Part Of Speech Tagging Using Hidden Markov Models

Muhammad Umar Salman
21010241

Muhammad Arsalan Manzoor
21010240

1. Introduction

Background

Parts of Speech (POS) tagging is a text processing technique to correctly understand the meaning of a text. POS tagging is the process of assigning the correct POS marker (Noun, Pronoun, Adverb, etc.) to each word in an input text. We have implemented POS tagging using Hidden Markov Models (HMMs) which are probabilistic sequence models. Our goal is that given a sequence of words we use HMMs to compute the most likely sequence of states (in our case POS Tags).

Part-of-Speech Tagging (POS Tagging)

Part-of-Speech (POS) such as (Noun, Verb, and Preposition) can help in understanding the meaning of a text by identifying how different words are used in a sentence. POS tagging is the process of assigning a POS marker (noun, verb, etc.) to each word in an input text. The input to a POS tagging algorithm is a sequence of tokenized words and a tag set (all possible POS tags) and the output is a sequence of tags, one per token. Words in the English language are ambiguous because they have multiple POS. For example, a book can be a verb (book a flight for me) or a noun (please give me this book). POS tagging aims to resolve those ambiguities.

Hidden Markov Models (HMMs)

The Hidden Markov Models (HMM) is a statistical model for modelling generative sequences characterized by an underlying process generating an observable sequence. HMMs have various applications such as in speech recognition, signal processing, and some low-level NLP tasks such as POS tagging. HMMs are also used in converting speech to text in speech recognition. HMMs are based on Markov chains. A Markov chain is a model that describes a sequence of potential events in which the probability of an event is dependent only on the state which is attained in the previous event. Markov model is based on a Markov assumption in predicting the probability of a sequence. If state variables are defined as $\boldsymbol{q_1, q_2, \ldots, q_i}$ a Markov assumption is defined as

$$\textbf{Markov Assumption:}\quad P(q_i = a|q_1 \ldots q_{i-1}) = P(q_i = a|q_{i-1})$$

The states are represented by nodes in the graph while edges represent the transition between states with probabilities. A first-order HMM is based on two assumptions. One of them is Markov assumption, that is the probability of a state depends only on the previous state as described earlier, the other is the probability of an output observation $\boldsymbol{O_i}$ depends only on the state that produced the observation $\boldsymbol{q_i}$ and not on any other states or observations

2. Methodology

Observation and Emission Probabilities

An HMM consists of two components, the A and the B probabilities. The A matrix contains the tag transition probabilities $\boldsymbol{P(t_i \mid t_{i-1})}$ and B the emission probabilities $\boldsymbol{P(w_i \mid t_i)}$
where $\boldsymbol{w}$ denotes the word and $\boldsymbol{t}$ denotes the tag

The transition probability, given a tag, how often is this tag is followed by the second tag in the corpus is calculated as

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

The emission probability, given a tag, how likely it will be associated with a word is given by

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Figure 2 shows an example of the HMM model in POS tagging. For a given sequence of three words, “word1”, “word2”, and “word3”, the HMM model tries to decode their correct POS tag from “N”, “M”, and “V”. The A transition probabilities of a state to move from one state to another and B emission probabilities that how likely a word is either N, M, or V in the given example.

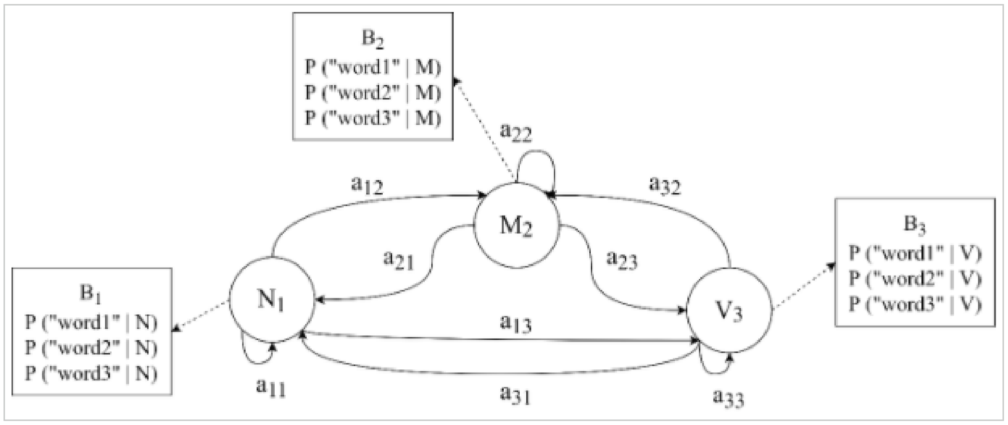


Figure 1

Viterbi Algorithm

The process of determining hidden states to their corresponding sequence is known as decoding. More formally, given A, B probability matrices and a sequence of observations
O = $o_1 \dots o_2, \dots o_T$, the goal of an HMM tagger is to find a sequence of states Q = $q_1 \dots q_2, \dots q_T$. For POS tagging the task is to find a tag sequence $t_1 \dots t_n$ that maximizes the probability of a sequence of observations of $w_1 \dots w_n$ words

$$\hat{t}_{1:n} = \underset{t_1 \dots t_n}{\operatorname{argmax}} P(t_1 \dots t_n | w_1 \dots w_n) \approx \underset{t_1 \dots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}}$$

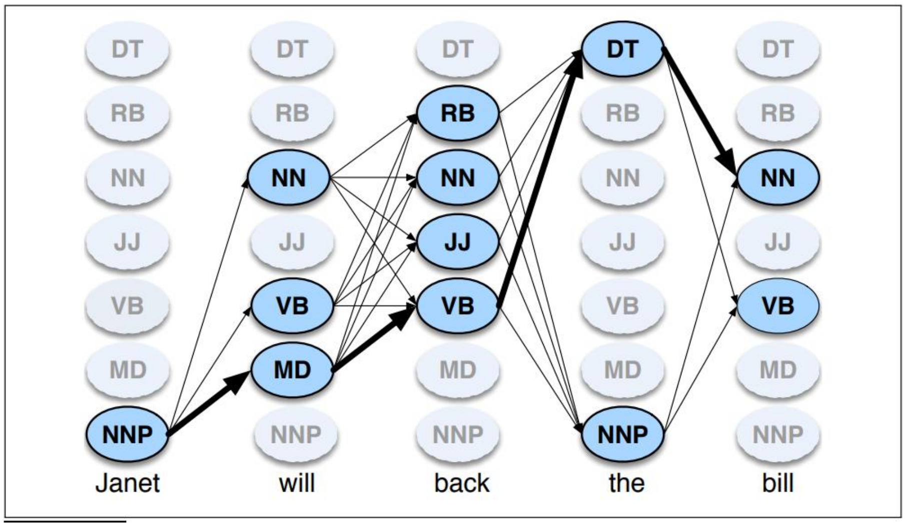


Figure 2

The decoding algorithm for the HMM model is the Viterbi Algorithm. Figure 3 shows an example of a Viterbi matrix with states (POS tags) and a sequence of words. The Viterbi algorithm works recursively to compute each cell value. For a given state ***q_j*** at time ***t***, the Viterbi probability at time ***t***, ***v_t(j)*** is calculated as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

3. Implementation

Dataset

In this assignment we use nltk library to use the brown corpus. The brown corpus consists of 15 categories including formal and informal writing styles. For a better result and better testing we only use 3 categories for testing our HMM results on it. The following categories for the brown corpus we use are:

1. News
2. Editorial
3. Reviews

We alongside with that use the “Universal Tagset” as our true labels for our experiment. That means we will consider the POS tags tagged by nltk’s library as perfectly annotated and will compare our results with this and use it as our training and testing set. The generic tags that we will use as states and are part of the “Universal Tagset” are as follows along with their count in our dataset:

1. NOUN (Nouns) - 56352
2. VERB (Verbs) - 29773
3. ADJ (Adjectives) - 15218
4. ADV (Adverbs) - 8429
5. PRON (Pronouns) - 6072
6. DET (Determiners & Articles) - 23525
7. ADP (Prepositions & Postpositions) - 24800
8. NUM (Numerals) - 3365
9. CONJ (Conjunctions) - 6032

10. PRT (Particles) - 4670
11. . (Punctuation) - 24381
12. X (A Catch-all for Other Categories) - 245
    e.g., Abbreviations or Foreign words)

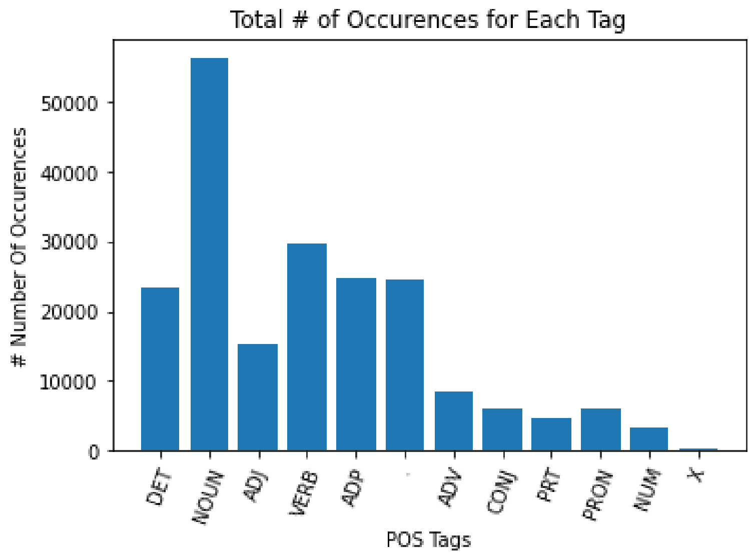
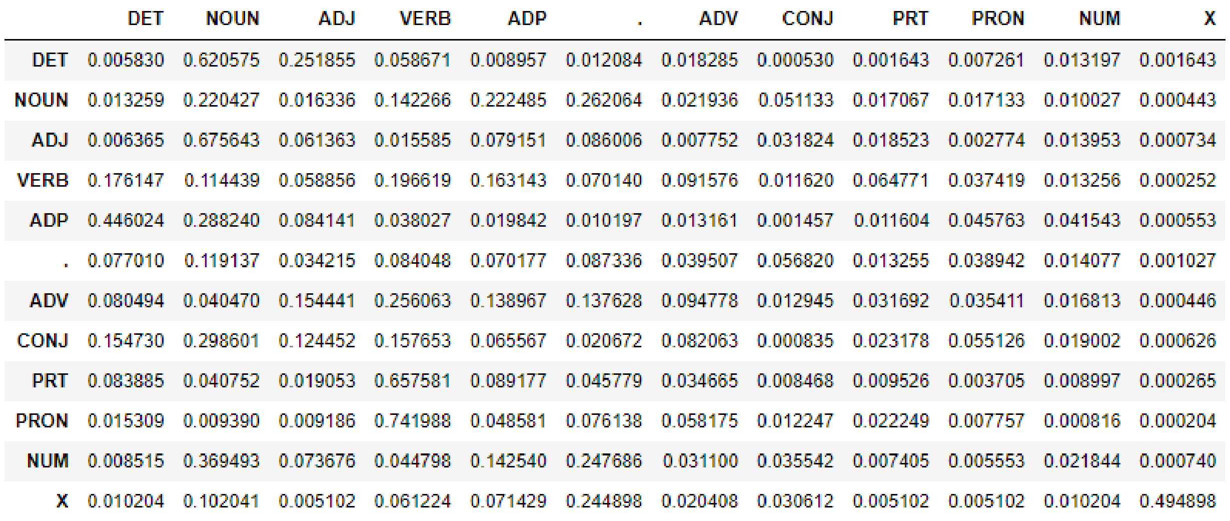


Figure 3

The total number of tagged sentences we got were 9371 sentences which we divided in to train test split based on the 80:20 rule. Thus, the training sentences were a total of 7496 and the testing sentences were a total of 1875 sentences. We before splitting into both sets shuffled them so that there was no bias that 'Review' categories were mostly being tested and 'News' category was mostly seen in training. After very little text processing such as extracting a unique set of words in training and lowering all the words so 'Apple' and 'apple' are the same we then began to calculate the transition and emission probabilities in the form of A and B matrices.

A & B Matrices

The A and B matrices which are also the Transition and Emission probabilities we formed by using the pandas library in Python. The transition probabilities as mentioned above are the state transitions to states. Here since the states are Tags we put the states in rows and columns we the rows are the previous state and the columns are the state they are transitioning too.



	DET	NOUN	ADJ	VERB	ADP	.	ADV	CONJ	PRT	PRON	NUM	X
DET	0.005830	0.620575	0.251855	0.058671	0.008957	0.012084	0.018285	0.000530	0.001643	0.007261	0.013197	0.001643
NOUN	0.013259	0.220427	0.016336	0.142266	0.222485	0.262064	0.021936	0.051133	0.017067	0.017133	0.010027	0.000443
ADJ	0.006365	0.675643	0.061363	0.015585	0.079151	0.086006	0.007752	0.031824	0.018523	0.002774	0.013953	0.000734
VERB	0.176147	0.114439	0.058856	0.196619	0.163143	0.070140	0.091576	0.011620	0.064771	0.037419	0.013256	0.000252
ADP	0.446024	0.288240	0.084141	0.038027	0.019842	0.010197	0.013161	0.001457	0.011604	0.045763	0.041543	0.000553
.	0.077010	0.119137	0.034215	0.084048	0.070177	0.087336	0.039507	0.056820	0.013255	0.038942	0.014077	0.001027
ADV	0.080494	0.040470	0.154441	0.256063	0.138967	0.137628	0.094778	0.012945	0.031692	0.035411	0.016813	0.000446
CONJ	0.154730	0.298601	0.124452	0.157653	0.065567	0.020672	0.082063	0.000835	0.023178	0.055126	0.019002	0.000626
PRT	0.083885	0.040752	0.019053	0.657581	0.089177	0.045779	0.034665	0.008468	0.009526	0.003705	0.008997	0.000265
PRON	0.015309	0.009390	0.009186	0.741988	0.048581	0.076138	0.058175	0.012247	0.022249	0.007757	0.000816	0.000204
NUM	0.008515	0.369493	0.073676	0.044798	0.142540	0.247686	0.031100	0.035542	0.007405	0.005553	0.021844	0.000740
X	0.010204	0.102041	0.005102	0.061224	0.071429	0.244898	0.020408	0.030612	0.005102	0.005102	0.010204	0.494898

Figure 4

We also plotted a heatmap to show the higher probabilities of transitions from one tag to another.

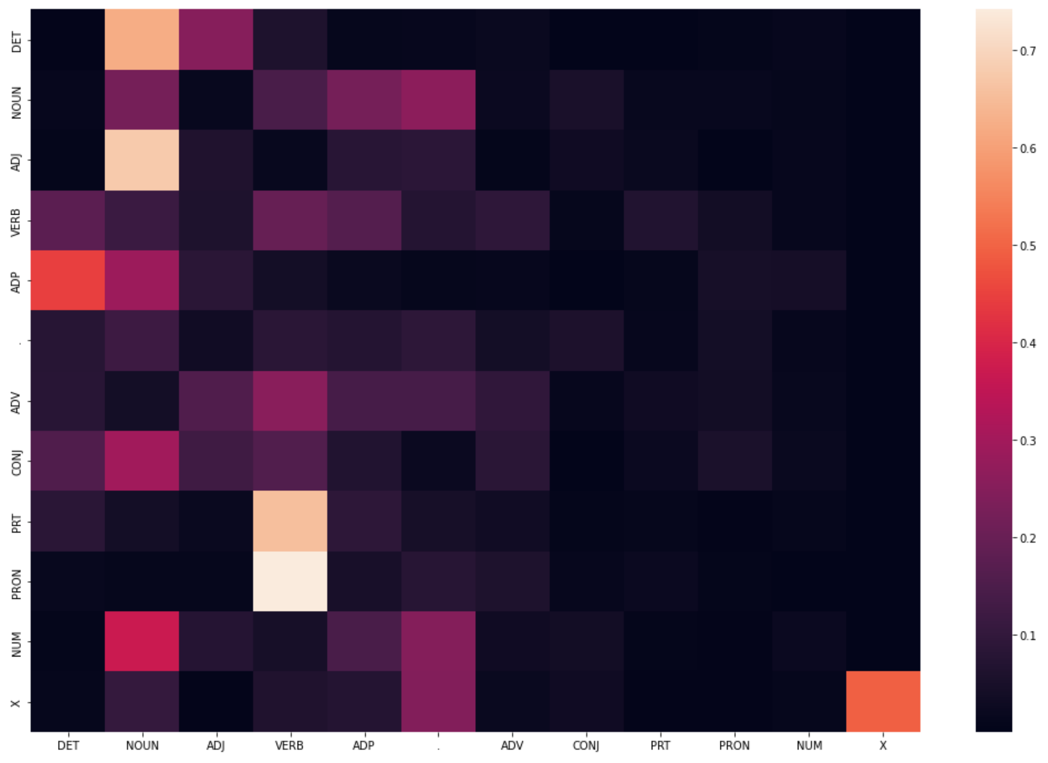
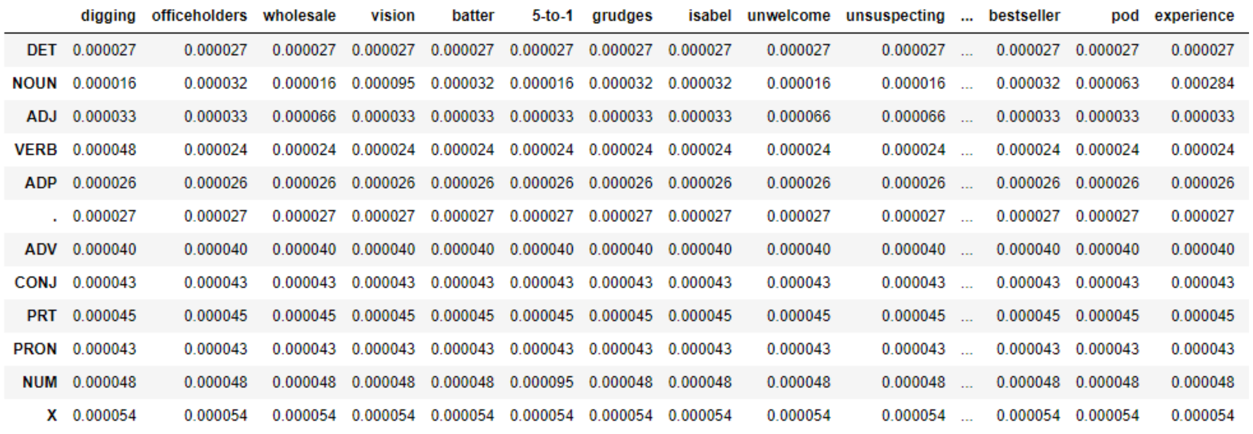


Figure 5

After that we using the set of 18247 unique words we got from the training and use that along with the count of all the words in each state. This gives us like mentioned above the emission probabilities which look something like this.



	digging	officeholders	wholesale	vision	batter	5-to-1	grudges	isabel	unwelcome	unsuspecting	...	bestseller	pod	experience
DET	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	...	0.000027	0.000027	0.000027
NOUN	0.000016	0.000032	0.000016	0.000095	0.000032	0.000016	0.000032	0.000032	0.000016	0.000016	...	0.000032	0.000063	0.000284
ADJ	0.000033	0.000033	0.000066	0.000033	0.000033	0.000033	0.000033	0.000033	0.000066	0.000066	...	0.000033	0.000033	0.000033
VERB	0.000048	0.000024	0.000024	0.000024	0.000024	0.000024	0.000024	0.000024	0.000024	0.000024	...	0.000024	0.000024	0.000024
ADP	0.000026	0.000026	0.000026	0.000026	0.000026	0.000026	0.000026	0.000026	0.000026	0.000026	...	0.000026	0.000026	0.000026
.	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	0.000027	...	0.000027	0.000027	0.000027
ADV	0.000040	0.000040	0.000040	0.000040	0.000040	0.000040	0.000040	0.000040	0.000040	0.000040	...	0.000040	0.000040	0.000040
CONJ	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	...	0.000043	0.000043	0.000043
PRT	0.000045	0.000045	0.000045	0.000045	0.000045	0.000045	0.000045	0.000045	0.000045	0.000045	...	0.000045	0.000045	0.000045
PRON	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	...	0.000043	0.000043	0.000043
NUM	0.000048	0.000048	0.000048	0.000048	0.000048	0.000095	0.000048	0.000048	0.000048	0.000048	...	0.000048	0.000048	0.000048
X	0.000054	0.000054	0.000054	0.000054	0.000054	0.000054	0.000054	0.000054	0.000054	0.000054	...	0.000054	0.000054	0.000054

Figure 6

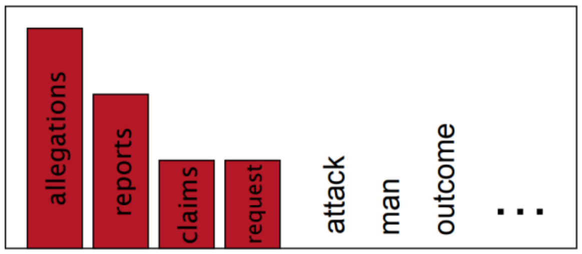
Add 1 Smoothing

As you know all the words can't be counted in every state and all the states don't transition to all states. That means that there will be some probabilities in our table which have a value of 0 because they have a count of 0. To solve this issue, we can use any smoothing or estimation technique. Some famous ones are Knesser Ney Smoothing and Good Turing Smoothing. However, for simplicity we have used Add -1 Smoothing or Laplace Smoothing which is a simple smoothing technique that Add 1 to the count of all n-grams in the training set before normalizing into probabilities. In simple words this means to assume we saw each word in each state and each state transitioning to each state 1 time more than we already did. The formula of Add 1 Smoothing can be shown as

MLE estimate:
$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

Add-1 estimate:
$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

Where compared to the regular MLE estimate we add 1 for each probability in the numerator and divide it by the set of total unique words. The effect can be seen as the following where word s like 'attack', 'man' etc. are not in the list of words but to smooth things out we add a single word in them which takes out a little mass from the rest of the probabilities but serves a greater purpose of and avoiding 0 probabilities which will cause entire multiplication of probabilities to go to 0.



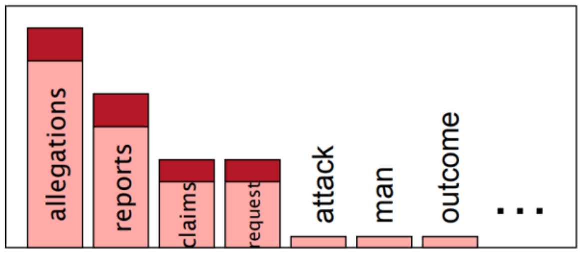


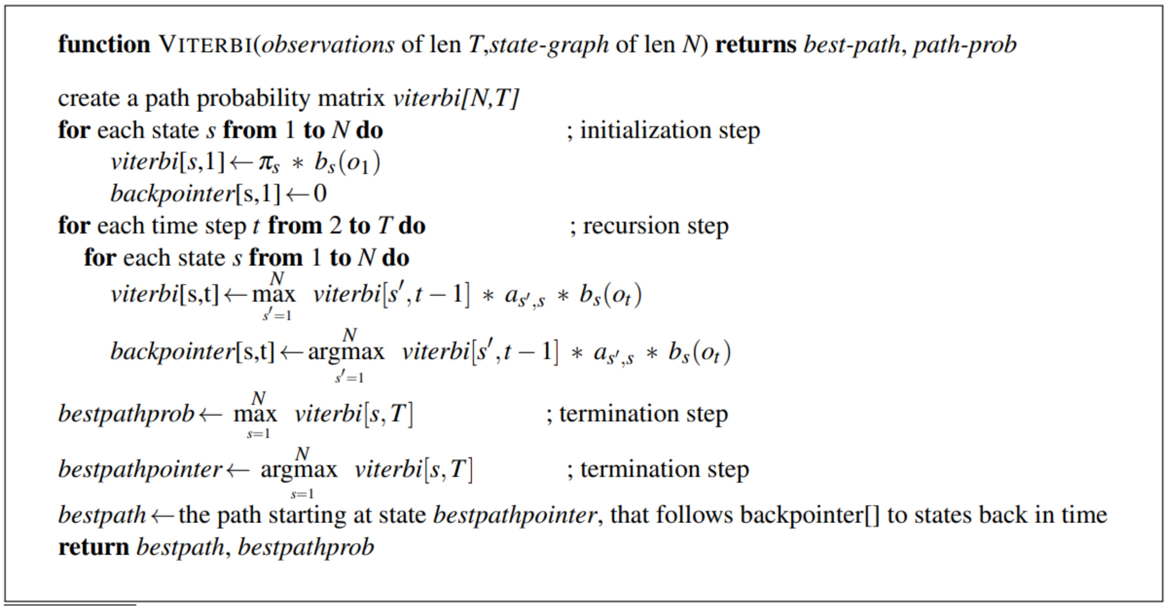
Figure 7

The Viterbi Algorithm

The Viterbi uses Dynamic Programming to help speed up its process of computing the most likely probability state sequence given the observation sequence. That is when we go over each word, we first compute the maximum probability for that state using the transition probability and emission probability compared to all other states and store that as our back pointer so we don't have to compute the Viterbi component coming from behind at every word instance. Once we get the back pointer for all the word sequence we can reconstruct the best path and find out the best probability for the most likely sequence of states. Another thing to mention is at the start we calculate the PI distribution which is basically the probabilities of each state coming at the start of the sentence. This will act as the Viterbi component that comes from the previous observation as the start state has no previous observation. The probabilities of the PI distribution on our corpus is shown below

```
{'DET': 0.242396,
 'NOUN': 0.204909,
 'PRON': 0.111393,
 'CONJ': 0.052561,
 'ADP': 0.114328,
 'ADV': 0.073906,
 'NUM': 0.016009,
 'ADJ': 0.044023,
 '.': 0.065902,
 'PRT': 0.031884,
 'VERB': 0.042156,
 'X': 0.000534}
```

Below is the pseudo code used for the Viterbi Algorithm that we have taken from Dan Jurafsky's book on Speech and Language Processing which we took reference from for the assignment.



```
function VITERBI(observations of len T,state-graph of len N) returns best-path, path-prob

create a path probability matrix viterbi[N,T]
for each state s from 1 to N do                          ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t from 2 to T do                      ; recursion step
  for each state s from 1 to N do
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$           ; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$     ; termination step
$bestpath \leftarrow$ the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

Figure 8

Evaluation

As mentioned, before we used the nltk "Universal Tagset" as true labels for the tags for training and testing of the brown corpus. Our testing data consists of 1875 tagged sentences which are a total of 40268 words that can be tagged. Our method of evaluation for this is to check out of these 40268 total tagged words how many did our HMM model manage to get right and then based on that we can calculate the accuracy of our model. Our HMM model managed to correctly identify 36461 tags and misclassified 3807 tags. This calculates up to be **90.55%** accurate for our test set. For each state we have shown a graph which shows as percentage out of 1 how many correctly tagged states were predicted.

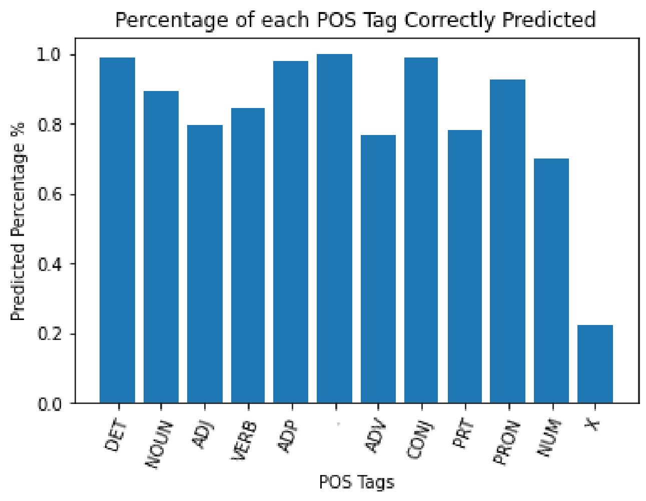


Figure 9

4. Conclusion

From our results of testing POS Tags on the brown corpus sentences (in the testing set) by our HMM we can see that the HMM we have designed works very well and gives results just as well most other language models that attempt to predict POS Tags. POS tagging resolves ambiguities for machines to understand natural language. In conversational systems, a large number of errors arise from natural language understanding (NLU) module. POS tagging is one technique to minimize those errors in conversational systems and in our assignment we have tried to do just that.

5. Refences

[1] Speech and Language Processing by Dan Jurafsky, https://web.stanford.edu/~jurafsky/slp3/A.pdf