**Reg No : RA1911003011043**

**Name : RAMNATH**

**Exp 10 - Implementation of NLP Programs**

**Aim:-**To Implement NLP programs

NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,** and **topic segmentation**.

**code:-**

!pip install -q wordcloud

import wordcloud

import nltk

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('punkt')

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

import pandas as pd

import matplotlib.pyplot as plt

import io

import unicodedata

import numpy as np

import re

import string

# Constants

# POS (Parts Of Speech) for: nouns, adjectives, verbs and adverbs

DI\_POS\_TYPES = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}

POS\_TYPES = list(DI\_POS\_TYPES.keys())

# Constraints on tokens

MIN\_STR\_LEN = 3

RE\_VALID = '[a-zA-Z]'

# Upload from google drive

from google.colab import files

uploaded = files.upload()

print("len(uploaded.keys():", len(uploaded.keys()))

for fn in uploaded.keys():

print('User uploaded file "{name}" with length {length} bytes'.format(name=fn, length=len(uploaded[fn])))

# Get list of quotes

df\_quotes = pd.read\_csv(io.StringIO(uploaded['quotes.txt'].decode('utf-8')), sep='\t')

# Display

print("df\_quotes:")

print(df\_quotes.head().to\_string())

print(df\_quotes.describe())

# Convert quotes to list

li\_quotes = df\_quotes['Quote'].tolist()

print()

print("len(li\_quotes):", len(li\_quotes)

# Get stopwords, stemmer and lemmatizer

stopwords = nltk.corpus.stopwords.words('english')

stemmer = nltk.stem.PorterStemmer()

lemmatizer = nltk.stem.WordNetLemmatizer()

# Remove accents function

def remove\_accents(data):

return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in string.ascii\_letters or x == " ")

# Process all quotes

li\_tokens = []

li\_token\_lists = []

li\_lem\_strings = []

for i,text in enumerate(li\_quotes):

# Tokenize by sentence, then by lowercase word

tokens = [word.lower() for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]

# Process all tokens per quote

li\_tokens\_quote = []

li\_tokens\_quote\_lem = []

for token in tokens:

# Remove accents

t = remove\_accents(token)

# Remove punctuation

t = str(t).translate(string.punctuation)

li\_tokens\_quote.append(t)

# Add token that represents "no lemmatization match"

li\_tokens\_quote\_lem.append("-") # this token will be removed if a lemmatization match is found below

# Process each token

if t not in stopwords:

if re.search(RE\_VALID, t):

if len(t) >= MIN\_STR\_LEN:

# Note that the POS (Part Of Speech) is necessary as input to the lemmatizer

# (otherwise it assumes the word is a noun)

pos = nltk.pos\_tag([t])[0][1][:2]

pos2 = 'n' # set default to noun

if pos in DI\_POS\_TYPES:

pos2 = DI\_POS\_TYPES[pos]

stem = stemmer.stem(t)

lem = lemmatizer.lemmatize(t, pos=pos2) # lemmatize with the correct POS

if pos in POS\_TYPES:

li\_tokens.append((t, stem, lem, pos))

# Remove the "-" token and append the lemmatization match

li\_tokens\_quote\_lem = li\_tokens\_quote\_lem[:-1]

li\_tokens\_quote\_lem.append(lem)

# Build list of token lists from lemmatized tokens

li\_token\_lists.append(li\_tokens\_quote)

# Build list of strings from lemmatized tokens

str\_li\_tokens\_quote\_lem = ' '.join(li\_tokens\_quote\_lem)

li\_lem\_strings.append(str\_li\_tokens\_quote\_lem)

# Build resulting dataframes from lists

df\_token\_lists = pd.DataFrame(li\_token\_lists)

print("df\_token\_lists.head(5):")

print(df\_token\_lists.head(5).to\_string())

# Replace None with empty string

for c in df\_token\_lists:

if str(df\_token\_lists[c].dtype) in ('object', 'string\_', 'unicode\_'):

df\_token\_lists[c].fillna(value='', inplace=True)

df\_lem\_strings = pd.DataFrame(li\_lem\_strings, columns=['lem quote'])

print()

print("")

print("df\_lem\_strings.head():")

print(df\_lem\_strings.head().to\_string())

# Add counts

print("Group by lemmatized words, add count and sort:")

df\_all\_words = pd.DataFrame(li\_tokens, columns=['token', 'stem', 'lem', 'pos'])

df\_all\_words['counts'] = df\_all\_words.groupby(['lem'])['lem'].transform('count')

df\_all\_words = df\_all\_words.sort\_values(by=['counts', 'lem'], ascending=[False, True]).reset\_index()

print("Get just the first row in each lemmatized group")

df\_words = df\_all\_words.groupby('lem').first().sort\_values(by='counts', ascending=False).reset\_index()

print("df\_words.head(10):")

print(df\_words.head(10))

df\_words = df\_words[['lem', 'pos', 'counts']].head(200)

for v in POS\_TYPES:

df\_pos = df\_words[df\_words['pos'] == v]

print()

print("POS\_TYPE:", v)

print(df\_pos.head(10).to\_string())

li\_token\_lists\_flat = [y for x in li\_token\_lists for y in x] # flatten the list of token lists to a single list

print("li\_token\_lists\_flat[:10]:", li\_token\_lists\_flat[:10])

di\_freq = nltk.FreqDist(li\_token\_lists\_flat)

del di\_freq['']

li\_freq\_sorted = sorted(di\_freq.items(), key=lambda x: x[1], reverse=True) # sorted list

print(li\_freq\_sorted)

di\_freq.plot(30, cumulative=False)

li\_lem\_words = df\_all\_words['lem'].tolist()

di\_freq2 = nltk.FreqDist(li\_lem\_words)

li\_freq\_sorted2 = sorted(di\_freq2.items(), key=lambda x: x[1], reverse=True) # sorted list

print(li\_freq\_sorted2)

di\_freq2.plot(30, cumulative=False)

**Output:-**









**Result:-**Thus the NPL program was implemented