CMPE561: Natural Language Processing

Assignment 2: Part-of-Speech Tagging using Hidden Markov Models

Name: Doruk Kilitçioğlu
Student ID: 2012400183

1. Introduction

This assignment felt very informative and satisfactory to implement. The algorithms were crystal clear, and they produced results immediately. The code and the report in Jupyter Notebook format can be found at GitHub (https://github.com/Xyllan/boun_cmpe561_assignments).

Below are the incremental steps I have taken during the assignment.

2. CoNLL Parsing

Before I could do anything, I had to be able to parse the input files. I created a parser that would take a CoNLL file, and return its sentences. Each word of the sentence is a tuple of word form, cpostag, and postag. The words which have form fields of '_' are ignored as per the instructor's instructions, all throughout the project.

In [1]:

```
import conll_parser as cpar

path = 'metu_sabanci_cmpe_561/train/turkish_metu_sabanci_train.conll'
train_sentences = cpar.get_sentences(path)
print(train_sentences[0])
```

```
[('Hayır', 'Adv', 'Adv'), (',', 'Punc', 'Punc'), ('şarklı', 'Noun', 'Zero'), ('değil', 'Verb', 'Verb'), ('.', 'Punc', 'Punc')]
```

After looping over the data, we get a list of cpostags:

In [2]:

```
print(cpar.tag_list(train_sentences, tag_ind = cpar.tag_ind('cpostag')))
```

```
{'Ques', 'Zero', 'Interj', 'Pron', 'Noun', 'Adj', 'Punc', 'Adv', 'Postp', 'Det', 'Num', 'Dup', 'Conj', 'Verb'}
```

And a list of postags:

In [3]:

```
print(cpar.tag_list(train_sentences, tag_ind = cpar.tag_ind('postag')))
```

```
{'Pron', 'APresPart', 'NPastPart', 'Verb', 'Num', 'Postp', 'AFutPart', 'QuesP', 'PersP', 'Punc', 'Adv', 'Dup', 'Det', 'Ord', 'NInf', 'Interj', 'Adj', 'Card', 'Range', 'Conj', 'Distrib', 'Ques', 'Zero', 'Prop', 'DemonsP', 'Noun', 'Real', 'APastPart', 'NFutPart', 'ReflexP'}
```

3. HMM PoS Tagger

The HMM PoS Tagger is a basic bi-gram part-of-speech tagger. The PoS tags are hidden, while the actual words are observable. We need to estimate the transition probabilities of states and observation likelihoods words in the given states, using the training data. Since we are using a bi-gram model, the transition probability only depends on the previous tag. The transition probability from tag T_{t-1} to tag T_t is estimated by:

$$P(T_t | T_{t-1}) = \frac{C(T_{t-1}, T_t)}{C(T_{t-1})}$$

where C is the number of occurences of the given tuple in the training data.

Similarly, the observation likelihood of any word W_t given a tag T_t is estimated by:

$$P(W_t | T_t) = \frac{C(W_t, T_t)}{C(T_t)}$$

Training the HMM PoS tagger therefore refers to calculating these probabilities for all words and tags. Since that creates two very sparse matrices, I decided to keep the actual counts in memory and do the calculations online.

During training, I also calculate the counts regarding the start and end states, since they are used in the Viterbi algorithm for calculating the highest likelihood tag set.

4. Task 1

After building the training portion of the aforementioned PoS tagger, the **train_hmm_tagger** program is pretty much finished. The instructions to run are in the README, but it is suffice to say that the program reads a training file, calculates the counts, and stores them to be used in the second task.

In [4]:

```
import train_hmm_tagger as hmm_train

tag_ind = cpar.tag_ind('cpostag')
hmm = hmm_train.HMM(cpar.tag_list(train_sentences, tag_ind), tag_ind)
hmm.train(train_sentences)

print('Adv count:',hmm.tag_count('Adv'))
print('Adv to Punc count:',hmm.tag_pair_count('Adv','Punc'))
print('\'Hayır\' as Adv count:',hmm.word_tag_count('Hayır','Adv'))
```

```
Adv count: 2673
Adv to Punc count: 390
'Hayır' as Adv count: 5
```

When running the two tasks separately, the trained HMM model is actually saved to a file called **hmm.conf**. This is not necessary here as we can keep everything in memory.

5. Task 2

In the second task, we implement the Viterbi algorithm. The algorithm is not explained in detail for the sake of brevity, and it already is properly commented in the code. In short we find the highest likelihood path from the start node to the end node, using the transition probability and the observation likelihood at each step. Since the probability of observing no words at the beginning and end states is 1, they are omitted. We say that the likelihood of a given tag list to the given word observations is *proportional to*:

$$P(END|\,T_n)\prod_{t=1}^{n}P(T_t|\,T_{t-1})P(W_t|\,T_t)$$

where $t = 1..n$ is the word index, and $T_0 = START$.

When expressed in log-space, the log-likelihood becomes a sum of the individual log transitions and log observation likelihoods. We find the highest likelihood state sequence and use it as our predicted PoS tag sequence.

One thing of note is how we handle a probability of 0 in the tagger. For each word in the sentence, we calculate the logarithm $P(T_t|\,T_{t-1})P(W_t|\,T_t)$ value. If that joined probability is zero (if the log probability is negative infinity), we plug in the value -10^{10}.If we have 0 probability for all possible word/tag combinations when trying to choose a tag, we would end up with a probability of 0 for the rest of the tagging, and that would mean we would be choosing random tags (since their cumulative probabilities are all the same). Instead, we are using a value where the rest of the calculations still affect the cumulative probability, and since the value is smaller than what any other calculations will produce, it does not disrupt our calculations.

In [5]:

```
import hmm_tagger

validation_path = 'metu_sabanci_cmpe_561/validation/turkish_metu_sabanci_val.conll'
test_sentences = cpar.get_sentences(validation_path)
pt_sentences = hmm_tagger.pos_tag(hmm, test_sentences)

print(pt_sentences[20])
```

```
[('Kalabalıktaki', 'Noun', 'Noun'), ('insanlar', 'Noun', 'Noun'), ('da', 'Conj', 'Conj'), ('onu', 'Pron', '
Pron'), ('dinlermiş', 'Noun', 'Noun'), ('.', 'Punc', 'Punc')]
```

5.1 Unknown Words

For unknown words, I tried two basic approaches:

- I assigned the most common tag to the unknown words. In most cases, this tag ended up being NOUN. While there definitely are some heuristics we are missing with this approach, it works very well compared to the random approach.
- I looked at the minimum edit (Levenshtein) distance between the vocabulary and the unknown word. This felt like it would be prone to false corrections, and it took way too much time (even when using dynamic programming) to go through the whole vocabulary each time. The results are not written here for the sake of brevity, but they are not an improvement over the most common tag solution, and take exponentially more time (~10 minutes versus the most common tag approach's ~1 second).

6. Task 3 & Results

The evaluation part is very straightforward and is pretty much a copy of the evaluation scheme of the first assignment (just calculating accuracy instead of the other indicators), so I won't be talking about the implementation. Between task 2 and task 3, the output is written to a file, but that is not necessary in this report since we can keep them in memory.

The only thing of significant change from the other assignment is that the stats for known and unknown words are separated, so that we can report them separately.

6.1 CPOSTAG Results

6.1.1 Unknown words

In [6]:

```
from evaluate_hmm_tagger import Tester

t = Tester(hmm.tags)
t.build(test_sentences, pt_sentences, hmm.tag_ind, vocab = hmm.vocab)

t.print_acc(0)
t.print_conf(0)
```

```
Accuracies:
Overall Accuracy: 0.662172878668
Ques Accuracy: 1.0
Zero Accuracy: 1.0
Interj Accuracy: 1.0
Pron Accuracy: 0.998413957177
Noun Accuracy: 0.662172878668
Adj Accuracy: 0.903251387787
Punc Accuracy: 1.0
Adv Accuracy: 0.965107057891
Postp Accuracy: 0.999206978588
Det Accuracy: 1.0
Num Accuracy: 0.990483743061
Dup Accuracy: 1.0
Conj Accuracy: 1.0
Verb Accuracy: 0.805709754163
Confusion matrix:
  Tags: ['Ques', 'Zero', 'Interj', 'Pron', 'Noun', 'Adj', 'Punc', 'Adv', 'Postp', 'Det', 'Num', 'Dup', 'Con
j', 'Verb']
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   2   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 835   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 122   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  44   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   1   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  12   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 245   0   0   0   0   0   0   0   0   0]]
```

We can see that, due to the high number of nouns, putting in nouns where we don't know the word works surprisingly well.

6.1.2 Known Words

In [7]:

```
t.print_acc(1)
t.print_conf(1)
```

```
Accuracies:
Overall Accuracy: 0.948063005534
Ques Accuracy: 1.0
Zero Accuracy: 1.0
Interj Accuracy: 1.0
Pron Accuracy: 0.990634312473
Noun Accuracy: 0.977862920392
Adj Accuracy: 0.970200085143
Punc Accuracy: 1.0
Adv Accuracy: 0.984248616433
Postp Accuracy: 0.99276287782
Det Accuracy: 0.990208599404
Num Accuracy: 0.997871434653
Dup Accuracy: 1.0
Conj Accuracy: 0.997871434653
Verb Accuracy: 0.994465730098
Confusion matrix:
  Tags: ['Ques', 'Zero', 'Interj', 'Pron', 'Noun', 'Adj', 'Punc', 'Adv', 'Postp', 'Det', 'Num', 'Dup', 'Con
j', 'Verb']
[[  1   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   2   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0  72   3   1   0   0   0   0   0   0   0   0]
 [  0   0   0   2 678  19   0   1   3   0   0   0   2   2]
 [  0   0   0   2  17 236   0   3   2   3   0   0   0   4]
 [  0   0   0   0   0   0 530   0   0   0   0   0   0   0]
 [  0   0   0   7   2   7   0 152   8   4   0   0   0   0]
 [  0   0   0   0   0   2   0   2  68   0   0   0   0   0]
 [  0   0   0   4   0   4   0   3   0 191   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   5  16   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   3   0   0   0   0   0   0   0   0 129   0]
 [  0   0   0   0   1   6   0   0   0   0   0   0   0 152]]
```

We can see that the accuracy is much higher in the known words case, presumably due to the fact that we are using the non-lemmatized versions of the words, which means that most of the words have only one actual tag that they are used as. The agglutinative nature of Turkish means there are a lot of unique word forms, which we are using as our vocabulary. These unique word forms end up having different PoS tags according to their suffixes. Obviously, this brings up our accuracy to levels that would not have been possible have we been using English.

6.1.3 All Words

In [8]:

```
t.print_acc(2)
t.print_conf(2)
```

```
Accuracies:
Overall Accuracy: 0.848199445983
Ques Accuracy: 1.0
Zero Accuracy: 1.0
Interj Accuracy: 1.0
Pron Accuracy: 0.993351800554
Noun Accuracy: 0.867590027701
Adj Accuracy: 0.946814404432
Punc Accuracy: 1.0
Adv Accuracy: 0.97756232687
Postp Accuracy: 0.995013850416
Det Accuracy: 0.993628808864
Num Accuracy: 0.995290858726
Dup Accuracy: 1.0
Conj Accuracy: 0.998614958449
Verb Accuracy: 0.928531855956
Confusion matrix:
  Tags: ['Ques', 'Zero', 'Interj', 'Pron', 'Noun', 'Adj', 'Punc', 'Adv', 'Postp', 'Det', 'Num', 'Dup', 'Con
j', 'Verb']
[[   1    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    2    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0   72    5    1    0    0    0    0    0    0    0    0]
 [   0    0    0    2 1513   19    0    1    3    0    0    0    2    2]
 [   0    0    0    2  139  236    0    3    2    3    0    0    0    4]
 [   0    0    0    0    0    0  530    0    0    0    0    0    0    0]
 [   0    0    0    7   46    7    0  152    8    4    0    0    0    0]
 [   0    0    0    0    1    2    0    2   68    0    0    0    0    0]
 [   0    0    0    4    0    4    0    3    0  191    0    0    0    0]
 [   0    0    0    0   12    0    0    0    0    5   16    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    3    0    0    0    0    0    0    0    0  129    0]
 [   0    0    0    0  246    6    0    0    0    0    0    0    0  152]]
```

Here we can see the effect of assigning noun to all unknown words in the confusion matrix. There is a trade-off between using word forms as our vocabulary and reaping the benefits of having one PoS tag each word, and having an abundance of unknown words, or using the word lemmas as our vocabulary and having more trouble tagging each word, while knowning more of the words. That is presumably why our training data contains different entries for word forms and word lemmas from time to time, to get the best of both words, but that also creates a difficulty in separating the word forms from their lemmas.

6.2 POSTAG Results

6.2.1 Unknown Words

In [9]:

```
pos_tag_ind = cpar.tag_ind('postag')
pos_hmm = hmm_train.HMM(cpar.tag_list(train_sentences, pos_tag_ind), pos_tag_ind)
pos_hmm.train(train_sentences)
ppt_sentences = hmm_tagger.pos_tag(pos_hmm, test_sentences)

pos_tester = Tester(pos_hmm.tags)
pos_tester.build(test_sentences, ppt_sentences, pos_hmm.tag_ind, vocab = pos_hmm.vocab)

pos_tester.print_acc(0)
pos_tester.print_conf(0)
```

```
Accuracies:
Overall Accuracy: 0.506740681998
Pron Accuracy: 1.0
APresPart Accuracy: 0.980967486122
NPastPart Accuracy: 0.984139571768
Verb Accuracy: 0.840602696273
Num Accuracy: 1.0
Postp Accuracy: 0.999206978588
AFutPart Accuracy: 0.995241871531
QuesP Accuracy: 1.0
PersP Accuracy: 1.0
Punc Accuracy: 1.0
```

```
Adv Accuracy: 0.965107057891
Dup Accuracy: 1.0
Det Accuracy: 1.0
Ord Accuracy: 0.999206978588
NInf Accuracy: 0.912767644726
Interj Accuracy: 1.0
Adj Accuracy: 0.942902458366
Card Accuracy: 0.991276764473
Range Accuracy: 1.0
Conj Accuracy: 1.0
Distrib Accuracy: 1.0
Ques Accuracy: 1.0
Zero Accuracy: 0.947660586836
Prop Accuracy: 0.966693100714
DemonsP Accuracy: 0.999206978588
Noun Accuracy: 0.506740681998
Real Accuracy: 1.0
APastPart Accuracy: 0.990483743061
NFutPart Accuracy: 0.992069785884
ReflexP Accuracy: 0.999206978588
Confusion matrix:
  Tags: ['Pron', 'APresPart', 'NPastPart', 'Verb', 'Num', 'Postp', 'AFutPart', 'QuesP', 'PersP', 'Punc', 'A
dv', 'Dup', 'Det', 'Ord', 'NInf', 'Interj', 'Adj', 'Card', 'Range', 'Conj', 'Distrib', 'Ques', 'Zero', 'Pro
p', 'DemonsP', 'Noun', 'Real', 'APastPart', 'NFutPart', 'ReflexP']
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  24   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  20   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0 201   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   6   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  44   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0 110   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  72   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  11   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  66   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  42   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0 639   0   0   0   0]
```

```
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  12   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  10   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0]]
```

We can see that the overall accuracy has dropped when compared to the cpostag version, since the tags are more detailed and simply assigning one tag to all loses its power. It is an expected result of our approach.

6.2.2 Known Words

In [10]:

```
pos_tester.print_acc(1)
pos_tester.print_conf(1)
```

```
Accuracies:
Overall Accuracy: 0.939123031077
Pron Accuracy: 0.997445721584
APresPart Accuracy: 1.0
NPastPart Accuracy: 0.999574286931
Verb Accuracy: 0.988931460196
Num Accuracy: 1.0
Postp Accuracy: 0.99318859089
AFutPart Accuracy: 0.999574286931
QuesP Accuracy: 0.995317156237
PersP Accuracy: 0.997871434653
Punc Accuracy: 1.0
Adv Accuracy: 0.98595146871
Dup Accuracy: 1.0
Det Accuracy: 0.990634312473
Ord Accuracy: 1.0
NInf Accuracy: 0.998297147722
Interj Accuracy: 1.0
Adj Accuracy: 0.97190293742
Card Accuracy: 0.997445721584
Range Accuracy: 1.0
Conj Accuracy: 0.998297147722
Distrib Accuracy: 1.0
Ques Accuracy: 1.0
Zero Accuracy: 0.985525755641
Prop Accuracy: 0.998722860792
DemonsP Accuracy: 0.996594295445
Noun Accuracy: 0.983397190294
Real Accuracy: 1.0
APastPart Accuracy: 0.999574286931
NFutPart Accuracy: 1.0
ReflexP Accuracy: 1.0
Confusion matrix:
  Tags: ['Pron', 'APresPart', 'NPastPart', 'Verb', 'Num', 'Postp', 'AFutPart', 'QuesP', 'PersP', 'Punc', 'A
dv', 'Dup', 'Det', 'Ord', 'NInf', 'Interj', 'Adj', 'Card', 'Range', 'Conj', 'Distrib', 'Ques', 'Zero', 'Pro
p', 'DemonsP', 'Noun', 'Real', 'APastPart', 'NFutPart', 'ReflexP']
[[  8   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    0   0   0   0   4   0   0   0   0   0   0   0]
 [  0  50   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   8   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   1   0   0]
 [  0   0   0 138   0   0   1   0   0   0   0   0   0   0   0   0   5   0
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0  67   0   0   0   0   2   0   0   0   0   0   3   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   5   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  36   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 530   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
```

```
 [  0   0   0   0   0   6   0   7   0   0 155   0   3   0   0   0   7   1
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   3   0 192   0   0   0   4   0
    0   0   0   0   0   0   3   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46   0   0   0
    0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   1   0   2   0   2   0   0   2   0   3   0   0   0 167   0
    0   0   0   0   9   0   0   8   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0  15
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
    0 130   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   1   0   0   0   0   0   0   0   0]
 [  1   0   0  17   0   0   0   0   0   0   1   0   0   0   0   0   2   0
    0   0   0   0  10   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    0   0   0   0   0  14   0   2   0   0   0   0]
 [  0   0   0   0   0   0   0   0   4   0   0   0   1   0   0   0   0   0
    0   0   0   0   0   0  10   0   0   0   0   0]
 [  0   0   0   1   0   3   0   0   1   0   0   0   0   0   3   0  16   0
    0   2   0   0   0   0   0 593   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  14   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   3   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   7]]
```

The overall accuracy is again down from the cpostag case. This again comes from the inherent difficulty of having more tags to choose from, plus the added fact that we have less data per tag. An accuracy of 93% is still very high and very similar, showing the 'one word-one tag' nature of things.

6.2.3 All Words

In [11]:

```
pos_tester.print_acc(2)
pos_tester.print_conf(2)
```

```
Accuracies:
Overall Accuracy: 0.788088642659
Pron Accuracy: 0.998337950139
APresPart Accuracy: 0.993351800554
NPastPart Accuracy: 0.994182825485
Verb Accuracy: 0.937119113573
Num Accuracy: 1.0
Postp Accuracy: 0.995290858726
AFutPart Accuracy: 0.998060941828
QuesP Accuracy: 0.996952908587
PersP Accuracy: 0.998614958449
Punc Accuracy: 1.0
Adv Accuracy: 0.978670360111
Dup Accuracy: 1.0
Det Accuracy: 0.993905817175
Ord Accuracy: 0.99972299169
NInf Accuracy: 0.968421052632
Interj Accuracy: 1.0
Adj Accuracy: 0.961772853186
Card Accuracy: 0.995290858726
Range Accuracy: 1.0
Conj Accuracy: 0.998891966759
Distrib Accuracy: 1.0
```

```
Ques Accuracy: 1.0
Zero Accuracy: 0.972299168975
Prop Accuracy: 0.987534626039
DemonsP Accuracy: 0.997506925208
Noun Accuracy: 0.816897506925
Real Accuracy: 1.0
APastPart Accuracy: 0.996398891967
NFutPart Accuracy: 0.997229916898
ReflexP Accuracy: 0.99972299169
Confusion matrix:
  Tags: ['Pron', 'APresPart', 'NPastPart', 'Verb', 'Num', 'Postp', 'AFutPart', 'QuesP', 'PersP', 'Punc', 'A
dv', 'Dup', 'Det', 'Ord', 'NInf', 'Interj', 'Adj', 'Card', 'Range', 'Conj', 'Distrib', 'Ques', 'Zero', 'Pro
p', 'DemonsP', 'Noun', 'Real', 'APastPart', 'NFutPart', 'ReflexP']
[[   8    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    1    0    0    0    0    0    4    0    0    0    0    0
     0    0]
 [   0   50    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0   24    0    0
     0    0]
 [   0    0    8    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0   20    0    1
     0    0]
 [   0    0    0  138    0    0    1    0    0    0    0    0    0    0
     0    0    5    0    0    0    0    0    0    0    0  202    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0   67    0    0    0    0    2    0    0    0
     0    0    3    0    0    0    0    0    0    0    0    1    0    0
     0    0]
 [   0    0    0    0    0    0    4    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    6    0    0
     0    0]
 [   0    0    0    0    0    0    0    5    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    0    0    0   36    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0  530    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    6    0    7    0    0  155    0    3    0
     0    0    7    1    0    0    0    0    0    0    0   45    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    3    0  192    0
     0    0    4    0    0    0    0    0    0    0    3    0    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    1
     0    0    0    0    0    0    0    0    0    0    0    1    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    46    0    0    0    0    0    0    0    0    0    0  111    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    2    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    1    0    2    0    2    0    0    2    0    3    0
     0    0  167    0    0    0    0    0    9    0    0   80    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    5    0
     0    0    0   15    0    0    0    0    0    0    0   11    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    0    2    0    0    0    0    0    0    0
     0    0    0    0    0  130    0    0    0    0    0    0    0    0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0]
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    1    0    0    0    0    0    0
    0    0]
[   1    0    0   17    0    0    0    0    0    0    1    0    0    0
    0    0    2    0    0    0    0    0   10    0    0   66    0    0
    0    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    1    0    0    0    0    0    0   14    0   44    0    0
    0    0]
[   0    0    0    0    0    0    0    0    4    0    0    0    1    0
    0    0    0    0    0    0    0    0    0    0   10    1    0    0
    0    0]
[   0    0    0    1    0    3    0    0    1    0    0    0    0    0
    3    0   16    0    0    2    0    0    0    0    0 1232    0    0
    0    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0   12    0   14
    0    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0   10    0    0
    3    0]
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    1    0    0
    0    7]]
```

Again, the combined accuracy is down due to the reasons explained above.

7. Conclusion

The 85% CPOSTAG accuracy over all words is definitely a good start for this very basic PoS Tagger. It shows how the HMM structure fits well to the problem. As stated above, there definitely is a trade-off between using word forms as our vocabulary and using the word lemmas as our vocabulary, due to the expected and empirical loss of accuracy when faced with unknown words. Also, our way of dealing with unknown words should definitely be replaced with another approach, since that is the primary thing that is keeping our accuracies down. If need be, we have a clear avenue to work on to increase our accuracy, and therefore, we can say that we are happy with the results.