CMPS 143 - Assignment 8
FINAL QA Competition

25 points + upto 25 points extra credit

Due Monday, June 12, 11:55 PM

This is the final phase of the Question Answering project. You will make a presentation after you submit Assignment 8 to present your work in the Finals slot on Wednesday June 14th between 12:00 and 3:00 PM. You will turn in the presentation by Tuesday afternoon so that we can put them all on the same machine to save time during the presentations. There will be a separate assignment posted for submitting presentations. The new dataset for this assignment contains new "hard" questions for both the old and the new stories, including:

- Fable "The Fox and The Crow": Easy, Medium, and Hard Questions
- Fable "The Wily Lion": Easy, Medium, and Hard Questions
- Fable "The Serpent and The Eagle": Easy, Medium, and Hard Questions
- Fable "The Fox and The Stork": Easy, Medium, and Hard Questions
- Fable "The Farmer and The Fox": Easy, Medium, and Hard Questions
- Fable "The Mouse and The Lion": Easy, Medium, and Hard Questions
- Blog "Protest Story": Easy, Medium, and Hard Questions
- Blog "Young Man Dies Story": Easy, Medium, and Hard Questions
- Blog "Storm Story": Easy, Medium, and Hard Questions
- Blog "Squirrel Story": Easy, Medium, and Hard Questions
- Blog "Teacher Story": Easy, Medium, and Hard Questions
- Blog "Oven Story": Easy, Medium, and Hard Questions

PLEASE NOTE: We have updated the files from HW7 so you MUST use the new dataset for this asignment.

The easy, medium, and hard questions were designed with different aspects of natural language processing in mind.

- **Easy:** questions that can be answers 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.
- **Hard:** questions that rely on semantic information to interpret, in this case that means using the .dict or .csv files provided showing the words in our stories as WordNet entries, then using the NLTK WordNet API to find synonyms, hyponyms and hypernyms that can be used in questions. Some hard questions require a combination of semantic information with the use of constituency or dependency parses.

In assignment 3, we required you to learn how to use the WordNet API for synonyms, hyponyms, and hypernyms to find other ways to say the same thing, or to find more general instances of verbs or nouns (hypernyms, *eat* is more general than *graze*) or to find more specific instances of verbs or nouns (*perched* is more specific than *sitting*).

In phase II, your work focused on increasing the precision of your system. In this last phase of the project you should try to increase both your precision and your recall. Paying attention to syntactic structures should help you increase precision, and also allow you to develop more general patterns that can answer more instances of the same type of question. You will need to use the WordNet API and provided files in order to answer the "hard" questions.

```
QuestionID: blogs-01-19
Question: Who blasted tear gas?
Answer: The police
Difficulty: Hard
Type: Sch

...

QuestionID: blogs-01-21
Question: Who fired a chemical weapon?
Answer: The police
Difficulty: Hard
Type: Sch
```

Figure 1: The answer key for story blogs-01, illustrating HARD questions.

The Task: Your system must conform to the following input and output specifications, but other than that you can design your system however you want!

Note: Your program must accept command line input as in Assignment 7 (see Input section below.)

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") and a question file named StoryID.questions (e.g., "fables-01.questions"). Your Q/A system should produce an answer for each question in the question file, based on the corresponding story file. A sample input file is shown below.
Your Q/A system should be placed within the dataset directory. The dataset contains the following:

- two WordNet .csv files that you can use to do Word Sense Disambiguation. One is called "Wordnet_nouns.csv" and the other is called Wordnet_verbs.csv". These files can be loaded into the provided stub code.

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

Figure 2: Sample input story process order file

- 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.

2 Output

Your Q/A system should produce a single *Response File*, which contains the answers that your system finds for all of the stories and questions in the input file. The output of your system should be formatted as follows Assignment 7.
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.

To obtain a **passing** grade, your Q/A system should be able to answer some easy, medium, and hard questions. It should use either the dependency parses or the constituency parses to find the right part of the sentence that has the answer to the question. It should use the WordNet files we have provided and the WordNet API to find synonyms, hyponyms, and hypernyms for the words used in the original story or in the Scheherazade output so that you can answer questions that mean the same things, independent of whether the words are the same. Your program should produce correctly formatted output and make a good faith attempt to answer all types of questions.

WARNING: You will be given the answer keys for some datasets, but 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. Your system must answer each question using general methods and you must use exactly the same system on both test sets. 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: Enhanced Scoring Function

We modified the scoring function for the final QA project to give more weight to the medium and hard questions. In addition to the scoring function you've been using, we're providing you with a Python script that will allow you to look at how well your system is performing on each different type of question. It takes your .answers file and the official .answers file and separated each into 3 new files: "easy", "medium", and "hard". This leaves you with a total of 6 new files: `dev_easy.answers`, `dev_medium.answers`, `dev_hard.answers`, `student_easy.answers`, `student_medium.answers`, `student_hand.answers`. You can then run the score-answers.pl script with the files for each type of question.

The performance of each Q/A system will continue be evaluated using the F-measure statistic, with this weighting change, 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, 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.

To run the question difficulty separation program:
`python separate-question-difficulty.py <response_file> <answerkey_file>`

For example:
`python separate-question-difficulty.py my_answer.txt hw8_dev.answers`

To run the scoring/evaluation program:
`perl score-answers.pl <response_file> <answerkey_file>`

To run the evaluation on an entire dataset:
`perl score-answers.pl my_answers.txt hw8_dev.answers`

To run the evaluation on a single question difficulty:
`perl score-answers.pl student_<difficulty>.answers dev_<difficulty>.answers`

You can also write the output to a file, as such:
`perl score-answers.pl student_easy.answers dev_easy.answers > easy_results.txt`

4 Provided Files

- the usual "stub" Python files; the same as those from Assignment 7.
- the .csv files containing the WordNet information. They are called `Wordnet_nouns.csv` and `Wordnet_verbs.csv`. We have also provided demo files called `wordnet_demo.py` and `wordnet_demo_comments.py` that demonstrate how to use these .csv files to help answer the "hard" questions.
- a file called `hw8.process_stories.txt` that should be passed in to your system as a command line argument, and indicates the order in which the stories should be processed.
- the MODIFIED `score-answers.pl` file and `separate-question-difficulty.py`.

5 What To Turn In

1. Your .py file
2. Your response file that contains the answers to ALL the questions (easy, medium, and hard difficulties) 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.

6 Tentative Rubric

This is a tentative rubric to give you an idea of how Assignment 8 will be graded.

1. Summary 6 pts.: 2 paragraphs to describe your approach and contributions.
2. Code 5 pts: clear code without any errors and exceptions.
3. Output 5 pts: your output file must be in the correct format so that the perl script can run it. Note that you need to output the "Answer:" even if there is no result.
4. Evaluation and Competition 9 pts: This part is based on how well your QA system performs both on the dev data (that you have) and the held-out data for HW8 and how your system ranks compared to other groups.

OPTIONAL PART (Extra Credit)
For this assignment, in addition to everything mentioned so far, we encourage to use Word2Vec models. You were given an introduction to this tool in the class. We have provided a demo code to get you started.

Depending on how you use this models, your implementation and ideas, you can earn anything between **10 to 25 extra credit points**.

In order to run the demo code, you will need a file called GoogleNews-vectors-negative300.bin (this is a large file, so be sure to have enough space in your computer). This the Google pretrained word vectors on Newswire text, and can be downloaded from the following URL
`http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/`

We will be using the gensim package to load and use this model. You should install both gensim and scikit-learn using the following commands:
pip install -U scikit-learn
pip install -U gensim
Depending on your version of scikit-learn, you might have to change the line

self.w2vecmodel=gensim.models.Word2Vec.load_word2vec_format(w2vecmodel, binary=True)
to
self.w2vecmodel=gensim.models.KeyedVectors.load_word2vec_format(w2vecmodel, binary=True)
in the file word2vec_extractor.py

To run the demo use:
$python3 baseline-stub-word2vec-demo.py

There are two methods implemented here: baseline_word2vec() and baseline_word2vec_verb() and you can try running each of these. Like the names suggest, the first method creates a vector for every word in the sentence and we use cosine similarity to get the sentence closest to the question, instead of the word overlap we used previously. The second method uses only the ROOT of the sentence and converts it to a vector.
To run this word2vec approach on the entire dataset, use the hw6_word2vec.py file. This is the solution to Assignment 6 using the word2vec approach.

7 Tentative Rubric – Extra Credit

This is a tentative rubric to give you an idea of how extra credit will be assigned.

1. Basic (5 pts.): use gensim to use the GoogleNews word embeddings described above.
2. Apply successfully to Q/A (5 pts.): show that the model has some positive results on your QA system.
3. Be Creative (0-15 pts): Come up with your own methods to apply the model in an attempt to improve your system's performance.