Patrick Yeh (psy2107)
Sameer Saxena (ss6167)

Project 1 README

Submitted files:

- project2.py
- transcriptSpanbert.txt
- transcriptGemini.txt
- README

How to run program:

1. Run 'python3 project2.py [google api key] [google engine id] [google gemini api key] -[spanbert/gemini] [r] [t] [q] [k]' where all the arguments are as specified in the project description/writeup
2. Note: You will need to clone the SpanBERT repository and place the project2.py file within this directory.

Here are some example calls:

1. `python3 project2.py -spanbert googleAPIKey engineKey geminiKey 2 0.7 "bill gates microsoft" 10`

2. `python3 project2.py -gemini googleAPIKey engineKey geminiKey 2 0.7 "bill gates microsoft" 10`

Dependencies:

The program uses the following libraries: sys, pprint, itertools, json, googleapiclient.discovery, BeautifulSoup, URLLib parse/request, requests, spacy, os, google.generativeai, time, spanbert. In particular, one should follow the setup steps in the project description/writeup to get the environment ready. Also, it's important to note that the file should be placed in the same directory as all the SpanBERT files.

Here is another look at the dependencies/libraries:

```
import sys
import string
import pprint
from itertools import permutations
import json
from googleapiclient.discovery import build
```

```
from bs4 import BeautifulSoup
import urllib.parse
import urllib.request
import requests

import spacy
from spacy_help_functions import get_entities, create_entity_pairs



# Load pre-trained SpanBERT model
from spanbert import SpanBERT

import os
import google.generativeai as genai
import time
```

Internal design:

At a high level, the program works as follows. First, we used the previously mentioned libraries/dependencies as explained above. We parse the command line arguments to get our inputs. Then we initialize the set of extracted tuples X to be empty. We will then repeatedly take our current seed query, and use the Google search engine to get 10 links. We will access each of those links via BeautifulSoup to get the associated text, then (after some light preprocessing) pass the text through to spaCy to get annotations. We then utilize either SpanBERT or Gemini components to process sentences that meet our requirements (i.e. we first filter out using the annotations), and any extracted tuples that meet our requirements are added on to X. If we have enough tuples, we can then return the top-K (in Gemini's case, all of the tuples in alphabetical order) tuples after processing each link. Otherwise, we will choose the top tuple that hasn't already been selected for querying as our new seed query, and we will repeat this process until we have either achieved our target or have exhausted all possible queries.

Step 3 methodology:

First, we retrieved the webpage by making a call to the Python requests library. This gave us HTML which we then passed

through BeautifulSoup to extract the relevant text we wanted. We then applied some light pre-processing (i.e. strip out whitespace if needed, and truncate the number of characters if it exceeds 10,000) before using spaCy to run the annotations and split sentences. From this, we were able to get some information such as the number of sentences to process. Once spaCy was applied, we then iterated through each sentence and checked if the sentence contained any entity pairs that match the type we need for our current relation. For example, if we were doing the schools_attended relation, we needed to check for entity pairs where the subject was a person and the object was an organization. Thus, we were able to cut down on the total number of computations to process (which was useful for time and resource consumption). Using SpanBERT, we then applied our SpanBERT model on the annotations to extract the potential tuples, using similar code to the starter example provided in the writeup. We then checked the associated confidence score so that we update the tuple if we found a higher score than previously. If the score was lower than our confidence threshold, we ignored the tuple. Otherwise, if it was a new tuple that met our threshold, we added it to our dictionary X. If we used Gemini instead, we once again used spaCy to filter out sentences, but instead we provided the raw sentence without annotations to our Gemini call. The main trick was to design the Gemini prompt appropriately so that 1) we got the desired tuples and 2) the tuples were in a nice format to parse. To do so, I designed the following prompt that is pasted below. Once we got the input, it was separated by semicolons and newlines, so parsing out the extracted tuples was fairly straightforward. I put the extracted tuples with a hardcoded confidence of 1.0 to X.

Example Gemini prompt:

```
prompt_text = """Given a sentence, extract all the (subject, object) relations
satisfying our intended relation. Output a list of (subject, object) tuples, one tuple
per line, with the subject and object separated by a semicolon.
Here are the four relations:
Schools_Attended: Subject: PERSON, Object: ORGANIZATION (i.e. "Joe Smith;Harvard"
where the subject "Joe Smith" attended Harvard, which is a school)
Work_For: Subject: PERSON, Object: ORGANIZATION (i.e. "Joe Smith;Apple" where the
subject "Joe Smith" works for the organization Apple, which is a company or entity)
```

```
Live_In: Subject: PERSON, Object: one of LOCATION, CITY, STATE_OR_PROVINCE, or COUNTRY
(i.e. "Bill Gates; Seattle" where the subject "Bill Gates" has lived in the object
"Seattle", which is a place)
Top_Member_Employees: Subject: ORGANIZATION, Object: PERSON (i.e. "Apple;Tim Cook"
where the subject "Apple" is an organization and the object "Tim Cook" is a
high-ranking or important employee, in this case the CEO, for that organization)
"""

relation_names = ["Schools_Attended", "Work_For", "Live_In", "Top_Member_Employees"]
relation_explanations = ["", "the first portion of the tuple is a person and the
second part is the organization that person has worked for. ", "", ""]

prompt_text += "We will use the " + relation_names[r-1] + " relation, which means " +
relation_explanations[r-1]

prompt_text += "Here is the sentence, please extract all possible tuples from it: "
#sentence_example = """Bill Gates stepped down as chairman of Microsoft in February
2014 and assumed a new post as technology adviser to support the newly appointed CEO
Satya Nadella. """
prompt_text += str(sentence)

prompt_text += "extracted:"
```

Miscellaneous notes:

There are some design choices that slightly differ the output from the canonical class reference implementation, but these are all justified or permitted by the writeup. In particular, for the Gemini runs, I simply return the first k tuples in alphabetical order instead of all the tuples or a random set. Moreover, perhaps because of text pre-processing, the number of sentences and annotations made slightly differ, but the main goal of filtering out sentences using these spaCy annotations is still implemented. More importantly, the end results (i.e. the list of k extracted tuples at termination) seem to correspond well with the canonical class reference implementation.

Also, to avoid my Gemini key from being flagged for being used too frequently in a short amount of time, I decided to add a sleep(1.8) for each call to Gemini. This could perhaps be too conservative, and perhaps makes the overall runtime a bit slower (about 30-50 seconds extra compared to SpanBERT), but I added it

just for the sake of safety. If there ever is some issue where the output says the Gemini key is being used at too frequent a pace, then the answer is to slightly increase the sleep time (perhaps by a second or so extra). Ultimately, the Gemini runtime should still be comparable to the SpanBERT runtime.

JSON API Key:
AIzaSyBOskhE799tyaHkMxJc08i3YLZiJj6vubw

Engine ID:
10e635b85174848d2

Sources used:

1. Class materials