COMS E6111-Advanced Database Systems (../index.html)
Spring 2022

Project 2

Due Date: Thursday March 31, 5 p.m. ET

Teams

You will carry out this project in **teams of two**. You can do the project with your same teammate as for Project 1 and you are also welcome to switch teammates if you wish. In this case, please be considerate and **notify your Project 1 teammate immediately**. If you want to form a new team but can't find a teammate, please follow these steps:

- Post a message on the Ed Discussion board asking for a teammate—**the best way**.
- Send email to Akhil (mailto:ak4590@columbia.edu) right away (and definitely **before Friday, March 4, at 5 p.m. ET**) asking Akhil to pair you up with another student without a teammate. Akhil will make a best effort to find you a teammate.

You do not need to notify us of your team composition. Instead, you and your teammate will indicate your team composition when you submit your project on Gradescope (click on "Add Group Member" after one of you has submitted your project). You will upload your final electronic submission on Gradescope exactly once per team, rather than once per student.

Important notes:

- If you decide to drop the class, or are even remotely considering doing so, please be considerate and **notify your teammate immediately**.
- On a related note, do not wait until the day before the deadline to start working on the project, just to realize then that your teammate has dropped the class or formed a new team. It is your responsibility to start working on the project and spot any problems with your teammate early on.
- You can do this project by yourself if you so wish. Be aware, however, that you will have to do exactly the same project as two-student teams will.
- Please check the Class Policies webpage (../proj-policies.html) for important information about what kinds of collaboration are allowed for projects, and how to compute the number of available grace late days for a team.

Overview

This project is about information extraction on the web, or the task of extracting "structured" information that is embedded in natural language text on the web. As we discussed in class, information extraction has many applications and, notably, is becoming increasingly important for web search.

In this project, you will implement a version of the **Iterative Set Expansion (ISE)** algorithm that we described in class: for a target information extraction task, an "extraction confidence threshold," a "seed query" for the task, and a desired number of tuples **k**, you will follow ISE,

starting with the seed query (which should correspond to a plausible tuple for the relation to extract), to return **k** tuples extracted for the specified relation from web pages with at least the given extraction confidence, and following the procedure that we outline below.

The objective of this project is to provide you with a hands-on experience on how to (i) retrieve and parse webpages; (ii) prepare and annotate text on the webpages for subsequent analysis; and (iii) extract structured information from the webpages.

You will develop and run your project on the Google Cloud infrastructure, using your LionMail account and VM as you did for Project 1 (../proj1.html).

IMPORTANT NOTE: When you restart your VM to work on this project, you must request at least 15 GB of RAM for the VM, so that your system will run without any memory issues. To do this, in the "VM instances" page of your Google Cloud account, click on the VM's name, then click on "EDIT" at the top, select **n1-standard-4 (4 vCPU, 15 GB memory)** as the "Machine type," and click on "Save."

Description

For this project, you will write a program that implements the ISE algorithm over the web. Your program will rely on:

- The Google Custom Search API (https://developers.google.com/custom-search/) to search the web. You used this API for Project 1 (../proj1.html).
- The Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/) toolkit to extract the actual plain text from a given webpage, and ignore HTML tags, links, images, and all other content that would interfere with the information extraction process. You might need to install `pip3` first:
  ```
  sudo apt update
  sudo apt install python3-pip
  ```
  And then Beautiful Soup proper:
  ```
  pip3 install beautifulsoup4
  ```
- The spaCy (https://spacy.io) library to process and annotate text through linguistic analysis (e.g., split sentences from paragraphs, tokenize text, detect entities). You will need to use Python 3.6 and follow these steps on your Google Cloud VM:
  ```
  sudo apt-get update
  pip3 install -U pip setuptools wheel
  pip3 install -U spacy
  python3 -m spacy download en_core_web_lg
  ```
- The SpanBERT (https://github.com/gkaramanolakis/SpanBERT) classifier to extract the following four types of relations from text documents:
  1. **Schools_Attended** (internal name: `per:schools_attended`)
  2. **Work_For** (internal name: `per:employee_of`)
  3. **Live_In** (internal name: `per:cities_of_residence`)
  4. **Top_Member_Employees** (internal name: `org:top_members/employees`)

  We have implemented the scripts for downloading and running the pre-trained **SpanBERT** classifier for the purpose of this project:
  ```
  git clone https://github.com/gkaramanolakis/SpanBERT
  ```

```
cd SpanBERT
pip3 install -r requirements.txt
bash download_finetuned.sh
```

Overall, your program should receive as input:

- Your **Google Custom Search Engine JSON API Key** from Project 1 (../proj1.html)
- Your **Google Engine ID** from Project 1 (../proj1.html)
- An integer **r** 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**
- A real number **t** between 0 and 1, indicating the "extraction confidence threshold," which is the minimum extraction confidence that we request for the tuples in the output
- A "seed query" **q**, 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)
- An integer **k** greater than 0, indicating the number of tuples that we request in the output

Then, your program should perform the following steps:

1. Initialize **X**, the set of extracted tuples, as the empty set.
2. Query your Google Custom Search Engine to obtain the URLs for the top-10 webpages for query **q**; you can reuse your own code from Project 1 for this part if you so wish.
3. For each URL from the previous step that you have not processed before (you should skip already-seen URLs, even if this involves processing fewer than 10 webpages in this iteration):
    a. Retrieve the corresponding webpage; if you cannot retrieve the webpage (e.g., because of a timeout), just skip it and move on, even if this involves processing fewer than 10 webpages in this iteration.
    b. Extract the actual plain text from the webpage using **Beautiful Soup**.
    c. If the resulting plain text is longer than 20,000 characters, truncate the text to its first 20,000 characters (for efficiency) and discard the rest.
    d. Use the **spaCy** library to split the text into sentences and extract named entities (e.g., PERSON, ORGANIZATION). See below for details on how to perform this step.
    e. Use the sentences and named entity pairs as input to **SpanBERT** to predict the corresponding relations, and extract all instances of the relation specified by input parameter **r**. See below for details on how to perform this step.
    f. Identify the tuples that have an associated extraction confidence of at least **t** and add them to set **X**.
4. Remove **exact duplicates** from set **X**: if **X** contains tuples that are identical to each other, keep only the copy that has the highest extraction confidence and remove from **X** the duplicate copies. (You do not need to remove approximate duplicates, for simplicity.)
5. If **X** contains at least **k** tuples, return the **top-k** such tuples sorted in decreasing order by extraction confidence, together with the extraction confidence of each tuple, and stop. (Alternatively, you can return all of the tuples in **X** sorted in decreasing order of extraction confidence, not just the top-**k** such tuples; this is what the reference implementation does.)
6. Otherwise, select from **X** a tuple **y** such that (1) **y** has not been used for querying yet and (2) **y** has an extraction confidence that is highest among the tuples in **X** that have not yet been used for querying. (You can break ties arbitrarily.) Create a query **q** from tuple **y** by

just concatenating the attribute values together, and go to Step 2. If no such **y** tuple exists, then stop. (ISE has "stalled" before retrieving **k** high-confidence tuples.)

Performing the Annotation and Information Extraction Steps

Steps 3.d and 3.e above require that you use the **spaCy** library and the pre-trained **SpanBERT** classifier to annotate the plain text from each webpage and extract tuples for the target relation **r**.

Relation extraction is a complex task that generally operates over text that has been annotated with appropriate tools. In particular, the **spaCy** library that you will use in this project provides a variety of text pre-processing tools (e.g., sentence splitting, tokenization, named entity recognition).

For your project, you should use **spaCy** for splitting the text to sentences and for named entity recognition for each of the sentences. You can find instructions on how to apply **spaCy** for this task here (https://spacy.io/usage/linguistic-features#named-entities) and in our example script (see below).

After having identified named entities for a sentence, you should use the pre-trained **SpanBERT** classifier for relation extraction. **SpanBERT** is a BERT-based relation classifier that considers as input (1) a sentence; (2) a subject entity from the sentence; and (3) an object entity from the sentence. **SpanBERT** then returns the predicted relation and the respective confidence value. You can find instructions on how to apply **SpanBERT** for this task here (https://github.com/gkaramanolakis/SpanBERT) and in our example script (see below).

We have put together two minimal Python scripts, namely, spacy_help_functions.py (spacy_help_functions.py) and example_relations.py (example_relations.py), that perform the full relation extraction pipeline, to illustrate how the **spaCy** library is integrated with **SpanBERT**. To run these scripts, you need to place them under the same directory as the spanbert.py file (provided here (https://github.com/gkaramanolakis/SpanBERT)).

As an example, consider the following sentence and the output from the full information extraction process:

- Sentence: "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."
- Script output:

```
spaCy extracted entities: [('Bill Gates', 'PERSON'), ('Microsoft',
'ORGANIZATION'), ('February 2014', 'DATE'), ('Satya Nadella', 'PERSON')]

Candidate entity pairs:
1. Subject: ('Bill Gates', 'PERSON')   Object: ('Microsoft',
'ORGANIZATION')
2. Subject: ('Microsoft', 'ORGANIZATION')   Object: ('Bill Gates',
'PERSON')
3. Subject: ('Bill Gates', 'PERSON')   Object: ('Satya Nadella', 'PERSON')
4. Subject: ('Satya Nadella', 'PERSON')   Object: ('Bill Gates', 'PERSON')
5. Subject: ('Microsoft', 'ORGANIZATION')   Object: ('Satya Nadella',
```

```
'PERSON')
6. Subject: ('Satya Nadella', 'PERSON')   Object: ('Microsoft',
'ORGANIZATION')

SpanBERT extracted relations:
1. Subject: Bill Gates   Object: Microsoft   Relation: per:employee_of
Confidence: 1.00
2. Subject: Microsoft   Object: Bill Gates   Relation:
org:top_members/employees   Confidence: 0.99
3. Subject: Bill Gates   Object: Satya Nadella  Relation: no_relation
Confidence: 1.00
4. Subject: Satya Nadella   Object: Bill Gates   Relation: no_relation
Confidence: 0.52
5. Subject: Microsoft   Object: Satya Nadella   Relation: no_relation
Confidence: 0.99
6. Subject: Satya Nadella   Object: Microsoft   Relation: per:employee_of
  Confidence: 0.98
```

Note that in the above example, **SpanBERT** runs 6 times for the same sentence, each time with a different entity pair. **SpanBERT** extracts relations for 3 entity pairs and predicts the `no_relation` type for the rest of the pairs (i.e., no relations were extracted). Each relation type predicted by **SpanBERT** is listed together with the associated extraction confidence score.

Unfortunately, the **SpanBERT** classifier is **computationally expensive**, so for efficiency you need to minimize its use. Specifically, you should **not** run **SpanBERT** over entity pairs that do not contain named entities of the right type for the relation of interest **r**. The required named entity types for each relation type are as follows:

- **Schools_Attended**: Subject: PERSON, Object: ORGANIZATION
- **Work_For**: Subject: PERSON, Object: ORGANIZATION
- **Live_In**: Subject: PERSON, Object: one of LOCATION, CITY, STATE_OR_PROVINCE, or COUNTRY
- **Top_Member_Employees**: Subject: ORGANIZATION, Object: PERSON

As an example, consider extraction for the **Work_For** relation (internal name: `per:employee_of`). You should only keep entity pairs where the subject entity type is PERSON and the object entity type is ORGANIZATION. By applying this constraint in the example sentence above ("Bill Gates stepped down ... appointed CEO Satya Nadella.") **SpanBERT** would run only for the first entity pair ('Bill Gates', 'Microsoft') and the sixth entity pair ('Satya Nadella', 'Microsoft'). Note that the subject and object entities might appear in either order in a sentence and this is fine.

So to annotate the text, you should implement **two steps**. First, you should use **spaCY** to identify the sentences in the webpage text together with the named entities, if any, that appear in each sentence. Then, you should construct entity pairs and run the expensive **SpanBERT** model, separately **only** over each entity pair that contains both required named entities for the relation of interest, as specified above. IMPORTANT: You **must not** run **SpanBERT** for any entity pairs

that are missing one or two entities of the type required by the relation. 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.

While running the second step over a sentence, **SpanBERT** looks for some predefined set of relations in a sentence. We are interested in just the four relations mentioned above. (If you are curious about the other relations available, please check the complete list (https://github.com/gkaramanolakis/SpanBERT/blob/master/relations.txt) as well as an article with a detailed description (https://tac.nist.gov/2015/KBP/ColdStart/guidelines/TAC_KBP_2015_Slot_Descriptions_V1.0.pdf).)

What to Submit and When

Your Project 2 submission will consist of the following three components, which you should submit on Gradescope by Thursday March 31 at 5 p.m. ET:

1. Your well-commented **Python code**, which should follow the format of our reference implementation (see below) and run on your Google Cloud VM, set up as detailed here (../gc-setup.html) (but with more memory, etc. for this project as indicated above). Your implementation should be called using the **same exact name and arguments as the reference implementation**.
2. A **README** file including the following information:
    a. Your name and Columbia UNI, and your teammate's name and Columbia UNI
    b. A list of all the files that you are submitting
    c. A clear description of how to run your program. Note that your project must **compile/run in a Google Cloud VM set up exactly following our instructions (../gc-setup.html)** (but with more memory, etc. for this project as indicated above). Provide all commands necessary to install the required software and dependencies for your program.
    d. A clear description of the internal design of your project, explaining the general structure of your code (i.e., what its main high-level components are and what they do), as well as acknowledging and describing all external libraries that you use in your code
    e. A detailed description of how you carried out Step 3 in the "Description" section above
    f. Your **Google Custom Search Engine JSON API Key and Engine ID** (so we can test your project)
    g. Any additional information that you consider significant
3. A **transcript** of the run of your program on input parameters: `2 0.7 "bill gates microsoft" 10` (i.e., for r=2, t=0.7, q=[bill gates microsoft], and k=10). The format of your transcript should closely follow the format of the reference implementation, and should print the same information (i.e., number of characters, sentences, relations, etc.) as the corresponding session of the reference implementation.

To submit your project, please follow these steps:

1. Create a directory named proj2.

2. Copy the source code files into the proj2 directory, and include all the other files that are necessary for your program to run.
3. Tar and gzip the proj2 directory, to generate a single file proj2.tar.gz.
4. **Submit on Gradescope exactly three files:**
    - Your proj2.tar.gz file with your code,
    - Your uncompressed README file, and
    - Your uncompressed transcript file.

Reference Implementation for Project 2

We created a reference implementation for this project. The reference implementation is called as follows:

```
python3 project2.py <google api key> <google engine id> <r> <t> <q> <k>
```

where:

- **<google api key>** is your Google Custom Search Engine JSON API Key (see above)
- **<google engine id>** is your Google Custom Search Engine ID (see above)
- **<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**
- **<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
- **<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)
- **<k>** is an integer greater than 0, indicating the number of tuples that we request in the output

Unfortunately, the **SpanBERT** classifier requires substantial amounts of memory to run. Therefore, a VM that could support many concurrent runs of the reference implementation, to accommodate the number of students in the class, would exceed our available Google Cloud budget. So rather than giving you direct access to the reference implementation, we provide the transcripts of a variety of runs of the reference implementation (./Proj2-Transcripts). We will keep adding and updating these transcripts periodically, so you have reasonably up-to-date runs available.

Please adhere to the format of the reference implementation for your submission and your transcript file. Also, you can use the transcripts of the reference implementation to give you an idea of how good your overall system should be. Ideally, the number of querying iterations that your system takes to extract the number of tuples requested should be at least as low as that of our reference implementation.

Grading for Project 2

A part of your grade will be based on the correctness of your overall system. Another part of your grade will be based on the number of iterations that your system takes to extract the number of tuples requested: ideally, the number of querying iterations that your system takes to extract the number of tuples requested should be at least as low as that of our reference implementation. We will not grade you on the run-time efficiency of each individual iteration, as

long as you correctly implement the two annotator "steps" described above; in particular, note that you must **not** run the second (expensive) step for all sentences, but rather you should restrict that second step to only those sentences that satisfy the criteria described above. We will also grade your submission based on the quality of your code, the quality of the README file, and the quality of your transcript.
