AMYA CODERS

204202001

SEMANTIC CODE SEARCH -1

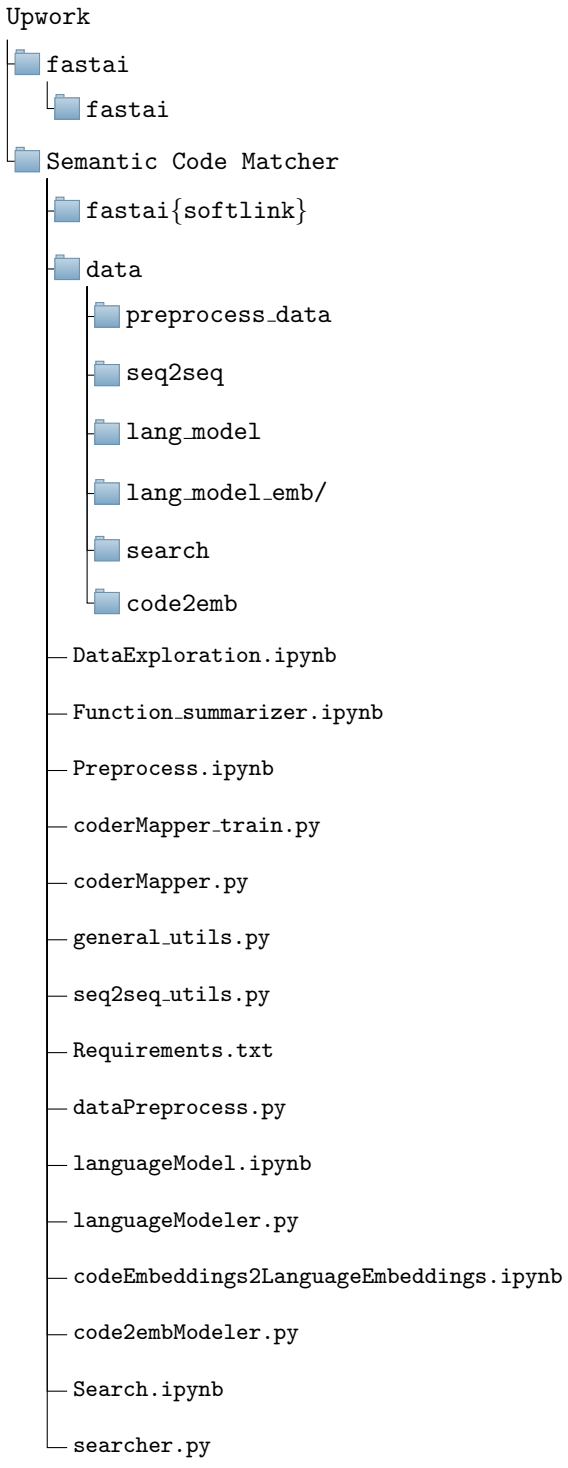
Instruction Manual

Author:
Atul Sahay

April 7, 2020

Contents

1 Project's Directory Structure



2 Requirements

2.1 How to install required modules for the project?

- Search for the **Requirements.txt** file in the project repository.
- Open the terminal and write (be sure you are in the project directory): See Fig: 2.1

user: Semantic-Code-Matcher$: **pip3 install -r Requirements.txt**

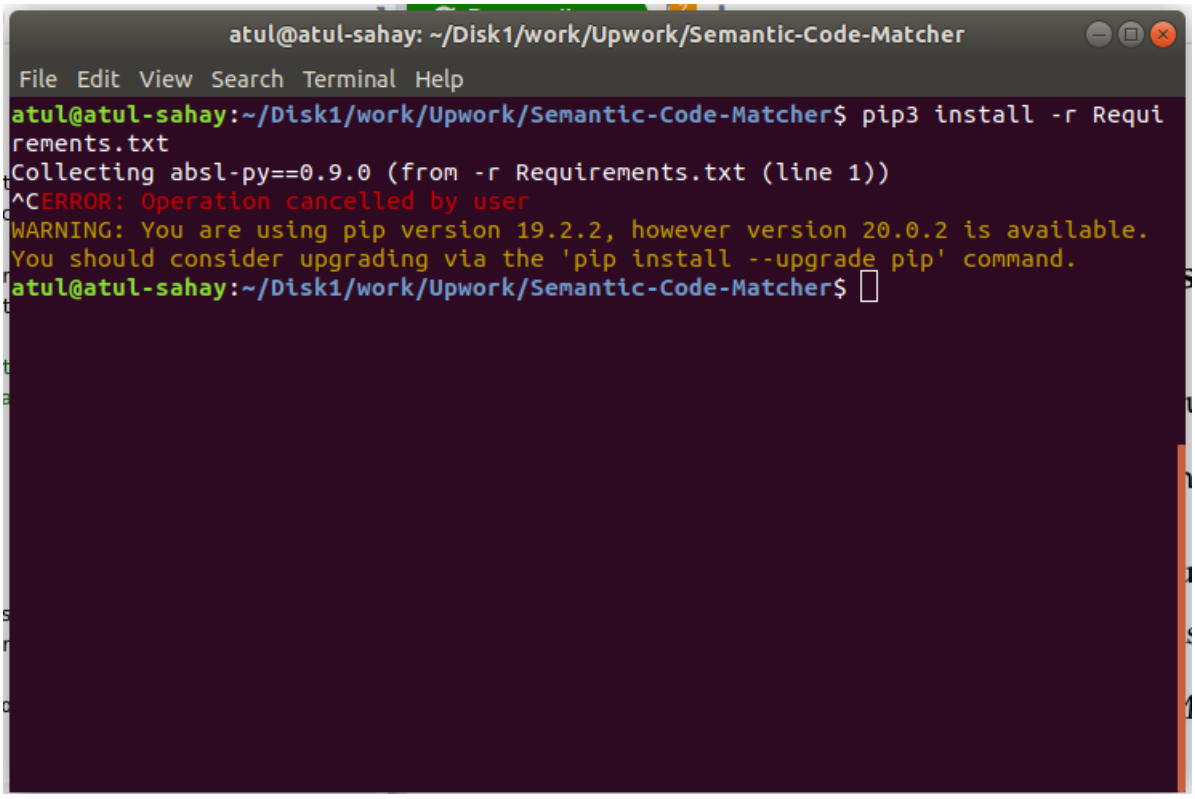


Figure 1: Installation of the required modules

3 Data Preprocessing

For the data loading and data preprocessing, you need to follow the given mentioned steps:

1. First you need to create a directory structure: as provided in section 1 for ”data” and ”preprocess_data”. Here the data will be stored.
2. Now just run the file name ”dataPreprocess.py”

$$-- > \textbf{\$ python3 dataPreprocess.py -t2t 0.87 -t2v 0.82}$$

Arguments:

(a) t2t[Optional, default=0.87] train to test split ratio

(b) t2v[Optional, default=0.82] train to valid split ratio

After these steps pleas run these command and check whether you are getting the similar directory structure or not.

$$-- > \textbf{\$ ls -lah ./data/processed_data/}$$

```
In [25]: !ls -lah ./data/processed_data/
total 2.6G
drwxrwxr-x 2 ritesh ritesh 4.0K Apr  4 05:22 .
drwxrwxr-x 3 ritesh ritesh 4.0K Apr  4 05:16 ..
-rw-rw-r-- 1 ritesh ritesh  16M Apr  4 05:19 test.docstring
-rw-rw-r-- 1 ritesh ritesh  55M Apr  4 05:19 test.function
-rw-rw-r-- 1 ritesh ritesh  18M Apr  4 05:19 test.lineage
-rw-rw-r-- 1 ritesh ritesh  25M Apr  4 05:19 test_original_function.json.gz
-rw-rw-r-- 1 ritesh ritesh  70M Apr  4 05:18 train.docstring
-rw-rw-r-- 1 ritesh ritesh 308M Apr  4 05:17 train.function
-rw-rw-r-- 1 ritesh ritesh  86M Apr  4 05:19 train.lineage
-rw-rw-r-- 1 ritesh ritesh 140M Apr  4 05:18 train_original_function.json.gz
-rw-rw-r-- 1 ritesh ritesh  16M Apr  4 05:19 valid.docstring
-rw-rw-r-- 1 ritesh ritesh  69M Apr  4 05:19 valid.function
-rw-rw-r-- 1 ritesh ritesh  19M Apr  4 05:19 valid.lineage
-rw-rw-r-- 1 ritesh ritesh  31M Apr  4 05:19 valid_original_function.json.gz
-rw-rw-r-- 1 ritesh ritesh 1.1G Apr  4 05:19 without_docstrings.function
-rw-rw-r-- 1 ritesh ritesh 344M Apr  4 05:22 without_docstrings.lineage
-rw-rw-r-- 1 ritesh ritesh 356M Apr  4 05:22 without_docstrings_original_function.json.gz
```

Figure 2: Directory structure of data/preprocess_data folder

4 Code and Doc-string Mapper

4.1 Pre-requisite

Make Sure you have the right files prepared from Step dataPreprocess

1. You should have these files in the root of the ./data/processed_data/ directory:
   (a) {train/valid/test.function} - these are python function definitions tokenized (by space), 1 line per function.
   (b) {train/valid/test.docstring} - these are docstrings that correspond to each of the python function definitions, and have a 1:1 correspondence with the lines in *.function files.
   (c) {train/valid/test.lineage} - every line in this file contains a link back to the original location (github repo link) where the code was retrieved. There is a 1:1 correspondence with the lines in this file and the other two files. This is useful for debugging.
2. **Set the value of use_cache appropriately.**

   In the file: **codeMapper**, if use_cache = True, preprocessed data will be downloaded where possible from the blog googleapis link instead of re-computing. However, it is highly recommended that you set use_cache = False
3. Make sure the data/seq2seq directory exists see the Section 1.

4.2 To Train the codeMapper Model

Run the command specified below

$ python3 codeMapper_train.py

After the completion of training you will find a "code_summary_seq2seq_model.h5" in data/seq2seq directory. see Section 1 for the directory structure.

4.3 Code Mapper Model Inference

You need to get ready with the model file stored in data/seq2seq folder or you can download the pretrained model file from the internet.

Further you need to provide the "input_file" that contains the code text one code per line and "output_file" where corresponding code and doctstring can be stored.

To run the command:

$ python3 Semantic-Code-Matcher/codeMapper.py -d True -I input_file -O out.csv

1. -d[Optional] if True, pretrained model will downloaded from the internet . False local saved model will be used
2. -I[Required] A text file containing the raw code segments one per line.
3. -O[Required] A csv file containg the code,docstring pairs

5 Language Model

5.1 Prerequisites

1. **[Imp] Before proceeding any further we need a fastai module**
   (a) Place the fastai folder in the same directory where this project directory is placed see Section 1.
   (b) For the creation of the softlink, you need to create a softlink:

   Semantic-Code-Matcher:$ ln -s ../fastai/fastai fastai

2. **Make Sure you have the right files prepared from Step dataPreprocess**
   (a) You should have these files in the root of the ./data/processed_data/ directory:
      i. {train/valid/test.function} - these are python function definitions tokenized (by space), 1 line per function.
      ii. {train/valid/test.docstring} - these are docstrings that correspond to each of the python function definitions, and have a 1:1 correspondence with the lines in *.function files.
      iii. {train/valid/test.lineage} - every line in this file contains a link back to the original location (github repo link) where the code was retrieved. There is a 1:1 correspondence with the lines in this file and the other two files. This is useful for debugging.
3. File to be used in this module is **languageModeler.py**

5.2 Model Characteristics

- The whole model is trained on the GPU specification:
  - Name: Geforce RTX 2080 ti
  - Memory: 11 GB
  - Cuda version 10.1
- The whole process takes around 8 hour for completion
- LSTM based model is used without the attention layer. [See the blog]
- Model File will be stored in data/lang_model directory as **./data/lang_model/lang_model_cpu_v2.torch** and **./data/lang_model/lang_model_gpu_v2.torch**

5.3 To Run The Model

.docstrings files placed in data/processed_data directory are needed to run the model.

Enter the command:

Semantic-Code-Matcher:$ python3 languageModeler.py -eval

Arguments

1. **-eval[Optional]**: When written model skips the training and enter into the evaluation mode. Make sure the necessary model files are intact before entering the evaluation mode.

Necessary Files for eval mode

1. text file: data/processed_data/test.docstring
2. encoder file: ./data/lang_model_emb/avg_emb_dim500_test_v2.npy
3. model file: **./data/lang_model/lang_model_cpu_v2.torch** or **./data/lang_model/lang_model_gpu_v2.torch**
4. vocab file: './data/lang_model/vocab_v2.cls'

```
read and parse data in pandas dataframes .

cosine dist:0.1695
--------------
 read csv into special data dictionary . example csv file :

cosine dist:0.1706
--------------
 load dataframe from csv file

cosine dist:0.1780
--------------
 read a pandas . dataframe from feather format

Search text: train random forest
cosine dist:0.1339
--------------
 train a network

cosine dist:0.1401
--------------
 train the scrnn model .

cosine dist:0.1416
--------------
 train a classifier

cosine dist:0.1416
--------------
 train the modeler .

cosine dist:0.1448
--------------
 train the classifier .
```

Figure 3: Manual inspection of the language model

6 Code To Embedding Module

6.1 Prerequisite

You should have completed all the previous steps and make sure the following files do exists before proceeding any further.

1. **Input to the Model** seq2seq (Code Mapper Model) preprocessed encoded data : ./data/seq2seq/py_t_code_vecs_v2.npy
2. seq2seq(CodeMapper Model): ./data/seq2seq/code_summary_seq2seq_model.h5
3. **Target File for the training** language model [from section 5] docstring embedding file: ./data/lang_model_emb/avg_emb_dim500_v2.npy

6.2 Model Characteristics

- The whole model is trained on the GPU specification:
  - Name: Geforce RTX 2080 ti
  - Memory: 11 GB
  - Cuda version 10.1
- The whole process takes around 3 hour for completion
- A dense layer is added on the encoder layer of the codeMapper Model(seq2seq) based [See the blog]
- Model File will be stored in data/code2emb directory as **./data/code2emb/code2emb_model.hdf5**.

6.3 To Run The Model

Meet the prerequisites before running the model Enter the command:

Semantic-Code-Matcher:$ python3 code2embModeler.py

After the model train is completed following files will be stored.

1. Model File : **./data/code2emb/code2emb_model.hdf5**.
2. Preprocessed file (code without docstring) : **./data/code2emb/nodoc_encinp.np**
3. Vectorised code without docstring file: **./data/code2emb/nodoc_vecs.np**

7 The Searcher Module

7.1 Prerequisite

You should have completed all the previous steps and make sure the following files do exists before proceeding any further.

1. **Input File on which search similarity results will given**: ./data/processed_data/without_docstrings.lineage and ./data/processed_data/without_docstrings_original_function.json.gz
2. vectorised code embedding file [see Subsection 6.3, 3r̂d point] : **./data/code2emb/nodoc_vecs.np**
3. Language Model : ./data/lang_model/lang_model_cpu_v2.torch
4. Language Model vocab : ./data/lang_model/vocab_v2.cls
5. **Optional** if you directly want to use the searcher and already build the index you must have this file: ./data/search/search_index.nmslib

7.2 Searcher Characteristics

- The whole searcher module is CPU extensive :
  - Name: Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz
  - Memory: 252GB
  - Cores: 56
  - Architecture : x86_64, (64 bits)
- Search Index File will be stored in data/search directory as **./data/search/search_index.nmslib**

7.3 To Run The Model

.docstrings files placed in data/processed_data directory are needed to run the model.

Enter the command:

Semantic-Code-Matcher:$ python3 searcher.py -direct

Arguments

1. **-direct[Optional]**: When written module skips building the search index and directly enters the search engine module based on the search index stored previously.

```
Activating the searcher
Search Code: read data into dataframes
WARNING:root:Processing 1 rows
cosine dist:0.8987  url: https://github.com/sk89q/Plumeria/blob/master/orchard/graphviz.py#L30
--------------

def render_dot(graph, format='png'):
    program = 'dot'
    if os.name == 'nt' and not program.endswith('.exe'):
        program += '.exe'
    p = subprocess.Popen([program, '-T' + format], env={'SERVER_NAME':
        'plumeria', 'GV_FILE_PATH': '/dev/null'}, shell=False, stdin=
        subprocess.PIPE, stderr=subprocess.PIPE, stdout=subprocess.PIPE)
    stdout, stderr = p.communicate(input=graph.to_string().encode('utf-8'))
    if p.returncode != 0:
        raise Exception(
            'Received non-zero return code from grapviz\n\nError: {}'.
            format(stderr.decode('utf-8')))
    return stdout

cosine dist:0.9002  url: https://github.com/LeastAuthority/kubetop/blob/master/src/kubetop/_te
--------------

def _render_pod(pod, node_allocable_memory):
    cpu, mem = _pod_stats(pod)
    mem_percent = node_allocable_memory.render_percentage(mem)
    return _render_row(_render_limited_width(pod['metadata']['name'], 46),
        '', _CPU(1000).render_percentage(cpu), mem.render('8.2'), mem_percent)

cosine dist:0.9048  url: https://github.com/LeastAuthority/kubetop/blob/master/src/kubetop/_te
--------------

def _render_pod_top(reactor, data):
    node_info, pod_info = data
    nodes = node_info['info']['items']
    node_usage = node_info['usage']['items']
    pods = pod_info['info']['items']
```

Figure 4: Basic Inline Search Engine