CSCI 544
Homework 3

Name: Yamini Haripriya Kanuparthi
Email ID: kanupart@usc.edu
USC ID: 9195653004

Task 1: Vocabulary Creation :

As part of Task 1 of the task, I made a script that uses the training dataset to build a vocabulary. As the script reads the input, it keeps track of how often each word appears and what part of speech it belongs to. Then, it filters the vocabulary based on a frequency threshold that has already been set. Words that happen less often than the threshold are changed with a special token "<unk>." This is an important step for dealing with things that the model doesn't know that it might find in future data. The finished vocabulary is saved in a text file called "vocab.txt." This file has a list of all the words, their corresponding index, and how often they are used, sorted from most common to least common. The first item in this file is set aside for the "^" token, which shows how many times it appears in all words that are less common than the threshold frequency. This process has helped me build a large vocabulary that will be very helpful in teaching the HMM how to correctly tag parts of speech.

What is the selected threshold for unknown words replacement?
Threshold I set is **2**.

What is the total size of your vocabulary and what is the total occurrences of the special token '< unk >' after replacement?

```
Vocabulary size after replacement: 23183
Total occurrences of '<unk>': 20011
```

Task 2: Model Learning

As part of the model learning part of the task, I built functions that use part-of-speech tagging training data to figure out the HMM's transition and emission probabilities. The script reads each line of input data and updates the counts of changes between tags and emissions of words given tags. It takes extra care with the beginning of sentences and words that aren't in the dictionary. Once you have the raw counts, you normalize them by the overall counts to get the probabilities. In the last step, these possibilities are stored in two dictionaries. State changes are shown by tag pairs, and emissions are shown by tag-word pairs. After that, these dictionaries are turned into a JSON file called hmm.json, which contains the learned HMM parameters. The number of emission and transition factors is shown by the script's output, which shows how complicated the learned model is.

How many transition and emission parameters in your HMM?

```
Number of emission parameters: 30303
Number of transition parameters: 1392
```

```
Number of transition parameters (excluding start): 1351
HMM model saved to hmm.json
```


```
'Transition": {
    "(start,NNP)": 0.1978962241934218,
    "(NNP,NNP)": 0.3782645420509543,
    "(NNP,,)": 0.13846908958086018,
    "(,,CD)": 0.021234939759036144,
    "(CD,NNS)": 0.15775891730703062,
    "(NNS,JJ)": 0.017196978862406887,
    "(JJ,,)": 0.029129343105320303,
    "(,,MD)": 0.010542168674698794,
    "(MD,VB)": 0.7990886934407121,
    "(VB,DT)": 0.22209580603397544,
    "(DT,NN)": 0.4734877816566169,
    "(NN,IN)": 0.24741637524111218,
    "(IN,DT)": 0.32807784039342325,
    "(DT,JJ)": 0.21834338305299905,
    "(JJ,NN)": 0.4491042345276873,
    "(NN,NNP)": 0.009519030219392476,
    "(NNP,CD)": 0.019176330928682313,
```

```
"Emission": {
    "(NNP,Pierre)": 6.848689617386 54e-05,
    "(NNP,Vinken)": 2.2828965391288468e-05,
    "(,,,)": 0.9999139414802065,
    "(CD,61)": 0.0007168253240050465,
    "(NNS,years)": 0.019530237301024905,
    "(JJ,old)": 0.003613599348534202,
    "(MD,will)": 0.3138709335593939,
    "(VB,join)": 0.0015693044058221193,
    "(DT,the)": 0.5016439225642653,
    "(NN,board)": 0.0023287907538381922,
    "(IN,as)": 0.0353954283543342,
    "(DT,a)": 0.2341478895588702,
    "(JJ,nonexecutive)": 0.00010179153094462541,
    "(NN,director)": 0.002422883309548826,
    "(NNP,Nov.)": 0.0026709889507807506,
    "(CD,29)": 0.0021218029590549374,
    "(.,.)": 0.9886228651373967,
    "(NNP,Mr.)": 0.044014245274404167,
```


Task 3: Greedy Decoding with HMM

In the greedy decoding stage, I have developed an algorithm to forecast part-of-speech tags for phrases. The algorithm uses the HMM parameters learned earlier, specifically the transition and emission probabilities, to carry out the tagging process. It follows a greedy heuristic by selecting the most probable tag for each word based on the preceding tag and the word itself. Decoding is performed individually for both the development and test datasets. The predicted tags are saved to an output file for evaluation, which is then compared to a gold-standard file using a given evaluation script. This provides the accuracy of the greedy decoding algorithm on the development data, serving as an indicator of the HMM's performance before being used on unseen test data.

What is the accuracy on the dev data?

```
The accuracy on validation data using the greedy method is
total: 131768, correct: 123203, accuracy: 93.50%
```

Task 4: Viterbi Decoding with HMM

For the fourth task, I applied the Viterbi decoding technique to improve the accuracy of part-of-speech tagging in the Hidden Markov Model. The program creates a matrix to save the probabilities of the most probable paths to each state at every time step. It then utilizes backpointers to reconstruct the best path of states after processing the full sentence. The Viterbi algorithm is more computationally demanding than the previous greedy technique, but it is also more precise as it evaluates the optimal sequence of tags for the entire sentence instead of making individual decisions. After annotating the words in the development and test datasets, I assessed the model's performance using an evaluation script that generated a quantitative measure of its accuracy on the development data. Viterbi decoding is anticipated to outperform greedy decoding due to its consideration of the complete context of the tag sequence.

What is the accuracy on the dev data?

```
The accuracy on validation data using the viterbi decoding method is
'1\tThat\tDT' '38\t.\t.' 131751
total: 131751, correct: 124912, accuracy: 94.81%
```