CS 6320 Natural Language Processing
Homework 3: POS tagging
Due: Oct 10th, 11:59pm.

In this assignment, you will implement an HMM POS tagger (including both training and Viterbi decoding).
You will train your HMM from the POS annotated data and tag the test set, and write up a report about your findings.

The training and test data are from the Wall Street Journal corpus with POS tags (data from LDC).
Important note: This data is used for this homework only. Please don't spread the data and discard it once you finish your homework.

A. Training

Training file is available from:
http://www.hlt.utdallas.edu/~yangl/cs6320/homework/hw3_train

The training data has been manually labeled with POS tags. Each line in the training file corresponds to a sentence. A slash / is used to separate the word and its POS tag.

From the training data, you need to get the model parameters that you will need for testing. These include:

- the transition probabilities $P(T_2|T_1)$ (note: this is similar to the bigram LM you implemented in homework 2);
- the observation probabilities P(w|T), the probabilities of observing word w given the POS tag T;
- and the possible POS tags for a word.

You can define a format to use for your model file. You will read this model later for testing, so try to have a data structure and format that are easy for yourself.

B. Testing

Test data is available from:
http://www.hlt.utdallas.edu/~yangl/cs6320/homework/hw3_test_00

File format: in the test set, each line corresponds to a sentence, and there are only words.

For testing, you need to read your HMM model file, and perform Viterbi decoding to find the best POS tags for the given sentences.

Output format: Your output file should look like the training file (i.e., put the hypothesized POS tag after each word, followed by the slash).

Note:

- You will use a sentence as your observation sequence in decoding. Similarly for training, sentences will be treated as sequences. Don't cross sentence boundaries for training/testing.
- Let's add a dummy start symbol and an end one for each sentence. This way you don't need initial probability vector for HMM, and your transition probabilities will be cleaner.
- In training, for observation probabilities (P(w|T)), save the probabilities for each word, i.e., use word as index, rather than states/tags. During testing, you can easily find all the possible tags for the seen words.
- For seen words, only the seen tags in the training set need to be considered in decoding. This also means that for observation probabilities, you don't need smoothing.
- For transition probability, if you want to do smoothing, you can use a simple linear interpolation of bigram and unigram probabilities:
  $p_s(t_2|t_1)=\alpha * p_{ml}(t_2|t_1)+ (1-\alpha) *p_{ml}(t_2)$
- Related to **unknown words**:
  - One thing you need to pay attention is how to deal with unknown words in the test sentences. You can use some simple morphology information to determine the possible tags, or allow them to have more possible POS tags.
  - For training, you can just use maximum likelihood for P(w|t) for all the seen words, no need for discounting.
  - For an unknown word during testing, our goal is to assign a probability P(w|t). To do this, we are going to use the word itself to figure out P(t|w) first. For this, you can either use uniform distribution, or prior probabilities of P(t), or do some morphology analysis and assign more reasonable probabilities to different tags (this part, I'm going to leave to you). Once you have P(t|w) for different tags, we are going to use Bayes rule to get P(w|t) (=P(w)*P(t|w) / P(t)). For the

P(w) term, since it's the same for all the tags for this word position, it doesn't matter for Viterbi decoding results, so you can use a small number for that (or if you want, consider using a Good-turing discount for unigram probability of the words). Now you may understand why I said earlier you don't need to change P(w|t) for the seen words. It's okay that these are not "valid" probabilities, since we only care about the Viterbi decoding results. This is different from perplexity calculation, for which you do need to use valid probabilities.

- **In your output file, for the unknown word, please use a special symbol to indicate it's an unknown word. This way later for the evaluation part, you can measure performance separately for known/unknown words.**

C. Evaluation

Ref tag file is available from:
http://www.hlt.utdallas.edu/~yangl/cs6320/homework/hw3_test_ref_00

To measure your POS tagging accuracy, compare your output tags to the references and get the tagging accuracy.
For performance, please report three numbers: one is the overall accuracy for the entire test set; the other two are accuracy results for the known and unknown words separately.

What should your program look like?

- for training: your program -train train_file -model model_file
- for testing: your program -test test_file -model model_file -o output
- evaluation: your program -ref ref_file -sys your_output (this will print the system's performance in stdout)

Please use the options as specified above. This will help the TA to grade your homework.
If you don't follow these, we will deduct some points.

How to turn in your homework?

1. Please submit your homework via eLearning.

2. Submit your source code and executable, with proper readme files. You don't need to submit your model file.
3. Submit your report, with the experiment results on the test set and any findings you have (e.g., effect of different training size, handling unknown words, confusion matrix, comparison to the baseline of using the most frequent tag).

Extra credits:

You can get extra credits (or just for fun) to implement a POS tagger using a trigram model for the transition probability.
Think what needs to be changed for this when you use Viterbi decoding!