CMPS 143 - Assignment 7

20 points

Due Monday, May 29, 11:55 PM

This homework assignment will expand upon the question answering task introduced in the previous assignment. The new dataset also has new medium difficulty questions for both the old and the new stories. In other words, the new dataset now includes:

- Fable “Fox and Crow”: Easy and Medium Questions
- Fable “Wily Lion”: Easy and Medium Questions
- Fable “Serpent and Eagle”: Easy and Medium Questions
- Fable “Fox and Stork: Easy and Medium Questions
- Blog “Protest Story”: Easy and Medium Questions
- Blog “Young Man Story”: Easy and Medium Questions
- Blog “Storm Story”: Easy and Medium Questions
- Blog “Squirrel Story”: Easy and Medium Questions

PLEASE NOTE: We have updated the files from HW6 so you MUST download the new dataset for this assignment.

The easy questions and the medium questions are designed with different aspects of natural language processing in mind:

- **Easy:** questions that can be answered with simple string matching.
- **Medium:** questions that can be answered with constituency or dependency parses, or advanced regular expressions or string matching on the .story or .sch files.

The kinds of patterns in constituency and dependency parses that the medium questions are based on have been discussed in the class.

The Task: You must improve your 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! This assignment focuses on improving precision.

Note: This homework will ask you to modify your program to accept command line input (see Input section below.)

```
QuestionID: blogs-01-2
Question: What is the summit meeting named?
Answer: G20 summit
Difficulty: Easy
Type: Sch

QuestionID: blogs-01-3
Question: Where did the protest happen?
Answer: on a street | along the street where I work | right in front of my store
Difficulty: Easy
Type: Story | Sch
...
```

Figure 1: Partial answer key for story blogs-01

1 Input

Your Q/A system should accept a single input file as a command-line argument. We should be able to run your program like this:

```
python <pythonfile> <inputfile>
```

This input file will contain a list of StoryIDs in the order that they should be processed and written to the `.answers` file. Your Q/A system should then assume that for each StoryID, the directory contains a story file named StoryID.story (e.g., "fables-01.story"), a question file named StoryID.questions (e.g., "fables-01.questions"), etc. Your Q/A system should produce an answer for each question in the question file, based on the corresponding story files. A sample input file is shown below.

```
fables-01
fables-02
fables-03
fables-04
blogs-01
blogs-02
blogs-03
blogs-04
```

Figure 2: Example input file

Your Q/A system should be placed within the dataset directory. The dataset contains the following:

- question files named StoryID.questions (e.g., "blogs-01.questions")
- answer files named StoryID.answers (e.g., "blogs-01.answers")
- story files named StoryID.story (e.g., "blogs-01.story")
- Scheherazade realization files named StoryID.sch (e.g., "blogs-01.sch")
- constituency parses of the story named StoryID.story.par (e.g., "blogs-01.story.par")
- dependency parses of the story named StoryID.story.dep (e.g., "blogs-01.story.dep")

- constituency parses of the Scheherazade story named StoryID.sch.par (e.g., “blogs-01.sch.par”)
- dependency parses of the Scheherazade story named StoryID.sch.dep (e.g., “blogs-01.sch.dep”)
- constituency parses of the questions named StoryID.questions.par (e.g., “blogs-01.questions.par”)
- dependency parses of the questions named StoryID.questions.dep (e.g., “blogs-01.questions.dep”)

Your Q/A system should produce an answer for each question in the question file, based on the corresponding story file.

Each question file will contain 4 lines per question indicating the QuestionID, the question itself, a difficulty rating, and the file you should use to find the answer. The *Difficulty* ratings are based upon the methods required for extracting the answer. The *Type* field indicates which file you should use to answer this question, where “Story” indicates the .story file and “Sch” indicates the .sch file. You can and should use this information in your program. The question file will be formatted like Figure 1, except that there will not be an answer line. For example, it would look like this:

```
QuestionID: blogs-01-2
Question: What is the summit meeting named?
Difficulty: Easy
Type: Sch

QuestionID: blogs-01-3
Question: Where did the protest happen?
Difficulty: Easy
Type: Story | Sch
...
```

Figure 3: Partial question file for story blogs-01

2 Output

Your Q/A system should produce a single *Response File*, which contains the answers that your system finds for all of the question of all of the stories in the **input file**. The output of your system should be formatted as follows:

```
QuestionID: fables-01-2
Answer: a piece of cheese

QuestionID: fables-01-3
Answer:

QuestionID: fables-01-4
Answer: snatched
...
```

Figure 4: An example response file for story fables-01

IMPORTANT: There should be a *QuestionID* and *Answer* for every question in every story specified by the input file, in exactly the same order. Also, be sure to print each answer on a single line. If your Q/A system can't find an answer to a question (or your system decides not to answer a question), then just leave the answer blank (as done for *fables-01-3* in Figure 4).

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.

Please note that 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. Your Q/A system is not allowed to use them 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.

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.

We have included this scoring function. To run the program:

```
perl score-answers.pl <response_file> <answerkey_file>
```

To run the evaluation on a single story:

```
perl score-answers.pl fables-01_my_answers.txt fables-01.answers
```

To run the evaluation on an entire dataset:

```
perl score-answers.pl train_my_answers.txt train.answers
```

4 Provided Files

We have provided you with the "stub" Python files. You are required to use parsing or chunking to complete this assignment. Please use the new stub code since some of the files have been updated compared to HW6 and the old stub file may not work with the new data.

We have provided you with a file called `process_stories.txt`. This is the input file that should be passed in to your system as a command line argument, and indicates the order in which the stories should be processed. When we run your code on the held out dataset, we will pass in a different file, so be sure to make this a command line argument, rather than hard coding the name of the file into your program.

The `score-answers.pl` file is also provided for evaluation.

5 What To Turn In

One submission required for each team (as attachment on eCommons):

1. Your question-answering system .py file
2. Your response file that contains the answers to all the questions for all the stories. The name of this file should include the last names of all members of your group, such as: `lastname1_lastname2_answers.txt` or `lastname1_lastname2_lastname3_answers.txt`.

Each student should have an individual inline submission on eCommons including:

1. Team Name
2. Name of the team member who submits the main solution
3. Describe **your team's approach** in this assignment in 1 paragraph. Each team member must write this individually.
4. Describe **your own contribution** in this assignment in 1 paragraph. Each team member must write this individually.