**Monday 1/6/2020**

Today, I was stuck wondering how in the world to deal with the following error:

---------------------------------------------------------------------------

UnpicklingError Traceback (most recent call last)

<ipython-input-4-7ffa7b3cfeba> in <module>()

4 with open('authorization.txt', 'rb') as input: # so often though

5 print(input.read(1))

----> 6 drive = pickle.loads(input.read())

UnpicklingError: invalid load key, '!'.

What was going on here was that for some reason, as the Google Colab code tried to grab and load the “auth.pkl” file in order to establish a connection to Google Drive (recall that my current research goal is to connect Google Colab itself to my Google Drive independent of the password. This is so that when Director runs this Google Colab file, password-filling will not be a problem), something was inherently wrong with loading this pickle file. I relooked at the code cell to see if there were any typos / trivial errors I could fix. Nope, that was not the case. I had planned on re-uploading the auth.pkl file, but it turned out my partner had tried that already, and that did not work out.

I decided to investigate this error online to see if I could get any hints as to how to resolve this error. By that point, however, the research period ended.

**Wednesday (Snow Day) 1/8/2020 - Thursday 1/9/2020**

My partner and I discussed the above error, and we concluded that we could use a different way (i.e. a different library) that can extract the object that the “auth.pkl” file stores. We also noticed something interesting: the input.read() method that was giving us the above error actually works out if we give a number as a parameter. For example, input.read(40000) takes in the first 40,000 “characters of data” (maybe they are bytes/representation of characters indicated by the ‘b’ in front of the string; I will refer to the following as a “byte string” for the rest of this journal report): “b'<!DOCTYPE html><html><head><meta name="google" content="notranslate">...”. This interestingly looked like Google Drive metadata from the keywords “html” and “meta name.” We thought that the reason the metadata from Google existed in the first place was that we had uploaded the file onto Google Drive, and then Google Drive automatically puts on metadata tags on every file we upload. When we played around with different numbers with input.read(#), we interestingly noted that we received different types of errors. On the “40000” case, the error was an invalid load key, similar to the above error. With different numbers though, we received an error something on the order of “MARK not found”, which indicated that the reader did not find a beginning mark to read from; we concluded that from Stack Overflow searches.

My partner suggested we tried using a .JSON file instead of a .pkl file since .JSON serves the same purpose of storing objects from Python. He introduced me to a link (<https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/>) in which he was reading about .JSON, and I read it for myself to get some sense as to whether .JSON files for our application were feasible, and if so, what methods we should use to accomplish that.

From an initial reading of the link, I was not too sure about it, since the example used was a nested dictionary as the object stored in the .JSON file and I was not too sure what data type our “auth.pkl” file held. I refreshed my memory on how our Pydrive script (i.e. iogoogle.py) worked. I realized the data type was an instance of a class found in the “PyDrive” package needed to authenticate a connection from the code to one’s Google Drive. Once we tried it though, we received an error of “Google Drive object not json serializable.” What a bummer.

I then thought, “Why not copy/paste the same code, create an auth.pkl file organically inside of the Colab file, upload that auth.pkl file to Google Drive, and see if that auth.pkl works?” However, this approach worked for neither of us.

We then thought of another approach. Remember the b’...’ string I mentioned either? That was vital because we realized that we could get rid of that metadata in the byte string of the .pkl file. We analyzed the entire string together and played around with numbers until we got to 10377 and -238 (you will see what I mean by that in the code below). We essentially played around with the index slicing of strings (i.e.

with open('fixed.pkl', 'rb') as input:

s = input.read()

s = s[10377: -238] ← that is the important part

drive = pickle.loads(s)

); however, that approach did not work either.

In our final approach, we were able to finally resolve the problem. At the time, we thought, “Well, what if we use a byte string read from the local version of the auth.pkl file instead of the one uploaded to Google Drive?” We looked at the byte string So, we did just that, and it worked out. This was the final code that worked:

s = b'\x80\x03cpydrive.drive\nGoogleDrive\nq\x00)...

drive = pickle.loads(s)