Homework Assignment #5

I implemented a Viterbi parts-of-speech tagger, trained and tested on the Wall Street Journal text provided on Moodle. The tagger uses Laplace smoothing and merges words that occurs only once in the training set into <UNKNOWN> to estimate the probability of unseen words. It then measures the tagging accuracies for different sizes of training data, different tagsets and whether smoothing is applied.

The program first allocates from the Wall Street Journal text data a training set and a test set. The tagger is trained on four different sizes of training data (90%, 70%, 50% and 30% of the text data) and tested on the remaining 10% of the text data not used for training. The function `hmm(training_sentences, reductedtagset)` takes a list of tagged sentences as a parameter, calculates both transition and emission probabilities with Laplace smoothing, and returns a tuple containing the transition and emission probabilities. For testing, the function `viterbi_tags(untagged_sentences, h)` is passed a list of untagged sentences and the probability tuple. In the case of unseen words, emission probability is multiplied by a very small number, .0001, to make sure the tagger does not give higher probability to unseen combinations of word and POS tag. Each cell also keeps a backpointer for backtracing. Another function, `maxsequence(probtable, tags)`, is then passed two arguments, the viterbi dynamic programming probability table and the list of all possible tags. It searches for the highest probability within the last column of the probability table, does backtrace and returns the most likely sequence of POS tags. To measure its accuracy, this sequence and the correct POS sequence obtained from the test data are passed to the function `compare(mytags, truetags, reducedtagset)`, which simply compares the two sequences and returns the number of correctly-tagged words divided by the total number of words.

The program then compares the accuracy of the tagger trained on different sizes of training data, different tagsets and whether Laplace smoothing is applied. They are all tested on the same test set. When trained on 90% of the data, it reaches almost 90% accuracy. As the training data size is reduced, the accuracy also decreases. This decrease is fairly slow, since even when trained on only 30% of the data, the accuracy still holds above 80%. The tagger does better without Laplace smoothing, although it might just be due to the fact that the test set is very similar to the training set (since both sets are taken from a Wall Street Journal corpus). The reduced tagset also improves tagging accuracy. In the reduced tagset, all verb types are collapsed into VB tag, all noun types into NN, adjective types into JJ and adverb types into RB. When trained on 90% of the WSJ data, the accuracy reaches 92% (Laplace applied) and 95% (no smoothing applied). The accuracies obtained from different sizes of training data, different tagsets and whether smoothing is done are summarized below:

Training data size:	90%	70%	50%	30%

Original tagset				
With Laplace smoothing	.8999	.8897	.8717	.8397
Without smoothing	.9386	.9351	.9247	.9053
Reduced tagset				
With Laplace smoothing	.9284	.9207	.9106	.8913
Without smoothing	.9569	.9532	.9447	.9319

The biggest challenge in building the tagger has been to get a reasonably accurate POS tag sequence. Initially, my tagger did very poorly and had an accuracy below .01. The tagger predicted some very weird sequence as the most likely sequence because I assigned too high probability for unseen words. The solution to this problem was to simply weigh down the probability for unseen words, for example, by multiplying it by a very small number, such as .0001.

The tagger has only been tested on the remaining 10% of the WSJ data not used for training. If I could build on what I've done so far, I would test the tagger on some other test sets from different genre and time period, then compare their accuracies using different smoothing methods and other POS tagging algorithms, such as the forward-backward algorithm and the unsupervised Baum-Welch algorithm.