CMPS 143 - Assignment 6

Due Wednesday, Feb 20, 11:59 PM

The next three assignments will be to design and build a question answering system. We will use stories from Aesop's Fables, the Blogs Corpus and MC test. For each homework you will receive new datasets with more difficult stories and questions.

The Task: Your must build a question answering (Q/A) system that can process a story and a list of questions, and produce an answer for each question. Your system must conform to the following input and output specifications, but other than that you can design your system however you want!

1 Input

Download from Canvas and unzip the file. Your Q/A system requires no command line arguments, but should be placed in the same directory as the dataset. The dataset contains the following:

- story files named `hw6-stories.tsv` contains following columns,
  - *sch*: Scheherazade realization of the story (empty for mcTest)
  - *sch_dep*: dependency parses of Scheherazade
  - *sch_par*: constituency parses of Scheherazade
  - *sid*: story ID
  - *story_dep*: dependency parses of the story
  - *story_par*: constituency parses of the story
  - *text*: raw text of the story
- question files named `hw6-questions.tsv`
  - *difficulty*: difficulty of the question, which is based upon the methods required for extracting the answer. (You have all 'Easy' questions for this assignment)
  - *qid*: unique question ID, which is the story ID *sid* followed by the question number. For example, "blogs-01-1" means that this is question #1 pertaining to story "blogs-01". "mc500.train.0.2" means that this is question #2 pertaining to story "mc500.train.0".
  - *text*: raw text of the question
  - *type*: indicates where the answer comes from either Sch, Story or both. You can use this information in your program.
  - *dep*: dependency parses of the question
  - *par*: constituency parses of the question
- answer files named `hw6-answers.tsv`
  - *answer*: the correct answer to the question
  - *difficulty*: the difficulty of the question, same as it is in hw6-questions.tsv
  - *qid*: the question ID
  - *sid*: the story ID

- *text*: raw text of the text
- *type*: where the answer comes from either Sch, Story or both, same as it is in hw6-questions.tsv

The Answer Key. In some cases, the answer key allows for several acceptable answers (e.g., "Toronto, Ontario" vs. "Toronto"), paraphrases (e.g., "Human Immunodeficiency Virus" vs. "HIV"), varying amounts of information (e.g., "he died" vs. "he died in his sleep of natural causes"), or occasionally different interpretations of the question (e.g., "Where did the boys learn how to survive a storm?" "camping tips from a friend" vs. "their backyard"). When more than one answer is acceptable, the acceptable answers are separated by a vertical bar (|). Below is a sample answer key in `hw6-answers.tsv`:

answer	difficulty	qid	sid	text	type
a police car | police car	Easy	blogs-01-4	blogs-01	What was burned?	Story | Sch
...	...	...	...	...	...

Project Files. Below is a listing of the files for this project as well as the file structure that is expected to run the starter code we provide for you:

```
+ data/
|   - hw6-answers.tsv
|   - hw6-questions.tsv
|   - hw6-stories.tsv
|   - HW6_turked_index_answers_pref_sch.csv
|   - HW6_turked_index_answers_pref_story.csv
+ qa_engine/
|   - __init__.py
|   - base.py
|   - score_answers.py
+ qa.py
+ constituency-demo-stub.py
+ dependency-demo-stub.py
+ chunk-demo.py
+ baseline-stub.py
```

2 Output

Your Q/A system should produce one single response file named `hw6-responses.tsv`, which contains the answers that your system finds for all of questions in `data/hw6-questions.tsv` with corresponding *qid*. The output of your system should be formatted as in Table 1.

answer	qid
YOUR_ANSWER	blogs-01-1
	blogs-01-2
...	...

Table 1: Sample of `hw6-responses.tsv` your system generates

IMPORTANT: Use "answer" and "qid" as your column names. There should be an *answer* and *qid* for every question. Also, be sure to print each answer on a single line. We use the actual qid to index and score your answers, be sure to get it correctly. If your Q/A system can't find an answer to a question (or your system decides not to answer a question), then just set the answer to the empty string `""`. This is analogous to leaving the answer blank as shown in Figure 1.

To obtain a Satisfactory grade, your Q/A system can still be simple, but it should produce correctly formatted output and make a non-trivial, good faith attempt to answer some questions.

WARNING: The answer keys given in *data/hw6-answers.tsv* are not allowed to use to answer questions! For example, you can not just look up the answer to each question, or use the answer keys as training data for a machine learning algorithm. Your system must answer each question using general methods. The answer keys are being distributed only to show you what the correct answers are, and to allow you to score your system's performance yourself.

3 Evaluation

The performance of each Q/A system will be evaluated using the F-measure statistic, which combines recall and precision in a single evaluation metric. Since Q/A systems often produce answers that are partially but not fully correct, we will score each answer by computing the *Word Overlap* between the system's answer and the strings in the answer key. Given an answer string from the answer key, your system's response will be scored for recall, precision, and F-measure.

As an example, suppose your system produces the answer "Elvis is great" and the correct answer string was "Elvis Presley". Your system's answer would get a recall score of 1/2 (because it found "Elvis" but not "Presley"), a precision score of 1/3 (because 1 of the 3 words that it generated is correct), and an F-measure score of .40.

Two important things to make a note of:

- This scoring measure is not perfect! You will sometimes receive partial credit for answers that look nothing like the true answer (e.g., they both contain "of" but all other words are different). And you may sometimes get a low score for an answer that seems just fine (e.g., it contains additional words that are related to the true answer, but these extra words aren't in the answer key). Word order is also not taken into account, so if your system produces all the correct answer words, but in the wrong order, it doesn't matter – your system will get full credit! Automatically grading answers is a tricky business. This metric, while not perfect, is meant to try to give your system as much partial credit as possible.
- The answer key often contains multiple answer strings that are acceptable for a question. Your system will be given a score based on the answer string that most closely matches your system's answer.

Running the scoring script. We have included this scoring function in `qa_engine/score_answers.py`, and to run the scoring you can use command,

```
python3 qa_engine/score_answers.py data/hw6-answers.tsv hw6-responses.tsv
```

Or you can use it in your program by calling the `main()` function. An example is in `qa.py` provided.

4 Provided Files

We have provided you with several "stub" Python files to get you started. These cover how to read in the various files, how to process the data in those files, and how to use a parser and chunker. All of these files will be reviewed during class and labs.

The "Easy" questions for this homework should be able to be answered with a simple string overlap or the use of regular expression patterns ('Where is A' → 'A is { in | under | at ...}'). It is not required to use the parser or chunker for this homework, however, these tools will be required for assignment 7.

Answer sentence selection. A usefull first step in answering a given question is to start by finding the story sentences that might contain the information necessary to answer the question. We have provided you with answer sentence annotations in the files

- `HW6_turked_index_answers_pref_sch.csv`
- `HW6_turked_index_answers_pref_story.csv`

The files contain the sentence index that answers each question. There are two files because some questions can be answered by either "story" or "sch" versions. The two files contain the answering sentence index in the case that you pick one version over the other. The *qid* column contains the question id and the *index* column contains the index of the answering sentence.

We do not provide you with any starter code to test your answer sentence identification method. Each team will need to develop their own sentence selection evaluation code.

5 What To Turn In

Team submission. One submission per team, zip all files and it should include,

1. Your question answering system, name it `qa.py`
2. Your response file that contains the answers to all the questions for all the stories. This should be called `hw6-responses.tsv`
3. README including all your team members
4. Include all files required to run your program

Individual submission. Each individual should submit the following,

1. Write down who is on your team, and who is responsible for turning in the code.
2. Write a summary of what got done during this phase of the project. This should be represented by a list of tasks and how they were completed.
3. Write down your individual contribution during this phase of the project.
4. Write down the contribution of your other team members for this phase of the project.