Semantic Code Search

CSE 6324: Advanced Topics in Software Engineering
Iteration 2

Team 7

Fotios Lygerakis Mohammad Rifat Arefin

Introduction

The project aims to perform code search based on natural language query. As developers spend a significant amount of time looking for relevant code snippets over the Internet, this code search process must be a sophisticated one. The keyword-based search approaches involve checking the textual similarity between source code and natural language query. However, these approaches lack the deep understanding of the natural language query and source code. In this project, we are going to incorporate machine learning models to jointly embed source code and corresponding natural language description into a high dimensional vector space. As code and the corresponding description will have similar vectors, the vector representation of the natural language query will retrieve the relevant code snippet by a nearest neighbor search in the vector space.

For training and evaluation purposes we are using the dataset from CodeSearchNet [1] challenge. The dataset consists of a total of six million methods of which 2 million have associated description as docstring comment. This dataset contains code from these six programming languages: Python, Javascript, Ruby, Go, Java, and PHP.

Project Plan: Features

Our project plan includes the following list of features. The list consists of both already implemented features and the proposed ones for future iterations.

Iteration 1:

1. Train an ML model on a subset of the CodeSearchNet [1] dataset
    - We trained a 1D-CNN model to capture the vector representation of both code and natural language descriptions for only Python code from the dataset.

2. Build a command-line tool to perform demo code search

    - We wrote a Jupyter notebook to perform code search utilizing the already trained 1D-CNN model. A user has to provide the search query in a cell, the 10 most relevant code snippets are retrieved for that query.

3. Collect feedback from potential customers and project mentor

    - We extensively collected feedback from the representative customers for the command-line code search tool. Also, we discussed with our project mentor regarding the project direction.

Iteration 2:

1. Expand the ML model for all six languages of the dataset

    - We expanded the ML model with more data from iteration 1. This time, the model is trained with the code from six programming languages: v The Neural Bag of Words model is used to encode both code and associated description.

2. Expand the command-line tool from iteration 1 to support the above programming languages.

    - We developed a command-line interface to support code search. A user can specify a language for which code snippets would be retrieved. We utilized the model trained during this iteration to develop the tool.

Future iterations:

1. Experiment with state of the art language models [planned for iteration 3]

    - We will experiment with state of the art language models like GPT-2 [3]. If GPT-3 [4] will be published in a reasonable time before the end of the course, we will focus on it, since it has shown great potential so far.

2. Develop a web app for code search and host it for our classmates [planned for iteration 3]

    - To demonstrate the performance of our code search tool in a better way, we will develop a web app for this. In this web app, a user can input a search query through a text field and specify among the six programming languages through a checkbox. The relevant code snippets will be retrieved and will be shown by the order of their ranking. Figure 1 describes the proposed layout of the webapp.

Figure 1. Proposed layout of the webapp

3. Examine the use of Code2Vec [6] in Code Search [planned for final iteration]

    - As Code2Vec [6] has shown great potential to capture a code snippet's functionality, we will incorporate it with our project in the final iteration.

4. Comparison between the two approaches: (a) Joint training of code and sentence encoder and (b) Finetuning [planned for final iteration]

    - So far, we have come across two different approaches for this problem. On the one [2], two different encoders are being developed for the code and query embeddings, using a code summarizer and a natural language model respectively. The models are being trained independently and then the encoder part of the code summarizer is being used to map code representations to the vector space of the language model. On the second approach [1], an end-to-end training is being deployed for the encoders of both the code and the queries. Our plan is to make a comparison between both approaches.

5. Use a domain-specific dataset like StaQC [5] to train the language model [planned for final iteration: low priority]

    - To capture the semantics of natural language queries in a better way, we plan to use the data from Stackoverflow. It has a rich discussion related to code, we believe a language model trained with this data will be helpful for our task.

Competitors (same from Iteration 1)

Our main competition so far are the entries in the Code Search Net Challenge [7]. We will try to outperform the leading approaches and provide the most efficient code search.

Risks

These are top five risks according to their severity

1. (old) The available data and modeling techniques may be insufficient to develop a better semantic code search engine than the state-of-the-art. There is an ambiguity about the probability of this happening, because we must experiment first on the proposed approaches of our plan and then decide. So we give a probability of 50% and an effect of 40 hours to redirect our efforts at the database retrieval system to speed up the code search, or create a better interface for the search engine, in order to provide a better experience to the user. Consequently, the risk effect will be 20 hours.
2. (old) At first, we will start with one programming language and gradually finetune the code encoder with other languages also. It is highly probable that the overall performance deteriorates after adding more languages. The probability of this risk is 40%, we have to focus on fine-tuning the model for overall better performance. We have to spend 30 more hours if this risk becomes reality. So the risk exposure is 12 hours.
3. (New) Hosting a web app for this project involves new risks. We have to find a suitable hosting service with available resources to host our tool. Additionally, we have to design the user interface by ourselves if there are no available designs. The probability of this risk is 70% and we have to work for 15 more hours if the risk becomes reality. So the overall risk exposure is 10.5 hours.
4. (New) We are developing our project in Singularity [12] containers for better reproducibility. It involves a new risk when we will need additional packages. We have to add these packages to the Singularity definition files and rebuild the container. The probability of this risk is 50% and the effect is 6 hours when this risk becomes reality. So the overall risk effect is 3 hours.
5. (New) For a better query representation, we plan to train a language model on Stackoverflow data. We are not prioritizing this step, If time allows, we will incorporate this feature. As this one is a low priority feature, the possibility of the risks associated will be very low - 5%. The risks associated with it involve data cleaning, fitting the data to the pipeline of the language model, and so on. These steps require 40 hours of work. So the overall risk exposure is 2 hours.

- <u>Eliminated</u>
    - Existing code performing semantic code search, on which we plan to base our own is built on libraries that are old or deprecated. From a thorough examination of the existing work, this is quite probable, so we give a probability of 70% happening and a 15 hour effect to mitigate it by installing it in docker environments. So the risk effect for this is 10.5 hours.

- Encountered
    - The team's hardware resources may be insufficient for training and deploying machine learning models. This is again very probable, due to the weak specifications of the team's laptops. The probability of this risk is 80% and the effect for overcoming it by transferring all the work on UTA's or commercially available servers, like AWS, is 20 hours, including the appropriate familiarization of the team members with these tools. Thus, the risk effect is 16 hours.
    - The size of the training data may be very large to fit in the RAM of the team's computers. This risk has a probability of 60% and an effect of 6 hours to be mitigated, by splitting the initial dataset into smaller subsets, either on the team's machines or on an online server. Thus, the risk effect will be approximately 3.5 hours.

Risk Mitigation

The risks mentioned above would be avoided if the team members are guided to the appropriate way. We can make the risk exposure minimum if we get directions from someone who has experience in the similar domain. That's why we plan to meet with our project mentor and show our plans and progress in advance. Additionally, the feedback from our customers and project mentor would help us to set our project plan that would incur minimum risks.

Specifications and Design

Inputs and outputs(same from Iteration 1)

The input of the system will be a natural language query for a particular language and output will be the corresponding code snippets ranked by their relevance.

Input	Output
Query	Url of code snippets

```
In [6]: query = 'matrix multiply'
```

```
Top ten results
https://github.com/apache/spark/blob/618d6bff71073c8c93501ab7392c3cc579730f0b/python/pyspark/mllib/linalg/distributed.py#L373-L389
https://github.com/apache/spark/blob/618d6bff71073c8c93501ab7392c3cc579730f0b/python/pyspark/mllib/linalg/distributed.py#L705-L720
https://github.com/pymupdf/PyMuPDF/blob/917f2d83482510e26ba0ff01fd2392c26f3a8e90/fitz/fitz.py#L270-L275
https://github.com/pikepdf/pikepdf/blob/07154f4dec007e2e9c0c6a8c07b964fd06bc5f77/src/pikepdf/models/matrix.py#L63-L79
https://github.com/tensorflow/cleverhans/blob/97488e215760547b81afc53f5e5de8ba7da5bd98/cleverhans/utils_tf.py#L561-L569
https://github.com/pymupdf/PyMuPDF/blob/917f2d83482510e26ba0ff01fd2392c26f3a8e90/fitz/fitz.py#L277-L284
https://github.com/pandas-dev/pandas/blob/9feb3ad92cc0397a04b665803a49299ee7aa1037/pandas/core/frame.py#L1026-L1030
https://github.com/quantumlib/Cirq/blob/0827da80dd7880e5b923eb69407e980ed9bc0bd2/cirq/ops/matrix_gates.py#L38-L47
https://github.com/materialsproject/pymatgen/blob/4ca558cf72f8d5f8a1f21dfdfc0181a971c186da/pymatgen/core/operations.py#L196-L202
https://github.com/keon/algorithms/blob/4d6569464a62a75c1357acc97e2dd32ee2f9f4a3/algorithms/matrix/multiply.py#L10-L28
```

Figure 2. Sample output of the given query “matrix multiply”

Use cases

In figure 3 diagram we see a typical use case of the semantic code search:

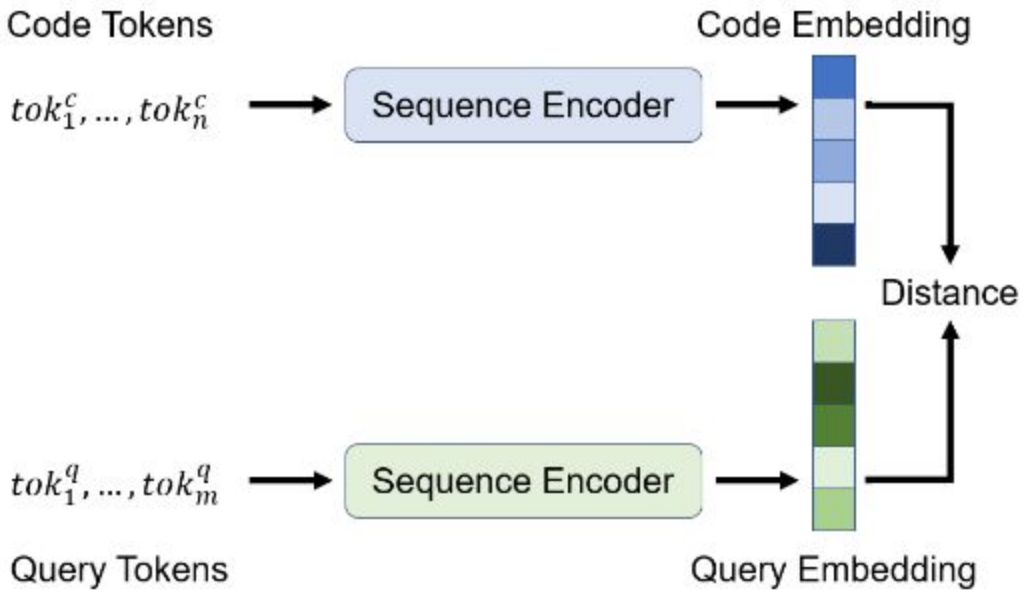

Figure 3. Sample of Semantic Code Search given the query “read text file line by line” [1]

- We take the query embedding from a **query token sequence** of size m
  $tok_i^q, for\ i = 1, .., m$

- We take the code snippet embedding from a **code snippet token sequence** of size n $tok_i^c,\ for\ i = 1, .., n$
- We measure the relevance based on their **cosine distance.**
- **Return** the code with a **smaller** distance.

More specifically, the user inputs a natural language query and the system retrieves a list of links to relevant GitHub repositories (Figure 4).

```
In [16]: query = 'read text file line by line'
```

```
In [17]: for idx, _ in zip(*query_model(query, model, indices, language)):
             predictions.append((query, language, definitions[idx]['identifier'], definitions[idx]['url']))

         df = pd.DataFrame(predictions, columns=['query', 'language', 'identifier', 'url'])
         # urls = list(df['url'])
         # print("Top ten results")
         # for link in urls[:10]:
         #     print(link)
         df.head(10)
```

Out[17]:

	query	language	identifier	url
0	read text file line by line	python	ReTextTab.replaceAll	https://github.com/retext-project/retext/blob/...
1	read text file line by line	python	readlines	https://github.com/spyder-ide/spyder/blob/f768...
2	read text file line by line	python	TextEditBaseWidget.__move_line_or_selection	https://github.com/spyder-ide/spyder/blob/f768...
3	read text file line by line	python	TextEditBaseWidget.__duplicate_line_or_selection	https://github.com/spyder-ide/spyder/blob/f768...
4	read text file line by line	python	BaseEditMixin.is_cursor_on_last_line	https://github.com/spyder-ide/spyder/blob/f768...
5	read text file line by line	python	ReTextTab.find	https://github.com/retext-project/retext/blob/...
6	read text file line by line	python	TrailingWhitespaceRule.match	https://github.com/ansible/ansible-lint/blob/b...
7	read text file line by line	python	BaseEditMixin.get_text_with_eol	https://github.com/spyder-ide/spyder/blob/f768...
8	read text file line by line	python	_set_line_eol	https://github.com/saltstack/salt/blob/e8541fd...
9	read text file line by line	python	TextEditBaseWidget.delete_line	https://github.com/spyder-ide/spyder/blob/f768...

Figure 4. Sample of Semantic Code Search given the query “read text file line by line”

Dataset Format[10] (same from Iteration 1)

- JSON-Lines[9] format is used to store the data. One row of the JSON-Line file has these following attributes:
  - **repo:** the owner/repo
  - **path:** the full path to the original file
  - **func_name:** the function or method name
  - **original_string:** the raw string before tokenization or parsing
  - **language:** the programming language
  - **code:** the part of the original_string that is code
  - **code_tokens:** tokenized version of code
  - **docstring:** the top-level comment or docstring, if it exists in the original string
  - **docstring_tokens:** tokenized version of docstring
  - **sha:** this field is not being used

- **partition:** a flag indicating what partition this datum belongs to of {train, valid, test, etc.}
- **url:** the url for the code snippet including the line numbers

Method

Neural Bag Of Words (NBOW) Encoder (Figure 5)

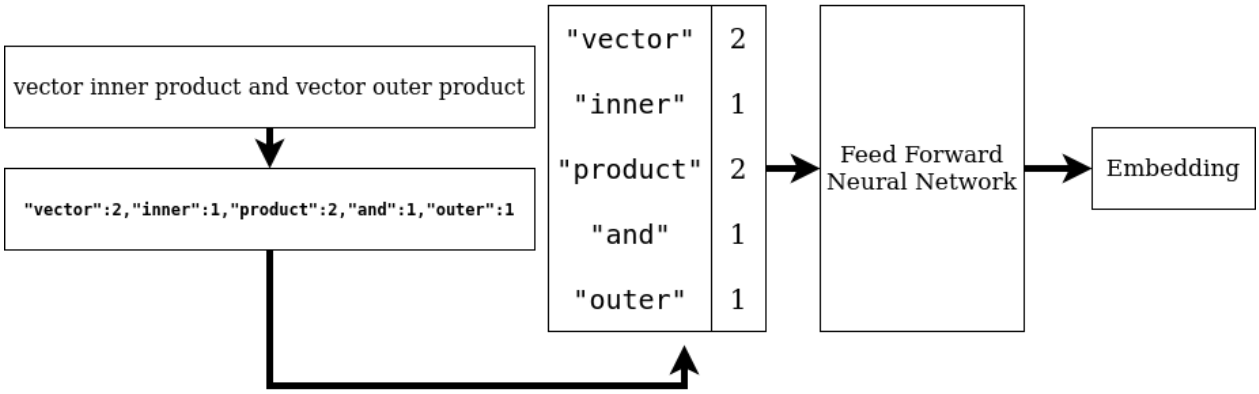

Figure 5. Schematic Diagram of Neural Bag Of Words encoder

We use one encoder per programming language and one for the queries. The optimization [1] is based on:

- Maximizing the inner product of the query and code embeddings
- Minimizing the inner product of the code snippet c_i and a distractor one $c_j (i \neq j)$

Data Structures (same from Iteration 1)

- Throughout the architecture of the semantic code search, we use different instances of DataFrames from the Pandas library [8], varying in the number of columns and types of information they contain. In Figure 4, for example, we use a DataFrame with four columns, containing information about the query, the type of programming language, the identifier, and the URL of the code snippet.

- **Annoy**[11], a C++ library with Python bindings is used to index all the functions from the CodeSearchNet corpus. It is useful at test time as it offers fast nearest neighbor indexing and thus the search becomes fast.

Code and Tests

The GitHub repository of this project: https://github.com/rifatarefin/semantic-code-search

- The existing CodeSearchNet [10] challenge uses Docker to containerize the development environment. As we had hardware constraints on our personal laptops, we set up our project on the HPC clusters in the Maverick2 server. However, these HPC clusters don't allow root privilege to the users. So we had to migrate everything to a rootless containerized environment called Singularity [12]. We had to replace everything specific to the Docker environment with Singularity. We have a separate branch with a Readme file to run our project inside a Singularity container. https://github.com/rifatarefin/semantic-code-search/tree/mav2

- Demo: https://docs.google.com/presentation/d/1l5rZ_5rYBs-SVwXbziZw07LTzuLo65BeGdYJ-lYsYO4/edit?usp=sharing

Customers and Users (same from Iteration 1)

Potential customers are code hosting and versioning services like Github, Gitlab, etc, general purpose search engines like Google, Bing, etc, and IDEs with integrated code search engines.

Feedback

The initial feedback will be obtained from the developers of the semantic code search software, but before the final presentation classmates will be asked to review the search engine.

References

[1] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, Marc Brockschmidt: CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. CoRR abs/1909.09436 (2019)

[2] https://towardsdatascience.com/semantic-code-search-3cd6d244a39c

[3] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D. & Sutskever, I. (2018), 'Language Models are Unsupervised Multitask Learners', .

[4] Brown, Tom B.; Mann, Benjamin; Ryder, Nick; Subbiah, Melanie; Kaplan, Jared; Dhariwal, Prafulla; Neelakantan, Arvind; Shyam, Pranav; Sastry, Girish; Askell, Amanda; Agarwal, Sandhini; Herbert-Voss, Ariel; Krueger, Gretchen; Henighan, Tom; Child, Rewon; Ramesh, Aditya; Ziegler, Daniel M.; Wu, Jeffrey; Winter, Clemens; Hesse, Christopher; Chen, Mark; Sigler, Eric; Litwin, Mateusz; Gray, Scott; Chess, Benjamin; Clark, Jack; Berner, Christopher; McCandlish, Sam; Radford, Alec; Sutskever, Ilya; Amodei, Dario (July 22, 2020). "Language Models are Few-Shot Learners". arXiv:2005.14165

[5] https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset

[6] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: learning distributed representations of code. *Proc. ACM Program. Lang.* 3, POPL, Article 40 (January 2019), 29 pages. DOI:https://doi.org/10.1145/3290353

[7] https://github.blog/2019-09-26-introducing-the-codesearchnet-challenge/

[8] https://pandas.pydata.org/docs/

[9] https://jsonlines.org/

[10] https://github.com/github/CodeSearchNet

[11] https://github.com/spotify/annoy

[12] https://sylabs.io/guides/3.0/user-guide/quick_start.html