HW5: Learning From Programs

> Teaching Assistant: li_wei@sjtu.edu.cn (李威)

The purpose of the homework is to let everyone have a better understanding of the basic procedure of source code learning, and the procedures include::

The first programming task is to learn to use Abstract Syntax Tree (AST) parser to extract code features. The second programming task is to implement a simple code search model. The TA has provided **the extracted code features** for task 2. Therefore, there is no dependency in between the two tasks, and it doesn't matter which one you start with.

Task1: Extract Code Features from AST

1.1 Setup

- We use a popular AST parser, **tree-sitter**. In this task we learn from the Java programming language. Tree-sitter provides an playground for visualizing AST. For example, for this code

```
class MyClass {
  public void readText(String file) {
  BufferedReader br = new BufferedReader(new FileInputStream(file));
  String line = null;
  while ((line = br.readLine())!= null) {
    System.out.println(line);
  }
  br.close();
  }
}
```

its AST is as follows:

```java
public class MyClass {
  public void readText(String file) {
  BufferedReader br = new BufferedReader(new FileInputStream(file));
  String line = null;
  while ((line = br.readLine())!= null) {
    System.out.println(line);
  }
  br.close();
  }
}
```

Tree 0.1 ms

```
program [0, 0] - [10, 0]
  class_declaration [0, 0] - [9, 1]
    modifiers [0, 0] - [0, 6]
    name: identifier [0, 13] - [0, 20]
    body: class_body [0, 21] - [9, 1]
      method_declaration [1, 2] - [8, 3]
        modifiers [1, 2] - [1, 8]
        type: void_type [1, 9] - [1, 13]
        name: identifier [1, 14] - [1, 22]
        parameters: formal_parameters [1, 22] - [1, 35]
          formal_parameter [1, 23] - [1, 34]
            type: type_identifier [1, 23] - [1, 29]
            name: identifier [1, 30] - [1, 34]
        body: block [1, 36] - [8, 3]
```

- Config tree-sitter

You can use following commands to get an AST parser:

```
pip3 install tree_sitter
cd ./task1/parser
git clone https://github.com/tree-sitter/tree-sitter-java # you can find tree-sitter-java folder
python3 build.py # you can find my-languages.so
python3 tree_sitter_hello_world.py # Hello Tree Sitter
```

- Config unittest

Teaching assistants use python `unittest`, on the one hand to demonstrate the basic usage of the AST parser, and on the other to evaluate whether your implementation is correct.

```
python -m unittest my_tests.MyTest.test_get_class_name
```

The above example demonstrates that the unittest `test_get_class_name` calls `ClassDeclarationVisitor.get_class_name` to get class name `MyClass`. The code line of `get_class_name` is only 5 ! Isn't it very simple? You must want to have a try. Let's go 😂

1.1 Extract Method Name

- Description: get all method names from a class.
- Example

```
public class MyClass {
    public void readText(String file) {
        System.out.println('Hello World.');
    }
    public static void printName() {
        System.out.println("MyClass");
    }
}
```

Expected output: `['readText', 'printName']`

- Hint code

```
class_body = root.children[0].children[3]
for child in class_body.children:
    ...
```

- Please fill the code in `method_declaration_visitor.py`

1.2 Extract API Sequence - object_creation_expression

- Description: get all constructor invocations from a method.
- Example

```
public class MyClass {
    public void readText(String file) {
        BufferedReader br = new BufferedReader(new FileInputStream(file));
    }
}
```

Expected output: `['BufferedReader', 'FileInputStream']`

- Hint code

```
def get_object_creation(self, code: str):
    for child in class_body.children:
        self._get_object_creation(child)
def _get_object_creation(node)
    # recursion
```

- Please fill the code in `object_creation_visitor.py`

1.3 Extract API Sequence - method_invocation

- Eescription: get all method calls from a method.
- Example

```
public class MyClass {
    public void readText(String file) {
        BufferedReader br = new BufferedReader(new FileInputStream(file));
        String line = null;
        while ((line = br.readLine())!= null) {
            System.out.println(line);
        }
        br.close();
    }
}
```

Expected output: `['BufferedReader.readLine', 'System.out.println', 'BufferedReader.close']`

- Hint

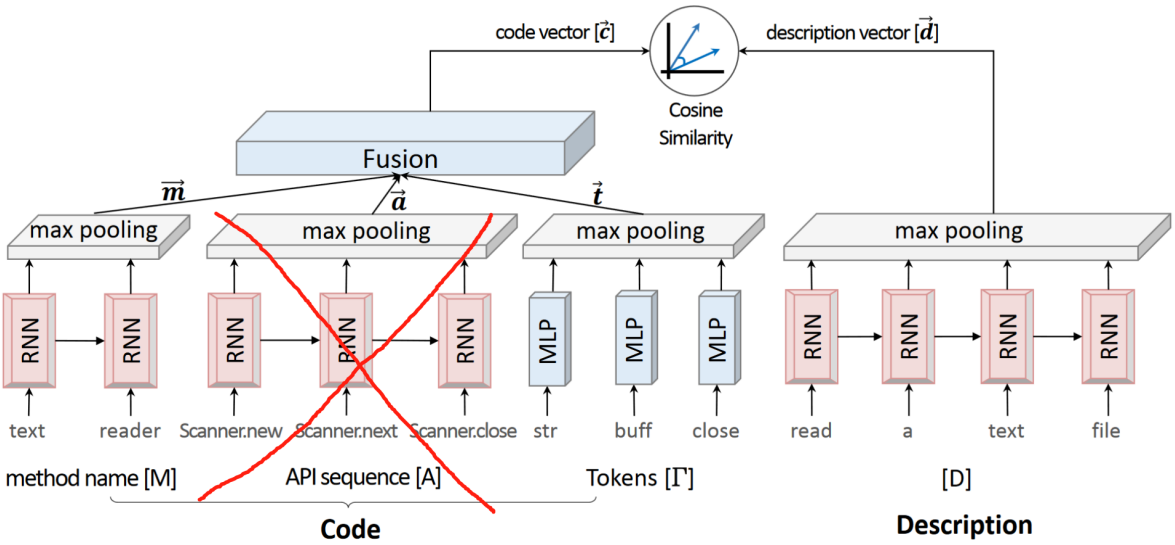
```
# first get all method invocations
# second if the object of a method invocation is Identifier, replace it by Type
```

- Please fill the code in `method_invocation_visitor.py`

!NOTES: TA uses the test cases in `my_tests.py` to benchmark your implementation, so please do not modify it.

Task2:

In this task, you need to implement a code search model. The model architecture is as follows:



- The model architecture is from [1]. To cut the training time, we ignore the API sequence features.

2.1 Setup

- Dependency.

  The example code uses `Pytorch` , ``. Other library used in TA environment can be found in `requirements.txt`
- Code Structures
  - `data/` : store datasets.
  - `main.py` : train and valid the model.
  - `model.py` : **TODO**. You need to implement a neural network model for code/desc representation and similarity measure.
  - `metrics.py` : Four metrics.
  - `data_loader.py` : A PyTorch dataset loader.
  - `utils.py` : utilities for models and training.
  - `configs.py` : configurations. You can change or add new hyper-parameters.

2.2 Notes

- Please report your results in the report. Please submit your logs.txt as evidence that you have completed the model training.
- Please DO NOT upload the datasets and your model checkpoints!

Submission

Please submit a zip file named `[studentID]_[name]_hw5.zip` . It should contain one folder named `[studentID]_[name]_hw5` . This folder should include one `task1` and one `task2` folder, and one report

file in PDF format.

Scoring Criteria

The scoring criteria for the task1 are as follows:

Aspect	Score
task 1.1	10 points
task 1.2	20 points
task 1.3	20 points
code/report is clear	5 points

The scoring criteria for the task2 are as follows:

Aspect	Score
model	15 points
complete training and validation process	15 points
result	10 points
code/report is clear	5 points

Reference

[1] X. Gu, H. Zhang, and S. Kim, “Deep code search,” in 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018, pp. 933–944.