TSE Final Report: Obfuscated Code2Seq

Team Deep Understanding (Shirish Singh, Aria Jiang, Yiqiao Liu)

April 2021

1 Synopsis

The project aims to build a code understanding tool that helps find code clones across multi-file projects. A motivating scenario for such a tool is that if a developer wants to understand a piece of code that s/he has encountered for the first time, it would be helpful to find similar and familiar code that the developer might have developed before. The tool can find behavioral similarity between the new code and code the developer is familiar with. More broadly, the applications of behavioral similarity are manifold: a) Program understanding for first-time developers, b) code search, c) code refactoring, and so on.

In terms of novelty of this project and why it is an interesting one, we think the basis lies in our attempt to perform an improvement method proven on a different model (code2vec) to a newer, state-of-the-art model (code2seq). There is a project researching whether obfuscation of variable names may influence the performance of code2vec. That project achieved a positive answer. Therefore, in our project we aim to find out whether obfuscation can improve code2seq, a state-of-the-art model. And if so, how it actually improves code2seq.

In terms of value to user community, investigating potential ways to improve the performance of code2seq may help researchers/developers understand obfuscated programs better. In addition, program understanding can improve developer's performance by recommending method names and similar methods.

2 Research Questions

We attempted to answer two research questions. Below are these two questions and our answers to them.

- **RQ1**: Does obfuscation of variable names yield an improved code2seq [2] model?

  Our results show that the obfuscation of variable names makes code2seq output irrelevant sequences of words instead of improving it. Because of that, obfuscated data trained code2seq got worse performance in the code similarity task. Code2seq uses AST representation [3] of the program to predict method names. The paths extracted from the AST representation for training the model consist of variable names used in the function. The model then learns to predict the method names using the paths. In the case of randomly obfuscated variable names, the model does not encounter repetitions of the same variable name in the training data, which forces the model to rely on the AST structural representation for predicting the method name. This observation is coherent with the prior research [4] on the code2vec model.

- **RQ2**: How can we utilize code2seq model's output for code similarity?

Code2seq outputs a sequence of words representing the code inputted. In our project, we use word2vec to represent those sequences as vectors, and then use similarity metrics (KDTree) to find the most similar vector for each vector. At last, we map these pairs of most similar vectors back to the original code (by remembering their positions) and are able to get the most similar lines of code for each set of code. In this way, we can utilize code2seq's output to find a similar set of code for some code we have. We can also use this method to check if two sets of code are similar or not.

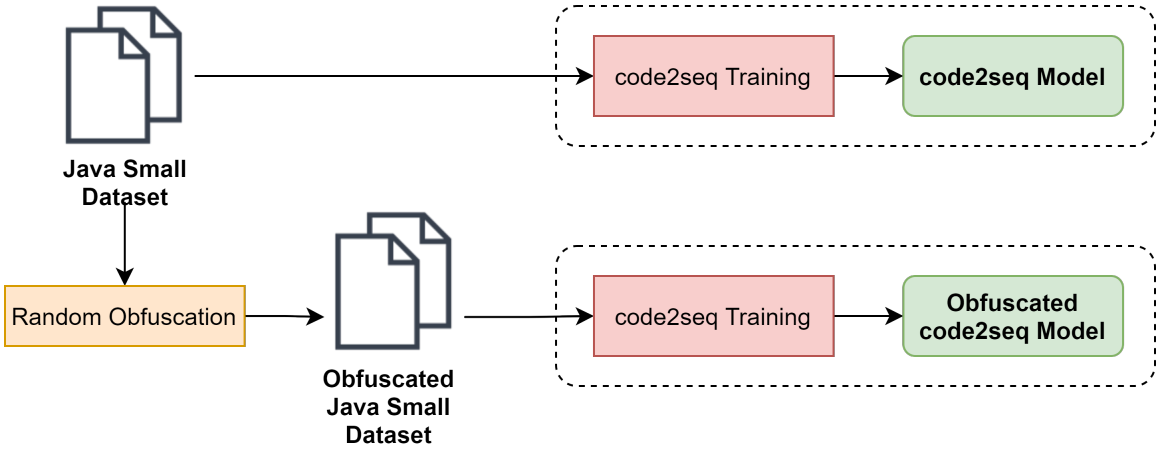

Figure 1: High level overview of generating code2seq models

2.1 Dataset

We used two datasets for this project. One is provided by the code2seq paper and the other is student submissions from a CS course. We do not have benchmarks but have a baseline (the original code2seq model).

2.1.1 Training Dataset

For training the model, we used Java-small dataset [1] consisting of about 700K Java files from 11 projects. We used the the obfuscator [4] provided to generate a new random obfuscated Java-small dataset to train the obfuscated code2seq model. Several Java files were not obfuscated because of technical limitations of the tool.

2.1.2 Test Dataset

Our dataset consists of student submissions for the first assignment of Columbia's 4156 ASE course offered in fall 2020. Students developed a two-player tic-tac-toe game. The user interface and the skeleton back-end were provided. Students had to complete the back-end of the game, where they wrote the game logic following software engineering best practices. All programs were written in Java programming language. In total we had 105 submissions collected from Github repositories of students.

2.2 Methodology and Metrics

After we trained the obfuscated code2seq model for 13 epoches (using obfuscated java-small dataset to re-train code2seq), we have two models for comparison (the original code2seq model is the external work we are comparing our model to). We evaluated both models on the student submission dataset

respectively and got two sets of results, i.e., the sequence representation of the code. To evaluate these two models' performance on the code similarity task, for each set of the results, we represent those sequences as vectors using word2vec, and then use similarity metrics to find the most similar vector for each vector. The similarity metrics used here is KDTree. After that, we mapped the vectors and their most similar vectors back to the original dataset and got the most similar method for each method as the final results.

To measure the accuracy of the results, we regarded two data points to be similar if they had the same file name and method name. If one's most similar data point had the same file name and method name, then we regarded the output to be correct. Finally, the accuracy score for each of the code2seq models was computed as $\frac{\text{\# of correct results}}{\text{\# of all the results}}$.

2.3 Results and Findings

Using the accuracy score calculation mentioned above, the original data trained code2seq got an accuracy score of 45.2%, while the one trained by the obfuscated dataset got only 30.8%. These two scores cannot be viewed alone because they cannot represent the real accuracy, but from the scores we can see that there is a discrepancy between the performance of these two models.

By manually analyzing the sequences outputted by code2seq, we found the code2seq model trained by obfuscated data will tend to output more irrelevant words. For instance, in obfuscated model, both the method named "set move validity" in the "Message" file and the method named "set draw" in the "Gameboard" file are represented by code2seq as "set is set". They are thus represented by the same vectors and paired as the most similar ones by KDTree, but obviously, the functionalities of these two methods are not similar. These kinds of errors are much less in the other model.

3 Deliverables

What were delivered and their locations:

- The vast majority of our code, scripts, configuration files, models, datasets and documentation can be found in our Github repository[1]
- (Reuse) To rerun this project, you will need to clone the original code2seq repository [2]
- (Reuse) The obfuscation tool [3] we used to construct the obfuscated dataset (based on Java-small dataset provided by code2seq repository) to re-train the model.

In our Github repository's README file, there is a complete list of what specific files are included and where to find them. Additionally, we will give a quick overview of the repository's content here. There are three directories in the repository: All_Project_Assignments, Code_Scripts_Configurations and Raw_Data_From_Experiments. In All_Project_Assignments, we have included all assignment submissions for this project. In Code_Scripts_Configurations, we have included all code, scripts and configurations we have either edited or created during the project. For example, there are files for preprocessing dataset, training the code2seq model and evaluating the models. Lastly, in the Raw_Data_From_Experiments directory, we have included datasets we used, models we used and data obtained during the final project such as the final csv output.

Our project is totally reproducible. Please refer to the README.md file in the GitHub repository:[4] We will also give a quick high-level overview for the reproduction steps here as well: 1) clone all three

[1] https://github.com/AriYJ/Code2seq_Obfuscated
[2] https://github.com/tech-srl/code2seq
[3] https://zenodo.org/record/3601003#.YFzjS69KjIU
[4] https://github.com/AriYJ/Code2seq_Obfuscated/blob/main/README.md

bullet items mentioned earlier in this section (our Github repository, the original code2seq repository, the obfuscation tool), 2) use the obfuscation tool to obfuscated the java-small data included in our repository and add the new dataset into the original code2seq repository clone, 3) replace some files within the original code2seq repository clone with the files in our repository clone (or simply add), 4) run relevant scripts.

4 Reuse

In this section, we outline the external materials we built our project on. We based our model on code2seq [2], and used the obfuscated code2vec tools [4] to obfuscate the java-small dataset (provided by code2seq). Additionally, we used the student submission data from Columbia's 4156 Advanced Software Engineering (fall 2020) as evaluation dataset for models. Following are the repository links:

- Code2seq: https://github.com/tech-srl/code2seq
- Obfuscated code2vec: https://github.com/basedrhys/obfuscated-code2vec
- Java-small dataset: https://s3.amazonaws.com/code2seq/datasets/java-small.tar.gz
- Columbia 4156 Fall Submission: https://tinyurl.com/3tacun7b

5 Team Task Division

In this section, we include an overview of what each one of us did during the project.

- **Aria Jiang**: Configured environment (installed necessary packages), adapted existing model (code2seq) so that it could evaluate the submission dataset and outputted the prediction results in csv file format. Trained code2seq model with obfuscated java-small data for 19 (we ended up using the model with 13 epochs) with Shirish's help. Evaluated submission data with the original small code2seq model and the obfuscated code2seq model. Prepared demo slides, prepared final submission repository and final report.
- **Yiqiao Liu**: Applied word2vec to the output of code2seq, calculated the similarity/distance between each pair of embedding, and found the "closest" code. Compared results from the original code2seq model and that from the obfuscated code2seq model. Prepared the slides for explaining the results, prepared final submission repository and final report.
- **Shirish Singh**: Completed initial environment setup. Extracted and cleaned ASE Dataset. Obtained obfuscation tool, set up the environment and obfuscated the java-small dataset. Pre-processed java-small dataset for training. Set up training & pre-processing scripts, trained obfuscated code2seq and troubleshot environment issues. Wrapped up code2seq training, prepared materials for public release, prepared demo slides, assisted in analysis, and addressed any technical challenge. Prepared final submission repository and final report.

6 Problems Encountered & Planned But Didn't Complete

Originally, we were thinking about retraining the code2seq large model based on obfuscated java-large data. However, we soon realized that due to computation and time constraint, it is impossible to do so (with 3,000 epochs). Therefore, we decided to retrain the code2seq small model based on obfuscated java-small data. The original code2seq small model was trained for 13 epochs and it was doable for us to train the obfuscated model also for 13 epochs. This is the main thing we planned but

didn't get to complete as originally thought (but complete in the sense that we found an alternative by comparing the two small models).

In terms of the problems we encountered during the project, we have summarized them below:

- The obfuscation script provided by the researchers [4] failed on some Java programs. We do not consider the failure on a small number of data samples to be a significant disadvantage since this behavior was also observed in their research [4].
- We faced technical challenges while pre-processing and training the code2seq model. The scripts were not able to fully pre-process all the files in the data set because of time and memory constraints. In addition, because of the large size of the model and limited GPU memory, we could only train the model on CPU. At last, we were able to successfully train the model for 19 epochs.
- Regarding performance evaluation, although all of the submission files serve similar roles, it is hard to tell which functions in these files are similar, i.e, there is no ground truth, so I need to manually select similar functions to test. At last, we are still not able to give a absolute value of accuracy, but it should be enough for comparing the relative accuracy between non-obfuscated-data-trained code2seq and obfuscated-data-trained model.

7 Threats to Validity

As mentioned in the above section, some files in the dataset were not obfuscated. Furthermore, some files were excluded from the training dataset because of the default time constraints of code2seq. Consequently, our obfuscated code model was trained on fewer samples than the original Java-small dataset. Our study results would likely change if we use a larger dataset or include the excluded files.

References

[1] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A Convolutional Attention Network for Extreme Summarization of Source Code. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2091–2100. JMLR.org, 2016.

[2] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations*, 2019.

[3] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. A general path-based representation for predicting program properties. *SIGPLAN Not.*, 53(4):404–419, June 2018.

[4] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. Embedding Java Classes with Code2vec: Improvements from Variable Obfuscation. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 243–253, New York, NY, USA, 2020. Association for Computing Machinery.

8 Self Evaluation

This section outlines what each one of us has learned from this final project.

- **Aria Jiang**:

  I truly learned a lot from working on this final project with my teammates. Prior to this, I had barely any experience working with models and data from published research papers. Therefore, the thought of altering the model, retraining the model and applying the model to different data sets are quite daunting to me. However, with lots of research as well as trial and error in addition to Shirish's help, I was able to complete a variety of tasks. My main tasks include 1) adapting code2seq model (editing interactive_predict.py file) so that it can evaluate new data sets and output prediction results into a csv file, 2) preprocess new data set for training (editing preprocess.sh), 3) training code2seq model based on preprocessed obfuscated data (editing train.sh and config.py).

  In terms of the main challenges I was faced with during the process, I would say they are 1) not knowing which files to edit due to lack of experience and instruction on Code2seq's Github page, 2) limitations stemming from things that are out of my control (e.g. the java extractor used in the project has a processing time limit), 3) difficulty in debugging in a large project (had to dig through files to understand what was really wrong and what needed to be changed), 4) extremely long training time when training code2seq on obfuscated data.

  In terms of what I have learned, the main points are 1) gaining knowledge and experience on how to alter and reuse state-of-the-art models from research papers, 2) gaining debugging and rewriting experience in working with large projects that were written by others, 3) obtaining specific programming know-hows (e.g. rglob in Python, using vi to directly edit scripts on vm).

  When it comes to comments about this course, I really want to emphasize how accessible professor made it through the option to write a discussion after watching course recording instead of attending class real-time, the option to present by recording instead of live presenting and the generous two-free-day late policy on assignments (even though I have never used it personally). All of these practices make it a great experience taking this course remotely.

- **Yiqiao Liu**:

  In our project, I am mainly responsible for applying the output we got from the code2seq model to the code similarity task, i.e., finding the most similar pairs of code. This includes a few steps: applying word2vec to the output of code2seq, normalizing the vectors, using similarity metrics to find the closest pairs of vectors, and mapping the pairs of closest vectors back to the pairs of code. After I got those most similar pairs of code, I evaluated the accuracy of this output, i.e., whether these pairs are really similar or not. I approached the evaluation part by both calculating an accuracy score and manually looking at the output, trying to understand what's going on in each step. Finally, I concluded that the obfuscated data trained code2seq model achieved a worse performance than the non-obfuscated one on this task, and I found the reason behind this is that the obfuscated code2seq model will tend to output unrelated words.

  There are two challenges in my work. One is to find the best similarity metrics. I considered Euclidean distance and cosine similarity, but since our data size is relatively large, I decided to use KDTree at last to minimize time cost. The KDTree given by the scipy package turned out to output the data point itself as the closest vector. At first I didn't notice that and outputted a few set of wrong results. Gladly I fixed them before the evaluation. The other is to measure the performance of the two models, i.e., the accuracy of their output on this specific task.

Since there is no ground truth regarding which methods are similar in our dataset, I had to think of a way that can directly compare the results. At last, I regarded two data points to be similar if they have the same file name and method name as the tic-tac-toe functionalities are quite simple. Although each score cannot really represent the true accuracy of that model, the discrepancy between these two scores does show that the obfuscated model performed worse on this task. To validate this result, I also manually analyzed the output and reached the same conclusion.

In terms of what I learned, besides how to look for problems, form questions, and find solutions to challenges as I mentioned above, I am really glad that we answered our research questions. I mentioned in our revised proposal that the reason I am interested in this project is that I want to find answers to these two research questions. Initially, I thought obfuscation of variable names would improve code2seq on this task because it did improve code2vec, but surprisingly, after these experiments, we found it didn't.

- **Shirish Singh**: I was primarily responsible for environment setup and troubleshooting. However, I assisted other members in other tasks. In fact, all members of the team ended up contributing to all aspects of the project. In terms of challenges, there were several system integration-based issues. The task of troubleshooting programs written in python and calling external Java programs was slightly challenging primarily because of library version mismatch and lack of logging in the code2seq codebase. Moreover, limitations on the computation power caused a delay in obtaining results. But I learned a lot about system integration, importance of documentation, and proper logging.

  Learning about code2seq was the most interesting part of the course. Other than that, it was interesting to see that the code2seq model, trained on obfuscated code, was able to predict the method names correctly for some methods. Explaining this observation helped me learn about the underlying representation of the code2seq model and its limitations. A widely used code2vec model is also built upon the same AST representation and therefore suffers from the same drawbacks. Using assignments from COMS 4156 allowed us to evaluate the models on familiar programs.

  Some papers presented in the class were helpful for my research work outside class. However, the lack of in-class discussions limited the learning opportunity. I guess online classes do not engage students like in-person classes. Anthony (TA) was very helpful in guiding the students.