COMP 150-06 Natural Language Processing
Spring 2016

Problem Set 3: POS Tagging with Hidden Markov models.

Build a Bigram HMM part-of-speech tagger using the template `pset3_template.py`.

1. **Preprocessing:** Similarly to problem set 2, build a static vocabulary from your training set, treating every word that occurs not more than once as an unknown token. Transform both of your training and test sets by introducing unknown and sentence boundary tokens. Assume that start and end symbols are always tagged as themselves, for example,
   `<S>/<S> a/DT cat/NN chased/VBN a/DT cat/NN </S>/</S>`
   After applying your transformation, print out the first sentences of both data sets.
2. **Baseline**:
   2.1. Implement the most common class algorithm (function `MostCommonClassBaseline`) that was introduced in class as a POS tagging baseline algorithm. Tag your test set according to this algorithm.
   2.2. Implement the function `ComputeAccuracy`: using the gold standard tags from the treebank, compute and report *sentence accuracy* (the percentage of the sentences that were tagged perfectly) and *tagging accuracy* (the percentage of the tags - excluding sentence boundary tokens - that were predicted correctly). This is the baseline accuracy that we will try to beat with our HMM tagger.
3. **Training:**
   3.1. Using the joint probability model that we introduced for the bigram HMM model,
   $$P(w_1^n, t_1^n) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$
   and a training set of $w_1, \ldots, w_n$, $t_1, \ldots, t_n$ (n words and n tags corresponding to each word in sequence) prove that the maximum likelihood estimates for the A and B matrices are the estimates that were introduced in class:
   $$P(w_i|t_i) = \frac{\mathrm{C}(w_i, t_i)}{\mathrm{C}(t_i)}, \quad P(t_j|t_i) = \frac{\mathrm{C}(t_i, t_j)}{\mathrm{C}(t_i)}$$
   3.2. Build a `BigramHMM` class by implementing the body of the methods in `pset3_template.py`.
      3.2.1. Implement the `Train` method for estimating A and B matrices and a tag dictionary that maps every word to a set of its candidate tags in the training set. Run it on the training set and train your HMM.
      3.2.2. Implement the `ComputePercentAmbiguous` method. Compute and report the *percent ambiguity* in your training set (the percentage of tokens in a data set that have more than one tag according to the tag dictionary). How many tags are there for the unknown token in the dictionary? List them.

3.2.3. Implement the `JointProbability` method for computing the HMM probability of a tagged sentence (given in 3.1). Print out the joint probability of the first tagged sentence of your training set.

4. **Testing:**
    4.1. Implement the `Viterbi` method of `BigramHMM` for POS tagging a test sentence using the tag dictionary and the A and B matrices estimated by `Train`.
    4.2. Implement the `Test` method of `BigramHMM`, which should apply Viterbi to your test set and predict new tags for every sentence. Using `ComputeAccuracy`, compute your *sentence accuracy* and *tagging accuracy*. Compare to the baseline and discuss your findings.
5. **Extra credit:** Build a confusion matrix and report on the most confused classes of your HMM. Using NLTK tools to look up what every tag means (try `nltk.help.upenn_tagset("JJ.*")` for example), discuss the potential reasons for some of these errors and implement your improvements.

Deliver the following by Wednesday Night 3/17/2016 02:59 AM.

1. A PDF write-up. All questions 1-5 attempted should have at least a paragraph describing your reasoning and final solution, explaining your decisions in detail. The write-up should include code blurbs if necessary.
2. Your code. This is a .py file that compiles and runs. It should demo the answers that you produced in your work.

How to submit your homework:

1. Log onto the Tufts server:
```
ssh your_username@homework.cs.tufts.edu
```
2. Navigate to the directory of your submission files and type the following command to submit all of your files together:
```
provide comp150nlp pset3 [file1 file2 ...]
```