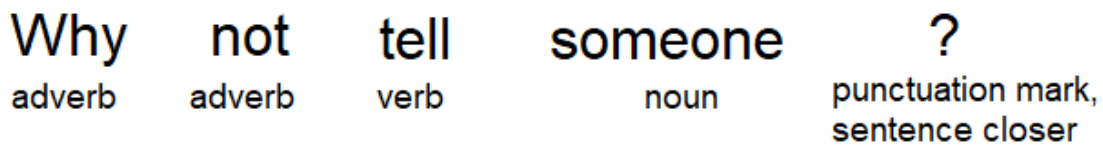
ASSIGNMENT 3

AIM :

Survey various techniques for POS tagging and implement any one of them.

THEORY :

Part-of-speech (POS) tagging is a popular Natural Language Processing process which refers to categorizing words in a text (corpus) in correspondence with a particular part of speech, depending on the definition of the word and its context.



In Figure, we can see each word has its own lexical term written underneath, however, having to constantly write out these full terms when we perform text analysis can very quickly become cumbersome especially as the size of the corpus grows. Thence, we use a short representation referred to as "**tags"** to represent the categories.

As earlier mentioned, the process of assigning a specific tag to a word in our corpus is referred to as part-of-speech tagging (POS tagging for short) since the POS tags are used to describe the lexical terms that we have within our text.

Lexical Term	Tag	Example
Noun	NN	Paris, France, Someone, Kurtis
Verb	VB	work, train, learn, run, skip
Determiner	DT	the, a
...	...	

Why	not	tell	someone	?
WRB	**RB**	**VB**	**NN**	**.**

Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

1. Rule-based POS Tagging :

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article then word must be a noun.

As the name suggests, all such kind of information in rule-based POS tagging is coded in the form of rules. These rules may be either −

- Context-pattern rules
- Or, as Regular expression compiled into finite-state automata, intersected with lexically ambiguous sentence representation.

We can also understand Rule-based POS tagging by its two-stage architecture −

- **First stage** − In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.
- **Second stage** − In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

2. Stochastic POS Tagging :

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

The simplest stochastic tagger applies the following approaches for POS tagging −

Word Frequency Approach: In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

Tag Sequence Probabilities: It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

3. Transformation Based Learning(TBL) :

In order to understand the working and concept of transformation-based taggers, we need to understand the working of transformation-based learning. Consider the following steps to understand the working of TBL −

- **Start with the solution** − The TBL usually starts with some solution to the problem and works in cycles.
- **Most beneficial transformation chosen** − In each cycle, TBL
- **Apply to the problem** − The transformation chosen in the last step will be applied to the problem.

The algorithm will stop when the selected transformation in step 2 will not add either more value or there are no more transformations to be selected. Such kind of learning is best suited in classification tasks.

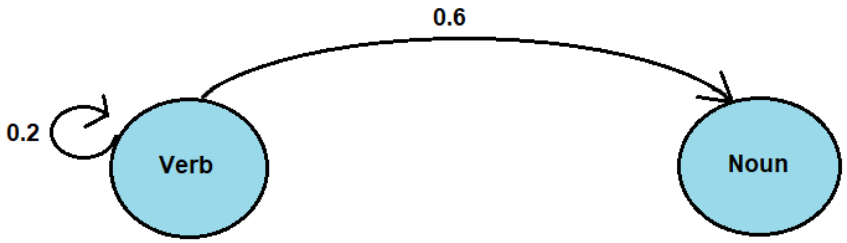
4. Hidden Markov Model (HMM) POS Tagging :

Before digging deep into HMM POS tagging, we must understand the concept of Markov Chains

Markov Model :

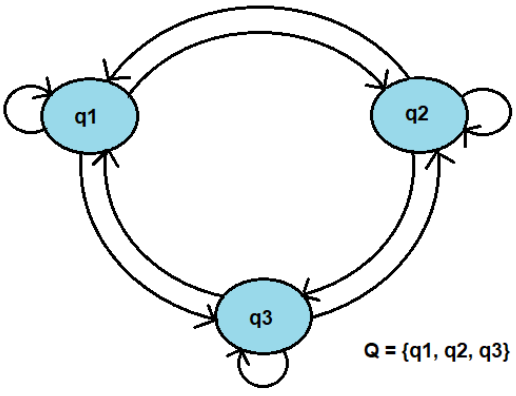
Taking the example text we used in Figure , “Why not tell someone?”, imaging the sentence is truncated to “Why not tell … ” and we want to determine whether the following word in the sentence is a noun, verb, adverb, or some other part-of-speech.

Now, if you are familiar with English, you'd instantly identify the verb and assume that it is more likely the word is followed by a noun rather than another verb. Therefore, the idea as shown in this example is that the POS tag that is assigned to the next word is dependent on the POS tag of the previous word.



By associating numbers with each arrow direction, of which imply the likelihood of the next word given the current word, we can say there is a higher likelihood the next word in our sentence would be a noun since it has a higher likelihood than the next word being a verb if we are currently on a verb. The image in figure above, is a great example of how a Markov Model works on a very small scale.

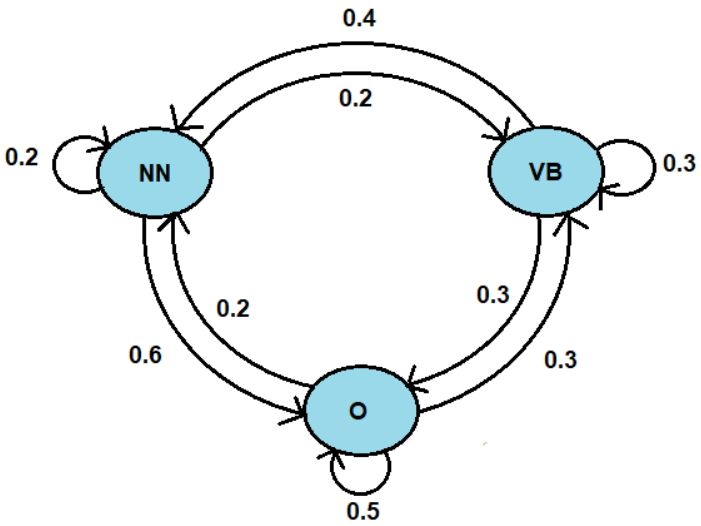
Given this example, we may now describe Markov models as "a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the Markov property)*.*". Therefore, to get the probability of the next event, it needs only the states of the current event.
We can depict a Markov chain as directed graph:



The lines with arrows are an indication of the direction hence the name "**directed graph**", and the circles may be regarded as the states of the model — a state is simply the condition of the present moment.

We could use this Markov model to perform POS. Considering we view a sentence as a sequence of words, we can represent the sequence as a graph where we use the POS tags as the events that occur which would be illustrated by the stats of our model graph.

For example, q1 in above figure would become NN indicating a noun, q2 would be VB which is short for verb, and q3 would be O signifying all other tags that are not NN or VB. Like in previous figure, the directed lines would be given a transition probability that define the probability of going from one state to the next.



A more compact way to store the transition and state probabilities is using a table, better known as a "**transition matrix**".

	NN	VB	O
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

This model only tells us the transition probability of one state to the next when we know the previous word. Hence, this model does not show us what to do when there is no previous word. To handle this case, we add what is known as the "**initial state**".

	NN	VB	O
(initial)	0.4	0.1	0.5
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

How did we populate the transition matrix? . Let's use 3 sentences for our corpus. The first is "<s> in a station of the metro", "<s> the apparition of these faces in the crowd", "<s> petals on a wet, black bough." (Note these are the same sentences used in the course). Next, we will break down how to populate the matrix into steps:

1. Count occurrences of tag pairs in the training dataset

$$C(t_{i-1}, t_i)$$

At the end of step one, our table would look something like this…

	NN	VB	O
(initial)	1	0	2
NN (noun)	0	0	6
VB (verb)	0	0	0
O (other)	6	0	8

Corpus:
<s> in a station of the metro

<s> the apparition of these faces in the crowd :

<s> petals on a wet, black bough.

2. Calculate the probability of using the counts

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1})}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

Appling the formula in above formula to the table in previous table, our new table would look as follows…

	NN	VB	O
(initial)	1/3	0	2/3
NN (noun)	0	0	1
VB (verb)	0	0	0
O (other)	6/14	0	8/14

You may notice that there are many 0's in our transition matrix which would result in our model being incapable of generalizing to other text that may contain verbs. To overcome this problem, we add smoothing.

Adding smoothing requires we slightly we adjust the formula from above formula by adding a small value, epsilon, to each of the counts in the numerator, and add N * epsilon to the denominator, such that the row sum still adds up to 1.

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}) + \epsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N * \epsilon}$$

	NN	VB	O
(initial)	0.3333	0.0003	0.6663
NN (noun)	0.0001	0.0001	0.9996
VB (verb)	0.3333	0.3333	0.3333
O (other)	0.4285	0.0000	0.5713

N = 3
epsilon = 0.001

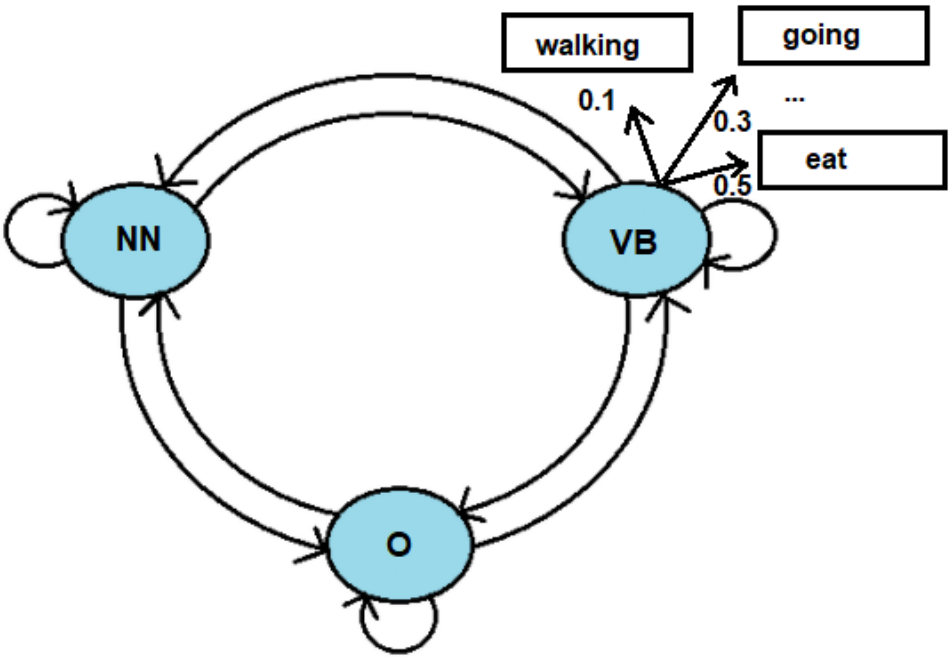
Hidden Markov Model

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobservable ("hidden") states .In our case, the unobservable states are the POS tags of a word.

If we rewind back to our Markov Model, we see that the model has states for part of speech such as VB for verb and NN for a noun. We may now think of these as hidden states since they are not directly observable from the corpus. Though a human may be capable of deciphering what POS applies to a specific word, a machine only sees the text, hence making it observable, and is unaware of whether that word POS tag is noun, verb, or something else which in-turn means they are unobservable.

Both the Markov Model and Hidden Markov model have transition probabilities that describe the transition from one hidden state to the next, however, the Hidden Markov Model also has something known as emission probabilities.

The emission probabilities describe the transitions from the hidden states in the model — remember the hidden states are the POS tags — to the observable states — remember the observable states are the words.



In Figure we see that for the hidden VB state we have observable states. The emission probability from the hidden states VB to the observable eat is 0.5 hence there is a 50% chance that the model would output this word when the current hidden state is VB.

We can also represent the emission probabilities as a table…

	walking	going	eat	...
NN (noun)	0.1	0.5	0.02	
VB (verb)	0.1	0.3	0.5	
O (other)	0.5	0.3	0.68	

Similar to the transition probability matrix, the row values must sum to 1. Also, the reason all of our POS tags emission probabilities are more than 0 since words can have a different POS tag depending on the context.

To populate the emission matrix, we'd follow a procedure very similar to the way we'd populate the transition matrix. We'd first count how often a word is tagged with a specific tag.

	in	a	...
NN (noun)	0	...	...
VB (verb)	0	...	...
O (other)	2	...	...

Corpus:
<s> in a station of the metro

<s> the apparition of these faces in the crowd :

<s> petals on a wet, black bough .

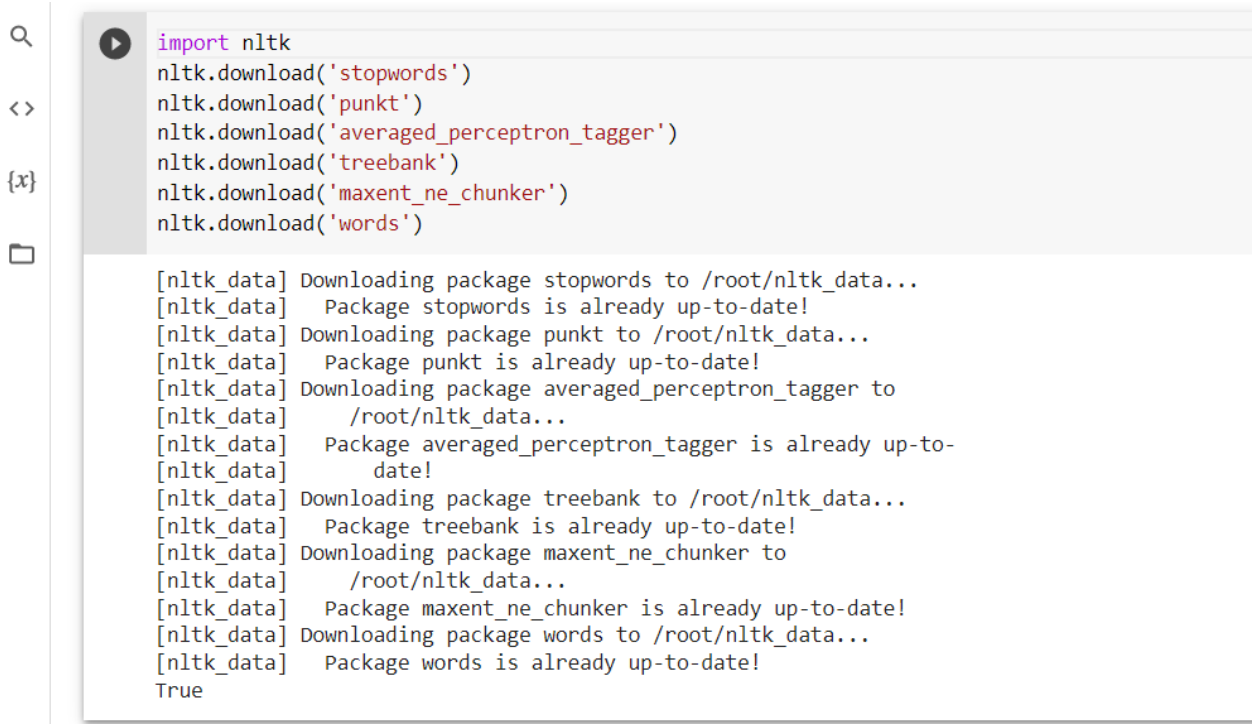
Since the process is so similar to calculating the transition matrix, I will instead provide you with the formula with smoothing applied to see how it would be calculated.

$$P(w_i|t_i) = \frac{C(t_i, w_i) + \epsilon}{C(t_i) + N * \epsilon}$$

CODE :

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('treebank')
nltk.download('maxent_ne_chunker')
nltk.download('words')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words('english'))

txt = "A and B are good friends"

tokenized = sent_tokenize(txt)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    wordsList = [w for w in wordsList if not w in stop_words]
```

```
    tagged = nltk.pos_tag(wordsList)
    print(tagged)

sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),("dog",
"NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(sentence)
print(result)
```



```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('treebank')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
True
```

```
[ ] from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize, sent_tokenize
    stop_words = set(stopwords.words('english'))

    txt = "A and B are good friends"

    tokenized = sent_tokenize(txt)
    for i in tokenized:
        wordsList = nltk.word_tokenize(i)
        wordsList = [w for w in wordsList if not w in stop_words]
        tagged = nltk.pos_tag(wordsList)
        print(tagged)
```

```
[('A', 'DT'), ('B', 'NNP'), ('good', 'JJ'), ('friends', 'NNS')]
```

```
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"), ("cat", "NN")]

grammar = "NP: {<DT>?<JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(sentence)
print(result)
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

CONCLUSION :

We studied various techniques for POS tagging and successfukky implemented one of them.