Brynne Wilkinson Methods in Computational Linguistics II
Spring 2021

Homework 2 Write Up

Overall, this assignment was challenging for me and there were a number of errors and adjustments I had to make to my code in order for it to run with success.

I began by reviewing the 'argparse' document, to better understand what its function was and how to incorporate it into the code. I reviewed lecture and practicum as well, before beginning. I modeled the block of code under "if __name__ == "__main__":" after the practicum exercise in the basic.py file. I then incorporated the generator function provided by Kyle, but struggled with understanding how and where to incorporate the "corpus = list(read_tags(conll2000.txt))" line of code. I placed it above main, and under the generator function. Beginning to try to divide the data into the three outputs, I tried to slice the data. I spent quite a while reviewing solutions on stackoverflow on how to slice a list based on percentages. I defined the list I was looking to slice as "args.input" using the len() function to determine the length of the input file. Next, I defined an index that would approximate 80% of a list under a variable. I then made a 90% index but did not store it under a variable, instead placing the line of code representing the index in its respective index location.

I ran the code to check for any errors up to this point. The first error message I received was for making a positional argument required. I then reviewed once again the difference between positional and optional arguments for argparse and removed "required=True" from all my arguments. However, I quickly was given another error message, this time for having indices that were floats, so I used the int() function around (len(args.inpput) * .81) to force the indices to be integers.

At this point I believed I had been successful in splitting the data into their respective amounts but was unable to proceed further as I could not figure out writing paths to the output files. After unsuccessful google attempts and enough frustration I reached out to Kyle for direction on how to proceed or for any helpful resources in understanding the next step. Kyle pointed out a few flaws in my code, and I made the following revisions based on his feedback.

I moved the line "corpus = list(read_tags(conll2000.tag))" to be under the main function, so the program could act on it. I also replaced "conll2000.tag" with "args.input" per Kyle's suggestion, so that the path was no longer hardcoded. I also changed "len(args.input)" to "len(corpus)" so as to obtain the length of the file and not the length of the name of the file.

My next objective being to write the paths to the files, I reviewed a video by Corey Schafer on youtube.com focusing on writing files, to clear up my understanding of the line "with open(args.output_path, "w") as sink:" from the practicum exercise and basic.py file. Modeled after this line, I wrote "with open(args.argument, "w") as sink:" for each output argument and, using a for loop, iterated through the variables of the sliced data as the material to be written into the file. As in practicum, I used a print() statement to write in the file.

Once again, I paused to complete a test run of the file. However, a new issue arose as I received an error message that "'args.input' file not found". Based on the file paths created for the output arguments, I tried to write a path to the 'args.input' file. With corpus placed above that statement, the "'args.input' file not found" error remained. However, I could not place corpus below that statement, because I called corpus to be used to write into the file, and that led to an error of calling a variable before it was defined. I then tried to write the path to the file and read it instead and have "pass" following it, but that did not resolve the issue. I proceeded to comment

out both the input path line, and the corpus line to identify which line was causing the error. Through this, I determined it was an issue with “corpus = list(read_tags(‘args.input’))”. After a bit of frustration, I tried removing the quotation marks around “args.input” which resolved the issue completely. I then removed the input path line of code as it was unnecessary.

After this, I tested the code with both tests provided by Kyle, and found the code to run successfully with the caveat that the formatting of the new files did not match the formatting of the initial document. However, I decided to proceed with steps 2 and 3 of the assignment and return to the formatting issue later.

I found steps 2 and 3 to be easier to complete and problem solve, and I modeled my code on the practicum exercise contained in the file numbers.py. I shuffled the corpus using “random.shuffle(corpus)” and stored it in a variable. I realize now in review, however, that storing it in a variable was likely not necessary. I tested whether or not this step was successful by running the code and looking at the files to see if the order matched the original document. After observing that the sentences were out of order, I moved onto step 3.

I added the “--seed” argument and set “required=True”. I also added the line “random.seed()” under the definition of corpus, so the seeding happened before the other commands. When I ran the seeding tests, everything seemed to be successful until I realized that when using the same seed, the SHA-256 checksum values of the files did not match when using the shasum command-line tool. After reviewing the ‘random’ docs, I realized this issue was because I did not specify that the seed was to be what the user inputted and thus specified “random.seed(args.seed)”. After that all tests ran successfully.

However, I had to loop back and resolve the formatting issue. I walked myself through the format of the original document, and what the current format of the output files were so that I could determine what needed to happen to correct the format. I decided to only work with one variable and file at a time. I tried to create a new variable of the sliced data, changing the original first from a list to a string then back to a list, replacing the new list variable in the paths block of code. I was aware this was WET and roundabout, but I was unclear of how else to proceed. I then ran the code to test it, but rather than write the data to the file, the data printed in the terminal. I promptly deleted the new variables.

Instead, I turned and considered that a better solution was probably to iterate through the lists under the paths block of code. I realized a second for statement was necessary as I was working with lists of lists, and that would allow me to iterate through the next level. I googled and used a solution from geeksforgeeks.org for turning a list into a string. Thus, I iterated through the levels of sliced data and in turning the final lists into strings matched the formatting of the original document. At this point, I was really proud and impressed that the program ran as well as it did.

My last act was to clean up the code. I added a second variable to represent the 90% index as well as comments to clarify what each piece or block of code did.

Ultimately, this assignment was quite difficult for me and challenged me in several ways. However, as pieces of the code began to work, I gained a better understanding of how to proceed and how the code functioned. I still think, were I to repeat this exercise from scratch, I would find it difficult. I want to practice it more to build a better understanding of argparse and to gain more confidence in my coding abilities. In addition, I am very aware that my code is not as concise or DRY as it could be, and I would like to write code that is less WET in the future.