**Name: Deepthi Thalagundamatada**

**Student email ID: dt7637@uncw.edu**

**CSC515: Artificial Intelligence**

**Project Report on**

**PARTS OF SPEECH TAGGING**

The Natural Language processing is one of the prominent research area in Artificial intelligence.

parts of speech tagging in the NLP with holds a important value in various applications like language translators. These translators need the parts of speech to convert to the other language.

Tagging parts of speech is not that easy because the words can hold different parts of speech depending on the sentence context.

example : position can be a

* BACK is ---noun (On my *back*)
* BACK is --- a verb (Could you [just](https://www.macmillandictionary.com/us/dictionary/american/just_1) back onto the [driveway](https://www.macmillandictionary.com/us/dictionary/american/driveway)?)

Input Data set:

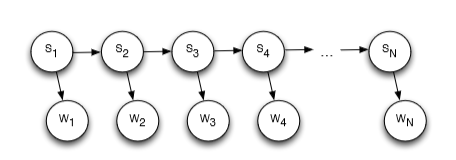
Universal TAGS of parts of speech:

1. ADJ (adjective)
2. ADV (adverb)
3. ADP (adposition)
4. CONJ (conjunction)
5. DET (determiner)
6. NOUN
7. NUM (number)
8. PRON (pronoun)
9. PRT (particle)
10. VERB

I decided to pick 10 of the universal tags to for the parts of speech.

I decided to do this with Markov model, but later learnt that version of Markov model is more needed for the implementation. So I considered HMM as the preferred algorithm.

**HMM Model:**



Here,

**Hidden States: Part of speech labels**

**Observed states/variables: Words**

Assumptions: When an unknown word or a transition sequence or unknown emission is observed we substitute with a log of small probability i.e. math.log(1.0/1000000000)

The probabilities are namely P(S1), P(Si+1|Si), P(Si+2|Si+1, Si) and P(Wi|Si)

**Initial State Distribution:**

For P(S1) (namely the initial state distribution) that is the prior of "part of speech" at position 1,

I have collected all the POS of the first position in each sentence and put it as a key in a dictionary (being startingProbs{} )of logarithm of probabilities.

Now, I have a total of 10 POS as keys and their corresponding logarithm of probabilities of them appearing in the first position.

For priors to be considered in positions other than first, I have collected all the POS appearing in the positions other than the first and found out the probabilities.

This forms my dictionary allSimplePOSProbs{}.This was done to make a distinction in distributions for POS according to their positions to get better accuracy.

**Transition probabilities:**

P(Si+1|Si): (Required for HMM)

Since P(Si+1|Si) = P(Si+1 , Si)/P(Si), We have found the probability of two POS tags appearing in sequence,

that is P(Si+1 , Si), for example [noun-adj], P(noun,adj) is the probability of noun-adj pair appearing and

is stored as the transition probabilities (in the form of log(P)), transitionProbs{} dictionary.

And to calculate -->> log(P(Si+1|Si)) = log(P(Si+1 , Si))-log(P(Si)),

For example, log(P(noun/adj)) = log(P(adj , noun))-log(P(adj)) = transitionProbs["adj\_noun"]-allSimplePOSProbs["adj"]

**Emission probabilities:**

**P(Wi/Si):**

P(Word/Part Of Speech) = probability of word given a part of speech.

For each POS we collected all the words that were tagged with this particular POS, and were stored in a

dictionary of dictionary whose outer key is POS and inner key is the word tagged with this particular POS.

For instance, P(found/VERB) = emissionProbs["VERB"]["found"] stored in the form of log(P)

**HMM Model By Viterbi:**

Since each state is dependent on the previous state we would need to calculate the sequence "s1,s2,s3,s4,....,sn"such that the probability **P(Si = si|W)** leads to maximum.

That is,

**P(S1,S2,S3,S4,....,Sn/W1,W2,W3,...,Wn) =P(W1/S1)\*P(S1)\*P(W2/S2)\*P(S2/S1)...\*P(Wn/Sn)\*P(Sn/Sn-1)**

so at every ith position, we find out the "si" leading to the highest probability of P(Si = si/W).

For every state at position t, we find the max of previous states' probability times the transition probability from the precious state to current state and multiply with emission probability of the current state.

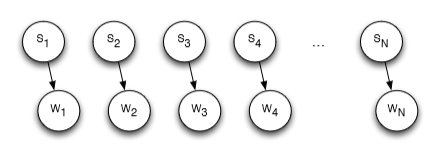
We store this probability in a dictionary dictOfPostions[position][POS] in the form of log(P), now after

we calculate the probability at each position for each state, to backtrack we append in the form of string

**maxPaths[position][POS] = maxPaths[position-1] [prevStateLeadingToMax] + "=>" + POS**

where POS is the current part of speech. (POS at position "t"), such that we don’t have to traverse or backtrack through the entire trace, rather at the last position find the POS that had the maximum probability and pull the string that maxPaths[LastPosition][maxPOS] had. Hence saving the requirement of return loop.

**Simple Naive Bayes Approach:**



In this we make a naive assumption that each POS at a given position is independent of any other positions.

That is,

**P(S1, S2,S3,S4,....,Sn/W1,W2,W3,...,Wn) = P(W1/S1)\*P(S1)\*P(W2/S2)\*P(S2)....\*P(Wn/Sn)\*P(Sn),**

so at every ith position, we find out the "si" leading to the highest probabilty of **P(Si = si/W).**

and output the sequence of the labels.

**Posterior Calculations:**

Say for a sentence of 4 words S1 = s1, S2 = s2,S3 = s3, S4 = s4 , w1,w2,w3,w4

**Simple Bayes :**

**P(S1,S2,S3,S4|W1,W2,W3,W4) (proportional to)**

**P(W1=w1|S1=s1)\*P(S1=s1)\*P(W2=w2|S2=s2)\*P(S2=s2)\*P(W3=w3|S3=s3)\*P(S3=s3)\*P(W4=w4|S4=s4)\*P(S4=s4)/P(W1,W2,W3,W4)**

in logs , I add the Emission Probabilities and Priors.

**HMM Model**:

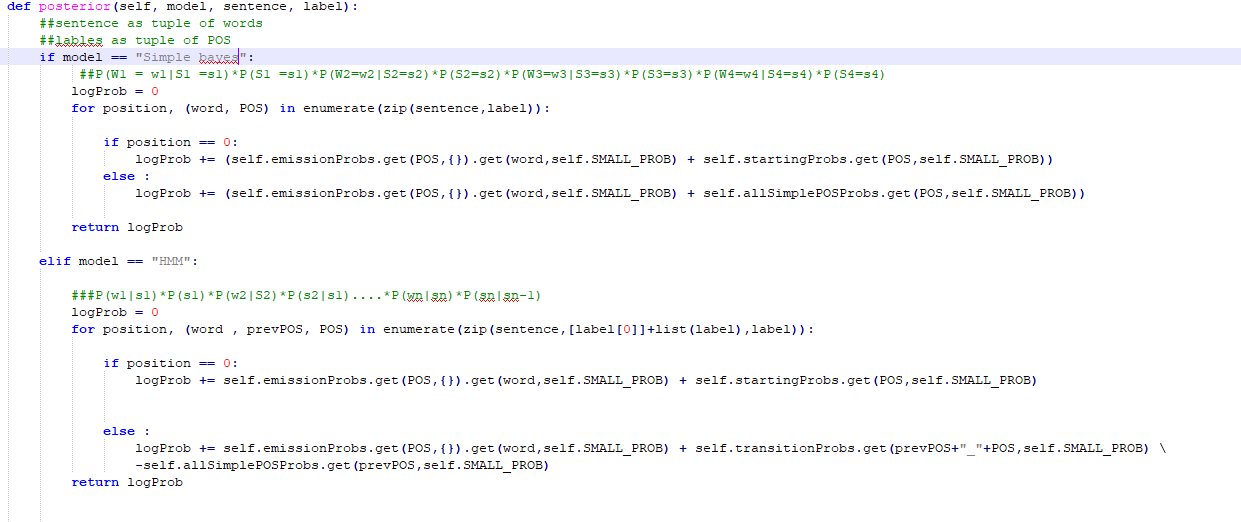
**P(S1,S2,S3,...Sn/W1,W2,W3,Wn)**

**(proportional to)**

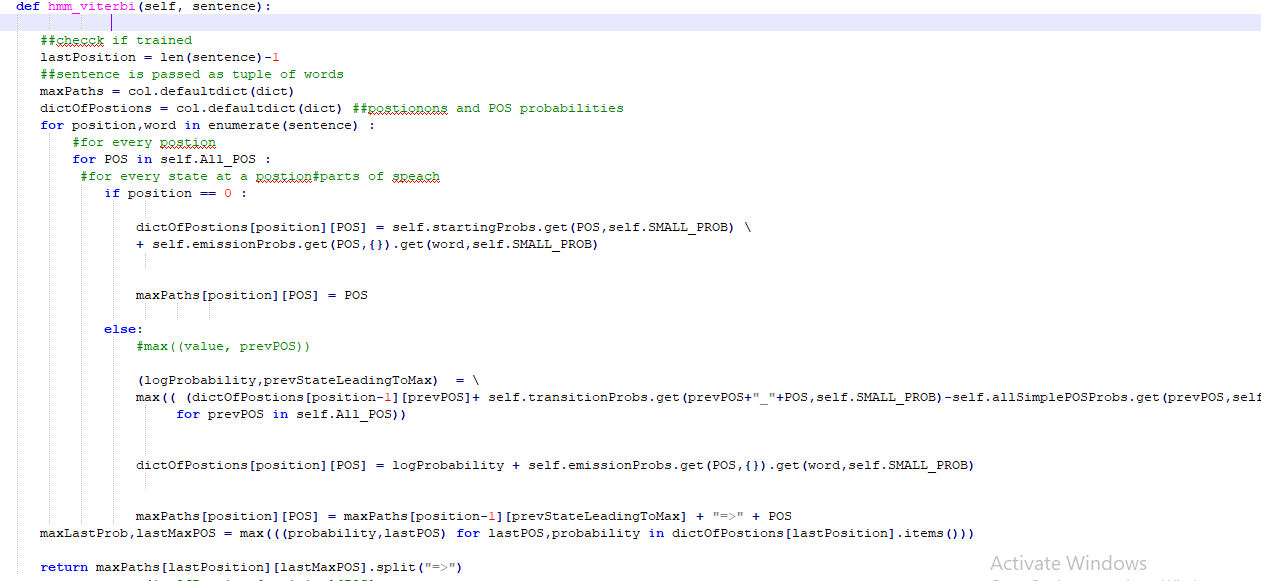
**P(w1|s1)\*P(s1)\*P(w2|S2)\*P(s2|s1)....\*P(wn|sn)\*P(sn|sn-1)**

**Implemented code:**

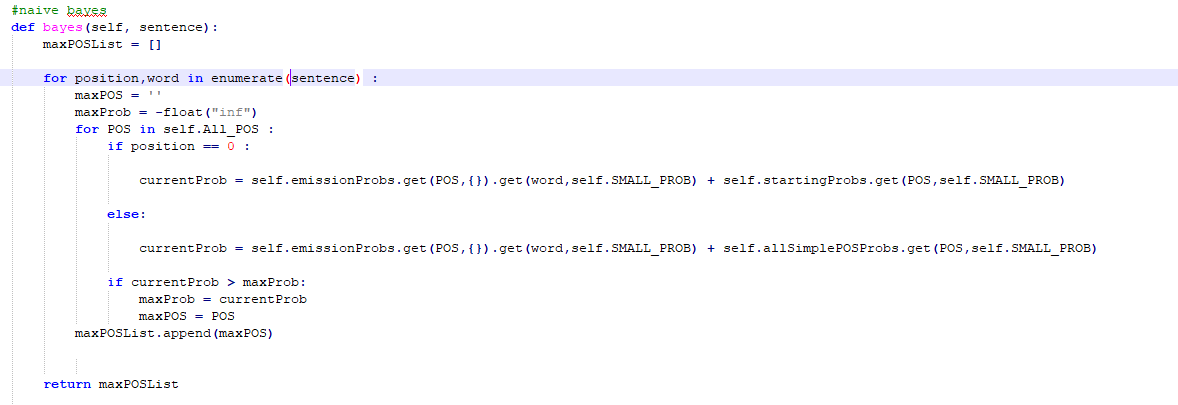
**Posterior probabilities:**



**For HMM:**

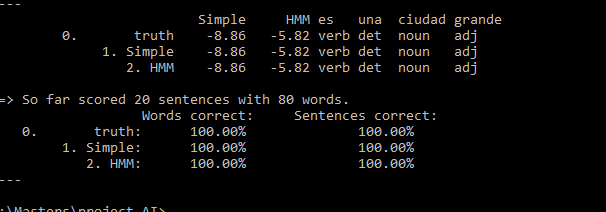


**Bayes Implementation**



**Results:**

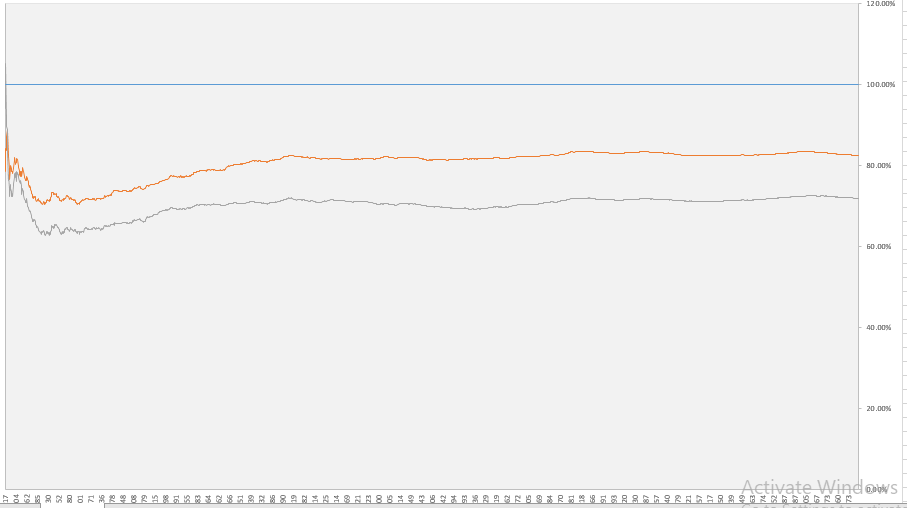
**For Spanish:**



**For English:**

**Interpretations:**

When I was dealing with very low values I decided to use log values to solve this issue. The Log probability when I had set it to 1/10 the words effected by this



**When I had set the log to 1.0/1000000000**

**And the number of words which are correct from the HMM and Bayes is Like this:**