PROJECT 2 README

NAMES AND UNI

Matthew Labasan, mjl2278

Phoebe Tang, ft2619

FILES SUBMITTED

1. proj2.tar.gz
    a. /proj2
        - project2.py
        - gemini_helper_6111.py
        - spacy_help_functions.py
        - requirements.txt
2. transcript_spanbert.txt
3. transcript_gemini.txt
4. Project 2 README.pdf

GETTING STARTED

Prerequisites

1. *Python 3.10.1 or above*
2. *Google Custom Search API Key and Google Search Engine Key*
3. *Gemini API Key*

Installation

Creating VM (credit from class website and project description)

1. Click “CREATE INSTANCE” to create your instance
2. Type any name (for example, “cs6111-instance”)
3. Change “Region” to “us-east1 (South Carolina)”
4. Change “Zone” to any “us-east1-x” (the last letter doesn’t matter) or, if you are far from New York right now, to a zone near you
5. In the "Machine configuration" tile set "Series" to "N1" and "Machine type" to "**n1-standard-4 (4 vCPU, 15 GB memory)**."
6. In the "OS and storage" tile, click on "Change" and then go to "Boot disk"; then, in the new popup menu, select "Ubuntu" as "Operating system" and **Ubuntu 22.04 LTS**

(x86/64, amd64 jammy image built on 2025-02-28) as "Version."; finally, change the boot disk size to 25 GB in the "Size (GB)" field and click the “SELECT” button

7. Now, you should be back on the “Create an Instance” page; scroll all the way down to the bottom of the page, and click the "Create" button; wait until the green checkmark shows up.

In the Google VM, after extracting the .tar.gz file, follow these steps:

1. Install Python 3.10 or newer with

   ```
   sudo apt update
   sudo apt install python3
   sudo apt install python3-venv
   sudo apt install -y python3-pip
   ```

   a. If it installs an older version of Python, make sure to explicitly install Python 3.10 and make 3.10 the default version.

   ```
   sudo add-apt-repository ppa:deadsnakes/ppa
   sudo apt update
   sudo apt install python3.10
   sudo apt install python3.10-venv python3.10-dev
   sudo update-alternatives --install /usr/bin/python3 python3
   /usr/bin/python3.8 1
   sudo update-alternatives --install /usr/bin/python3 python3
   /usr/bin/python3.10 2
   sudo update-alternatives --config python3
   ```

   - Select the index of python 3.10

2. Create a virtual environment

   ```
   python3 -m venv dbproj
   ```

3. Move directory in /home location to /opt/ to avoid memory issues

   ```
   sudo mv ~/dbproj /opt/
   ln -s /opt/dbproj ~/dbproj
   ```

4. Activate the virtual environment

   ```
   source dbproj/bin/activate
   ```

5. Navigate into the file extracted from the .tar.gz file

   ```
   cd proj2
   ```

6. Install necessary libraries

   ```
   pip install -r requirements.txt
   ```

7. Clone the SpanBERT repository and install all necessary requirements

   ```
   git clone https://github.com/Shreyas200188/SpanBERT
   cd SpanBERT
   pip3 install -r requirements.txt
   bash download_finetuned.sh
   ```

8. Remove this file from the SpanBERT repository

   ```
   rm spacy_help_functions.py
   ```

9. Move these files in `/proj2` into `/SpanBERT`

   ```
   cd ..
   mv gemini_helper_6111.py ./SpanBERT
   mv project2.py ./SpanBERT
   mv spacy_help_functions.py ./SpanBERT
   cd SpanBERT
   ```

10. **If any errors occur when installing packages in requirements,** then please manually install dependencies. Listed below for your convenience:

    ```
    pip3 install --upgrade google-api-python-client
    pip3 install beautifulsoup4
    sudo apt-get update
    pip3 install -U pip setuptools wheel
    pip3 install -U spacy
    python3 -m spacy download en_core_web_lg
    pip install -q -U google-generativeai
    ```

Usage

1. Run & replace with your parameters, using a query in quotations:

   ```
   python3 project2.py [-spanbert|-gemini] <google api key> <google engine id> <google gemini api key> <r> <t> <q> <k>
   ```

   a. **[-spanbert|-gemini]** is either **-spanbert** or **-gemini**, to indicate which relation extraction method we are requesting>
   b. **<google api key>** is your Google Custom Search Engine JSON API Key (see above)
   c. **<google engine id>** is your Google Custom Search Engine ID (see above)
   d. **<google gemini api key>** is your Google Gemini API key (see above)

e. **<r>** is an integer between 1 and 4, indicating the relation to extract: 1 is for **Schools_Attended**, 2 is for **Work_For**, 3 is for **Live_In**, and 4 is for **Top_Member_Employees**
f. **<t>** is a real number between 0 and 1, indicating the "extraction confidence threshold," which is the minimum extraction confidence that we request for the tuples in the output; **t** is ignored if we are specifying **-gemini**
g. **<q>** is a "seed query," which is a list of words in double quotes corresponding to a plausible tuple for the relation to extract (e.g., "bill gates microsoft" for relation Work_For)
h. **<k>** is an integer greater than 0, indicating the number of tuples that we request in the output

Example usage: `python3 project2.py -gemini <google api key> <google engine id> <google gemini api key> 1 0.8 “Obama Columbia” 10`

DESCRIPTION OF PROJECT

Internal Design

Our logic is located mainly in the main() method, with a few helper functions for calling SpanBERT and Gemini.

1. Extract the arguments to obtain the type of relation extraction, Google API key, Google Search Engine ID, r, t, q, and k.
2. Set up initial variables and load in SpanBERT if needed.
3. Enter a loop. The program calls the Google Custom Search API to retrieve search results for a given query. This method is defined as `search()`
    a. It uses the build function from `googleapiclient.discovery`
    b. Makes a request to see `cse().list()` with given engine ID
    c. Returns a dictionary of the top 10 query results

    Source Code via Google: (https://github.com/googleapis/google-api-python-client/blob/main/samples/customsearch/main.py)
4. For each URL obtained from the `search()` method not already seen, fetch the website text using `requests` and `BeautifulSoup` and clean/trim text as necessary with string functions and `re`.

5. Use `Spacy` to obtain an object representing the split sentences, tokens, relations, etc. of the text.
6. Utilize the specified relation extraction method to extract relations. This will either call the modified SpanBERT or Gemini helper methods originally provided in the project instructions. See "*Step 3 - Information Extraction Method*" for more information on this and the modifications made on the helper functions.
7. If a URL was already seen, print a message stating it will skip over it.
8. Once all URLs have been processed and if `k` relations were not extracted, find a new query by concatenating relations (sorted depending on relation extraction method) into a single string, and checking if the lowercase version of the string exists in the used set. If not, set this new string as the current query, and mark the new string as used by lowercasing it for standardization, then storing it in a set. Do this for all relations until a new query is found. If a new query could not be found in this step, print a message and exit the loop.
9. Continue loop until extracted relations (`X`) reach a length equal to or greater than `k`.
10. Return the top-k extracted relations (X) if at least k exist; otherwise, return all available extracted relations in sorted order (if necessary).

Notable External Libraries Used

1. `googleapiclient:` For Google Search
2. `google.generativeai`: For using Google Gemini to extract relations
3. `time`: For spacing out timeouts to avoid rate limitation
4. `requests:` To fetch websites from URLs
5. `BeautifulSoup:` For processing raw text from webpage to ignore HTML tags, images, and other content that would interfere with information extraction process
6. `re:` For using regular expressions to parse returned text
7. `spacy:` process and annotate text through linguistic analysis
8. `spanbert:` For extracting relations using bert
9. `spacy_help_functions:` Started functions via project instructions. Modified for our project.

Step 3 - Information Extraction Method

Project specification step 3 involves steps 4 through 7 from above. Here is a more detailed explanation:

1. While looping through the resulting URL's from the Google Search API, we skip URL's that have already been by keeping a set of previously processed URL's. If the URL is present in this set, we skip over it and print a message.
2. Retrieve corresponding webpage using `requests` and `BeautifulSoup`. We use `BeautifulSoup`'s `.get_text()` method to extract plain text from the HTML received from the request. Next we strip extra starting and trailing spaces, remove excess characters like "\n" and "\xa0" that may appear in the plain text using `re`'s functions. We can also remove things like "\t" and "\r", and have left this line commented out in case this is desired. Finally, we trim the text to <= 10,000 characters.
3. We then utilize `Spacy`'s `.load` method as used in the example script to extract sentences, tokens, etc. and pass the resulting object to the following extraction methods:

1. SpanBERT

   a. We call the `extract_relations` helper method from `spacy_help_functions.py`, provided by the SpanBERT repository.
   b. To ensure that we only run SpanBERT on entity pairs that contain the named entities that are desired for the relation (such as PERSON and ORGANIZATION for School_Attended), we passed in the `entities_of_interest` parameter, which was defined when parsing arguments in from the command line.
   c. With this parameter defined, the provided `create_entity_pairs` function will automatically ignore unrelated entity pairs.
   d. We also included an additional parameter, `relation_of_interest`, to the `extract_relations` function so that we only return entity pairs that satisfy our desired relation.
   e. Within our version of the `spacy_help_functions.py` file, we have the following changes:
      i. Added counter variables for extracted relations and sentences and updated the print statements to better match the transcripts
      ii. Added an in a progress log as seen in the transcript
      iii. Added a check to prevent SpanBERT from running on empty examples
      iv. Added a check to only return SpanBERT's predicted relations that match the relation of interest.

f. Once relations have been extracted, we return back to the `main` method and add them to our `extracted_tuples` default dictionary. If the value of the relation has a higher confidence than -1 (not present) or the confidence for an already present relation, then we add it to the dictionary / update the confidence value.

g. After all sentences have been processed and all relations have been obtained, we then sort this dictionary by confidence value, ensuring that `X` is sorted in descending confidence for query creation or final return.

2. Gemini

a. We included a `filter_sentences_by_entity_types` method to iterate through all spacy sentences and check if the terms are labeled with required types. Sentences that don't contain both required entity types will be discarded.

b. The `extract_relations` function takes in filtered sentences, relation type, and gemini API key to extract relations. We first configure `genai` with the api key and specify the model name. We then iterate through each sentence and feed in a prompt that we fine tuned to achieve the best results.

c. Design of prompt

   i. Give initial instruction to extract relation from the following sentence

   ii. Set up instructions for each relation to give model context on the relation. We also included a few examples for the model to learn

   iii. Provided instructions for the final output in JSON format with a 1-shot example

   iv. Instructed model to not consider common nouns that wouldn't provide much information with few-shot prompt

   v. Instructed model to only consider proper nouns with few-shot prompt

   vi. Instructed model to not include any additional comment. Just the JSON

d. We then process the prompt response with the `extract_json_from_text` function, which attempts to parse the json returned by using regular expressions. If the parse was successful, and we have valid subjects and objects, then print the extraction found and append it to the dictionary with preset value of 1 for confidence.

e. For query expansion, since we don't have access to the confidence interval, we decided to filter out any pairs of entities if they contain common nouns like "he",

“they”, “it” since those pairs wouldn’t provide too much information for a new iteration. This is our approach to expand the query with as much information as we can.

API KEYS

Google Custom Search Engine: XXXXX

JSON API Key and Engine ID: XXXXX