ANISH ADNANI NLP

D17B/01 (1)

EXPERIMENT 6

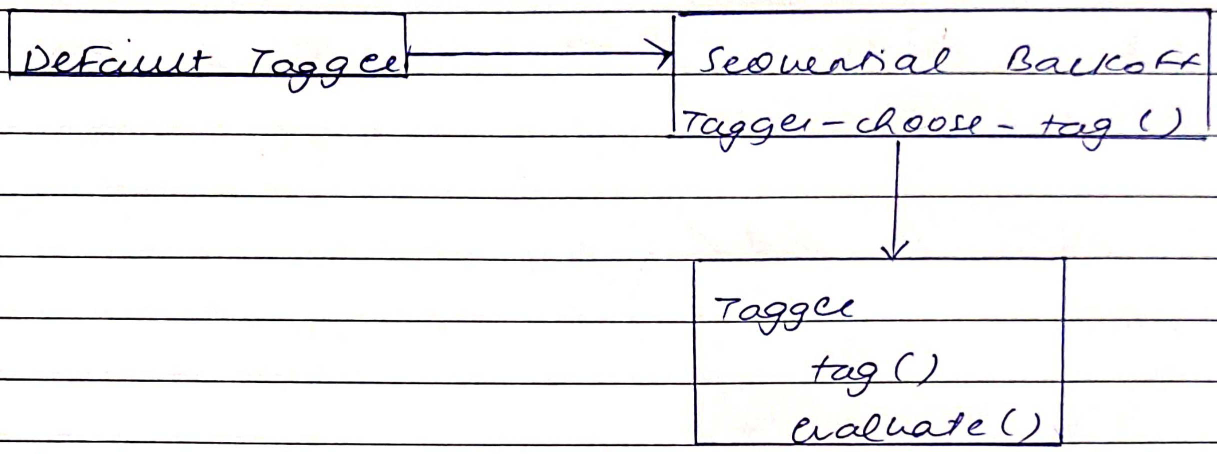
Aim: Implement viterbi encoding for POS (part of speech Tagging) on the given text.

Theory:

Part of speech Tagging (POS):
It is a process of converting sentence to form list of words, list of tuples. The tag in case is a part of speech tag, and signifies whether the word is a noun, adjective, verb & so on

POS	TAG
Noun	n
verb	v
Adjective	a
Adverb	r

Default Tagging:
It is a basic step for the part-of-speech tagging and is performed using Default Tagger class.



POS Tag Ambiguity

In English, many common words have multiple meanings and therefore multiple meanings mean multiple POS. The job of POS tagger is to resolve this ambiguity accurately based on context of use

For example: The word 'shot' can be a noun/verb

Viterbi encoding method for POS:

The intuition behind the viterbi algorithm is to use dynamic programming to reduce the number of computations by storing the calculations that are seperated.

$$\begin{tikzcd}
S1 & S2 & S3 & S4 & S5 \\
 & & RB \arrow[ddr] \arrow[dddddr] & & \\
VBN \arrow[dr] & & NN \arrow[dr] \arrow[ddddr] & & \\
 & TO \arrow[uur] \arrow[ur] \arrow[dr] \arrow[dddr] & & DT \arrow[r] \arrow[dddr] & VB \\
 & & JJ \arrow[ur] \arrow[ddr] & & \\
VBD \arrow[uur] & & & & \\
 & & VB \arrow[uuur] \arrow[r] & NNP \arrow[uuur] \arrow[r] & NN
\end{tikzcd}$$

Promised to back the bill

Conclusion:

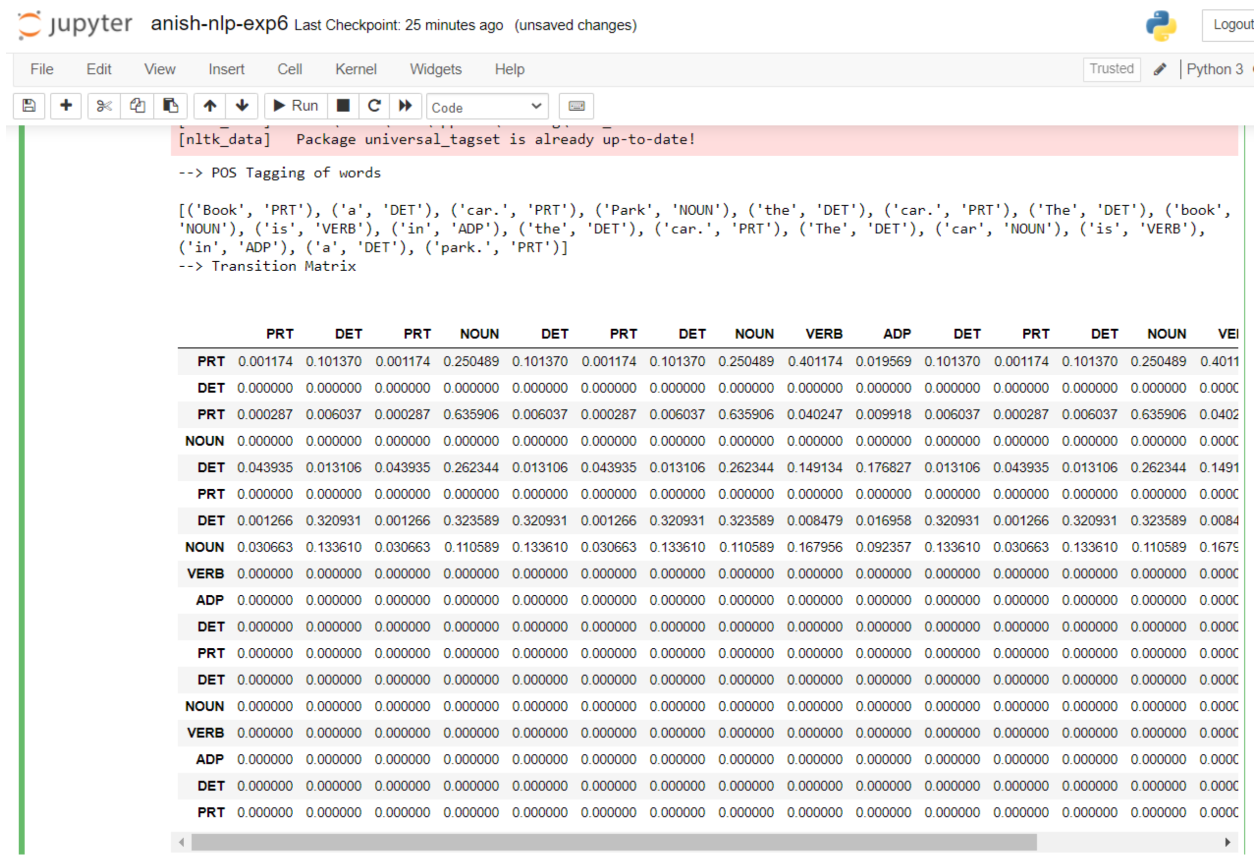
POS Tagging and its method well understood, ambuiguities got clear, viterbi encoding method understood, POS tagging done for the chosen corpus text

Code

```
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
from IPython.display import display
nltk.download('treebank')
nltk.download('universal_tagset')
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state =
101)
train_tagged_words = [ tup for sent in train_set for tup in sent ]
tags = {tag for word,tag in train_tagged_words}

def word_given_tag(word, tag, train_bag = train_tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)
    w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
    count_w_given_tag = len(w_given_tag_list)

    return (count_w_given_tag, count_tag)

def t2_given_t1(t2, t1, train_bag = train_tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2:
            count_t2_t1 += 1

    return (count_t2_t1, count_t1)

tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
for i, t1 in enumerate(list(tags)):
    for j, t2 in enumerate(list(tags)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]

tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
def Viterbi(words, train_bag = train_tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))
    for key, word in enumerate(words):
        p = []
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]

            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = emission_p * transition_p
            p.append(state_probability)
        pmax = max(p)
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))



text = "Book a car. Park the car. The book is in the car. The car is in a park."
words = text.split()
POS_tagging = Viterbi(words)
print("--> POS Tagging of words")
print()
print(POS_tagging)
pos_tags = []


for i in POS_tagging:
    if i not in pos_tags:
        pos_tags.append(i[1])

Transition_matrix = np.zeros((len(POS_tagging), len(POS_tagging)), dtype='float32')


for i, t1 in enumerate(tags):
    for j, t2 in enumerate(POS_tagging):
        if t2_given_t1(t2[1], t1, POS_tagging)[1] != 0:
            Transition_matrix[i, j] = t2_given_t1(t2[1], t1)[0]/t2_given_t1(t2[1], t1)[1]
        else:
            Transition_matrix[i, j] = 0.0
print("--> Transition Matrix")
print()
Transition_df = pd.DataFrame(Transition_matrix, columns = list(pos_tags), index=list(pos_tags))
display(Transition_df)
words = []
```

```
for i in POS_tagging:
    if i not in words:
        words.append(i[0])

Emission_matrix = np.zeros((len(POS_tagging), len(POS_tagging)), dtype='float32')


for i, t1 in enumerate(tags):
     for j, t2 in enumerate(POS_tagging):
            Emission_matrix[i, j] = word_given_tag(t2[0], t2[1], POS_tagging)[0]



print("--> Emision Matrix")
print()
Emision_df = pd.DataFrame(Emission_matrix, columns = list(words), index=list(pos_tags))
display(Emision_df)
```

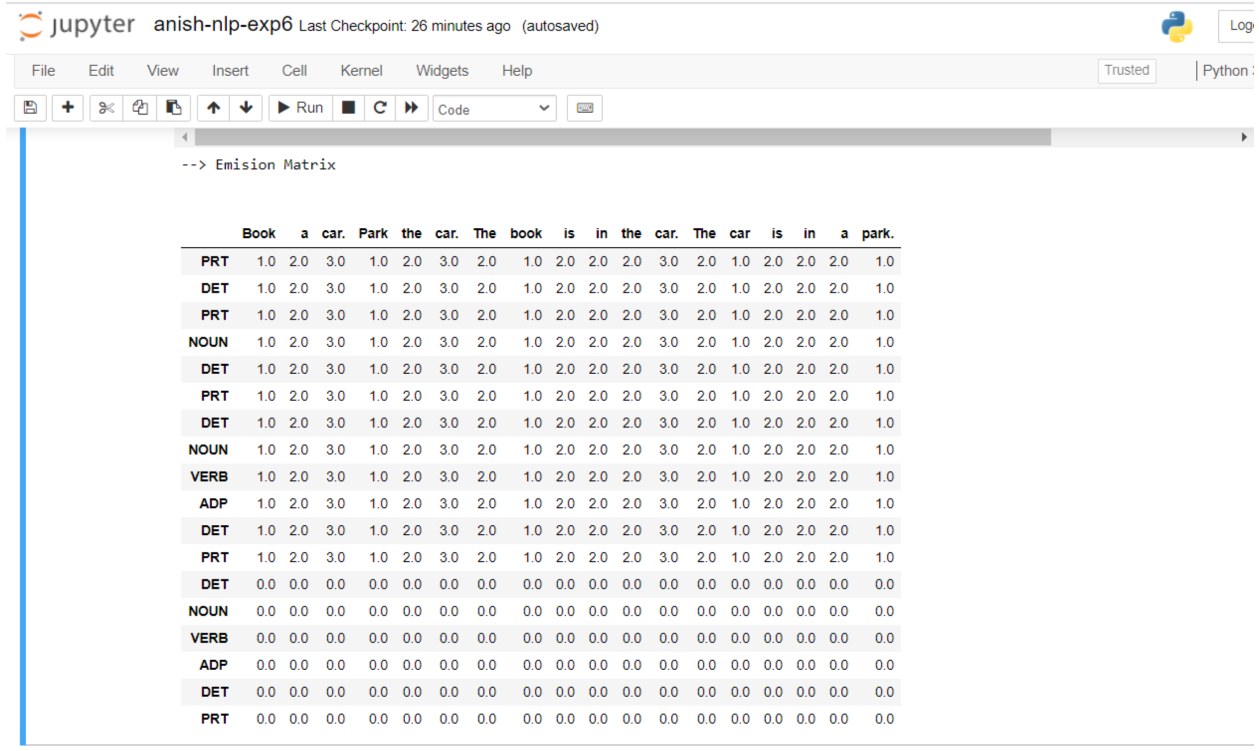
Output


jupyter anish-nlp-exp6 Last Checkpoint: 25 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
[nltk_data]   Package universal_tagset is already up-to-date!
```

```
--> POS Tagging of words

[('Book', 'PRT'), ('a', 'DET'), ('car.', 'PRT'), ('Park', 'NOUN'), ('the', 'DET'), ('car.', 'PRT'), ('The', 'DET'), ('book',
'NOUN'), ('is', 'VERB'), ('in', 'ADP'), ('the', 'DET'), ('car.', 'PRT'), ('The', 'DET'), ('car', 'NOUN'), ('is', 'VERB'),
('in', 'ADP'), ('a', 'DET'), ('park.', 'PRT')]
--> Transition Matrix
```

	PRT	DET	PRT	NOUN	DET	PRT	DET	NOUN	VERB	ADP	DET	PRT	DET	NOUN	VEI
PRT	0.001174	0.101370	0.001174	0.250489	0.101370	0.001174	0.101370	0.250489	0.401174	0.019569	0.101370	0.001174	0.101370	0.250489	0.4011
DET	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
PRT	0.000287	0.006037	0.000287	0.635906	0.006037	0.000287	0.006037	0.635906	0.040247	0.009918	0.006037	0.000287	0.006037	0.635906	0.0402
NOUN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
DET	0.043935	0.013106	0.043935	0.262344	0.013106	0.043935	0.013106	0.262344	0.149134	0.176827	0.013106	0.043935	0.013106	0.262344	0.1491
PRT	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
DET	0.001266	0.320931	0.001266	0.323589	0.320931	0.001266	0.320931	0.323589	0.008479	0.016958	0.320931	0.001266	0.320931	0.323589	0.0084
NOUN	0.030663	0.133610	0.030663	0.110589	0.133610	0.030663	0.133610	0.110589	0.167956	0.092357	0.133610	0.030663	0.133610	0.110589	0.1679
VERB	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
ADP	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
DET	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
PRT	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
DET	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
NOUN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
VERB	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
ADP	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
DET	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
PRT	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000


jupyter anish-nlp-exp6 Last Checkpoint: 26 minutes ago (autosaved)
Log
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code

```
--> Emision Matrix
```

	Book	a	car.	Park	the	car.	The	book	is	in	the	car.	The	car	is	in	a	park.
PRT	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
DET	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
PRT	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
NOUN	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
DET	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
PRT	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
DET	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
NOUN	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
VERB	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
ADP	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
DET	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
PRT	1.0	2.0	3.0	1.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	3.0	2.0	1.0	2.0	2.0	2.0	1.0
DET	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NOUN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
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
ADP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DET	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PRT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0