Homework 3

1. Proof of MLE results
Let's first denote that $P(w_i \mid t_j) = \alpha_{ij}$, $P(t_j \mid t_k) = \pi_{jk}$, $C(w_i, t_j) = n_{ij}$ and $C(t_k, t_j) = m_{kj}$. We use N_w to denote total number of distinct words and N_t the total number of distinct tages. Then the likelihood function can be express as:

$$\begin{aligned} P(w^n, t^n) &= \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1}) \\ &= \prod_{i=1}^{N_w} \prod_{j=1}^{N_t} \prod_{k=1}^{N_t} (\alpha_{ij})^{n_{ij}} (\pi_{jk})^{m_{kj}} \end{aligned}$$

The log likelihood function has the form:

$$\log P(w^n, t^n) = \sum_{i=1}^{N_w} \sum_{j=1}^{N_t} \sum_{k=1}^{N_t} \left[n_{ij} \log(\alpha_{ij}) + m_{kj} \log(\pi_{jk}) \right]$$

Since this is a constrained optimization problem, we need to introduce the Lagrange Multiplier and the unconstrained objective function is the following:

$$f = \sum_{i=1}^{N_w} \sum_{j=1}^{N_t} \sum_{k=1}^{N_t} \left[n_{ij} \log(\alpha_{ij}) + m_{kj} \log(\pi_{jk}) \right] + \sum_{j=1}^{N_t} \lambda_j \Big(\sum_{i=1}^{N_w} \alpha_{ij} - 1 \Big) + \sum_{k=1}^{N_t} \mu_k \Big(\sum_{j=1}^{N_t} \pi_{jk} - 1 \Big)$$

Take the derivative of f w.r.t. α_{ij} and set it to zero we have:

$$\frac{n_{ij}}{\alpha_{ij}} = \lambda_j$$

which leads to

$$\alpha_{ij} = \frac{n_{ij}}{\lambda_j}$$

Since we know that $\sum_{i=1}^{N_w} \alpha_{ij} = 1$, then $\lambda_j = \sum_{i=1}^{N_w} n_{ij} = C(t_j)$. Hence:

$$P(w_i \mid t_j) = \alpha_{ij} = \frac{n_{ij}}{\lambda_j} = \frac{C(w_i, t_j)}{C(t_j)}$$

Again, take the derivative of f w.r.t π_{jk} and set it to zero we have:

$$\frac{m_{kj}}{\pi_{jk}} = \mu_k$$

With a similar line of reasoning, $\mu_k = \sum_{j=1}^{N_t} m_{kj} = C(t_k)$ and hence

$$P(t_j \mid t_k) = \pi_{jk} = \frac{m_{kj}}{\mu_k} = \frac{C(t_k, t_j)}{C(t_k)}$$

2. This preprocessing is pretty much the same as what we've done in hw2. The general idea is the same, the only differences are details in implementation, like in this problem set, every element in each line of the dataset in a tuple comprising of a word and its corresponding tag, we need to make sure we're dealing with the word instead of the tuple.

3. This part is relatively easy compared with the Viterbi, the basic idea is to tag the word with the most probable tag learned from the training set. This baseline algorithm is reasonable, from my perspective, since when we want to tag a word, the first idea is to use the most common tag we know for this word. However, this tagging scheme only considers the correlation between word and tag while ignoring tag-tag correlation, which is why hidden markov model theoretically performs better. Then computing accuracy is nothing but error count divided by the total number of sentences/tags.

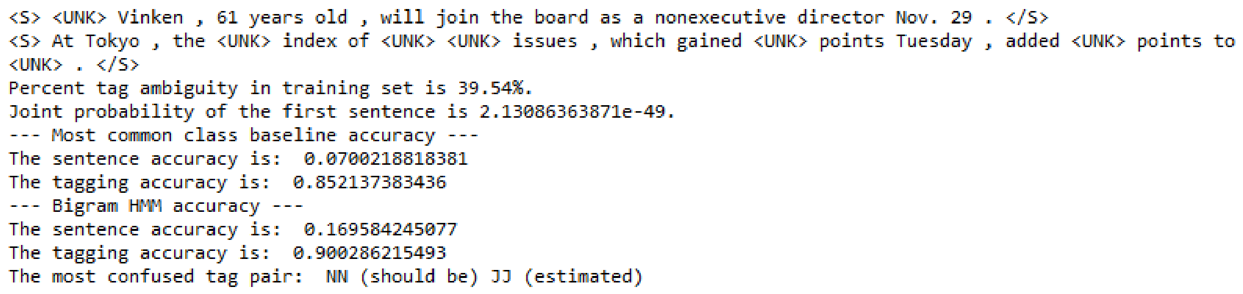
4. The proof of MLE formula is provided on the first page. It's pretty simple and straight forward, basically solving an constrained optimization problem with everything being linear. For implementing those A and B matrices, what I did is using a dictionary with keys being tuples(word-tag or tag-tag) to store the counts. Another way to do it is to use a nested dictionary - a dictionary of dictionaries of counts.
   The concept of *percent ambiguity* is a little vague, I'm not sure whether we should consider every unique word or literally every word in the training set. I chose to work on every word in the training set, from my perspective, this is more reasonable because this value can be used to measure how many words are non-trivial in the training set.
   When calculating the joint probabilities, I use log probability and then convert it back to normal probability. The reason why I implemented it this way is to reduce the risk of underflow.
   All results are attached in the back.

5. This is probably the most complicated algorithm we've implemented so far in this course, plus I didn't find the pesudo code that helpful. The 's' and 't' are counter intuitive at least for our homework since I usually confuse 't' with 'tags' instead of 'states'. With a dictionary of word to tags, the computation load of Viterbi can be greatly reduced, but still I noticed that a lot of paths have zero probability. I tried to ignore that tag in that state and its following paths, but for several sentences, the backtracking fails. One way to solve this is to use smoothing for both transition and emission probabilities but that requires a lot more computations.
   From the results, the hidden markov model helps increase tagging accuracy by 5% and the sentence accuracy by 10%.

6. From the confusion matrix, the most confused classes are 'NN' and 'JJ'. It seems 'NN' stands for noun and 'JJ' for adjective. If you think about this, it's reasonable sometimes that the algorithm incorrectly tags a noun as an adjective, like when you

have a noun right in front of a noun, it’s hard to distinguish from an adjective. I don’t know if there are ways to improve this, probably higher n-grams.

Results:



```
<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.
--- Most common class baseline accuracy ---
The sentence accuracy is:  0.0700218818381
The tagging accuracy is:  0.852137383436
--- Bigram HMM accuracy ---
The sentence accuracy is:  0.169584245077
The tagging accuracy is:  0.900286215493
The most confused tag pair:  NN (should be) JJ (estimated)
```