Assignment III
Applied Natural Language Processing

1 Overview

In this assignment you will write a Hidden Markov Model part-of-speech tagger for Italian, Japanese, and a surprise language. The training data provided is tokenized and tagged; the test data provided will be tokenized, and your tagger will add the tags. The assignment will be graded based on the performance of your tagger, that is how well it performs on unseen test data compared to the performance of a reference tagger.

2 Data

A set of training and development data will be made available as a compressed ZIP archive on **Blackboard**. The uncompressed archive will have the following files:

- Two files (one Italian, one Japanese) with tagged training data in the word/TAG format, with words separated by spaces and each sentence on a new line.
- Two files (one Italian, one Japanese) with untagged development data, with words separated by spaces and each sentence on a new line.
- Two files (one Italian, one Japanese) with tagged development data in the word/TAG format, with words separated by spaces and each sentence on a new line, to serve as an answer key.
- A readme/license file (which you won't need for the exercise).

3 Programs

You will write two programs: **hmmlearn.py** will learn a hidden Markov model from the training data, and **hmmdecode.py** will use the model to tag new data. If using Python 3, you will name your programs **hmmlearn3.py** and **hmmdecode3.py**. The learning program will be invoked in the following way:

¿ **python hmmlearn.py /path/to/input**

The argument is a single file containing the training data; the program will learn a hidden Markov model, and write the model parameters to a file called **hmmmodel.txt**. The format of the model is up to you, but it should follow the following guidelines:
- The model file should contain sufficient information for **hmmdecode.py** to successfully tag new data.
- The model file should be human-readable, so that model parameters can be easily understood by visual inspection of the file.
The tagging program will be invoked in the following way:

¿ **python hmmdecode.py /path/to/input**

The argument is a single file containing the test data; the program will read the parameters of a hidden Markov model from the file **hmmmodel.txt**, tag each word in the test data, and write the results to a text file called **hmmoutput.txt** in the same format as the training data.

4 Submission

All submissions will be completed through Vocareum; please consult the instructions for how to use Vocareum given during previous assignments.

Multiple submissions are allowed; only the final submission will be graded. Each time you submit, a submission script is invoked.

The performance of your tagger will be measured automatically; failure to format your output correctly may result in very low scores, which will not be changed.

If you have any issues with Vocareum with regards to logging in, submission, code not executing properly, etc., please contact the TAs.

5 Grading

After the due date, we will train your model (separately for each language) on a combination of the training and development data, run your tagger on new (unseen) test data, and compute the accuracy of your output compared to a reference annotation. Your grade will be the accuracy of your tagger, scaled to the performance of a reference HMM tagger.

Since part-of-speech tagging can achieve high accuracy by using a baseline tagger that just gives the most common tag for each word, only the performance above the baseline will be scaled:

- If your accuracy $\leq$ baseline accuracy, your grade is your accuracy.
- If baseline accuracy $<$ your accuracy $<$ reference accuracy, your grade is baseline + (1 – baseline) x (yours – baseline) / (reference – baseline).
- If reference accuracy $\leq$ your accuracy, your grade is 100.

For example, if the baseline is 90%, the reference in 95%, and your accuracy is 93%, then your grade will be 0.9 + 0.1 x 0.03 / 0.05 = 96%.
Each language (Italian, Japanese, and the surprise language) is worth one-third of the overall grade for this assignment.
Italian: Baseline: 0.8926 Reference: 0.9423
Japanese: Baseline: 0.8623 Reference: 0.9189

6 Notes

- **Tags:** Each language has a different tagset; the surprise language will have some tags that do not exist in the Italian and Japanese data. You must therefore build your tag sets from the training data, and not rely on a precompiled list of tags.
- **Slash character:** The slash character '/' is the separator between words and tags, but it also appears within words in the text, so be very careful when separating words from tags. Slashes never appear in the tags, so the separator is always the last slash in the word/tag sequence.
- **Smoothing and unseen words and transitions:** You should implement some method to handle unknown vocabulary and unseen transitions in the test data, otherwise your programs won't work.
  - **Unseen words:** The test data may contain words that have never been encountered in the training data: these will have an emission probability of zero for all tags.
  - **Unseen transitions:** The test data may contain two adjacent unambiguous words (that is, words that can only have one part-of-speech tag), but the transition between these tags was never seen in the training data, so it has a probability of zero; in this case the Viterbi algorithm will have no way to proceed.

  The reference solution will use add-one smoothing on the transition probabilities and no smoothing on the emission probabilities; for unknown tokens in the test data it will ignore the emission probabilities and use the transition probabilities alone. You may use more sophisticated methods which you implement yourselves.
- **End state:** You may choose to implement the algorithm with transitions ending at the last word of a sentence, or by adding a special end state after the last word.
- **Runtime efficiency:** Vocareum imposes a limit on running times, and if a program takes too long, Vocareum will kill the process. Your program therefore needs to run efficiently. In our experience, using log probabilities contributes to runtime-efficient code.

7 Collaboration and details

- This is an individual assignment. You may not work in teams or collaborate with other students. You must be the sole author of 100%
- You may not look for solutions on the web, or use code you find online or anywhere else
- You may not download the data from any source other than the files provided on Blackboard, and you may not attempt to locate the test data on the web or anywhere else
- You may use packages in the Python Standard Library. You may not use any other packages. You may use numpy, but are **NOT** allowed to use scikit, pandas, nltk, etc
- You may use external resources to learn basic functions of Python (such as reading and writing files, handling text strings, and basic math), but the extraction and computation of model parameters, as well as the use of these parameters for classification, must be your own work
- Failure to follow the above rules is considered a violation of academic integrity, and is grounds for failure of the assignment, or in serious cases failure of the course. We use plagiarism detection software to identify similarities between student assignments, and between student assignments and known solutions on the web. Any attempt to fool plagiarism detection, for example the modification of code to reduce its similarity to the source, will result in an automatic failing grade for the course

8 Quick Reference

- Write a script **hmmlearn.py** which takes argument as path to input. This should write model parameters to **hmmmodel.txt** any format, should be human readable
- Write a script **hmmdecode.py** which takes argument as path to input of test file which has to be tagged. This should write result to **hmmoutput.txt**. Format of this output file should be similar to the format of training data

Good Luck!!
