Assignment 1

Natural Language Processing
Fall 2018
Total points: 80
Issued: 09/14/2018 Due: 09/27/2018

All the code has to be your own (exceptions to this rule are specifically noted below). The code must run on the CAEN Linux environment without additional installation or additional files (except for the data files specified in the assignment).

You can discuss the assignment with others, but the code is to be written individually. You are to abide by the University of Michigan/Engineering honor code; violations will be reported to the Honor Council.

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 available from the Files section of the Canvas class webpage.

Assumptions:

- Your program should only focus on occurrences of the period (.), thus assuming that no other punctuation marks can 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 (periods that are followed by a whitespace). 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.

<u>Programming guidelines:</u>
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 occurrence, consists of eight features, and is assigned an EOS or a NEOS label.
- For example, for the case of "Mt. Pleasant":
    1. The word to the left of "." is **"Mt"**.
    2. The word to the right of "." is **"Pleasant"**.
    3. Length of "Mt" is two, so **True** for "Length of L < 3".
    4. "Mt" is capitalized, so **True** for "Is L capitalized".
    5. "Pleasant" is capitalized, so **True** for "Is R capitalized".

  So the feature vector for ""Mt. Pleasant" is: [**"Mt"**, **"Pleasant"**, **True**, **True**, **True**] (core features) plus your choice of three additional features.
- Using the sklearn Python library (available on CAEN), train a DecisionTree 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. It is part of your assignment to determine what options to use for this classifier.
- 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 description of your own three additional features
- The accuracy of your system on the test data using all eight features (core + your own)
- The accuracy of your system on the test data using only the five core features from the class
- The accuracy of your system on the test data using only your own three additional features

2. [30 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 available from the Files section of the Canvas class webpage.

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 a word. You should also discard bigrams that occur less than 5 times.
- 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 (high to low) of their chi-square (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:

- Top 20 bigrams along with their chi-square scores, in reverse order of the scores
- 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.

General submission instructions:

- Include all the files for this assignment in a folder called *[your-uniqname].Assignment1/* **Do not** include the data files.
  For instance, *mihalcea.Assignment1/* will contain *SBD.py, SBD.answers, Collocations.py, Collocations.answers*
- Archive the folder using zip and submit on Canvas by the due date.
- Include your name and uniqname in each program and in each *.answers file
- Make sure all your programs run correctly on the CAEN machines using a Linux environment.