Assignment 2
Natural Language Processing
Total points: 70
Due Saturday, Feb 23 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. [60 points] Viterbi Part-of-speech Tagger

Write a Python program *Viterbi.py* that implements the Viterbi algorithm for part-of-speech tagging, as discussed in class. Specifically, your program will have to assign words with their Penn Treebank tag. You will train and test your program on subsets of the Treebank dataset, consisting of documents drawn from various sources, which have been manually annotated with part-of-speech tags. The datasets (*POS.train* and *POS.test*) are uploaded on Canvas; each line in these files corresponds to a sentence.

Programming guidelines:

Your program should perform the following steps:

- Starting with the training file, collect and store all the raw counts required by the Viterbi algorithm. Please make sure to also cover the **"beginning of a sentence"** in your raw counts.
- Implement the Viterbi algorithm and apply it on the test data. Make sure to strip off the part-of-speech tags in the test data before you make your tag predictions.
- Compare the tags predicted by your implementation of the Viterbi algorithm against the provided (gold-standard) tags and calculate the accuracy of your system.
- The *Viterbi.py* program should be run using a command like this: % *python Viterbi.py* POS.train POS.test
- The program should produce at the standard output the accuracy of the system, as a percentage. It should also generate a file called POS.test.out, which includes the words in the test file along with the part-of-speech tags **predicted** by the system.

Write-up guidelines:

Create a text file called Viterbi.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, the accuracy of your system on the test data
- If your program is complete, the accuracy of **a simple baseline program *(baseline.py)*** that assigns to each word its most frequent tag (according to the training data)
- If your program is complete, identify three errors in the automatically tagged data, and analyse them (i.e., for each error, write **one brief sentence** describing the possible reason for the error and how you think it could be fixed)

2. [10 points] Training on Large Data

Train your Viterbi tagger on the large training file (POS.train.large), which is uploaded on Canvas. Test the tagger on the same test file as before (POS.test).

Write-up guidelines:

Create a text file called Viterbi.large.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, the accuracy of your system on the test data
- If your program is complete, the accuracy of **a simple baseline** that assigns to each word its most frequent tag (according to the **large** training data)

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, ***Viterbi.py, baseline.py, Viterbi.answers,*** **and *Viterbi.large.answers***, but **do not** include the data files.
3. **Four screenshots** showing the four runs of the two programs (baseline on the smaller training set, viterbi on the smaller training set, baseline on the larger training set, and viterbi on the larger training set) (whether succeeded or failed), and the output or part of the output (as your screenshot allows). **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.