NLP Problem Set 2

SUBMISSION INSTRUCTIONS:

Due-date: The due-date for this homework assignment is as listed on Canvas (see "Due" within the Assignment page). The completed assignment needs to be submitted into by that due-date.

What to submit:

1. Self-grading text file named **selfgradesNLP_2.txt**. See instructions below. This must be **a separate file (i.e., must not be part of another file you submit)**.
2. Solutions to programming problems. Submit code files that include code segments you were asked to implement. Include screenshots of the output related to this assignment, and the results' plots (if a particular programming problem calls for graphs or plots of results).

   Note for Jupyter Notebook users: Do **not** submit your code files as .ipynb files. You must convert your Jupyter Notebook .ipynb files to plain-text (.txt) files prior to submission, and submit your codes as .txt files.

Do **not** zip, tar, or compress in any other way the files you submit. Any files you submit must be uncompressed. Any files you submit must be either in pdf (i.e., ".pdf') or text (i.e., ".txt") formats. This means that for any ".py" file you submit, you need to change its file-extension to ".txt" prior to submission. Any file formats other than ".pdf' or ".txt" are **not** allowed.

GRADING:

This problem set is self-graded. We will randomly choose a subset of submissions to verify the grades.

How to self-grade problems: The maximal number of points you are allowed to claim for any particular problem is listed in the problem section-headings. For any programming problem you may only claim a full credit or zero, i.e., there is **no** partial credit for any of the programming problems.

Reminder: Copying solutions from **any** source – e.g., online, solutions manuals, assignment solutions from prior semesters, or solutions from any other sources – is prohibited.

Self-grading file filling and submission instructions:

To receive any credit for this assignment, a separate self-grading text file **must** be included as part of this assignment submission. Use the self-grading text file that is provided with this assignment. You need to edit that file by filling in your self-assigned grades, and submit the edited file with your self-grades filled in. The submitted self-grading file must be **exactly** in format and according to the following specifications.

Self-grading file must contain **one line only** (i.e., single line inside the file). This line must contain text fields separated by be **commas**, i.e., comma-separated format. The first three comma-separated fields must contain (in this order) your last name, your first name, and your Northeastern University student email address, respectively. The remaining fields must contain your self-assigned grades for **all** gradable problems (in the order in which those problems appear in the assignment).

Gradable problem is any problem in this assignment for which a point-value (maximum number of points you can obtain for that particular problem) has been listed in the heading of that problem's respective section or subsection (look for square parentheses, e.g., [X points]).

The last field of the self-grading file line must contain your TOTAL self-grade for this homework assignment. Obviously, this total must be equal to the sum of all problem self-grades.

Do the following to complete your self-grading file:

1. Get the "starter" self-grading file, provided within this assignment on Canvas. This file contains the following single line:

   Lastname,Firstname,Email,

2. Replace last-name, first-name, and email-address fields with your data. You are reminded that the email address must be your Northeastern University email address.

3. Continuing this same single line, enter your self-grades for all gradable problems, in the order of gradable problems and including any zero-grades you may have self-assigned for any of the problems.

   Each self-assigned grade must be followed by a **single comma** (i.e., self-grade fields are comma-separated fields).

   Make sure that the number of these comma-separated self-grade fields equals the number of gradable problems. Do not skip self-grades of any gradable problems. If you have not attempted a particular problem, enter self-grade of (numerical zero) for that problem. Do **not** use any blanks.

4. Put a comma after the last self-grade field and then continue in the same line by entering the total self-grade for this homework. Do not add any characters after the total self-grade field. Other than filling the fields in the single line according to the specifications provided here, do **not** alter anything in the file format. Do **not** add any extra lines.

   Example:

   - Johnny Doe is listed in Northeastern University student records as: LAST=Doe, FIRST=Charles, MIDDLE=John.
   - Suppose a hypothetical homework assignment consisting of six gradable problems (six here is just an example) with their maximum point-values of 15, 15, 10, 20, 10, 30, respectively, and that Johnny self-graded them as 15, 0, 10, 20, 0, 30.
   - Johnny needs to change the line in the self-grade file, putting his data for last-name, first-name and email-address, appending (in the same line) comma-separated fields containing all self-grades (non-zeros as well as zeros, as applicable), and ending the line with the field containing the total self-grade for this homework. Therefore, Johnny changes the single line in the self-grading file to the following:

     Doe,Charles,doe_ch@northeastern.edu,15,0,10,20,0,30,75.

5. Prior to submitting your homework, check the self-grading file and make sure that it is completed **exactly** according to **all** of the instructions provided herein. Also, check that you did not overlook any gradable problems.

6. Be sure to submit your completed self-grading file together with your homework.

Please note that a credit of zero will be given for this homework if the self-grading file, filled in according to the instructions provided herein, is not submitted as part of your homework.

Programming Problems

In each of the problems listed below, you are asked to tokenize the text provided in the file NLPdataHW2.txt.

The output of your code (which needs to be submitted in addition to the source code showing your solutions) needs to include the list of all tokens resulting from your code execution. Additional items that the output needs to include are specified within each of the questions below.

1 Tokenization using NLTK [33 points]

Tokenize the text using word_tokenize. Convert these tokens to lowercase. Also, calculate the number of tokens and determine the frequency of each word. List ten most-frequent tokens.

2 Tokenization using NLTK blankline tokenizer [33 points]

Tokenize the text using NLTK blankline tokenizer. Your output should include the list of tokens.

Notes:

A blankline tokenizer is a subclass of Regexp Tokenizer that uses a predefined regular expression. It tokenizes a string, treating any sequence of blank lines as a delimiter. Blank lines are defined as lines containing no characters, except space or tab characters.

3 Tokenization using Tensorflow Keras API [34 points]

Tokenize the text using Tensorflow Keras API. Follow the instructions provided in the starter code regarding the use of various functions for this question. Also determine the indices of the generated word tokens.

Notes:

In Tensorflow Keras API, the tf.keras.preprocessing.text.Tokenizer class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count. By default, all punctuations are removed, transforming the text into space-separated sequences of words (words may also include the ' character). These sequences are then split into lists of tokens. They will then be indexed or vectorized. 0 is a reserved index that will not be assigned to any word.

The text_to_word_sequence is a function of tf.keras.preprocessing.text. It transforms a string of text into a list of words while ignoring filters which include punctuations by default.

Tokenization in Keras can also be performed using 'fit_on_sequences', 'fit_on_text', 'sequences_to_texts', 'texts_to_sequences' and others.

The 'texts_to_sequences' function transforms each text in texts to a sequence of integers. Only top $num\_words - 1$ most frequent words will be taken into account. Only words known by the tokenizer will be taken into account.