Assignment 1
Natural Language Processing
Total points: 70
Due Thursday, October 8 at 11:59 pm on Canvas

Please Work Individually

All the code has to be your own. The code is to be written individually. Violations will be reported to the University.

1. [50 points] Sentence Boundary Detection.

Write a Python program *SBD.py* that detects sentence boundaries in text. Specifically, your program will have to predict if a period (.) is the end of a sentence or not. You will train and test your program on subsets of the Treebank dataset, consisting of documents drawn from various sources, which have been manually tokenized and annotated with sentence boundaries. The datasets (*SBD.train* and *SBD.test*) are uploaded on Canvas.

Assumptions:

- Your program should only focus on occurrences of the period (.), thus assuming that no other punctuation marks will end a sentence. In other words, in your program, you should not handle question marks (?) or exclamation signs (!).
- Your program should only handle periods that end a word. That is, you should ignore those periods that are embedded in a word, e.g., *Calif.-based* or *27.4*
- Your program will only make use of the EOS (End of Sentence) and NEOS (Not End of Sentence) labels, and ignore all the TOK labels.

Programming guidelines:

Your program should perform the following steps:

- ➢ Identify all the period occurrences where the period ends a word, i.e., sequences of the form "L. R" Each of these periods is labeled as either EOS or NEOS.
- ➢ For each such period occurrence:
  - ➔ Extract the set of five core features described in class, namely:
    - ◆ Word to the left of "." (L) (values: English vocab)
    - ◆ Word to the right of "." (R) (values: English vocab)
    - ◆ Length of L < 3 (values: binary)
    - ◆ Is L capitalized (values: binary)
    - ◆ Is R capitalized (values: binary)
  - ➔ Extract three additional features of your own choosing.

- ➢ The two steps above will create, for both the training and the test dataset, a collection of feature vectors. Each feature vector corresponds to one period instance, consists of eight features, and is assigned an EOS or a NEOS label.
- ➢ Using the sklearn library, train a Decision Tree classifier. Use the feature vectors obtained from the examples in **SBD.train** to train the classifier, and apply the resulting model to predict the labels for the feature vectors obtained from the examples in **SBD.test.**
- ➢ Compare the labels predicted by the classifier for the feature vectors obtained from SBD.test against the provided (gold-standard) labels and calculate the accuracy of your system.

The *SBD.py* program should be run using a command like this:
% *python SBD.py* SBD.train SBD.test

The program should produce at the standard output the accuracy of the system, as a percentage. It should also generate a file called SBD.test.out, which includes the first two columns from the input file SBD.test, along with the label EOS or NEOS predicted by the system for each period occurrence in the test data. The version of the program that you will submit should include all eight features mentioned before (core + your own).

Write-up guidelines:

Create a text file called **SBD.answers**, and include the following information:

- A **brief** description of your own three additional features
- How complete your program is. Even if your program is not complete or you are getting compilation errors, you will get partial credit proportionally. Just mention **clearly and accurately** how far you got.
- If your program is complete, the accuracy of your system on the test data using all eight features (core + your own).
- If your program is complete, the accuracy of your system on the test data using only the five core features from the class.
- If your program is complete, the accuracy of your system on the test data using only your own three additional features.

2. [20 points] Collocation identification.

Write a Python program *Collocations.py* that identifies collocations in text. Specifically, your program will have to implement the **chi-square** and the **pointwise mutual information (PMI)** measures of association for the identification of bigram collocations. You will run your program on a subset of the Treebank corpus. The dataset (*Collocations*) is uploaded on Canvas.

Programming guidelines:

- ➢ Collect the raw counts from the corpus for all the unigrams (individual words) and bigrams (sequences of two words). Unigrams and bigrams that include tokens consisting only of punctuation should not be included, i.e., you should not collect counts for "," (unigram) or for "today ," (bigram). You should instead collect counts for e.g., "U.S." or "34.7", where the punctuation is part of the word.

- ➢ Implement the chi-square and the PMI measures as **two separate functions**. Each of the two functions should use the raw counts from before, and calculate the chi-square (or PMI) for all the bigrams in the corpus. Again, bigrams that include punctuation as a separate token should not be included.
- ➢ Rank the bigrams in reverse order of their chisquare (or PMI) score, and output the top 20 bigrams.

The *Collocations.py* program should be run using a command like this:
% *python Collocations.py Collocations <measure>*

where the <measure> parameter could be either "chi-square" or "PMI". Depending on the parameter provided in the command line, one of the two measures should be used.

Your program should produce the following output:

Bigram1 Score1
Bigram2 Score2
…
Bigram20 Score20

representing the top 20 bigrams in reverse order of their chi-square (PMI) score.

Write-up guidelines:

Create a text file called Collocations.answers, and include the following information:

- How complete your program is. Even if your program is not complete or you are getting compilation errors, you will get partial credit proportionally. Just mention **clearly and accurately** how far you got.
- If your program is complete, top 20 bigrams along with their chi-square scores, in reverse order of the scores
- If your program is complete, top 20 bigrams along with their PMI scores, in reverse order of the scores
- A **brief** discussion of which of the two measures you believe works better to identify collocations, based on your analysis of the top 20 bigrams produced by each measure.

There will be two links for submission on Canvas:

1. Please save your full code as PDF **(plain text)** and submit it by itself.
2. Please Submit a zip file that includes all your files **SBD.py, SBD.answers, Collocations.py, Collocations.answers,** but **do not** include the data files.
3. **Three** screenshots showing the run of your **three programs**, sentence boundary, collocation detection with Chi-square, and collocation detection with PMI, (whether succeeded or failed), and the output or part of the output (as your screenshot allows). If a fewer number of screenshots show all the results, it should be acceptable as well. **Please make sure the date is shown in the screenshots.**

General Grading Criteria:

- Submitting all required files (even with a non-compiling code): 15/70.
- A **reasonable attempt** and a reasonable code that doesn't compile: 35-40/70.
- A **reasonable program/code,** which runs successfully, but doesn't give the correct output: (45-50/70) (ex: few mistakes in functions or metrics, accuracy rates not rational, etc.)
- A successful program with the correct output, but not fulfilling all requirements (ex: Write-up not complete, etc.): 60/70.
- A program fulfilling all the requirements: 70/70.