CSE5525: Lab 2 Report

Team Members
Adam Stiff
Deblin Bagchi
Kristin Barber
Akin Solomon

Step 2:
For unseen words, a probability of .001 was used.

Step 4:

Tagger	Accuracy
In-built POS Tagger	99.77%
Bigram Tagger	86.54%
Bigram Tagger with Backoff	88.85%
Unigram Tagger	86.13%
Hmm Tagger using Viterbi	89%

Summary of Results:

We obtained competing results with the Unigram Tagger and Bigram Tagger with Backoff. The similarity to the Unigram and Bigram Tagger with Backoff is probably due to our uniform choice of 0.001 for the probability of unseen words. We expected the in built POS tagger to outperform our model due to its usage of MEMM's compared to our use of HMM's. HMM's only rely on the transition and emission probabilities compared with MEMM's which can make of use of multiple useful features. There's only so much useful knowledge that can be reasonably encoded within the probabilities themselves so HMM's are bound to lose valuable information.One way to improve our accuracy, other than increasing our training data size, would be to use an unseen word probability that was more representative of our training data.

Step 5:

Using the trained tagger from the first training set to label the second training set, and then re-estimating a new model based on the concatenation of the first training set with the newly labeled second training set to label the test set gave an error rate of **0.110756123536** (error rate indicating the number of mismatches between the labels assigned by the model and the correct labels for the test set).

The next suggestion was to continue re-estimating the labels for the second training set with this new model in an iterative fashion. The table below shows what happened to the error rate for the labels assigned by the model to the test set as the model was continually refined:

Iteration 1	0.110756123536
Iteration 2	0.168554555136
Iteration 3	0.171846258108
Iteration 4	0.171168554555
Iteration 5	0.171362184142
Iteration 6	0.171265369348
Iteration 7	0.171071739762
Iteration 8	0.171071739762

As can be seen from the table above, the error rate increases for the first two refinements, then decreases, increases and converges. The increase in error rate might be explained by the fact that we are labeling the test set based on a model that is not perfect (part of the model may have incorrect labelings since it is based in part on the assigned labels from the second training set) and these imperfections cause incorrect labelings to occur for the test set which propagate through the refinement phases.

Step 6:

At this point, we have calculated the forward and backward probability matrices for a given sentence, and used those to calculate the gamma and ksi matrices for the E-step. We have also implemented the update of the state transition probability matrix, but have run out of time to implement the emission probability matrix.

Running the HMM Part-of-Speech Tagger

The scripts which encompass this assignment are:

`prob1.py`: This script estimates the transition and emission probabilities under a naive Bayes' rule assumption which are then used by the Viterbi algorithm whn traversing the HMM.
`Viterbi.py`: The file defines a Viterbi class which encapsulates creating the model and defines findShortestPath() method which is what determines the most probable path through the HMM and assigns labels accordingly. **Step 5** is also part of this file.
`fwdback.py`: This file is where alpha and beta probabilities are calculated and instantiates a ForwardBackward object to run the algorithm given these probabilities.
`ForwardBackward.py`: This file defines a ForwardBackward class which implements the mechanics of the forward backward algorithm.

The commands to run are:

```
python Viterbi.py
python ForwardBackward.py
```