Code2seq Model Trained On Obfuscated Data

Shirish Singh, Aria Jiang, Yiqiao Liu, April 13th

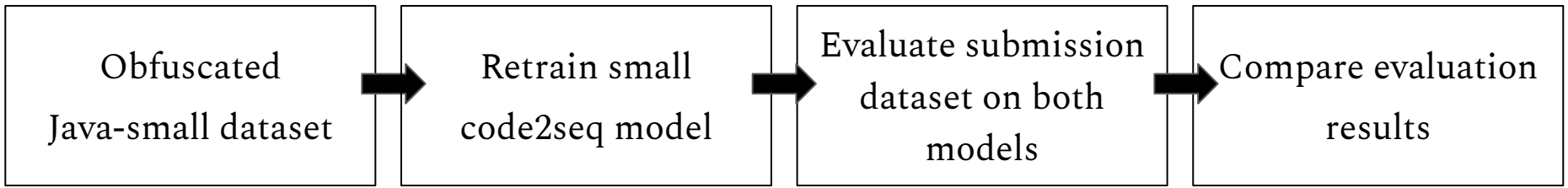
Project Overview (1/2)

- Build a code similarity tool for multi-file projects
- **Research Questions**:
  - Does obfuscation of variable names yield an improved code2seq model?
  - How can we utilize code2seq model's output for code similarity?
- **Motivation**: code2vec learns to rely on variable names for prediction, causing it to be easily fooled by typos or adversarial attacks [1].
- **Datasets**:
  - Training: Java-small dataset provided by code2seq
  - Evaluation: 105 student submissions from Advanced Software Engineering fall 2020

[1] Compton, Rhys, 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*, pp. 243-253. 2020.

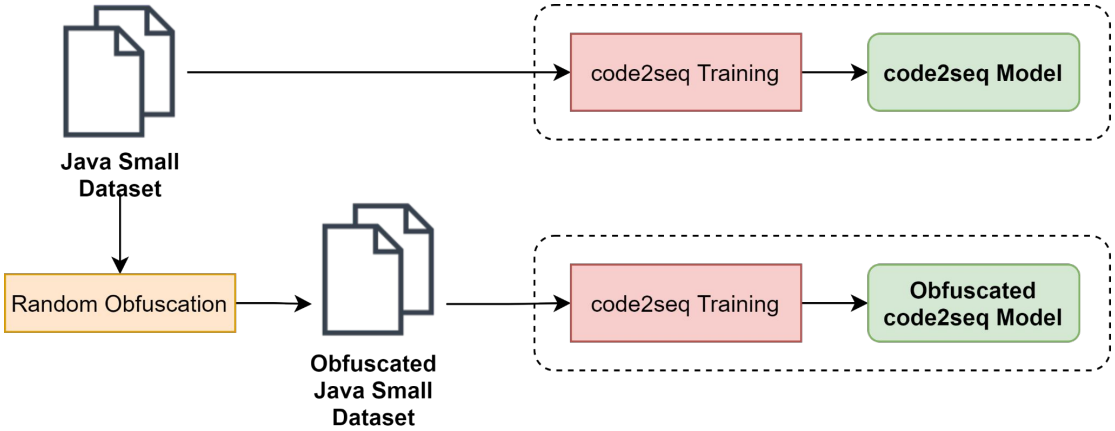
Project Overview (2/2)

- Novelty - aim to find out whether obfuscation can improve code2seq, and if so, how it improves code2seq specifically (similar research on code2vec)
- Value to User Community
  - May help researchers/developers understand obfuscated programs better
  - Program understanding can improve developer's performance by recommending method names



Environment Setup & Data Obfuscation

- System spec: Dell 8930, i5 (6 cores), 56 GBs RAM, Nvidia GTX 1070 (8 GB)



Model Training & Evaluation (1/3) - Data Preprocessing

Edited preprocess.sh to point to obfuscated datasets

```
#VAL_DIR=my_val_dir
#TEST_DIR=my_test_dir
TRAIN_DIR="/home/TSE_Project/code2seq/data/obfuscated/"
VAL_DIR="/home/TSE_Project/code2seq/data/obfuscated_validation/"
TEST_DIR="/home/TSE_Project/code2seq/data/obfuscated_test/"

DATASET_NAME=my_dataset
MAX_DATA_CONTEXTS=1000
MAX_CONTEXTS=200
SUBTOKEN_VOCAB_SIZE=186277
TARGET_VOCAB_SIZE=26347
NUM_THREADS=64
PYTHON=python3
##########################################################

TRAIN_DATA_FILE=${DATASET_NAME}.train.raw.txt
VAL_DATA_FILE=${DATASET_NAME}.val.raw.txt
TEST_DATA_FILE=${DATASET_NAME}.test.raw.txt
EXTRACTOR_JAR=JavaExtractor/JPredict/target/JavaExtractor-0.0.1-S
```

Model Training & Evaluation (2/3) - Model Training

Edited train.sh to point to right preprocessed data;
Edited config.py to alter hyper-parameters

```
type=java-small-model-19-epochs
dataset_name=java-small
data_dir=data/java-small-preprocessed/java-small
data=${data_dir}/${dataset_name}
test_data=${data_dir}/${dataset_name}.val.c2s
model_dir=models/${type}
```

```
class Config:
    @staticmethod
    def get_default_config(args):
        config = Config(args)
        config.NUM_EPOCHS = 13
        config.SAVE_EVERY_EPOCHS = 1
        config.PATIENCE = 10
        config.BATCH_SIZE = 512
        config.TEST_BATCH_SIZE = 256
        config.READER_NUM_PARALLEL_BATCHES = 1
        config.SHUFFLE_BUFFER_SIZE = 10000
```

Model Training & Evaluation (3/3) - Evaluation

Edited interactive_predict.py to customize script for submission data;
Outputted csv file with four columns: submission id, java file name, original method name and predicted method name

```
def predict(self):
    fileDir = os.path.dirname(os.path.realpath('__file__'))
    #start_num, end_num = 1, 105
    with open('../output.csv', 'w+') as csvfile:
        filewriter = csv.writer(csvfile, delimiter = ',', quotechar = '|', quoting = csv.QUOTE_MINIMAL)
        filewriter.writerow(['submission_id', 'file_name', 'method_name', 'prediction'])
    for path in Path('../dataset').rglob('*.java'):
        print(path)
        input_filename = os.path.join(fileDir, path)
        input_filename = os.path.abspath(os.path.realpath(input_filename))
        submission_num_regex = re.compile(r'(Submission_)(\d+)')
        submission_num = submission_num_regex.search(input_filename).groups()[1]
```

```
69,Message,|get||code|,|get||code|
69,Message,|get||message|,|get||message|
69,GameBoard,|get||player||by||id|,|get||player|
69,GameBoard,|start||game|,|start||game|
69,GameBoard,|attempt||move|,move
69,GameBoard,|in||win||state|,|is||valid|
69,GameBoard,|check||all||same||type|,|is||same||type|
69,GameBoard,|in||draw||state|,|is||empty|
69,GameBoard,|get||p|,|get||player|
69,GameBoard,|set||p|,|set||p|
69,GameBoard,|get||p|,|get||player|
69,GameBoard,|set||p|,|set||p|
```

Prediction Results & Findings

- To compare the code2seq results, we represent those sequences as vectors using word2vec, and then use similarity metrics (KDTree) to find the most similar vector for each vector. At last, we map these pairs back to the methods
- To measure accuracy, we regard two data points to be similar if they have the same file name and method name
- If one's most similar data point has the same file name and method name, we regard the output to be correct
- The non-obfuscated data trained code2seq got an accuracy score of 45.2%, while the obfuscated one (13 epoch) got only 30.8%

Prediction Results & Findings

- The accuracy scores can't be viewed alone, but there is a discrepancy between the two models
- Example:

  In obfuscated model, both the method named "set move validity" in "Message" file and the method named "set draw" in "Gameboard" file are represented by code2seq as "set is set". They are thus paired as the most similar ones by KDTree.
- In non-obfuscated model, there are fewer errors like this.

Challenges

- The obfuscation script provided by the researchers failed on some Java programs
- Code2seq's scripts were not able to fully pre-process all java files in the submission dataset because of time and memory constraints
- Due to large size of the model and limited GPU memory, we could only train the model on CPU. At last, we were able to train the model for 19 epochs
- There is no ground truth for performance evaluation, so we needed to manually select similar functions to test

Questions?