PROBLEM SET #3

MARIA MOYA

1. Text Preprocessing

We will consider a Bigram LM. I split the testing data by the last 3000 words and I remove traces on line 258. We need to account for words that occur in our test set but not our training data. Thus we convert any word in the training/test set that is not in our vocabulary V to $< UNK >$. From lines 18-29 I define a *getVoc* function that finds the vocabulary from the training set. I then define a function *PreprocessText* on lines 31-41 to impute the unknown tokens to $< UNK >$ given our vocabulary.

The first two sentences are:

1.) $< S >< UNK >$ Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 . $< /S >$

and

2.) $< S >$ At Tokyo , the $< UNK >$ index of $< UNK >< UNK >$ issues , which gained $< UNK >$ points Tuesday , added $< UNK >$ points to $< UNK >$. $< /S >$

2. Baseline

The baseline approach essentially tags every word with its most frequent tag. I define a function *MostCommonClassBaseline* on line 162. I begin by building a dictionary of counters. I then take the maximum to find the dictionary of the most common class or the tags that occurred the most given a word(hence, occurred the maximum amount of times).

Next I define another function *ComputeAccuracy* that iteratively counts the number of tokens, tokens that were tagged incorrectly and the number of incorrect tagged lines. This allows me to compute the percentage of the sentences that were tagged perfectly and the percentage of the tags - excluding sentence boundary tokens. I calculated the accuracy of the sentence by:

$$(1 - \frac{lineswithmistakes}{sizeoftestset}) * 100 = 7\%$$

And the tagging accuracy by:

$$(1 - \frac{incorrectlytaggedtokens}{Total\#oftokens - 2*(sizeoftestset)}) * 100 = 85\%$$

Note that the tagging accuracy is a bit lower than the standard 90% accuracy.

3. Training

Next we will consider the following HHM model:

$$P(w_1, ..., w_n, t_1, ..., t_n) = \prod_{i=1}^{n} P(t_i|t_{i-1}) \prod_{i=1}^{n} P(w_i|t_i)$$

where $P(t_i|t_{i-1})$ are the transition (bigram) probabilities for matrix A and $P(w_i|t_i)$ are the emission probabilities for matrix B where each conditional probability is estimated by MLE(which is again done by counts). These estimates are computed using lines 55-87. Next we have to consider a metric that accounts for ambiguity. We define ambiguity by words having more than one possible tag. For instance, the word book can have two different types of POS tags, you can book a flight which would be a verb or you can read a book which is a noun. I define a function *ComputePercentAmbiguous* that counts the number of tags associated with each given word. I then take the number of ambiguous words to the total number of tokens which was 39.54%

I then define a function *JointProbability* to compute the HMM probability of a tagged sentence which can be found in lines 103-109. The Joint probability of the first sentence is 2.13086363871e-49.

4. Testing

Testing Brute force would be incredibly inefficient because we would have to search through all possible tag sequences, thus 45^n possibilties. The Viterbi algorithm allows us to apply the HHM to testing data.This allows us to find the most probable sequence of tags(or hidden states) for a given sequences of observed states(which is our sequence of words).This basically says take the product of your previous viterbi(expect for your initial and end) times your transition and emission probabilities. We consider the following equation:

$$v_t(j) = \max_{1 \leq i \leq n} \; v_{t-1}(i) a_{ij} b_j(o_t)$$

I define my verbiti algorithm in lines 111-153 and test it on line 155. This produces an output of 16.9584245077 for the sentence accuracy and 90.0286215493 the tagging accuracy. This clearly gave a better accuracy than the baseline as expected since it gives us the best tag sequence in the entire sentence rather than simply tagging words by their most frequent tags.

5. Extra credit

Lastly I consider the confusion matrix from slide 85 which measures how often a tag i is mistagged as a tag j.I define a function called *buildConfusionMatrix* and *matrixmax* which searches through tag idicies i and j to find what the tag should be versus what it was estimated to be. This produces the following results for the most confused tag pair: NN (should be) JJ (estimated).

6. Output

```
<S> <UNK> Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
</S>
<S> At Tokyo , the <UNK> index of <UNK> <UNK> issues , which gained <UNK> points Tuesday ,
added <UNK> points to <UNK> . </S>
Percent tag ambiguity in training set is 39.54%.
Joint probability of the first sentence is 2.13086363871e-49.
Sanity Check  ========>>>> -426082.457238
------------- Most common class baseline accuracy -----------
The sentence accuracy is:  7.00218818381
The tagging accuracy is:  85.2137383436
-------------Bigram HMM accuracy ----------------------------
The sentence accuracy is:  16.9584245077
The tagging accuracy is:  90.0286215493
The most confused tag pair:  NN (should be) JJ (estimated)
```