CSCI-544 HOMEWORK 2

Name: Vorapoom Thirapatarapong

In [1]:
```
import task1 as t1
import task2 as t2
import task3 as t3
import task4 as t4
from collections import defaultdict
import json
```

In [2]:
```
path_train = 'data/train'
path_dev = 'data/dev'
path_test = 'data/test'
occ_thres = 0
```

Task1

- What is the selected threshold for unknown words replacement?
  The selected threshold for unknown words replacement used is 0, meaning that I chose to not replace any word with low frequency to be unknown. This is because the threshold at 0 gives the highest accuracy on the dev data on both greedy and viterbi decoding on task3 and task4. Results of different thresholds used can be seen below.
- What is the total size of your vocabulary?
  The total size of vocaulary is 43,193 words.
- What is the total occurrences of the special token '< unk >' after replacement?
  As mentioned earlier that I chose to use the unknown words threshold of 0, there are 0 occurrences of the special token in my vocabulary.
- Explanation
  Task1 was done in 2 simple steps. First, each word of the input file is kept in a dictionary along with its occurrences. Next, each word in the dictionary is checked whether its occurrences are above the threshold. If not, those words will be added to the < unk > token instead.

In [3]:
```
corpus = t1.create_corpus(path_train, occ_thres)
t1.query(corpus, occ_thres)
```

```
Threshold of unknown word replacement = 0
Total size of vocabulary = 43193
Total occurrences of special token < unk > after replacement = 0
```

Task2

- How many transition parameters in your HMM?
  There are 1,392 transition parameters. These include the initial transition parameters as

well.

- How many emission parameters in your HMM?
  There are 50,286 emission parameters in total.
- Explanation
  Firstly, the train data is transformed to a list of lists for ease of processing (first layer represents each sentence and the layer inside represents each word in a particular sentence). Then, the nested for-loop is performed to collect the information of transition and emission parameters. When these raw values are collected, the final values which are joint/conditional probabilities are calculated. Note that 2 versions of emission parameters are being collected: the original word, and a lowercase word which will be used when training the greedy decoding.

In [4]:
```
data = t2.read_file_to_list(path_train)
transition, emission, list_pos, emission_lower = t2.get_emission_transition(d
t2.query(transition, emission)
```

```
Number of transition parameters = 1392
Number of emission parameters = 50286
```

Task3

- What is the accuracy of Greedy Decoding on the dev data?
  The highest accuracy achieved on dev data is 93.78%
- Explanation
  In the greedy decoding, when a new sentence comes in for a prediction, firstly the code will predict the POS of the first word based on the product of the emission prob and the initial transition prob if this first word is in the corpus. If not, the code will try to check if the lowercase of this first word is in corpus (found a few cases where this is the case and it helps improve the accuracy) and find POS based on the same method. If none of the above cases happen, POS prediction will be NNP. After the prediction of the first word, subsequent words' POS are predicted similarly using the product of the emission prob and the transition prob. However, if the word is not in the corpus (emission prob = 0), the POS predicted will be the one that gives the highest transition probability based on the previous word's POS.

In [5]:
```
train = t2.read_file_to_list(path_train)
dev = t2.read_file_to_list(path_dev)
corpus = t1.create_corpus(path_train, occ_thres)
corpus = list(corpus.keys())
transition, emission, list_pos, emission_lower = t2.get_emission_transition(t
transition_init = {k: v for k, v in transition.items() if k[0] == 'start'}
transition_def = t3.get_transition_default(transition)
```

In [6]:
```
prediction_dev = t3.make_prediction(dev, transition, emission, list_pos, corp
acc = t3.get_accuracy(dev, prediction_dev)
print('Greedy Decoding: Accuracy on dev data =', acc)
```

```
Greedy Decoding: Accuracy on dev data = 0.937799769291482
```

Task4

- What is the accuracy of Viterbi Decoding on the dev data?
  The highest accuracy achieved on dev data is 94.36%
- Explanation
  In the Viterbi decoding, when a new sentence comes in for a prediction, firstly the code will calculate the probability of each POS of the first word from the product of emission and initial transition probabilities in a dictionary. The code will only store the values for POS that exist for that first word in the emission parameters. However, if the word is not in the corpus, the default dictionary will be used (initial transition parameters). For the subsequent words, for each of the possible POS, the dictionary will be updated to store the maximum probability of each of the previous words' POS. Finally, the sequence that gives the highest probability at the end will be used as a prediction.

In [7]:
```
transition_init = {k[1]: (v, [k[1]]) for k, v in transition.items() if k[0] =
transition_def = t4.get_transition_default(transition)
```

In [8]:
```
prediction_dev = t4.make_prediction(dev, transition, emission, list_pos, corp
acc = t4.get_accuracy(dev, prediction_dev)
print('Viterbi Decoding: Accuracy on dev data =', acc)
```

```
Viterbi Decoding: Accuracy on dev data = 0.9436206059134236
```

Test different threshold

In [9]:
```
# greedy
train = t2.read_file_to_list(path_train)
dev = t2.read_file_to_list(path_dev)
test = t2.read_file_to_list(path_test)

thres_list = [0, 1, 3, 5, 10, 20]
print('Threshold performance variation: greedy decoding')
for thres in thres_list:
    # get corpus
    corpus = t1.create_corpus(path_train, thres)
    corpus = list(corpus.keys())

    # get transition / emission parameters
    transition, emission, list_pos, emission_lower = t2.get_emission_transiti
    transition_init = {k: v for k, v in transition.items() if k[0] == 'start'
    transition_def = t3.get_transition_default(transition)


    # predict dev
    prediction_dev = t3.make_prediction(dev, transition, emission, list_pos,

    # get accuracy
    acc = t3.get_accuracy(dev, prediction_dev)
    print('threshold:', thres, ', dev accuracy: ', acc)
```

```
Threshold performance variation: greedy decoding
threshold: 0 , dev accuracy:  0.937799769291482
threshold: 1 , dev accuracy:  0.9298919312731467
threshold: 3 , dev accuracy:  0.9193051423714407
threshold: 5 , dev accuracy:  0.9105245583146135
threshold: 10 , dev accuracy:  0.892121000546415
threshold: 20 , dev accuracy:  0.8661890595592253
```

In [10]:

```
# viterbi

thres_list = [0, 1, 3, 5, 10, 20]
print('Threshold performance variation: viterbi decoding')
for thres in thres_list:
    # get corpus
    corpus = t1.create_corpus(path_train, thres)
    corpus = list(corpus.keys())

    # get transition / emission parameters
    transition, emission, list_pos, emission_lower = t2.get_emission_transiti
    transition_init = {k[1]: (v, [k[1]]) for k, v in transition.items() if k[
    transition_def = t4.get_transition_default(transition)


    # predict
    prediction_dev = t4.make_prediction(dev, transition, emission, list_pos,

    # get accuracy
    acc = t4.get_accuracy(dev, prediction_dev)
    print('threshold:', thres, ', dev accuracy: ', acc)
```

```
Threshold performance variation: viterbi decoding
threshold: 0 , dev accuracy:  0.9436206059134236
threshold: 1 , dev accuracy:  0.9349310910084391
threshold: 3 , dev accuracy:  0.9223863153421165
threshold: 5 , dev accuracy:  0.9128544107825876
threshold: 10 , dev accuracy:  0.8925535790176674
threshold: 20 , dev accuracy:  0.8641703600267137
```