Project: Information Extraction System

Team Members:

- Pranjal Srivastava (ps3392)
- Shreyas Chatterjee (sc5290)

Submitted Files:

- `project2.py` : Main entry point script for the Information Extraction System.
- `ISE/spanbert_ise.py` : Python script for information extraction using SpanBERT model.
- `ISE/gemini_ise.py` : Python script for information extraction using Google Gemini API.
- `ISE/base_ise.py` : Python script containing the base class for information extraction system.
- `ISE/utils.py` : Python script containing utility functions and constants used in information extraction.
- `README.txt` : This README file.
- `QueryManager/__init__.py` : Initialization file for the QueryManager package.
- `QueryManager/query_manager.py` : Python script containing the QueryManager class for querying search results.
- `requirements.txt` : Mentions all the libraries and packages that need to be installed

Folder Structure:

- Main Folder - Iterative Relation Extraction
  - ISE Folder
    - **init**.py
    - base_ise.py
    - gemini_ise.py
    - spanbert_ise.py
    - utils.py
  - Query Manager Folder
    - **init**.py
    - query_manager.py
  - SpanBERT Folder (imported)
  - project2.py (main file)
  - README.md
  - requirements.txt

Running the Program:

To run the program, follow these steps:

1. Set up the system like mentioned in the project description (python environment and apt-get package error)
2. Make sure that the environment is activated and navigate to inside the main folder
3. Install the required libraries and dependencies:

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

4. When inside the main folder run the following-

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

5. In SpanBERT/spanbert.py, change the imports to 'from SpanBERT.pytorch_pretrained_bert.modeling' and 'from SpanBERT.pytorch_pretrained_bert.tokenization', i.e., the code should look like-

```
#from transformers import AutoTokenizer, AutoModel, BertForSequenceClassification
from SpanBERT.pytorch_pretrained_bert.modeling import BertForSequenceClassification
from SpanBERT.pytorch_pretrained_bert.tokenization import BertTokenizer
```

6. Execute the main entry point script `project2.py` using Python:

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

External Libraries Used:

- `spacy` : Used for natural language processing tasks such as tokenization, part-of-speech tagging, and entity recognition.
- `requests` : Used for making HTTP requests to fetch webpage content.
- `beautifulsoup4` : Used for parsing HTML content fetched from webpages.
- `re` : Used for regular expression operations.
- `ast` : Used for parsing Gemini API output string to extract related entities.
- `google.generativeai` : Used for accessing the Google Gemini API.

Internal Design:

- `project2.py` : Main entry point for the Information Extraction System. Orchestrates the execution of SpanBERT and Gemini information extraction scripts. It parses the arugements that are passed when we run the main file.
- `ISE/utils.py` : Contains variables and dictionaries which contain information about relations and the corresponding entities. These have been made to refer to whenver needed.
- `ISE/base_ise.py` : Contains the base class `BaseISE` for information extraction system. This file basically provides a base template for both spanbert based extraction and gemini based extraction. It has 4

completely defined functions - match_relation, process_url, download_text_from_url and process_sentence. The functions ise and filter_entity_blocks have been kept undefined here because spanbert and gemini need slightly different versions of the same.

- `ISE/spanbert_ise.py` : Implements information extraction using the SpanBERT model. It contains a class `SpanBertISE` which inherits from `BaseISE` .
- `ISE/gemini_ise.py` : Implements information extraction using the Google Gemini API. It contains a class `GeminiISE` which inherits from `BaseISE` .
- `QueryManager/query_manager.py` : Python script containing the QueryManager class for querying the url and obtain the search results.

Step 3 Description:

Step 3 involves designing and implementing the information extraction process using two approaches: SpanBERT model and Google Gemini API. The SpanBERT approach utilizes a pre-trained BERT model for extracting relations from text, while the Gemini approach uses the Google Gemini API for natural language processing tasks.

Base Information Extraction Class (`base_ise.py`):

- `BaseISE` Class: Processes urls and creates a base template for both spanbert and gemini based extractions.
- `process_sentence` : Takes sentence and relation as input. Uses the relation and utils.py to get the entity of interest. Then uses the spacy_help_fucntions to create entity pairs. It finds the candidate pairs which match the entities of interest dscription and filters them accoridng to the requiements. It also keeps track of the extracted and accepted relations from the url. This function takes care of the condition - If a sentence is missing one or two entities of the type required by the relation, you should skip it and move to the next sentence.
- `download_text_from_url` : Takes url as a input and used BeautifulSoup to parse and url and extract the text. It is here that the 10000 character filter is applies to the text.
- `level_log` : Logs info and is used to maintian the appropriate printing format.
- `process_url` : Makes the right calls to the functions to download the text, extract the sentences using spacy. It also maintains the printing format and initialzations required for each url that is being processed.

SpanBERT Information Extraction (`spanbert_ise.py`):

- `SpanBertISE` Class: Implements information extraction using the SpanBERT model.
- `get_query_from_bank` : Retrieves the query with maximum confidence from the query bank for different iterations.
- `filter_sentence` : Finds entity predictions from SpanBERT and applies the filters mentioned in the description to accept the extracted entities or not. All tuples conditions are checked inside this function. This is where the query bank is updated too.
- `print_start` : Prints information about the start of the extraction process for each iteration. Used to maintian the appropriate printing format.
- `log_relations` : Logs extracted relations and iterations. Used to maintian the appropriate printing format. This is where the decreasing order of confidence occures.
- `match_relation` : Matches the relation type to the required predefined relation types of SpanBERT

- `ise` : Executes the information extraction process using SpanBERT. It keeps track of the queries, runs the query to obtains urls, processes the urls and runs the iterations. This checks for already processed urls.

Google Gemini API Information Extraction (`gemini_ise.py`):

- `generate_prompt_template` : Generates the prompt for Gemini API based on relation instruction.
- `parse_gemini_output` : Parses Gemini API output to extract related entities using ast library.
- `GeminiISE` Class: Implements information extraction using the Google Gemini API.
- `get_gemini_completion` : Fetches completion from Gemini API based on prompt.
- `filter_sentence` : Finds entity predictions from Gemini API and applies the filters mentioned in the description to accept the extracted entities or not. All tuples conditions are checked inside this function. This is where the query bank is updated too.
- `print_start` : Prints information about the start of the extraction process for each iteration. Used to maintian the appropriate printing format.
- `get_query_from_bank` : Retrieves the first query from the query bank since all have the same confidence.
- `log_relations` : Logs extracted relations and iterations. We have not put a filter on the number of tuples that are being printed.
- `ise` : Executes the information extraction process using Gemini API. It keeps track of the queries, runs the query to obtains urls, processes the urls and runs the iterations. This checks for already processed urls.

Google Custom Search Engine API Key and Engine ID:

Pranjal Srivastava

- **API Key:** AIzaSyDuSM_JnE7BUlaW54cgZLlcNgEwm79URsg
- **Engine ID:** 3430a164067404160

Shreyas Chatterjee

- **API Key:** AIzaSyBdyvstyytiLSY0jPXM1FV9JfGmDaoZ3iY
- **Engine ID:** 91edb5a90f6c44a6a

Additional Information:

The initial setup has to be same for it to run end to end.

Details about Query Manager Class from Project 1 (`QueryManager/query_manager.py`):

- `QueryManager` Class: Implements querying of search results using the Google Custom Search Engine API.
- `__init__` : Initializes the QueryManager object with required parameters such as API key, engine ID, and number of results.
- `__repr__` : Provides a string representation of the QueryManager object.
- `query` : Executes a query to retrieve search results based on the specified query string.
- `__verify_results` : Verifies the integrity of search results to ensure they meet the required conditions.

- `__parse_results` : Parses the search results and extracts relevant information such as URLs, titles, and snippets.

Details about output:

Once the program is run, the output gets stored in a output log file in the main folder. The format of the log file name is out_{spanbert/gemini}_rel_{1/2/3/4}.log based on the model and the relation type specified by the user.

To see the output in the terminal, after the program has finished execution, run:

```
cat out_{spanbert/gemini}_rel_{1/2/3/4}.log
```