NLP ASSIGNMENT - 1

💡 Implement a program to perform
a) Tokenization
b) Word analysis.
Count the frequency of words
Identify words belonging to different POS tags
Explore library functions to do word analysis

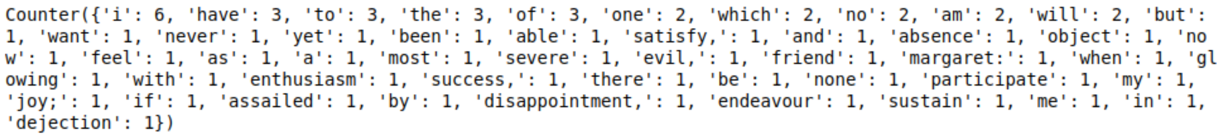
```
#TOKENIZATION
corpus = [] #Made a corpus list
sentence = ""
with open("text.txt") as f:
    lines = f.readline()
    lines = lines.lower()
    sentence+=lines
word=""
for char in sentence:
    if char == " " or char == ".":
        corpus.append(word)
        word=""
    word+=char

#Removing all the special chracters and punctuation marks using the repalce method

for i in range(len(corpus)):
    corpus[i] = corpus[i].replace(" ","")
    corpus[i] = corpus[i].replace(" ,","")
    corpus[i] = corpus[i].replace("'' ","")
    corpus[i] = corpus[i].replace("'","")
    corpus[i]=corpus[i].replace(".","")
    corpus[i]=corpus[i].replace("!","")
    corpus[i]=corpus[i].replace("$","")
    corpus[i]=corpus[i].replace("(","")
    corpus[i]=corpus[i].replace(")","")
    corpus[i]=corpus[i].replace("*","")
    corpus[i]=corpus[i].replace("%","")
    corpus[i]=corpus[i].replace("@","")

for char in corpus:
    if char=="" or char=="”":
        corpus.remove(char)
corpus
```

```
['but',
 'i',
 'have',
 'one',
 'want',
 'which',
 'i',
 'have',
 'never',
 'yet',
 'been',
 'able',
 'to',
 'satisfy,',
 'and',
 'the',
 'absence',
 'of',
 'the',
 'object',
 'of',
 'which',
 'i',
 'now',
 'feel',
 'as',
 'a',
 'most',
 'severe',
 'evil,',
 'i',
 'have',
 'no',
 'friend',
 'margaret:',
 'when',
 'i',
 'am',
 'glowing',
 'with',
 'the',
 'enthusiasm',
 'of',
 'success,',
 'there',
 'will',
 'be',
 'none',
 'to',
 'participate',
 'my',
 'joy;',
 'if',
 'i',
 'am',
 'assailed',
 'by',
 'disappointment,',
 'no',
 'one',
 'will',
 'endeavour',
 'to',
 'sustain',
 'me',
 'in',
 'dejection']
```

```
#SENTENCE TOKENIZATION

corpus_sentence = []
for sen in sentence.split("."):
    corpus_sentence.append(sen)

corpus_sentence
```

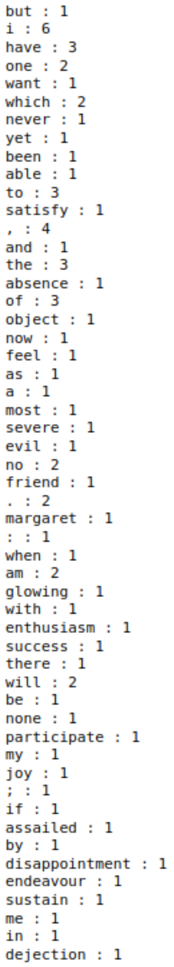
```
['but i have one want which i have never yet been able to satisfy, and the absence of the object of which i now feel as a most severe evil,
 ' margaret: when i am glowing with the enthusiasm of success, there will be none to participate my joy; if i am assailed by disappointment
 '']
```

```
1 b) Word Analysis. Count the frequency of words

from collections import Counter
freq_word = Counter()
for txt in corpus:
    freq_word[txt]+=1
print(freq_word)
```



```
Counter({'i': 6, 'have': 3, 'to': 3, 'the': 3, 'of': 3, 'one': 2, 'which': 2, 'no': 2, 'am': 2, 'will': 2, 'but':
1, 'want': 1, 'never': 1, 'yet': 1, 'been': 1, 'able': 1, 'satisfy,': 1, 'and': 1, 'absence': 1, 'object': 1, 'no
w': 1, 'feel': 1, 'as': 1, 'a': 1, 'most': 1, 'severe': 1, 'evil,': 1, 'friend': 1, 'margaret:': 1, 'when': 1, 'gl
owing': 1, 'with': 1, 'enthusiasm': 1, 'success,': 1, 'there': 1, 'be': 1, 'none': 1, 'participate': 1, 'my': 1,
'joy;': 1, 'if': 1, 'assailed': 1, 'by': 1, 'disappointment,': 1, 'endeavour': 1, 'sustain': 1, 'me': 1, 'in': 1,
'dejection': 1})
```

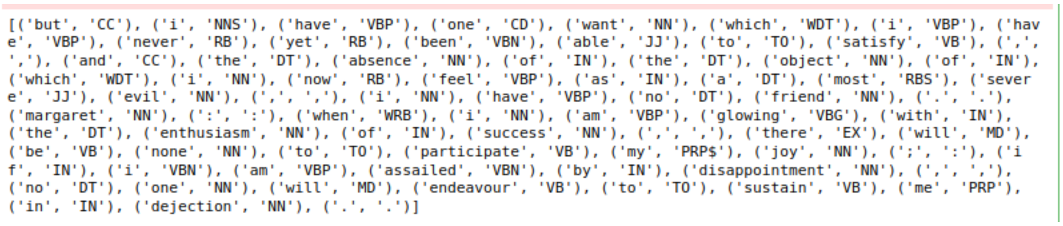
```
1)b) Finding the frequency using the nltk library

from nltk.probability import FreqDist
nltk_freq_word = nltk.FreqDist(tokens)
for key, value in nltk_freq_word.items():
    print(key,":", value)
```



```
but : 1
i : 6
have : 3
one : 2
want : 1
which : 2
never : 1
yet : 1
been : 1
able : 1
to : 3
satisfy : 1
, : 4
and : 1
the : 3
absence : 1
of : 3
object : 1
now : 1
feel : 1
as : 1
a : 1
most : 1
severe : 1
evil : 1
no : 2
friend : 1
. : 2
margaret : 1
: : 1
when : 1
am : 2
glowing : 1
with : 1
enthusiasm : 1
success : 1
there : 1
will : 2
be : 1
none : 1
participate : 1
my : 1
joy : 1
; : 1
if : 1
assailed : 1
by : 1
disappointment : 1
endeavour : 1
sustain : 1
me : 1
in : 1
dejection : 1
```

💡 1)b) Identify POS tagging

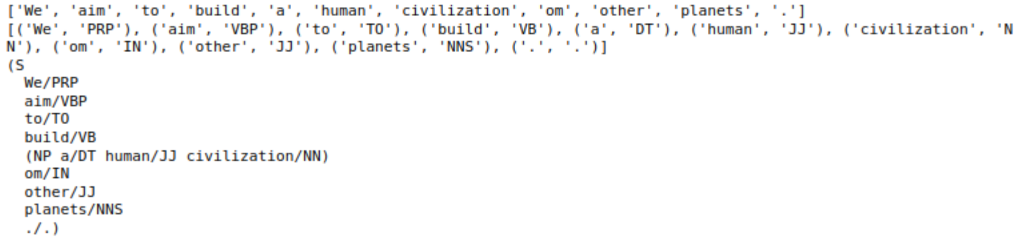
```
nltk.download('averaged_perceptron_tagger')
pos_tagged = nltk.pos_tag(tokens)
print(pos_tagged)
```



```
[('but', 'CC'), ('i', 'NNS'), ('have', 'VBP'), ('one', 'CD'), ('want', 'NN'), ('which', 'WDT'), ('i', 'VBP'), ('hav
e', 'VBP'), ('never', 'RB'), ('yet', 'RB'), ('been', 'VBN'), ('able', 'JJ'), ('to', 'TO'), ('satisfy', 'VB'), (',',
','), ('and', 'CC'), ('the', 'DT'), ('absence', 'NN'), ('of', 'IN'), ('the', 'DT'), ('object', 'NN'), ('of', 'IN'),
('which', 'WDT'), ('i', 'NN'), ('now', 'RB'), ('feel', 'VBP'), ('as', 'IN'), ('a', 'DT'), ('most', 'RBS'), ('sever
e', 'JJ'), ('evil', 'NN'), (',', ','), ('i', 'NN'), ('have', 'VBP'), ('no', 'DT'), ('friend', 'NN'), ('.', '.'),
('margaret', 'NN'), (':', ':'), ('when', 'WRB'), ('i', 'NN'), ('am', 'VBP'), ('glowing', 'VBG'), ('with', 'IN'),
('the', 'DT'), ('enthusiasm', 'NN'), ('of', 'IN'), ('success', 'NN'), (',', ','), ('there', 'EX'), ('will', 'MD'),
('be', 'VB'), ('none', 'NN'), ('to', 'TO'), ('participate', 'VB'), ('my', 'PRP$'), ('joy', 'NN'), (';', ':'), ('i
f', 'IN'), ('i', 'VBN'), ('am', 'VBP'), ('assailed', 'VBN'), ('by', 'IN'), ('disappointment', 'NN'), (',', ','),
('no', 'DT'), ('one', 'NN'), ('will', 'MD'), ('endeavour', 'VB'), ('to', 'TO'), ('sustain', 'VB'), ('me', 'PRP'),
('in', 'IN'), ('dejection', 'NN'), ('.', '.')]
```

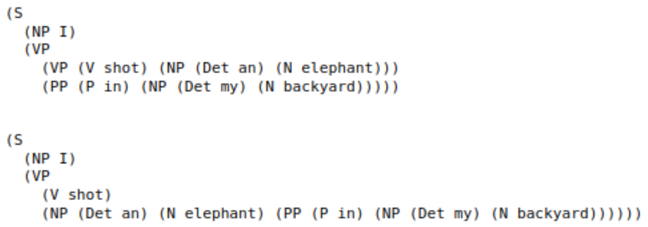
💡 2) Explore different parsers and write code snippets to show their implementation.

```
Regex Parser
from nltk import word_tokenize, pos_tag, RegexpParser

text = "We aim to build a human civilization om other planets."

tokens = word_tokenize(text)
print(tokens)

tag = pos_tag(tokens)
print(tag)

grammar = "NP: {<DT>?<JJ>*<NN>}"

cp = RegexpParser(grammar)

result = cp.parse(tag)

print(result)
```



```
['We', 'aim', 'to', 'build', 'a', 'human', 'civilization', 'om', 'other', 'planets', '.']
[('We', 'PRP'), ('aim', 'VBP'), ('to', 'TO'), ('build', 'VB'), ('a', 'DT'), ('human', 'JJ'), ('civilization', 'N
N'), ('om', 'IN'), ('other', 'JJ'), ('planets', 'NNS'), ('.', '.')]
(S
  We/PRP
  aim/VBP
  to/TO
  build/VB
  (NP a/DT human/JJ civilization/NN)
  om/IN
  other/JJ
  planets/NNS
  ./.)
```

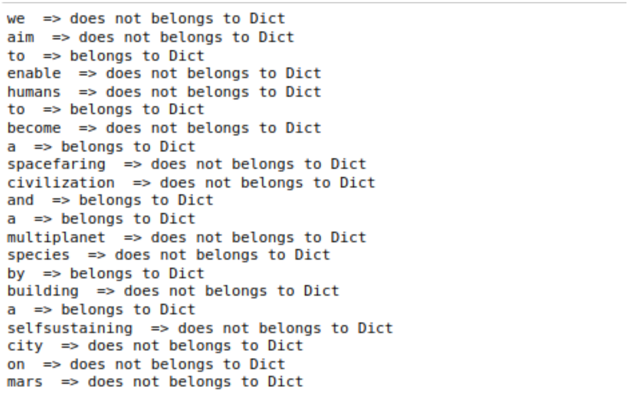
```
NLTK Chart Parser

from nltk import BottomUpChartParser, CFG

grammar = CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'backyard'
V -> 'shot'
P -> 'in'
""")

sentence_tokens = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'backyard']

parser = BottomUpChartParser(grammar)
trees = parser.parse(sentence_tokens)

for tree in trees:
    print(tree)
    print("\n")
```



```
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N backyard)))))

(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N backyard))))))
```

💡 Write a program that identifies whether a string or strings are part of the dictionary or not.

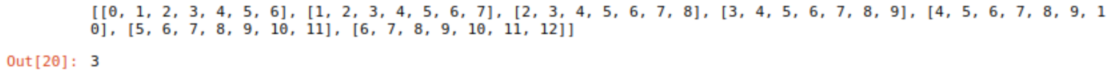
```
dictionary = freq_word
string = "We aim to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mar
string = string.replace(".","")
string = string.replace(",","")
string = string.replace("'","")
string = string.replace('"',"")
string = string.replace("-","")
string = string.lower()
for txt in string.split(" "):
  if txt in dictionary:
    print(txt," => belongs to Dict")
  else:
    print(txt,  " => does not belongs to Dict")
```



```
we  => does not belongs to Dict
aim  => does not belongs to Dict
to  => belongs to Dict
enable  => does not belongs to Dict
humans  => does not belongs to Dict
to  => belongs to Dict
become  => does not belongs to Dict
a  => belongs to Dict
spacefaring  => does not belongs to Dict
civilization  => does not belongs to Dict
and  => belongs to Dict
a  => belongs to Dict
multiplanet  => does not belongs to Dict
species  => does not belongs to Dict
by  => belongs to Dict
building  => does not belongs to Dict
a  => belongs to Dict
selfsustaining  => does not belongs to Dict
city  => does not belongs to Dict
on  => does not belongs to Dict
mars  => does not belongs to Dict
```

💡 Write a program to find minimum edit distance between two strings

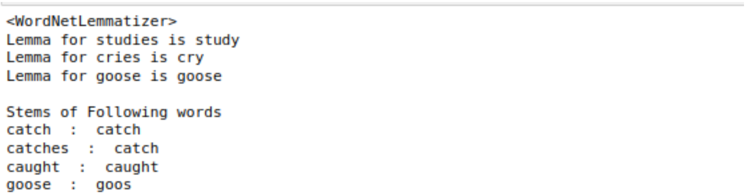
```
def med(string1,string2):
    n = len(string1)
    m = len(string2)
    matrix = [[i+j for j in range(m+1)] for i in range(n+1)]
    print(matrix)
    for i in range(1,n+1):
        for j in range(1,m+1):
            if string1[i-1] == string2[j-1]: #checking if the characters are simillar
                d=0
            else:
                d=1
            matrix[i][j] = min(matrix[i-1][j]+1, matrix[i][j-1]+1,matrix[i-1][j-1]+d)
    distance_score = matrix[n][m]

    return distance_score

med("Kitten","siettn")
```



```
[[0, 1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7], [2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9], [4, 5, 6, 7, 8, 9, 1
0], [5, 6, 7, 8, 9, 10, 11], [6, 7, 8, 9, 10, 11, 12]]
Out[20]: 3
```

💡 Write a program that takes a word and gives its morphological form as output

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
def lemmentization(string1):
    wordnet_lemmatizer = WordNetLemmatizer()
    print(wordnet_lemmatizer)


    tokenization = nltk.word_tokenize(string1) #reurns the root form of the word because lemmentisation
    #works with the context

    for w in tokenization:
        print("Lemma for {} is {}".format(w, wordnet_lemmatizer.lemmatize(w)))


def stemming(string1):
    ps = PorterStemmer()
    print(ps.stem(string1))

lemmentization("studies")
stemming("studies")
```

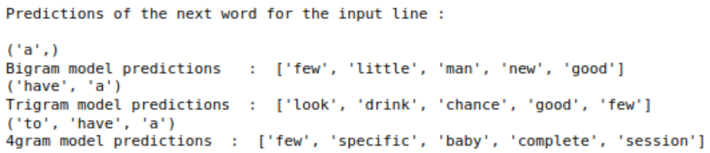


```
<WordNetLemmatizer>
Lemma for studies is study
Lemma for cries is cry
Lemma for goose is goose

Stems of Following words
catch  :  catch
catches  :  catch
caught  :  caught
goose  :  goos
```

💡 Write a program for word generation given a context

```
import nltk
from nltk import word_tokenize, sent_tokenize
from nltk.util import ngrams
from nltk.corpus import brown
# nltk.download('brown')
```

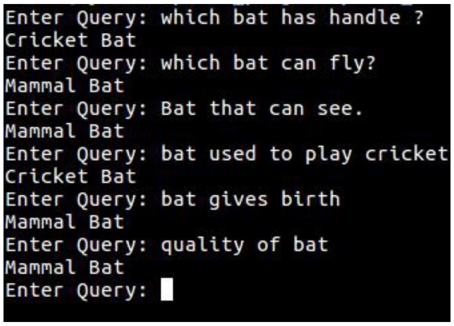
```
corpus_tokens = brown.words()
input_string =""
# with open("frankestien.txt") as f:
#   lines = f.readline()
#   lines=lines.lower()
#   input_string+=lines
# input_string = """From fairest creatures we desire increase,
#   That thereby beauty's rose might never die,
#   But as the riper should by time decease,
#   His tender heir might bear his memory:
#   But thou, contracted to thine own bright eyes,
#   Feed'st thy light's flame with self-substantial fuel,
#   Making a famine where abundance lies,
#   Thy self thy foe, to thy sweet self too cruel:
#   Thou that art now the world's fresh ornament,
#   And only herald to the gaudy spring,
#   Within thine own bud buriest thy content,
#   And tender churl mak'st waste in niggarding:
#     Pity the world, or else this glutton be,
#     To eat the world's due, by the grave and thee."""
# input_string = input("Enter input string: ")
input_string ="I would like to have a"
input_string_tokens = word_tokenize(input_string)


def word_predictor(n):
    frequencies = nltk.FreqDist(ngrams(corpus_tokens, n))
    frequencies_list = [(k, v) for k, v in dict(frequencies).items()]
    frequencies_list = sorted(
        frequencies_list, key=lambda x: x[-1], reverse=True)
    ngram = tuple(ngrams(input_string_tokens, n-1))[-1]
    predictions = []
    count = 0
    print(ngram)
    for each in frequencies_list:
        if each[0][:-1] == ngram:
            count += 1
            predictions.append(each[0][-1])
            if count == 5:
                break

    if count < 5:
        while(count != 5):
            predictions.append("NONE")
            count += 1

    return predictions


print("\nPredictions of the next word for the input line : \n")
print("Bigram model predictions   : ", word_predictor(2))
print("Trigram model predictions  : ", word_predictor(3))
print("4gram model predictions  : ", word_predictor(4))
```



```
Predictions of the next word for the input line :

('a',)
Bigram model predictions   :  ['few', 'little', 'man', 'new', 'good']
('have', 'a')
Trigram model predictions  :  ['look', 'drink', 'chance', 'good', 'few']
('to', 'have', 'a')
4gram model predictions  :  ['few', 'specific', 'baby', 'complete', 'session']
```

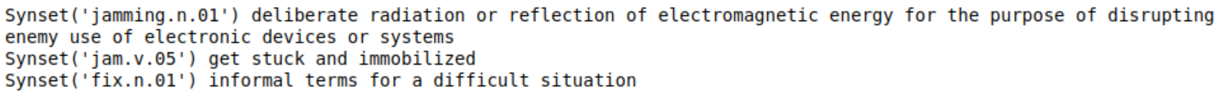
💡 Write a program that take a sentence as input and performs word sense disambiguation using the context

```
import nltk
import codecs
from nltk.tokenize import PunktSentenceTokenizer,sent_tokenize, word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer, PorterStemmer

def filteredSentence(sentence):

    filtered_sent = []
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words("english"))
    words = word_tokenize(sentence)

    for w in words:
        if w not in stop_words:
            filtered_sent.append(lemmatizer.lemmatize(w))

    return filtered_sent

def simlilarityCheck(word1, word2):

    word1 = word1 + ".n.01"
    word2 = word2 + ".n.01"
    try:
        w1 = wordnet.synset(word1)
        w2 = wordnet.synset(word2)

        return w1.wup_similarity(w2)

    except:
        return 0

def synonymsCreator(word):
    synonyms = []

    for syn in wordnet.synsets(word):
        for i in syn.lemmas():
            synonyms.append(i.name())

    return synonyms

if __name__ == '__main__':

    cricfile = codecs.open("cricketbat.txt", 'r', "utf-8")
    sent2 = cricfile.read().lower()
    vampirefile = codecs.open("vampirebat.txt", 'r', 'utf-8')
    sent1 = vampirefile.read().lower()
    sent3 = "start"

    # FOR TEST , replace the above variables with below sent1 and sent 2
    # sent1 = "the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots of mo
    # sent2 = "the river bank has water in it and it has fishes trees . lots of water is stored in the banks. boats float in it and animals
    # sent3 = "from which bank should i withdraw money"

    while(sent3 != "end"):

        sent3 = input("Enter Query: ").lower()

        filtered_sent1 = []
        filtered_sent2 = []
        filtered_sent3 = []

        counter1 = 0
        counter2 = 0
        sent31_similarity = 0
        sent32_similarity = 0

        filtered_sent1 = simpleFilter(sent1)
        filtered_sent2 = simpleFilter(sent2)
        filtered_sent3 = simpleFilter(sent3)

        for i in filtered_sent3:

            for j in filtered_sent1:
                counter1 = counter1 + 1
                sent31_similarity = sent31_similarity + simlilarityCheck(i,j)

            for j in filtered_sent2:
                counter2 = counter2 + 1
                sent32_similarity = sent32_similarity + simlilarityCheck(i,j)

        filtered_sent1 = []
        filtered_sent2 = []
        filtered_sent3 = []
        filtered_sent1 = filteredSentence(sent1)
        filtered_sent2 = filteredSentence(sent2)
        filtered_sent3 = filteredSentence(sent3)

        sent1_count = 0
        sent2_count = 0

        for i in filtered_sent3:

            for j in filtered_sent1:

                if(i==j):
                    sent1_count = sent1_count + 1

            for j in filtered_sent2:
                if(i==j):
                    sent2_count = sent2_count + 1

        if((sent1_count + sent31_similarity)>(sent2_count+sent32_similarity)):
            print ("Mammal Bat")
        else:
            print ("Cricket Bat")

        #-----------------------------------------------
        #Sentence1: the river bank has water in it and it has fishes trees . lots of water is stored in the banks. boats float in it and an
        #sentence2: the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots
        #query: from which bank should i withdraw money.

        #sen1: any of various nocturnal flying mammals of the order Chiroptera, having membranous wings that extend from the forelimbs to t
        #sen 2: a cricket wooden bat is used for playing criket. it is rectangular in shape and has handle and is made of wood or plastic a
    print ("\nTERMINATED")
```



```
Enter Query: which bat has handle ?
Cricket Bat
Enter Query: which bat can fly?
Mammal Bat
Enter Query: Bat that can see.
Mammal Bat
Enter Query: bat used to play cricket
Cricket Bat
Enter Query: bat gives birth
Mammal Bat
Enter Query: quality of bat
Mammal Bat
Enter Query: 
```

```
Using the NLTK library

from nltk.wsd import lesk
from nltk.tokenize import word_tokenize

a1= lesk(word_tokenize('This device is used to jam the signal'),'jam')
print(a1,a1.definition())
a2 = lesk(word_tokenize('I am stuck in a traffic jam'),'jam')
print(a2,a2.definition())
a3 = lesk(word_tokenize('Today is a jam packed day'),'jam')
print(a3,a3.definition())
```

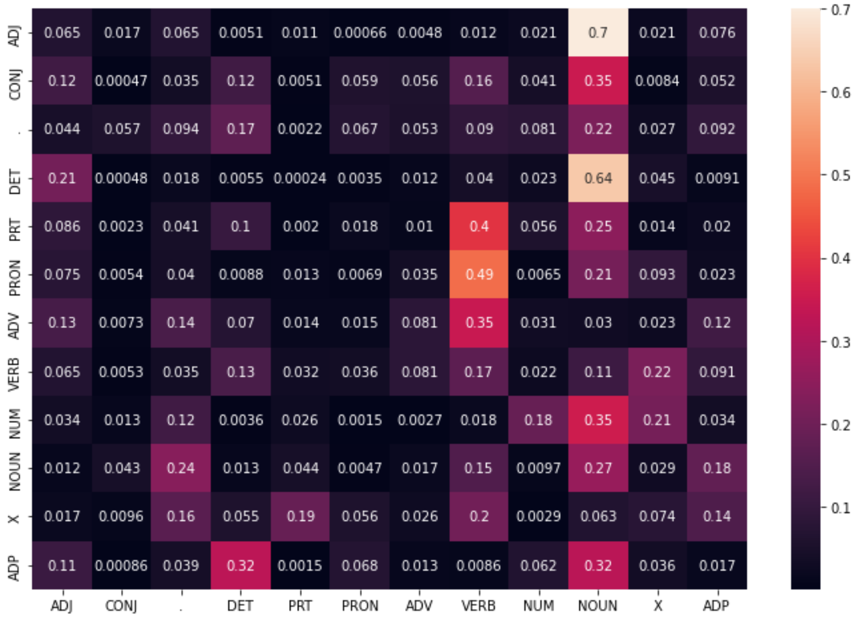


```
Synset('jamming.n.01') deliberate radiation or reflection of electromagnetic energy for the purpose of disrupting
enemy use of electronic devices or systems
Synset('jam.v.05') get stuck and immobilized
Synset('fix.n.01') informal terms for a difficult situation
```

💡 Identify the most probable POS tag using Viterbi algoIdentify the most probable POS tag using Viterbi algo

```
pos_tag_dataset = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))
print(pos_tag_dataset[:5])

random.seed(1000)
train_set, test_set = train_test_split(pos_tag_dataset, train_size = 0.95, test_size = 0.05)

print(len(train_set)) #3718
print(len(test_set))#196

#  Split train and test set tags and tokens
train_set_tuples = [tup for sent in train_set for tup in sent]
test_set_tuples = [tup for sent in test_set for tup in sent]

train_tagged_tokens = [tag[0] for tag in train_set_tuples]
train_tagged_pos_tags = [tag[1] for tag in train_set_tuples]
test_tagged_tokens = [tag[0] for tag in test_set_tuples]
test_tagged_pos_tags = [tag[0] for tag in test_set_tuples]
print(train_tagged_tokens[:10])
print(train_tagged_pos_tags[:10])

/*['In', 'September', ',', 'the', 'custom-chip', 'maker', 'said', '0', 'excess', 'capacity']
['ADP', 'NOUN', '.', 'DET', 'ADJ', 'NOUN', 'VERB', 'X', 'ADJ', 'NOUN']*/


# Build vocabulary sets for words and tags
train_vocab_set = set(train_tagged_tokens)
train_pos_tag_set = set(train_tagged_pos_tags)
len_vocab_set = len(train_vocab_set)
len_pos_tags = len(train_pos_tag_set)
print(f"Number of words in training set are: {len_vocab_set}")
print(f"Number of pos tags in training set are: {len_pos_tags}")

/* Number of words in training set are: 12088
Number of pos tags in training set are: 12 */

#  Compute emission probabilities for a given word for a given tag
def word_given_tag(word, tag, train_bag = train_set_tuples):
    tag_list = [pair for pair in train_bag if pair[1]==tag] # get all pairs containing given tag
    tag_count = len(tag_list)
    word_given_tag_count = 0
    for pair in tag_list:
        if pair[0] == word:
            word_given_tag_count+=1
    return (word_given_tag_count, tag_count)


#  compute transition probabilities of a previous and next tag
def t2_given_t1(t2, t1, train_bag = train_set_tuples):
    tags = [pair[1] for pair in train_bag]
    t1_tag_count = 0
    for tag in tags:
        if tag==t1:
            t1_tag_count +=1

    t2_given_t1_count = 0
    for indx in range(len(tags)-1):
        if tags[indx]== t1 and tags[indx+1]==t2:
            t2_given_t1_count+=1

    return (t2_given_t1_count, t1_tag_count)

# create a transition matrix for training_pos_tag_set
# M(i, j) represents p(tj given ti)
tags_matrix = np.zeros((len_pos_tags, len_pos_tags), dtype='float32')
for i, t1 in enumerate(list(train_pos_tag_set)):
    for j , t2 in enumerate(list(train_pos_tag_set)):
        tupl = t2_given_t1(t2, t1)
        tags_matrix[i, j] = tupl[0]/tupl[1]

tags_df = pd.DataFrame(tags_matrix, columns=list(train_pos_tag_set), index=list(train_pos_tag_set))
tags_df
```

```
# Visualizations
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.heatmap(tags_df, annot=True)
plt.show()


# Viterbi algorithm
def viterbi_algo(words, train_bag = train_set_tuples):
    state = []
    T = list(set([pair[1] for pair in train_bag]))

    for key, word in enumerate(words):
        p = [] # Probability for observations
        for tag in T:
            if key==0: # start word
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]

            # compute emission and state probabilities
            tupl = word_given_tag(words[key], tag)
            emission_p = tupl[0]/tupl[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))

# testing viterbi algorithm
random.seed(1000)

# choosing 5 random sentences
rndom = [random.randint(1, len(test_set)) for x in range(5)]
#  list of sentences
test_run = [test_set[i] for i in rndom]
# list of tagged words
test_run_base = [tup for sent in test_run for tup in sent]
# list of untagged words
test_tagged_words = [tup[0] for sent in test_run for tup in sent]

# finding tags using viterbi algo
tagged_seq = viterbi_algo(test_tagged_words)
# finding accuracy
viterbi_word_check = [i for i, j in zip(tagged_seq, test_run_base) if i==j]
viterbi_accuracy = len(viterbi_word_check)/len(tagged_seq)*100
print(viterbi_accuracy)


OUTPUT : 92.66055045871559
```



	ADJ	CONJ	.	DET	PRT	PRON	ADV	VERB	NUM	NOUN	X	ADP
ADJ	0.065	0.017	0.065	0.0051	0.011	0.00066	0.0048	0.012	0.021	0.7	0.021	0.076
CONJ	0.12	0.00047	0.035	0.12	0.0051	0.059	0.056	0.16	0.041	0.35	0.0084	0.052
.	0.044	0.057	0.094	0.17	0.0022	0.067	0.053	0.09	0.081	0.22	0.027	0.092
DET	0.21	0.00048	0.018	0.0055	0.00024	0.0035	0.012	0.04	0.023	0.64	0.045	0.0091
PRT	0.086	0.0023	0.041	0.1	0.002	0.018	0.01	0.4	0.056	0.25	0.014	0.02
PRON	0.075	0.0054	0.04	0.0088	0.013	0.0069	0.035	0.49	0.0065	0.21	0.093	0.023
ADV	0.13	0.0073	0.14	0.07	0.014	0.015	0.081	0.35	0.031	0.03	0.023	0.12
VERB	0.065	0.0053	0.035	0.13	0.032	0.036	0.081	0.17	0.022	0.11	0.22	0.091
NUM	0.034	0.013	0.12	0.0036	0.026	0.0015	0.0027	0.018	0.18	0.35	0.21	0.034
NOUN	0.012	0.043	0.24	0.013	0.044	0.0047	0.017	0.15	0.0097	0.27	0.029	0.18
X	0.017	0.0096	0.16	0.055	0.19	0.056	0.026	0.2	0.0029	0.063	0.074	0.14
ADP	0.11	0.00086	0.039	0.32	0.0015	0.068	0.013	0.0086	0.062	0.32	0.036	0.017