Clinical Protocol Chatbot

(Pre-processing for training data)

Table of Contents

1. Overview

The objective of this project is to further pre-process the training data used for a Clinical Protocol Chatbot to retrain the model. Currently, the information in the raw PDFs on Clinical Protocol is extracted by Adobe API into a JSON file output. However, only plain textual information is extracted and hence, the textual information in figures and tables are just extracted and outputted without interpretation. As such, we will pre-process textual information from figures and tables to obtain a textual summary of them before re-inserting it back to the JSON file output. This way, the scripts will make sense of the texts from figures and tables before outputting it in the JSON file output.

The next steps will be to proceed with text chunking using the JSON file output and then to generate QA pairs from the text chunks.

This document will guide you through the generation of image summaries, text chunking, and the generation of QA pairs.

2. Getting started

2.1. Installation

Ensure that you have installed VSC locally by following the instructions in the links below:

- https://code.visualstudio.com/download

2.2. Prerequisites

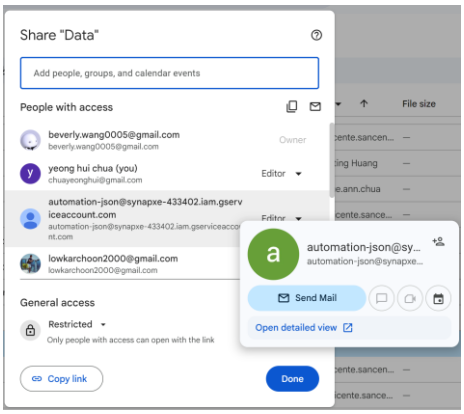
After the installation, make sure that you are using the correct python for this project and have set up a virtual environment by following the steps below:

1. Clone the remote repository into your local directory
    a. Open VSC
    b. Change directory to where you would like to clone the remote repository
    c. Go to github repository
    d. Copy the html link from github repository
    e. Paste the html link copied into the terminal in this format:
    Git clone <html link>
    f. Change directory to the cloned project folder
2. If you have more than 1 python versions installed locally, check python version
    a. Window+R
    b. Type “sysdm.cpl”
    c. Click on “Advanced”
    d. Click on “Edit” under “System variables”
    e. Make sure the path of the python version 3.11.0 is placed before all other versions
    f. Then save the changes
    g. Close ALL VSC windows and restart the application
    h. Double check the current python version by typing “python –version” into command prompt
    i. List the paths of python versions installed by typing “where python” into command prompt
3. Set up virtual environment
    a. Make sure you are using the python version 3.11.0 locally before you start creating virtual env
    b. Click on any existing .py files in your VSC project folder.
    c. Ctrl + Shift + P to open palette
    d. Then click on “Create virtual environment” → Choose python interpreter.
    e. Wait for the virtual environment to get created
    f. Note that once this is completed successfully, it would also mean that requirements.txt had been ran successfully too
    g. To check what is installed so far in the environment: pip list
    h. To rerun requirements.txt if there are any changes: pip install -r requirements.txt

3. Connection to Database

For this project, the scripts will interact with Google Drive to upload the output files accordingly. Create a service account with google to obtain a JSON key file to the API service by following the steps below:

1. Set up the Google API by following what is documented in this article: https://medium.com/@matheodaly.md/using-google-drive-api-with-python-and-a-service-account-d6ae1f6456c2
2. Once you have obtained the JSON key file, add the JSON file to your project folder
3. Share your project's main google drive with your google service account url and give editor access



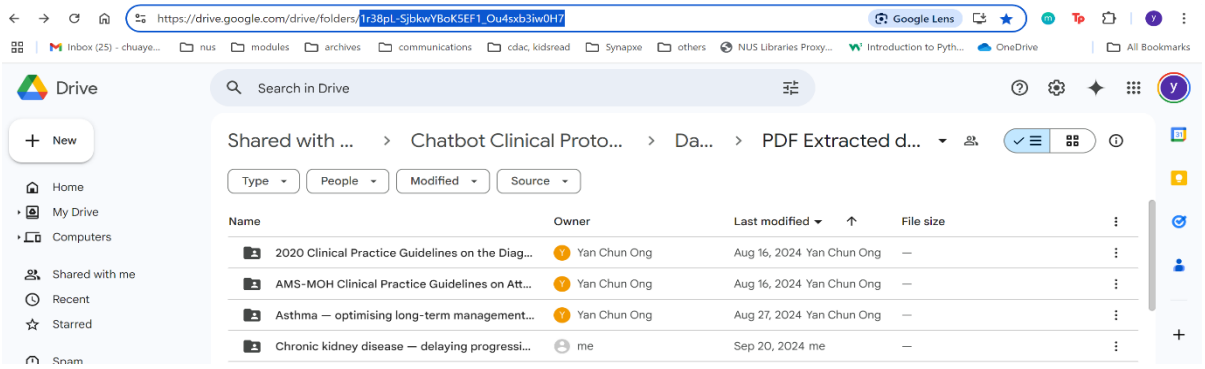
4. In each script that require a connection to the Google Client, replace this masked portion of the script with your path to the JSON key file:

```
# Authenticate and build the google Drive API client
SCOPES = ['https://www.googleapis.com/auth/drive']
SERVICE_ACCOUNT_FILE =
creds = service_account.Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE, scopes=SCOPES)
service = build('drive', 'v3', credentials=creds)
```

5. Under the "main" method, check that the folder id corresponds to the one you are interacting with

```
# Main application
def main():

    # Replace with the folderid of the folder you are interacting with
    root_folder_id = '1r38pL-SjbkwYBoK5EF1_Ou4sxb3iw0H7'
```



4. Methodology

4.1 Add image summaries

image_summaries.py

Firstly, we will run the script called **image_summaries.py** to generate the textual summaries of figures and tables for each and output them to a xlsx file called **combined_{PDF_name}.xlsx**.

Steps to run the script:

1. cd to the folder containing the script called add_image_summaries
2. ctrl+f for "Requires user specific inputs" and edit the portion stated in the docstring.
3. In the terminal, run "python image_summaries.py"

Once the xlsx file is obtained for each PDF, we will require human review of the generated textual summaries. It is recommended to have at least 2 reviewers to review the generated textual summaries for completeness and accuracy. The reviewers can edit directly on the xlsx file.

edit_json.py

After the review is completed, we will run another script called **edit_json** to find and replace the old text content of figures/tables in the JSON file output with the new textual summaries. The original JSON file by Adobe API will be duplicated, and the changes will be made reflected in the new JSON file called **structuredData_edited.json**.

Steps to run the script:

4. cd to the folder containing the script called add_image_summaries
5. ctrl+f for "Requires user specific inputs" and edit the portion stated in the docstring.
6. In the terminal, run "python edit_json.py"

Once the JSON file is outputted, we can move on to text chunking based on the new JSON file.

4.2 Text Chunking

The full list of Clinical Protocol PDFs can be categorised into 4 main categories:

1. PDFs with numbered sections
2. By HealtierSG organisation
3. By ACG organisation
4. By ACE organisation

There are 4 scripts available for text chunking based on the corresponding categories above:

1. text_chunking_numbering.py
2. text_chunking_healtiersg.py
3. text_chunking_acg.py
4. text_chunking_ace.py

Steps to perform text chunking:

1. For each PDF, identify which category it belongs to.
2. Identify the corresponding script by looking at the naming of the script.
3. cd to the folder containing the script called text_chunking.
4. ctrl+f for "Requires user specific inputs" and edit the portions stated in the docstring.
5. In the terminal, run "python {script name}.py"

Each run of the script generates text chunks for one PDF only. Hence, to process for all PDFs, we need to run the script multiples times. We cannot bulk generate the text chunks because the structure varies too much among PDFs. The output will be in a xlsx file.

Once the text chunks are generated for all PDFs, we need at least 2 reviewers to review all the xlsx files containing the text chunks. More specifically, the review should check whether the extraction of section headers is correct. In addition, we should also check the if the text chunks for each section headers are complete and accurate. Reviewers can edit directly in the xlsx file.

4.3 QA Pairs Generation

This is the final script documented, and it is for exploratory purposes on the type of LLM to use for the generation of QA pairs. In script **generate_qa_gemini.py**, Gemini LLM is used to generate the QA pairs for each text chunk in a PDF. The same LLM is used to evaluate the faithfulness of the QA pairs. The script will take in a xlsx file containing the text chunks and output a xlsx file containing the generated QA pairs and their corresponding faithfulness scores.

Steps to run the script:

1. cd to the folder containing the script called generate_qa
2. ctrl+f for "Requires user specific inputs" and edit the portion stated in the docstring.
3. Upload the xlsx file containing the text chunks of the PDF into the same folder containing the script
4. In the terminal, run "python generate_qa_gemini.py"

For using Gemini to run the LLM for QA generation, I had to adapt and edit the custom faithfulness class under the script: _faithfulness_custom.py. As such, I have also attached the edited script in resources.

For now, each run of the script generates QA pairs with faithfulness scores for one PDF only. Hence, to process for all PDFs, we need to run the script multiples times. Bulk processing can be implemented in the future. In addition, the quality of the QA pairs should also be evaluated.

5. Resources

5.1 Github

The relevant scripts documented in this word document, requirements.txt, and this word document have been uploaded to a public Github repository here:

https://github.com/c0desnippet/clinical_protocol_pre_processing_synapxe

5.2 Google Drive

In addition to Github, the scripts documented in this word document, requirements.txt, and this word document have been uploaded to the main project folder in Google Drive under the folder called “Preprocessing_by_Yeong_Hui”:

https://drive.google.com/drive/folders/1K37wWTFcZ1N85WW7rA4xhidTwiHnb_jn