilities of code embeddings in determining the similarity between source code files in the presence of pervasive modifications.

We will divide the work as follows: Sam will download the preprocessed dataset and trained model called java14_model, and manually generate code embeddings for all 300 classes one-by-one. Jessica will download the preprocessed dataset and trained model called java-large-released-model, and manually generate code embeddings for the same 300 classes one-by-one. We will then perform the similarity analysis and write the final report together.

So far, we have both downloaded our respective preprocessed datasets and trained models, as well as the Java datasets with 300 classes. Sam has generated all 300 code embeddings with the java14_model, and completed Scenarios 1 and 3 for this model. Jessica has been getting errors when running java-large-released-model due to the compatibility of her device (MacBook Pro with M1 chip) and the model. However, once this compatibility issue is resolved, she will be able to generate the code embeddings seamlessly.

The user community for which our project might prospectively benefit may include researchers studying code similarity tools. Our project may provide extra data and analysis on the efficacy of the tool developed by the authors of "A comparison of code similarity analyzers". Furthermore, as the authors of code2vec state 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". 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). We hope that our final project can provide value by empirically testing the validity of this remark with respect to the experimental process laid out in the paper "A comparison of code similarity analysers", and by comparing the results of our experiment with the results obtained in the original experiment.

A few research questions that our project will answer include:

- RQ1: How accurate are code embeddings at determining the similarity between code source files with pervasive modifications?
- RQ2: How does the performance of code embeddings as a similarity metric compare to the code similarity analyzers observed in the original experiment?
- RQ3: How does the training of the code embedding model impact its performance as a code similarity metric?

For our class demo, we are thinking about two options. We could create a toy example in which we demonstrate how the code embeddings were created and then show the computation for the code similarity. We could also present a small slideshow which would allow us to go into some detail about the experimental process. We are likely going to pursue the former option rather than the latter given it is more conducive to the demonstration than the presentation.

We have put our code on GitHub and have added both instructors to the repository. The repository is public, so it should be easily accessible. Here is the link:
https://github.com/Sam-Coburn/COMS6156_FinalProject