CMPT470-816AssignmentMachineLearning

This Repository contains the code for the machine learning assignment of CMPT 470/816.

Getting Started

The assignment will give you a basic understanding of how deep neural networks can be used for source code. Here I use the source code as a feature and feed them into the neural network.

By this assignment, you will solve the following Software engineering problem:

```
 Method Call Recommendation using Reccurent Neural Network(RNN) and
Long Short Term Memory(LSTM) Based neural network.
```

Let me give an example of the problem. Consider the follwoing code snippet:

```
 1. public String toString(){
2.   int baseLength = base.length();
3.   StringBuilder builder = new StringBuilder(baseLength);
4.   for (int i = baseLength - 1; i >= 0; --i){
5.       builder.append(base.charAt(i));
6.   }
7.   return builder.toString();
8. }
```

The `toString()` method tries to create a string of a `CharSequence` object `base` taking each character at a time using a `for` loop and appending the character in a `StringBuilder` object, `builder`. Consider, the user is writing the line 7 where (s)he wrote `return builder.` and expecting to have a method call recommendation. We like to recommend a method name such as `toString` to the user. Therefore our `label` for the problem is a method name. So what would be the feature? The feature is the previous source code. In different researches, the last four lines of code are found very important. Therefore, we collect all code tokens of the last four lines. In our example, last four lines of `return builder.` are:

```
 3.   StringBuilder builder = new StringBuilder(baseLength);
4.   for (int i = baseLength - 1; i >= 0; --i){
5.       builder.append(base.charAt(i));
6.   }
```

One way to collect the code tokens is using regular expression. However, in that case tokens such as `i`, `baseLength` will be collected. If you look at the code, they are identifier and in our dataset, such identifier can have different name. For example, a developer can write `baseLength`, `baseLen`, `baseSize` and so on as the name of the identifier and if we try to train neural network keeping these names, we might end up having a million of the distinct word in our training vocabulary. So why not we replace these tokens by their data type? To do that, we need to parse the code and we can do that using Eclipse JDT Core. The library can create

Abstract Syntax Tree(AST) from java code. Each node will contain either operand such as `i`, `baseLength` or an operation such as `=`, `for` and so on. Each node will have multiple information embedded in it. We will extract the type information of the operand nodes and replace the operand nodes values by the type information. Then we will visit the AST from line 7 to line 3 to create the feature of our task. Since we have the code tokens as a feature, let us call them context. Therefore the context for the example is:

```
 StringBuiler, return, }, int, charAt, CharSequence, append,StringBuilder,
{, --, >=, -, int, =, int, for, int, StringBuilder, new, =, StringBuilder
```

The above process is done by the Java code in this repository. It takes each java file of a subject system, parses them to AST, finds each method call, collects all tokens in the previous four lines for each method call, and forms context from those tokens and label from the method call name.

In data/repository folder, you will see one subject system,**JEdit**. Our java program will collect the dataset for machine learning from this subject system. If you like to play with more, just put more subject system in the repository folder.

After the successful run of the java code, you will see the dataset atdata/dataset folder.

Now you have a dataset, you need to do some machine learning task which includes data collection, preprocessing, machine learning configuration, training, and testing.

In scripts folder you will find the python scripts for a machine learning algorithm. It collects dataset, splits train and test files, converts/encodes both the context and label into the numerical vectors, configure the machine learning algorithm, feed the training data to create a model and test the model.

While doing so, we need to create a vocabulary class to store all the distinct tokens from context and label.

It is a multi-class classification task which means the number of class/ distinct labels in more than two. So how many classes do we have? The answer is simple, the length of the label vocabulary. Therefore while testing the algorithm gives the probability value for each class and the class having the highest probability is chosen as the predicted label.

In the machine learning code, you will notice, we do encoding and padding. What are they?

Since the neural network does not accept string value so we need to encode the context and value. We create the vocabularies for context and label. Let us consider the vocabularies look like:

```
 context_vocabulary = {for : 1, int : 2, String : 3, return : 4, ......}
lable_vocabulary = {toString : 1, append : 2, .......}
```

In the vocabulary, each token is associated with an id. For example the id of `for` is 1, `int` is 2 in context vocabulary. Let us have a context such as `{String, int, return}` and the label as `{append}`. We encode the context and label based on the id in the vocabularies. Therefore encoded context will be `{3,2,4}` and encoded label as `{2}`. One more problem is, the context can be of any size whereas we need a fixed size of the input to the neural network. So we select a maximum length of input and cut off the long context or pad short one. For example, we choose the length should be 5. Therefore, the padded version of the encoding context should look like `{3,2,4,0,0}` where 0 is chosen as the padded token. Now, we feed the neural network with the padded context and we are expecting that the network should give the highest probability score in the second node of the output layer.

Lastly, in the vocabulary, we save the frequency of each token. Its because in the real world the vocabulary can be huge. So we need to cut off some words to maintain good accuracy. One simple and most accepted way is to remove the words from the vocabulary that appear less than a threshold value. Now if any context or label has one of those removed token, we put the id of `UNK` or Unknown token in the encoded version.

There are three parts of this assignment:

1. Run the existing code: The Java code in the repository collects the dataset from any number of java projects. Then you use the dataset and run the python written machine learning code situated at scripts folder. It will train a recurrent neural network based model, evaluate the top-1 recommendation and return the accuracy value. Neural Network is developed using Keras library.

2. Train and test with the LSTM model: In this task, you need to change the machine learning code in such a way that it will train an LSTM model and test using the same. To do that, you need to first understand the code, locate where the change is required (concept location), determine the lines of code that are going to be affected due to the change (effect analysis), and update the code accordingly.

3. Write the machine learning code using DeepLearning4j: Now you need to write the whole machine learning code using the DeepLearning4j library in Java. DeepLearning4J is a java based library maintained by Eclipse foundation used primarily for deep learning tasks. The architecture of the library is very similar to Keras. Therefore the coding style will be similar.

Prerequisites

Before running the code, you must have the following packages installed on your computer:

```
 java 8
maven
python 3.x
pip
```

TASK 1

A step by step instruction is given in the following

- Step 1: Use any editor to import the java project. I will recommendIntelliJ editor. To import the project, click File-> Open. In the pop up dialogue, choose the pom.xml file and click open. Another pop up window will appear and choose open as project option. IntelliJ will open the project as maven project and install all the dependencies automatically.

- Step 2:

```
Open src/main/java/config/Config.java
change the following global variable
public static final String LOG_PATH = ROOT_PATH+"cms500_codeparser.log";
```

Put your NSID in the logfile name. Otherwise, you will not get any marks.

- Step 3: If any .log file exists, delete it.
- Step 4:

  ```
  Open src/main/java/extraction/MethodCallCollector.java
  Run the main function
  ```

  If the main method is run properly, you will see 10 CSV files is written in "./data/dataset/" folder. Moreover **your_nsid_codeparser.log** will be created. It will store all log statements regarding code extraction.

- Step 5: Now, you need to run the machine learning code written in python. To do that go toScripts folder and open the README.MD file. All instructions of running the machine learning code are given there. Follow them and you will able to run the machine learning code. After the successful run, you will be done with the Task 1.

TASK 2

Task 2 is more like building an LSTM based neural network instead of RNN. Before doing any change in the machine learning part, update the following global variable:

```
log_file_path = root_folder+'your_nsid_keras_rnn.log'
to
log_file_path = root_folder+'your_nsid_keras_lstm.log'
```

Next, you can follow the following steps to complete the task:

- Understand the code: First, you need to understand how the current machine learning code is working. I tried to write the code as readable as I can. Moreover, the code is also checked by some experts and revised. So I hope, you will not find any difficulty to understand it.
- Concept location: Next, try to find the location of the code where the change is required to fulfill the task. You might need to do some online search about how LSTM is implemented through Keras. Since Keras is a very popular framework in the machine learning community, you will find a lot of discussion, blogs, tutorial and so on. Then detect the area where you need to change.
- Impact analysis: Before changing the code, try to find the code where the change will be impacted. The impacts are but not limited to parameters, data types of the contexts and labels, any functions and so on. Hint: think simple and straight.
- Update: Finally, update the code and run it. If you find any error solve it because once you are able to run

the RNN code, I am not going to touch your code. It is your responsibility to make the code runnable.

After the successful run, you will find**your_nsid_keras_lstm.log** file in the project directory. It will keep all the log statements along with the accuracy of the model.

TASK 3

This is a bit tricky because you need to learn how DeepLearning4J works. However, the working mechanism of Keras and deeplearning4j are almost the same.

However, if you understand the machine learning code, you will find the following five sections there: 1. Data Collection 2. Data Prepossessing 3. Neural Network Configuration 4. Training 5. Testing

DeepLearning4j will come to the play from third. Before that, you need to do the data collection and prepossessing. You should do these two steps by yourself.

For neural network configuration, training and testing you can look at the following code examples:

RNN Example using DeepLearning4j

Another Example

DeepLearning4j is created based on the concept of Keras library. Therefore, you just need to relate the code I provided for Keras and the above code examples.

First, try for yourself. If you fail then come to me. I will not help you if you don't anything.

Hint: In the pom.xml file, I included all the libraries required for the machine learning code in Java. Therefore, you can use the current project for writing machine learning code or you can create a new maven project.

One last thing: You need to create a log file for the machine learning task. You can see how I have done logging while parsing the code and follow that. But your completion of Task 3 will depends on whether you provide a log file or not. Therefore, after a successful run of your machine learning code, **your_nsid_deeplearning4j_rnn/lstm.log** file needs to be generated.

What to hand in

1. your_nsid_codeparser.log
2. For each of the task provide the following three files:
    - Zip file of the full project
    - your_nsid_keras/deeplearning4j_rnn/lstm.log
3. Lastly one file name as README.txt file that will have the followings:
    - Status of all three tasks.
    - How to run each of your project(s): requirements, steps to run
    - Changes in python script for completing task 2.
    - Overall what you learn from the assignment out of 10 in three categories: 1. Usability(10 as in very

useful, 0 as in not required for future), 2. Complexity(10 as in very hard, 0 as in very easy), 3. New learning(10 as in everything were new, 0 as in nothing learned)
  - Any additional comment. Your assessment will not be considered for marking and will be helpful for the future.

Marking

1. Dataset Creation: 5 (will be considered as complete if your_nsid_codeparser.log file is provided with all log statements)
2. Task 1: 10 (will be considered as complete if your_nsid_keras_rnn.log file and the zip file of the project are provided)
3. Task 2: 20 (will be considered as complete if your_nsid_keras_lstm.log file, the zip file of the project and the README.txt file are provided. For task 2, the main important line in README.txt is "4. Changes in python script for completing task 2.")
4. Task 3: 40 (will be considered as complete if your_nsid_deeplearning4j_rnn/lstm.log file and the zip file of the project are provided)
5. README.txt: 5 (will be considered as complete if the descriptions of first two points are provided and Task 1 is complete. Even if you write you cant finish Task 2 or 3, you will get full marks for README.txt file)

Total Marks: 80

Issues

If you have any problem with the code provided in the repository. Please create an issue. I will try to solve it as soon as possible.

Authors

- **C M Khaled Saifullah** - *Initial work* - khaledkucse

See also the list of contributors who participated in this project.

License