**NLP - LAB 2** **Date : 02- 02- 24**

**AIM :**

Write a python NLTK program to split the text from genesis corpus and display it into a list of words . Remove the Punctuations and Stopwords from given text and perform Stemming and POS tagging .

**RESOURCES** **:** Google Colab / Jupyter notebook.

**OBJECTIVES :**

To preprocess and analyze text data for extracting useful insights. This involves several tasks, including tokenization to split the text into individual words or sentences, part-of-speech tagging to label each word with its grammatical category, stemming to reduce words to their root forms, and punctuation removal to clean the text. Additionally, the codes utilize chunking to identify and group words based on specific patterns or combinations of parts of speech , for various nlp applications.

**PROCEDURE :**

* Import the required libraries.
* Download the required packages , with a particular language
* Load the Genesis corpus.
* Tokenize the text into words.
* Remove punctuation.
* Remove Stopwords
* Perform Stemming.
* Perform POS tagging.
* Display the list of words after performing each operation.

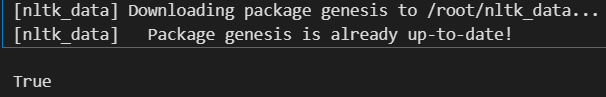
**Code :**



import nltk

nltk.download('genesis')

**Output :**

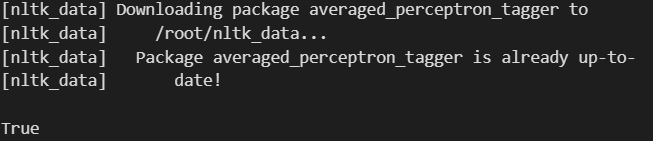


Here we have imported the NLTK library for natural language processing in Python and downloads the "genesis" corpus, and gives access to the corpus for tasks such as text analysis within NLTK.

**Code :**

nltk.download('averaged\_perceptron\_tagger')

**Output :**



**Code :**

from nltk.corpus import genesis

#access text of specific translation

genesis\_text = genesis.words('english-kjv.txt')

#genesis\_text = genesis.words('english-web.txt')

#genesis\_text = genesis.words('lolcat.txt')

print(genesis\_text)

**Output :**



In this segment we have imported the NLTK corpus module and accesses genesis ,It then assigns the words from this text to the variable genesis\_text. By printing genesis\_text, it displays the words in genesis . We can also choose different variations like 'lolcat.txt'

**Code :**

# Analyze word freq, POS tag (Part of Speech), etc.

from nltk import FreqDist, pos\_tag

fdist = FreqDist(genesis\_text)

fdist.most\_common(50) # Top 50 freq words

tagged\_words = pos\_tag(genesis\_text)

print(tagged\_words[:5]) # Display first 15 words with POS tags

**Output :**



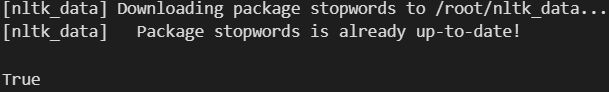
Here we have utilized NLTK's FreqDist to compute the frequency distribution of words in genesis ,retrieves the top 50 most common words. Additionally, it employs NLTK's pos\_tag function to assign parts-of-speech tags to each word in the Genesis text and prints the tags for the first five words. This helps analyze word frequencies and syntactic structures within the text.



**Code :**

nltk.download('stopwords')

**Output :**



****

**Code :**

from nltk.corpus import stopwords

en\_stops = set(stopwords.words('english'))

all\_words = [""]

word\_str=""

for word in genesis\_text[:31]:

if word not in en\_stops:

word\_str += f"{word} "

word\_str

**Output :**



In this segment we import the NLTK's stopwords for the English language, which are common words often filtered out during text analysis. It then iterates over the first 31 words of genesis .The resulting string, word\_str, concatenates the non - stopwords into a single string for further processing or analysis.

**Code :**

from nltk.tokenize import punkt

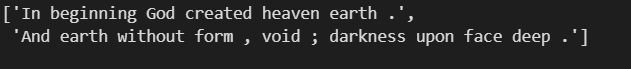
tokenizer = punkt.PunktSentenceTokenizer()

text="This is a sentence. This is another sentence. This is the last sentence."

sentences=tokenizer.tokenize(word\_str)

sentences

**Output :**



This code initializes a tokenizer from the punkt module, specifically a PunktSentenceTokenizer. It then applies this tokenizer to the string word\_str, which likely contains concatenated non-stopword words from genesis.The tokenizer splits the text into individual sentences, which are stored in the variable sentences.



**Code :**

from nltk.stem import SnowballStemmer

# Create a Snowball Stemmer object for English

stemmer = SnowballStemmer(language='english')

# Define a list of words to be stemmed

words= ['running', 'ran', 'runs']

# Iterate over each word and stem it using Snowball Stemmer

stemmed\_words=[]

for word in genesis\_text[:31]:

stemmed\_word=stemmer.stem(word)

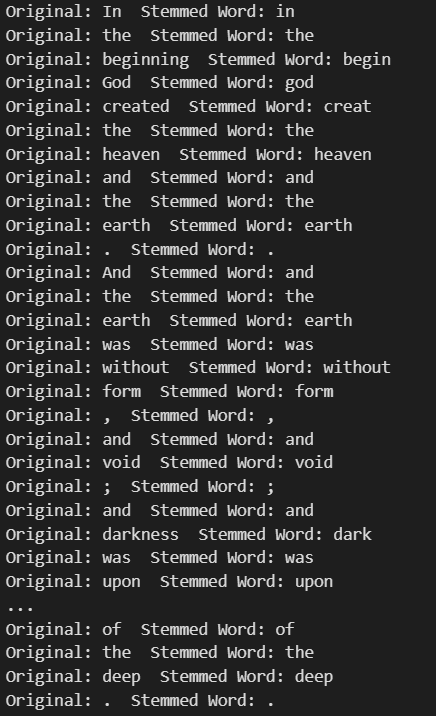
stemmed\_words.append(stemmed\_word)

# Print original and stemmed words

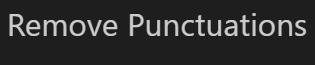
for i in range(31):

print(f'Original: {genesis\_text[i]} Stemmed Word: {stemmed\_words[i]}')

**Output :**



Here we have imported the SnowballStemmer from NLTK to perform stemming on English words. It initializes the stemmer object and defines a list of words to be stemmed. Then, it iterates over the first 31 words of genesis ,, stemming each word using the Snowball Stemmer, and prints both the original and stemmed versions of each word. This process helps in reducing words to their root forms for analysis or comparison purposes.



**Code :**

import string

text\_with\_punctuation = word\_str

text\_without\_punctuation = text\_with\_punctuation.translate(str.maketrans('', '', string.punctuation))

print(text\_without\_punctuation)

**Output :**



Here we have imported the string module and retrieved the variable word\_str containing a string of words. It then utilizes the translate() method to remove all punctuation from the text by creating a translation table where each punctuation character is mapped to None. The resulting text without punctuation is then printed. And it is helpful for text preprocessing before nlp tasks.



**Code :**

from nltk import pos\_tag

from nltk import RegexpParser

text=word\_str.split()

print("After Split:", text)

tokens\_tag=pos\_tag(text)

print("After Token",tokens\_tag)

patterns="""mychunk:{<NN.?>\*<VBD.?>\*<JJ.?>\*<CC>?}"""

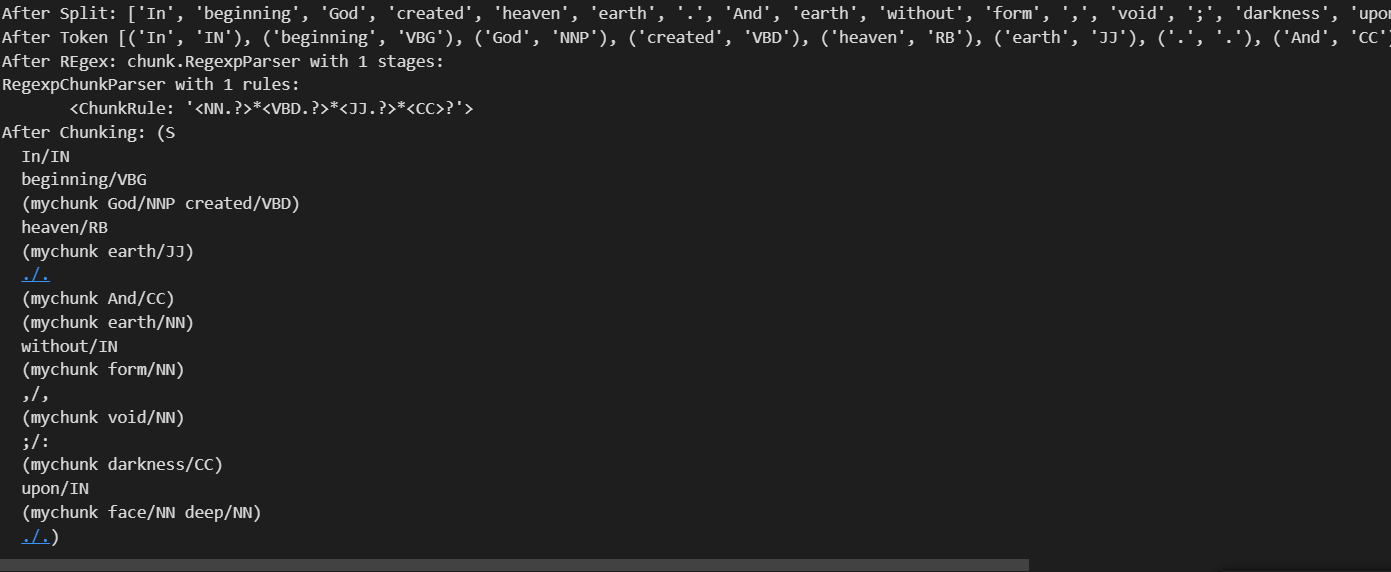
chunker = RegexpParser(patterns)

print("After REgex:",chunker)

output = chunker.parse(tokens\_tag)

print("After Chunking:",output)

**Output :**



Here we have applied the technique in which we ,tokenize the input text into a list of words and assigns part-of-speech tags to each word using NLTK's pos\_tag function. It then defines a regular expression pattern for chunking based on specific combinations of noun, verb, adjective, and conjunction tags. This pattern is applied to the tagged tokens using a RegexpParser object, resulting in the identification of chunks based on the defined pattern .