University of New York in Tirana
Fall 2020

Handed by: Ledjo Pilua

Advanced Java
COURSE ASSIGNMENT

Notes and Documentation [Report]

In this assignment, as per instructions requirements, I wrote a program which constructs a language model for languages: Albanian, English, Spanish, French, German and Italian. That said, this program, constructed from text files located in dedicated folders in "LocalFolder" classifies a txt, that is Mystery.txt file, and selects the language whose model is closer to the text in Mystery.txt file. To quantify the similarity of texts to each other, as required, I used cosine similarity and it is then displayed.

CONCURRENCY

To meet this requirement, I have created four classes: MyHashTable, ThreadPool, ModelFile, and ModelLanguage.

- My**HashTable**
  MyHashTable extends HashTable. I chose HashTable because it is thread-safe, and the average time complexity put/get is O(1). I had to create MyHashTable because HashTable ensures that there is only one writer writing at a time. We need to read before writing, and we have to ensure that just one Thread at a time can do this. e.g If we have a model that contains the bigram "rd" and two Threads that find "rd", both will try to see if it already exists on the model (read), if it does then they will increase the value associated with "rd" by one. The problem is that if they read at the same time, they will write the same value and we are going to lose one unit. I solved this problem by

creating a synchronized method on the java class MyHashTable that does exactly that (read first, write after). Each MyHashTable object represents a model.

➢ **ThreadPool**

This class is intended to run multiple threads (pool) and make the calling thread wait until all the started threads (pool) finish their tasks. It does the following:

- Start each thread
- Wait until all threads finish
- Disable new tasks from being submitted
- Wait a while for existing tasks to terminate
- Cancel currently executing tasks
- Re/Cancel if current thread also interrupted
- Preserve interrupt status

➢ **ModelFile**

This class represents each text file and extends Thread. It has a reference to a model (MyHashTable). It will update this model as it reads the text file. This class is responsible for updating the model, reading one text file, getting each word, and extracting the n-gram of each word.

➢ **ModelLanguage**

This class represents each folder or language, and extends Thread. It has a model (MyHashTable) object. It will create a pool of ModelFiles, one ModelFile for each ".txt", once created all the ModelFiles it will execute the pool and wait until they finish. All of this using ThreadPool. Each ModelFile is given the reference to the MyHashTable. In this manner, they will update the same model.
Once the pool finishes, the model of this language is ready. The next step will be to compare this model with the Mystery model. For that, it needs to verify if the Mystery model is completed. If it is still calculating the model then we will have to wait until it finishes. When the Mystery model and the language model are ready, we can calculate the Document Distance. *Each ModelLanguage is in charge of calculating the document distance of the language that it represents.*

Execution Flow

This program has a total of six Java classes.
The four mentioned before and the Java classes *Main* and *FileManager*.

- **Main**
  Receives and validates the arguments. Creates the FileManager object and starts its thread.
- **FileManager**
  Validates that the local folder and mystery file exist. It also creates a pool of ModelLanguages and executes them with ThreadPool.

  Finally, to wrap things up, the main thread creates a ModelFile for the mystery file and starts it, then it creates one thread per folder and waits until they finish. Each folder will create one thread per text file and wait until they complete the model. When the pool of threads destined to construct the model of a language finishes, the folder thread will calculate the document distance. When the pool of folders finishes, the main thread will request the lowest distance among all the languages, and that will be our answer.

  Particularities
  In this project, 4 bytes are used to represent the norm and the Dot product, and 8 bytes to represent the cosine similarity. If the model of a language is fed with tens of thousands of lines, the numbers will be too large to fit into 8 or 4 bytes. The maximum value that norm and Dot product can take is: 2147483647

CONCLUSION

I had a lot of fun writing this program and doing the necessary research regarding NLP and it immensely helped me better understand and reinforce concepts discussed in lectures. It also added more nuance to my current research interests regarding NPL as I am closely following the Sentiment Analysis topics.

I am also attaching to the zip file the executable jar file named LedjoPiluaNLP.jar and here is screenshot of me running it from the command line for default n value which is 2 and then for n=3

```
C:\Users\ledjo>java -jar  "C:\Users\ledjo\Desktop\LedjoPiluaNLP.jar" "D:\eclipse\LedjoPiluaAssignment\LocalFolder"
al angle: 61.972194
fr angle: 70.779774
it angle: 66.083804
es angle: 67.287953
en angle: 50.624502
ge angle: 73.315583
Nearest: en angle: 50.624502

C:\Users\ledjo>java -jar  "C:\Users\ledjo\Desktop\LedjoPiluaNLP.jar" "D:\eclipse\LedjoPiluaAssignment\LocalFolder" 3
al angle: 84.744339
it angle: 84.093810
en angle: 67.045501
es angle: 83.534784
fr angle: 85.744742
ge angle: 87.518317
Nearest: en angle: 67.045501
```