Homework Number 3

Due February 25, 2019

1. Download the zip file at this link. It a link to a file on NYUClasses -- so you will need the appropriate login to download it. It will include the following files:

    - A training file (WSJ_02-21.pos) consisting of about 950K words. Each line consists of a token, a single blank, and the part-of-speech of that token using the Penn Treebank tag set. Sentence boundaries are marked by an empty line (about 40K sentences).
    - A Development file (WSJ_24.pos) in the same format as the training file, consisting of about 32.9 K words (about 1350 sentences).
    - A Development test file (WSJ_24.words) -- the same as WSJ_24.pos, but without the POS tags
    - A Test file (WSJ_23.words) in the same format as the Development test file. (56.7K words, about 2415 sentences.)
    - There is also a README.txt file describing the data
    - There is a scoring program (score.py -- written in python2) for you to see how well you are doing on the development set.

2. For development purposes, use the development corpus as your test corpus. Use it for debugging and improving your code, initially ignoring the POS. Later when you are ready to submit the HW, merge the development and training corpora. So you can train with more data when running the system you are submitting.
3. Make a table of the prior probabilities for each of POS tags (assuming bigrams) using the training corpus. (update your training corpus before submitting as per 2)
4. Make a likelihood table including all words in the training corpus. (update your training corpus before submitting as per 2)
5. Make a list of all the words that you find in the training corpus. Any word that is not in this list is out of vocabulary (OOV). You may find OOV words when running the system and will have to treat them specially because otherwise they will have a likelihood of 0 for all POS tags.
6. Implement a Viterbi HMM POS tagger using the prior probabilities and likelihood tables. This program should take a corpus in the format of the test corpus and produce output in the format of the training corpus. As per 2, in the development stage of your program-writing use the training corpus to create the

tables and run the system on the development corpus. For the final system, merge the training and development and run on the test.

- If you plan to use a language other than python or java, please let me know.
- Use some strategy to handle the "likelihood" of out of vocabulary (OOV) items. Possible strategies include:
  - use 1/1000 (or other number) as your likelihood for all OOV items and use this same likelihood for all parts of speech -- effectively only use the transition probability for OOV words
  - make up some hard coded probabilities based on features like capitalization and endings of words, e.g., ends_in_s --> .5 NNS, begins with capital letter --> .5 NNP, nonletter/nonnumber --> .5 punctuation tag, a token consisting of digits --> 1.0 CD, All other probabilities for OOV are 1/1000.
  - use the distribution of all items occurring only once as the basis of computing likelihoods for OOV items, e.g., suppose there are 50K words occurring 1 time in the corpus, 46,010 NN, 3704K JJ, 243 VBD, 40 RB, 2 IN, and 1 DT, then the likelihoods would be the total number of words of each category divided by these numbers. If there are a total of 200K NNs in the corpus then there is a 46010/200K change that the NN will be an unknown word. In other words, we are pretending that *UNKNOWN_WORD* is a single word.
  - use some modified version of the above that take into account the endings of the words, e.g., have classes like OOV_ends_in_s, OOV_with_capital_letter, etc.
  - see section 5.8 of Jurafsky and Martin for other possibilities

8. Submit your homework through GradeScope in the form of a zip file called NetID-HW3.zip, e.g., alm4-HW3.zip
   - Your program filename(s) should include your NETID, the appropriate file type (.py, .java, etc.) and "HW3", e.g., alm4trainHMM_HW3.py, alm4_viterbi_HW3.py, main_alm4_HW3.py, etc.
   - A 1-page write-up called NETID_HW3_README.txt, e.g., alm4_HW3_README.txt, that explains how to run your system, what you did to handle OOV items, etc.
   - Your output for the test data. Use WSJ_23.words to produce an output file called **submission.pos**
9. You may discuss methods with anybody you want, including fellow students. However, you should submit your own code.

10. There is more information and more options at the end of the lecture 3 HMM slides on the website.
11. You have the option of collaborating with one other person and doing a joint entry for this project. However, if you do this, you should add at lease one interesting strategy aimed at achieving a higher score. It is OK if your attempt is unsuccessful, provided you include a description of what you tried along with an explanation of why you think it didn't work. There are more details in the HMM slides.
12. Note that implementing a trigram version (as per J and M) is an option, but past experience suggests that handling the OOV items well has a bigger effect on the final result.
13. Grading is based on: your score on the test corpus and whether or not you implemented anything extra (e.g., handling OOV words, trigram model, etc.).

    - When you submit your program on GradeScope, it will immediately give you an accuracy score for your test file (alm4_WSJ_23.pos). This will be a major factor in your final grade. If you submit an incorrect format file, you will get a particularly low score and you probably want to debug your system and resubmit.