**Practical No: 1**

**AIM a): Study different libraries used for NLP in python.**

**NLTK (Natural Language Toolkit)**

NLTK is a comprehensive library for natural language processing in Python, providing easy-to-use interfaces to over 50 corpora and lexical resources. It supports various text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Its educational resources and practical tools make it ideal for beginners and experts alike..

**SpaCy**

SpaCy is an advanced NLP library designed for high performance and production use. It features state-of-the-art tokenization, part-of-speech tagging, named entity recognition, and dependency parsing. SpaCy is known for its speed and efficiency, capable of processing large volumes of text quickly.

**Gensim**

Gensim is a robust library for topic modeling and document similarity analysis using algorithms like Word2Vec, LDA, and LSI. It excels at processing large text corpora and is optimized for performance and scalability. Gensim's efficient implementations of various vector space algorithms make it a favorite for researchers and developers working on unsupervised machine learning tasks.

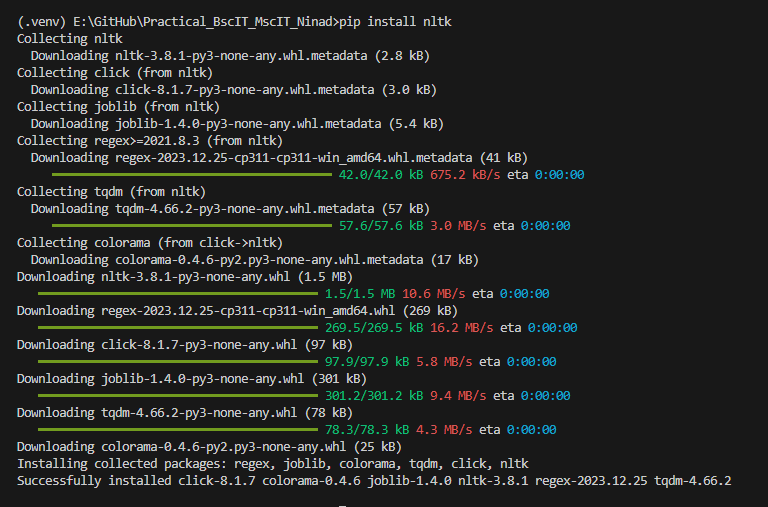
**TextBlob**

TextBlob is a simple library for processing textual data, offering easy-to-use tools for common NLP tasks. It supports part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. TextBlob is built on top of NLTK and Pattern, providing a user-friendly interface while leveraging the power of these libraries.

**Polyglot**

Polyglot is a multilingual NLP library that supports tokenization, named entity recognition, part-of-speech tagging, sentiment analysis, and word embeddings in multiple languages. It is designed to handle text in over 160 languages, making it a versatile tool for global NLP applications. Polyglot utilizes efficient algorithms and data structures, ensuring high performance and accuracy in processing diverse languages.

**AIM b): Install NLTK Package.**



**AIM c): Convert the given text into speech.**

**Description:**

**Code:**

# Import the required module for text to speech conversion

#!pip install gtts

from gtts import gTTS

# This module is imported so that we can play the converted audio

import os

# The text that you want to convert to audio

mytext = "Hello Everyone!My name is Ninad"

# Language in which you want to convert

language = "en"

# Passing the text and language to the engine, here we have marked slow=False. Which tells the module that the converted audio should have a high speed

myobj = gTTS(text=mytext, lang=language, slow=False)

# Saving the converted audio in a mp3 file named welcome

myobj.save("welcomeNK.mp3")

# Playing the converted file

os.system("mpg321 welcomeNK.mp3")

**Output:**

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**AIM d): Convert the speech to text**

**Description:**

**Code:**

#Aim: Convert audio file Speech to Text.

#Note: required to store the input file "NLP\_test.wav" in the current folder before running the program.

!pip install SpeechRecognition pydub

import speech\_recognition as sr

filename = "/content/NLP\_test.wav"

# initialize the recognizer

r = sr.Recognizer()

# open the file

with sr.AudioFile(filename) as source:

# listen for the data (load audio to memory)

audio\_data = r.record(source)

# recognize (convert from speech to text)

text = r.recognize\_google(audio\_data)

print(text)

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**Practical No: 2**

**AIM a) Study of various corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories.**

**Code:**

import nltk

from nltk.corpus import brown

nltk.download('brown')

# Display file ids of brown corpus

print('File ids of brown corpus\n', brown.fileids())

# Pick out the first of these texts — Emma by Jane Austen — and give it a short name, ca01

ca01 = brown.words('ca01')

# Display first few words

print('\nca01 has the following words:\n', ca01[:20])

# Total number of words in ca01

print('\nca01 has', len(ca01), 'words')

# Categories or files in brown corpus

print('\n\nCategories or files in brown corpus:\n')

print(brown.categories())

# Display other information about each text by looping over all the values of fileid

# and then computing statistics for each text.

print('\n\nStatistics for each text:\n')

print('AvgWordLen\tAvgSentenceLen\tNo. of Times Each Word Appears On Avg\tFileName')

for fileid in brown.fileids():

num\_chars = len(brown.raw(fileid))

num\_words = len(brown.words(fileid))

num\_sents = len(brown.sents(fileid))

num\_vocab = len(set(w.lower() for w in brown.words(fileid)))

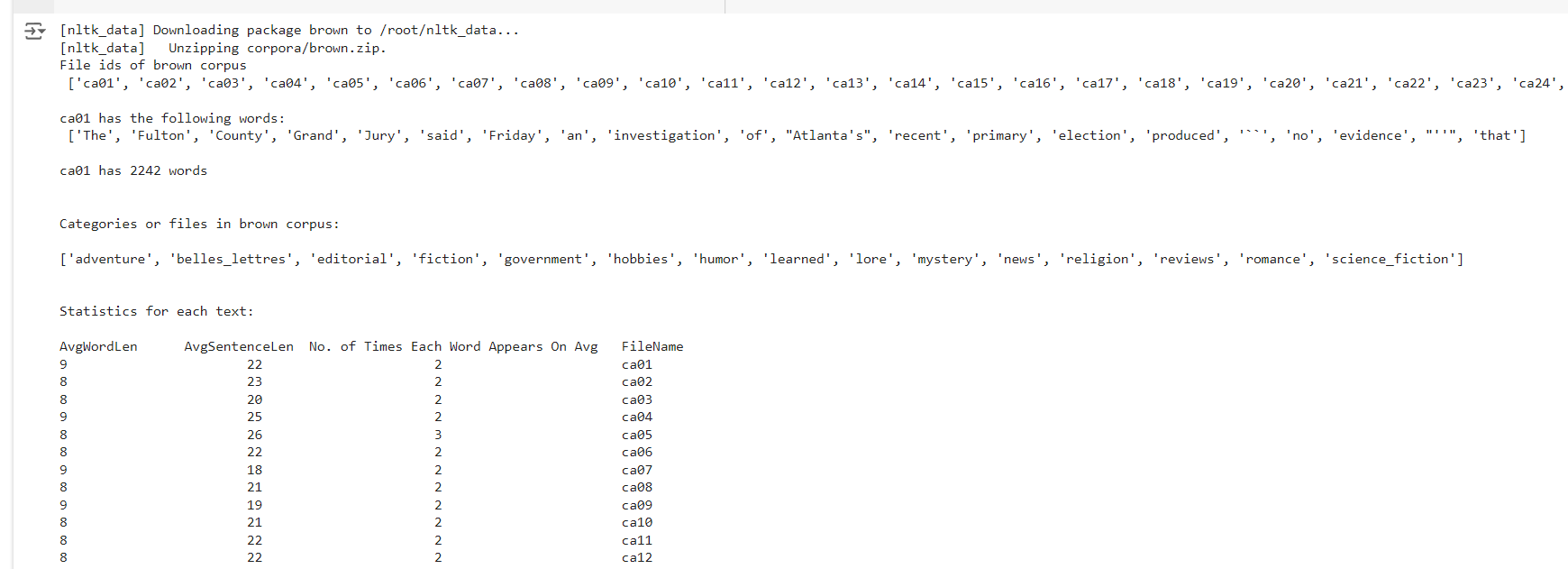
print(f"{int(num\_chars / num\_words)}\t\t\t"

f"{int(num\_words / num\_sents)}\t\t\t"

f"{int(num\_words / num\_vocab)}\t\t\t"

f"{fileid}")

**Output:**



**AIM b) Create and use your own corpora (plaintext, categorical).**

**Code:**

import os

import nltk

nltk.download('punkt')

from nltk.corpus import PlaintextCorpusReader

# Set the path to your corpus

corpus\_root = '/content/uni' #path of files

filelist = PlaintextCorpusReader(corpus\_root, '.\*')

# Display file list

print('\nFile list:\n')

print(filelist.fileids())

print(filelist.root)

# Display other information about each text by looping over all the values of fileid

# and then computing statistics for each text.

print('\n\nStatistics for each text:\n')

print('AvgWordLen\tAvgSentenceLen\tNo. of Times Each Word Appears On Avg\tFileName')

for fileid in filelist.fileids():

num\_chars = len(filelist.raw(fileid))

num\_words = len(filelist.words(fileid))

num\_sents = len(filelist.sents(fileid))

num\_vocab = len(set(w.lower() for w in filelist.words(fileid)))

print(f"{int(num\_chars / num\_words)}\t\t\t"

f"{int(num\_words / num\_sents)}\t\t\t"

f"{int(num\_words / num\_vocab)}\t\t"

f"{fileid}")

**Output:**

A screenshot of a computer program

Description automatically generated

**AIM c) Study Conditional frequency distributions**

**Code:**

# Process a sequence of pairs

text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]

import nltk

from nltk.corpus import brown

nltk.download('inaugural')

nltk.download('udhr')

fd = nltk.ConditionalFreqDist(

(genre, word)

for genre in brown.categories()

for word in brown.words(categories=genre)

)

genre\_word = [

(genre, word)

for genre in ['news', 'romance']

for word in brown.words(categories=genre)

]

print(len(genre\_word))

print(genre\_word[:4])

print(genre\_word[-4:])

cfd = nltk.ConditionalFreqDist(genre\_word)

print(cfd)

print(cfd.conditions())

print(cfd['news'])

print(cfd['romance'])

print(list(cfd['romance']))

from nltk.corpus import inaugural

cfd = nltk.ConditionalFreqDist(

(target, fileid[:4])

for fileid in inaugural.fileids()

for w in inaugural.words(fileid)

for target in ['america', 'citizen']

if w.lower().startswith(target)

)

from nltk.corpus import udhr

languages = [

'Chickasaw', 'English', 'German\_Deutsch',

'Greenlandic\_Inuktikut', 'Hungarian\_Magyar', 'Ibibio\_Efik'

]

cfd = nltk.ConditionalFreqDist(

(lang, len(word))

for lang in languages

for word in udhr.words(lang + '-Latin1')

)

cfd.tabulate(conditions=['English', 'German\_Deutsch'], samples=range(10), cumulative=True)

**Output:**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

**AIM d) Study of tagged corpora with methods like tagged\_sents, tagged\_words.**

**Code:**

import nltk

from nltk import tokenize

nltk.download('punkt')

nltk.download('words')

para = "Hello! My name is Ninad Karlekar. Today you'll be learning NLTK."

sents = tokenize.sent\_tokenize(para)

print("\nSentence tokenization\n===================\n", sents)

# Word tokenization

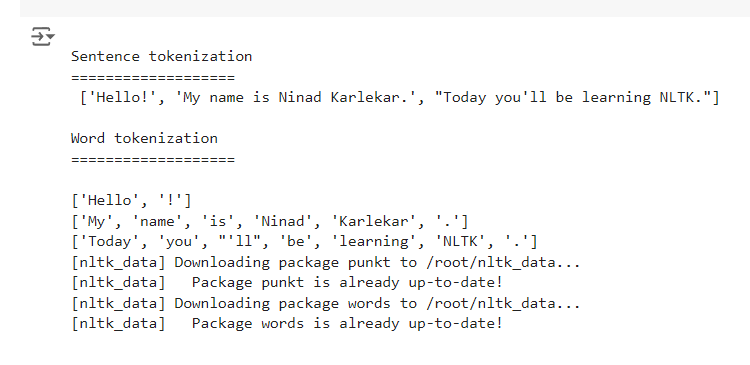
print("\nWord tokenization\n===================\n")

for index in range(len(sents)):

words = tokenize.word\_tokenize(sents[index])

print(words)

**Output:**



**AIM e) Write a program to find the most frequent noun tags.**

**Code:**

import nltk

from collections import defaultdict

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

text = nltk.word\_tokenize("Ninad likes to play football. Ninad does not like to play cricket.")

tagged = nltk.pos\_tag(text)

print(tagged)

addNounWords = []

count = 0

for words in tagged:

val = tagged[count][1]

if val in ('NN', 'NNS', 'NNPS', 'NNP'):

addNounWords.append(tagged[count][0])

count += 1

print(addNounWords)

temp = defaultdict(int)

# Memoizing count

for sub in addNounWords:

for wrd in sub.split():

temp[wrd] += 1

# Getting max frequency

res = max(temp, key=temp.get)

# Printing result

print("Word with maximum frequency : " + str(res))

**Output:**

A white background with colorful text

Description automatically generated

**AIM f) Map Words to Properties Using Python Dictionaries**

**Code:**

# Creating and printing a dictionary by mapping word with its properties

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(thisdict)

print(thisdict["brand"])

print(len(thisdict))

print(type(thisdict))

**Output:**

A white background with black text

Description automatically generated

**AIM g)** **Find different words from a given plain text without any space by comparing this text with a given corpus of words. Also find the score of words.**

**Code:**

from \_\_future\_\_ import with\_statement

import re

words = [] # corpus file words

testword = [] # test words

ans = [] # words matches with corpus

print("MENU")

print("-----------")

print(" 1. Hash tag segmentation")

print(" 2. URL segmentation")

print("Enter the input choice for performing word segmentation:")

choice = int(input())

if choice == 1:

text = "#whatismyname" # hash tag test data to segment

print("Input with HashTag:", text)

pattern = re.compile("[^\w']")

a = pattern.sub('', text)

elif choice == 2:

text = "www.whatismyname.com" # URL test data to segment

print("Input with URL:", text)

a = re.split('\s|(?<!\d)[,.](?!\d)', text)

splitwords = ["www", "com", "in"] # remove the words which is containing in the list

a = "".join([each for each in a if each not in splitwords])

else:

print("Wrong choice...try again")

exit()

print(a)

for each in a:

testword.append(each) # test word

test\_lenth = len(testword) # length of the test data

# Reading the corpus

with open('words.txt', 'r') as f:

lines = f.readlines()

words = [e.strip() for e in lines]

def Seg(a, lenth):

ans = []

for k in range(0, lenth + 1): # this loop checks char by char in the corpus

if a[0:k] in words:

print(a[0:k], "- appears in the corpus")

ans.append(a[0:k])

break

if ans != []:

g = max(ans, key=len)

return g

return ""

test\_tot\_itr = 0 # each iteration value

answer = [] # Store each word that contains the corpus

Score = 0 # initial value for score

N = 37 # total number of corpus

M = 0

C = 0

while test\_tot\_itr < test\_lenth:

ans\_words = Seg(a, test\_lenth)

if ans\_words != "":

test\_itr = len(ans\_words)

answer.append(ans\_words)

a = a[test\_itr:test\_lenth]

test\_tot\_itr += test\_itr

Aft\_Seg = " ".join([each for each in answer])

# print segmented words in the list

print("Output")

print("---------")

print(Aft\_Seg) # print after segmentation the input

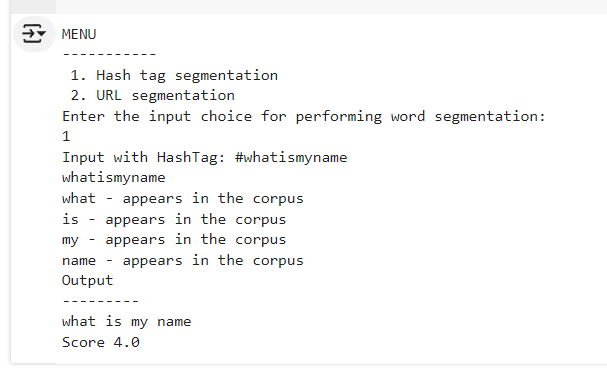
# Calculating Score

C = len(answer)

score = C \* N / N # Calculate the score

print("Score", score)

**Output:**



A screenshot of a computer code

Description automatically generated

**Practical No: 3**

**Aim A): Study of Wordnet Dictionary with methods as synsets, definitions,**

Code:

# NLP 3A. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms

import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')

synsets = wordnet.synsets("phone")

print("\*\*Word:\*\* phone")

print("  \* Synsets:")

for synset in synsets:

    word = synset.lemmas()[0].name()

    print(f"      - Word: {word}")

    print(f"        - Definition: {synset.definition()}")

    print(f"          - Examples: {synset.examples()}")

print("-"\*40)

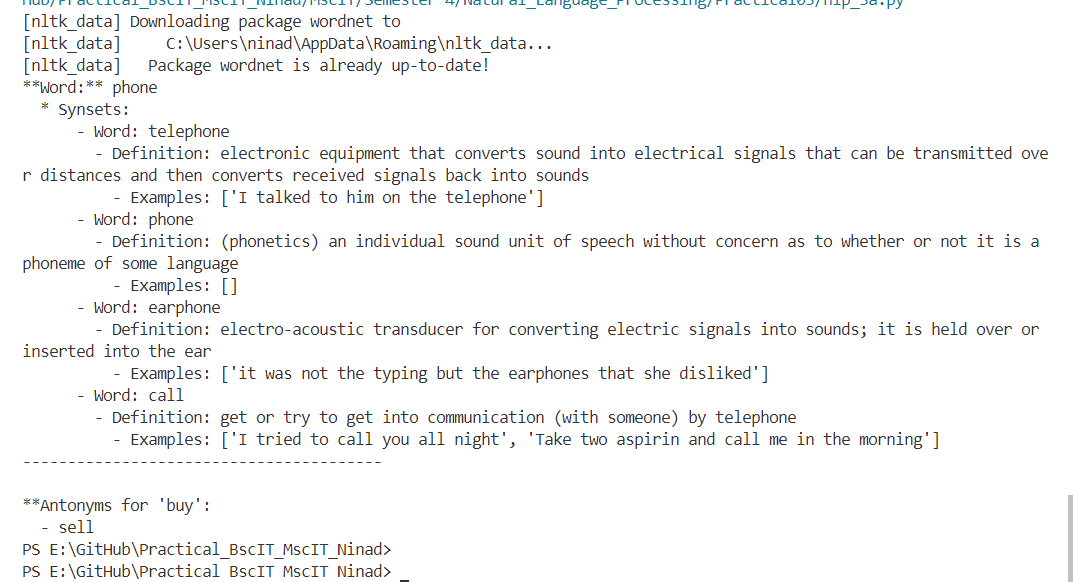
antonyms = wordnet.lemma('buy.v.01.buy').antonyms()

print("\n\*\*Antonyms for 'buy':")

for antonym in antonyms:

    print(f"  - {antonym.name()}")

OUTPUT:



**Aim B): Study lemmas, hyponyms, hypernyms.**

Code:  
  
# NLP 3B Study lemmas, hyponyms, hypernyms.

import nltk

from nltk.corpus import wordnet

nltk.download('wordnet')

print("\n\*\*Lemmas\*\*")

synsets = wordnet.synsets("computer")

print(" \* Synsets and Lemmas:")

for synset in synsets:

lemma\_names = [lemma.name() for lemma in synset.lemmas()]

print(f" - Synset: {synset} --> Lemmas: {lemma\_names}")

print("\n\*\*Hyponyms\*\*")

computer\_synset = wordnet.synset("computer.n.01")

hyponyms = computer\_synset.hyponyms()

print(" \* Hyponyms of 'computer.n.01':")

for synset in hyponyms:

lemma\_names = [lemma.name() for lemma in synset.lemmas()]

print(f" - Synset: {synset} --> Lemmas: {lemma\_names}")

print("\n\*\*Hypernyms\*\*")

vehicle\_synset = wordnet.synset("vehicle.n.01")

car\_synset = wordnet.synset("car.n.01")

lowest\_common\_hypernym = car\_synset.lowest\_common\_hypernyms(vehicle\_synset)

print(f" \* Lowest common hypernym of 'vehicle' and 'car': {lowest\_common\_hypernym[0]}")

**OUTPUT:**A screenshot of a computer code

Description automatically generated

**Aim C): Write a program using python to find synonym and antonym of word "active" using Wordnet.**

**Code:**

# NLP 3C. Write a program using python to find synonym and antonym of word "active" using Wordnet.

import nltk

from nltk.corpus import wordnet

nltk.download('omw-1.4')

def get\_synonyms\_antonyms(word):

synonyms = []

antonyms = []

for syn in wordnet.synsets(word):

for lemma in syn.lemmas():

synonyms.append(lemma.name())

if lemma.antonyms():

antonyms.append(lemma.antonyms()[0].name())

return set(synonyms), set(antonyms)

def main():

word = input("Enter the word:-")

synonyms, antonyms = get\_synonyms\_antonyms(word)

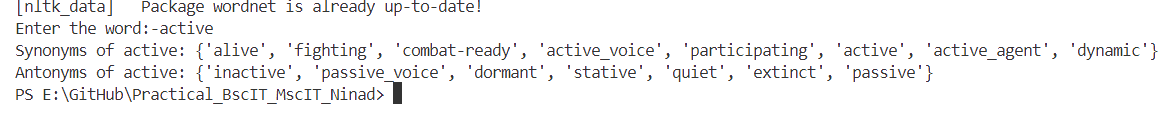
print("Synonyms of", word + ":", synonyms)

print("Antonyms of", word + ":", antonyms)

if \_\_name\_\_ == "\_\_main\_\_":

nltk.download('wordnet')

main()

**Output:**

**Aim D): Compare two nouns.**

**Code:**

import nltk

from nltk.corpus import wordnet

syn1 = wordnet.synsets('football')

syn2 = wordnet.synsets('soccer')

# A word may have multiple synsets, so need to compare each synset of word1 with synset of word2

for s1 in syn1:

for s2 in syn2:

print("Path similarity of: ")

print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')

print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')

print(" is", s1.path\_similarity(s2))

print()

print(f"The Path Similarity between '{noun1}' and '{noun2}' is: {path\_similarity\_score}")

**Output:**

**A screenshot of a computer

Description automatically generated**

**Aim 3E\_1): Using nltk, add or remove stop words in NLTK's Default stop word list**

**Code:**

# NLP 3E\_a: Using nltk, add or remove stop words in NLTK's Default stop word list

# Import necessary libraries

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Download stopwords (if not already downloaded)

nltk.download('punkt')

nltk.download('stopwords')

print("-"\*40)

# Sample text

text = "Ninad likes to play Chess, however he is not too good with the football."

print("Given Text:- ",text)

# Remove stop words from text

text\_tokens = word\_tokenize(text)

stop\_words = stopwords.words('english')

tokens\_without\_sw = [word for word in text\_tokens if word not in stop\_words]

print("Tokens without stop words:", tokens\_without\_sw)

# Add custom stop word ('not')

custom\_stop\_words = stop\_words + ['not']

tokens\_without\_sw = [word for word in text\_tokens if word not in custom\_stop\_words]

print("\nTokens without 'not' (custom stop word):", tokens\_without\_sw)

**Output:**

A computer screen with text

Description automatically generated

**Aim 3E\_2): Using Gensim, add or remove stop words in Default Gensim stop words List.**

**Code:**

# pip install gensim nltk

import gensim

from gensim.parsing.preprocessing import remove\_stopwords, STOPWORDS

import nltk

from nltk.tokenize import word\_tokenize

# Ensure necessary NLTK data is downloaded

nltk.download('punkt')

nltk.download('stopwords')

text = "Harshad likes to play cricket, however he is not too fond of basketball."

filtered\_sentence = remove\_stopwords(text)

print(filtered\_sentence)

all\_stopwords = gensim.parsing.preprocessing.STOPWORDS

print(all\_stopwords)

# The following script adds 'likes' and 'play' to the list of stop words in Gensim:

all\_stopwords\_gensim= STOPWORDS.union(set(['likes', 'play']))

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords\_gensim]

print(tokens\_without\_sw)

# Output:

#['Harshad', 'cricket', ',', 'fond', 'basketball', '.']

# The following script removes the word "not" from the set of stop words in Gensim:

sw\_list = {"not"}

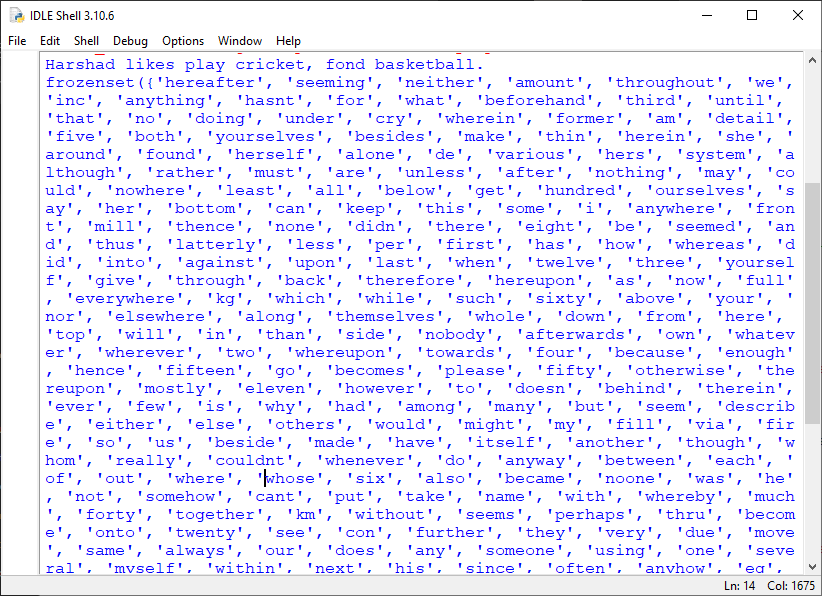
all\_stopwords\_gensim = STOPWORDS.difference(sw\_list)

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in all\_stopwords\_gensim]

print(tokens\_without\_sw)

**Output:**



**Aim 3E\_3): Using SpaCy, add or remove Stop Words in Default SpaCy stop words List.**

**Code:**

# NLP 3E\_c: Using SpaCy, add or remove Stop Words in Default SpaCy stop words List.

#python -m spacy download en\_core\_web\_sm

import spacy

import nltk

from nltk.tokenize import word\_tokenize

print("NLP 3E 3 Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List")

sp = spacy.load("en\_core\_web\_sm")

# Get default stop words from spaCy

all\_stopwords = sp.Defaults.stop\_words

text = "Ninad likes to play Chess, however he is not too good with the football."

text\_tokens = word\_tokenize(text)

all\_stopwords.add("play")

tokens\_without\_sw = [word for word in text\_tokens if word not in all\_stopwords]

print("Original sentence (tokens):", text\_tokens)

print("Stop word 'play' added:", tokens\_without\_sw)

all\_stopwords.remove("not")

tokens\_without\_sw = [word for word in text\_tokens if word not in all\_stopwords]

print("Original sentence (tokens):", text\_tokens)

print("Stop word 'not' removed:", tokens\_without\_sw)

**Output:**

A screen shot of a computer screen

Description automatically generated

**Practical No: 4**

**Text Tokenization**

**Aim 4A): Tokenization using Python’s split() function.**

**Code:**

# 4A. Tokenization using Python’s split() function

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Tokenizing the text using split()

tokens = text.split()

# Printing the tokens

print("="\*60)

print("4A. Tokenization using Python’s split() function")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4B): Tokenization using Regular Expression (RegEx).**

**Code:**

# 4b. Tokenization using Regular Expressions (RegEx)

import re

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Define the regex pattern for tokenization (splitting by whitespace)

pattern = r'\s+'

# Tokenizing the text using re.split()

tokens = re.split(pattern, text)

# Printing the tokens

print("="\*60)

print("4b. Tokenization using Regular Expressions (RegEx)")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A white background with black text

Description automatically generated

**Aim 4C): Tokenization using NLTK.**

**Code:**

#4c. Tokenization using NLTK

import nltk

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

# Sample text to tokenize

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Tokenizing the text using NLTK's word\_tokenize()

tokens = word\_tokenize(text)

# Printing the tokens

print("="\*60)

print("4c. Tokenization using NLTK")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4D): Tokenization using spaCy library.**

**Code:**

#4d. Tokenization using the spaCy library

import spacy

# Load the English language model

nlp = spacy.blank("en")

# Text to be tokenized

text = "Hello ! My name is Ninad Karlekar I live in mumbai"

# Process the text with SpaCy

doc = nlp(text)

# Extract tokens

tokens = [token.text for token in doc]

# Print tokens

print("="\*60)

print("4d. Tokenization using the spaCy library")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

**Output:**

A close up of a computer screen

Description automatically generated

**Aim 4E): Tokenization using Keras.**

**Code:**

#4e. Tokenization using Keras [COLAB]

import keras

from tensorflow.keras.preprocessing.text import text\_to\_word\_sequence #works on colab

# Create a string input

str = "Hello ! My name is Ninad Karlekar I live in mumbai"

# tokenizing the text

tokens = text\_to\_word\_sequence(str)

print("="\*60)

print("4e. Tokenization using Keras")

print("-"\*10)

print("Tokens:", tokens)

print("="\*60)

####

# to run on local IDE(jupyter) use tenserflow version 2.13.0

# pip uninstall tensorflow     ## to uninstall latest version if installed

# pip install tensorflow==2.13.0

####

**Output:**

A screenshot of a computer code

Description automatically generated

**Aim 4F): Tokenization using Gensim.**

**Code:**

# 4f. Tokenization using Gensim

#pip install gensim

from gensim.utils import tokenize

# Create a string input

str = "Hello ! My name is Ninad Karlekar I live in mumbai"

# tokenizing the text

# Tokenizing the text

tokenized\_words = list(tokenize(str))

# Printing each tokenized word separately

print("="\*60)

print("4f. Tokenization using Gensim")

print("-"\*10)

print("Tokens:", tokenized\_words)

print("="\*60)

**Output:**

A computer code with black text

Description automatically generated

**Practical No: 5**

**Aim 5A): Word tokenization in Hindi**

**Code:**

import sys

from indicnlp import common

# The path to the local git repo for Indic NLP library

INDIC\_NLP\_LIB\_HOME=r"indic\_nlp\_library"

# The path to the local git repo for Indic NLP Resources

INDIC\_NLP\_RESOURCES=r"indic\_nlp\_resources"

# Add library to Python path

sys.path.append(r'{}\src'.format(INDIC\_NLP\_LIB\_HOME))

# Set environment variable for resources folder

common.set\_resources\_path(INDIC\_NLP\_RESOURCES)

from indicnlp.tokenize import indic\_tokenize

indic\_string='सुनो, कुछ आवाज़ आ रही है। फोन?'

print('Input String: {}'.format(indic\_string))

print('Tokens: ')

for t in indic\_tokenize.trivial\_tokenize(indic\_string):

**Output:**

**A screenshot of a computer program

Description automatically generated**

**Aim 5B): Generate similar sentences from a given Hindi text input.**

**Code:**

synonyms = {

"खुश": ["प्रसन्न", "आनंदित", "खुशी"],

"बहुत": ["अधिक", "बहुत ज्यादा", "काफी"]

}

# Function to generate similar sentences by replacing some words with synonyms

def generate\_similar\_sentences(input\_sentence, num\_sentences=5):

similar\_sentences = []

# Replace some words with synonyms

for word, word\_synonyms in synonyms.items():

for synonym in word\_synonyms:

modified\_sentence = input\_sentence.replace(word, synonym)

similar\_sentences.append(modified\_sentence)

return similar\_sentences[:num\_sentences]

input\_sentence = "मैं आज बहुत खुश हूँ।"

similar\_sentences = generate\_similar\_sentences(input\_sentence)

print("Original sentence:", input\_sentence)

print("Similar sentences:")

for sentence in similar\_sentences:

print("-", sentence)

**Output:**

A screenshot of a computer

Description automatically generated

**Aim 5C): Identify the Indian language from the given text..**

**Code:**

import nltk

import langid

# Download necessary NLTK data

nltk.download('punkt')

def identify\_language(text):

lang, \_ = langid.classify(text)

return lang

# Identify the Indian Language from the given text

language = identify\_language("नमस्ते, आप कैसे हैं?")

print("Identified language:", language)

**Output:**

A screenshot of a computer error

Description automatically generated

**Practical No: 6**

**Aim 6A): Part of speech Tagging and chunking of user defined text.**

**Code:**

# 6. Illustrate part of speech tagging.

## a) sentence tokenization, word tokenization, Part of speech Tagging and chunking of user defined text.

import nltk

from nltk import tokenize

from nltk import tag

from nltk import chunk

nltk.download('punkt')

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

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

# Paragraph to be tokenized

para = "Hello! My name is Ninad Karlekar. Today you'll be learning NLTK."

# Sentence tokenization

sents = tokenize.sent\_tokenize(para)

print("\nsentence tokenization\n===================\n", sents)

# Word tokenization

print("\nword tokenization\n===================\n")

for index in range(len(sents)):

    words = tokenize.word\_tokenize(sents[index])

    print(words)

# POS Tagging

tagged\_words = []

for index in range(len(sents)):

    tagged\_words.append(tag.pos\_tag(words))

print("\nPOS Tagging\n===========\n", tagged\_words)

# Chunking

tree = []

for index in range(len(sents)):

    tree.append(chunk.ne\_chunk(tagged\_words[index]))

print("\nchunking\n========\n", tree)

**Output:**

A black text on a white background

Description automatically generated A black text on a white background

Description automatically generated A close-up of a text

Description automatically generated A text on a white background

Description automatically generated

**Aim 6B): Named Entity recognition of user defined text.**

**Code:**

# !pip install -U spacy

# !python -m spacy download en\_core\_web\_sm

import spacy

nlp = spacy.load("en\_core\_web\_sm")

# Process whole documents

text = (

    "When Sebastian Thrun started working on self-driving cars at "

    "Google in 2007, few people outside of the company took him "

    "seriously. “I can tell you very senior CEOs of major American "

    "car companies would shake my hand and turn away because I wasn’t "

    "worth talking to,” said Thrun, in an interview with Recode earlier "

    "this week."

)

doc = nlp(text)

# Analyse syntax

print("Noun phrases:", [chunk.text for chunk in doc.noun\_chunks])

print("Verbs:", [token.lemma\_ for token in doc if token.pos\_ == "VERB"])

**Output:**

A close-up of a computer code

Description automatically generated

**Aim 6C): Named Entity recognition with diagram using NLTK corpus - treebank**

**Code:**

# 6C: Named Entity recognition with diagram using NLTK corpus - treebank

import nltk

nltk.download('treebank')

from nltk.corpus import treebank\_chunk

treebank\_chunk.tagged\_sents()[0]

treebank\_chunk.chunked\_sents()[0]

treebank\_chunk.chunked\_sents()[0].draw()

# Note: It runs on Python IDLE, VScode

**Output:**

A diagram of a person's beard

Description automatically generated

**Practical No: 7**

**Aim 7A): Define grammar using nltk. Analyse a sentence using the same.**

**Code:**

import nltk

from nltk import tokenize

grammar1 = nltk.CFG.fromstring("""

S -> VP

VP -> VP NP

NP -> Det NP

Det -> 'that'

NP -> 'flight'

VP -> 'Book'

""")

sentence = "Book that flight"

all\_tokens = tokenize.word\_tokenize(sentence)

print(all\_tokens)

parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens):

print(tree)

tree.draw()

**Output:**

A screenshot of a computer

Description automatically generated

1. **Aim 7B): Accept the input string with Regular expression of FA: 101+**

**Code:**

def FA(s):

# if the length is less than 3 then it can't be accepted, Therefore end the process.

if len(s) < 3:

return "Rejected"

if s[0] == '1':

if s[1] == '0':

if s[2] == '1':

for i in range(3, len(s)):

if s[i] != '1':

return "Rejected"

return "Accepted" #if all 4 nested if true

return "Rejected" # else of 3rd if

return "Rejected" # else of 2nd if

return "Rejected" # else of 1st if

inputs = ['1', '10101', '101', '10111', '01010', '100', '', '10111101', '1011111']

for i in inputs:

print(FA(i))

**Output:**

**A screenshot of a computer

Description automatically generated**

1. **Aim 7C): Accept the input string with Regular expression of FA: (a+b)\*bba**

**Code:**

def FA(s):

size=0

#scan complete string and make sure that it contains only 'a' & 'b'

for i in s:

if i=='a' or i=='b':

size+=1

else:

return "Rejected"

#After checking that it contains only 'a' & 'b'

#check it's length it should be 3 atleast

if size>=3:

#check the last 3 elements

if s[size-3]=='b':

if s[size-2]=='b':

if s[size-1]=='a':

return "Accepted" # if all 4 if true

return "Rejected" # else of 4th if

return "Rejected" # else of 3rd if

return "Rejected" # else of 2nd if

return "Rejected" # else of 1st if

inputs=['bba', 'ababbba', 'abba','abb', 'baba','bbb','']

for i in inputs:

print(FA(i))

**Output:**

**A screenshot of a computer

Description automatically generated**

1. **Aim 7D): Implementation of Deductive Chart Parsing using context free grammar and a given sentence.**

**Code:**

#!pip install nltk

#nltk.download('punkt')

import nltk

from nltk import tokenize

grammar1 = nltk.CFG.fromstring("""

S -> NP VP

PP -> P NP

NP -> Det N | Det N PP | 'I'

VP -> V NP | VP PP

Det -> 'a' | 'my'

N -> 'bird' | 'balcony'

V -> 'saw'

P -> 'in'

""")

sentence = "I saw a bird in my balcony"

for index in range(len(sentence)):

all\_tokens = tokenize.word\_tokenize(sentence)

print(all\_tokens)

# all\_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']

parser = nltk.ChartParser(grammar1)

for tree in parser.parse(all\_tokens):

print(tree)

tree.pretty\_print()

**Output:**

**A screenshot of a computer program

Description automatically generated**

**Practical No: 8**

**Aim 8A): Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer.**

**Code:**

# PorterStemmer

import nltk

from nltk.stem import PorterStemmer

word\_stemmer = PorterStemmer()

print("Output of PorterStemmer:-")

print(word\_stemmer.stem('Ninad is running'))

print("\*"\*50)

# LancasterStemmer

import nltk

from nltk.stem import LancasterStemmer

Lanc\_stemmer = LancasterStemmer()

print("Output of LancasterStemmer:-")

print(Lanc\_stemmer.stem('jumping'))

print("\*"\*50)

#RegexpStemmer

import nltk

from nltk.stem import RegexpStemmer

Reg\_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)

print("Output of RegexpStemmer:-")

print(Reg\_stemmer.stem('writing'))

print("\*"\*50)

# SnowballStemmer

import nltk

from nltk.stem import SnowballStemmer

english\_stemmer = SnowballStemmer('english')

print("Output of SnowballStemmer:-")

print(english\_stemmer.stem ('writing'))

print("\*"\*50)

**Output:**

A screenshot of a computer program

Description automatically generated

**Aim 8B): Study WordNet Lemmatizer**

**Code:**

# WordNetLemmatizer

print("Output of WordNetLemmatizer:-")

import nltk

nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("word :\tlemma")

print("rocks :", lemmatizer.lemmatize("books"))

print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"

print("worse :", lemmatizer.lemmatize("worse", pos ="a"))

print("\*"\*50)

**Output:**

A screenshot of a computer program

Description automatically generated

**Practical No: 9**

**Aim 9): Implement Naive Bayes classifier.**

**Code:**

# 9. Implement Naive Bayes classifier

# pip install pandas

# pip install sklearn

import pandas as pd

import numpy as np

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

nltk.download('stopwords')

sms\_data = pd.read\_csv("MscIT\\Semester 4\\Natural\_Language\_Processing\\Practical09\\spam.csv", encoding='latin-1')

stemming = PorterStemmer()

corpus = []

for i in range(len(sms\_data)):

s1 = re.sub('[^a-zA-Z]', ' ', sms\_data['v2'][i])

s1 = s1.lower()

s1 = s1.split()

s1 = [stemming.stem(word) for word in s1 if word not in set(stopwords.words('english'))]

s1 = ' '.join(s1)

corpus.append(s1)

countvectorizer = CountVectorizer()

x = countvectorizer.fit\_transform(corpus).toarray()

print(x)

y = sms\_data['v1'].values

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, stratify=y, random\_state=2)

multinomialnb = MultinomialNB()

multinomialnb.fit(x\_train, y\_train)

y\_pred = multinomialnb.predict(x\_test)

print(y\_pred)

print(classification\_report(y\_test, y\_pred))

print("accuracy\_score: ", accuracy\_score(y\_test, y\_pred))

**Output:**A screenshot of a computer

Description automatically generated

**Practical No: 10**

**Aim: Speech tagging**

**Source Code:**

1. **Speech Tagging using spaCy**

import spacy

nlp = spacy.load("en\_core\_web\_sm")

sp = spacy.load('en\_core\_web\_sm')

sen = sp(u"I like to play cricket. I hated it in my childhood though")

print(sen.text)

print(sen[7].pos\_)

print(sen[7].tag\_)

print(spacy.explain(sen[7].tag\_))

for word in sen:

print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.explain(word.tag\_)}')

sen = sp(u'Can you google it?')

word = sen[2]

print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.explain(word.tag\_)}')

sen = sp(u'Can you search it on google?')

word = sen[5]

print(f'{word.text:{12}} {word.pos\_:{10}} {word.tag\_:{8}} {spacy.explain(word.tag\_)}')

# Finding the Number of POS Tags

sen = sp(u"I like to play football. I hated it in my childhood though")

num\_pos = sen.count\_by(spacy.attrs.POS)

for k, v in sorted(num\_pos.items()):

print(f'{k}. {sen.vocab[k].text:{8}}: {v}')

# Visualizing Parts of Speech Tags

from spacy import displacy

sen = sp(u"I like to play football. I hated it in my childhood though")

displacy.serve(sen, style='dep', options={'distance': 120})

**Output:**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**Source Code:**

1. **Speech tagging using nktl**

import nltk

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

# Create our training and testing data:

train\_text = state\_union.raw("2005-GWBush.txt")

sample\_text = state\_union.raw("2006-GWBush.txt")

# Train the Punkt tokenizer:

custom\_sent\_tokenizer = PunktSentenceTokenizer(train\_text)

# Tokenize:

tokenized = custom\_sent\_tokenizer.tokenize(sample\_text)

def process\_content():

try:

for i in tokenized[:2]:

words = nltk.word\_tokenize(i)

tagged = nltk.pos\_tag(words)

print(tagged)

except Exception as e:

print(str(e))

process\_content()

process\_content()

**Output:**

A screenshot of a computer program

Description automatically generated

**Practical no: 10B**

**Aim: Study statistical parsing**

**Source Code:**

1. **Usage of Give and Gave in the Penn Treebank sample**

import nltk

def give(t):

return t.label() == 'VP' and len(t) > 2 and t[1].label() == 'NP' \

and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP') \

and ('give' in t[0].leaves() or 'gave' in t[0].leaves())

def sent(t):

return ' '.join(token for token in t.leaves() if token[0] not in '\*-0')

def print\_node(t, width):

output = "%s %s: %s / %s: %s" % \

(sent(t[0]), t[1].label(), sent(t[1]), t[2].label(), sent(t[2]))

if len(output) > width:

output = output[:width] + "..."

print(output)

for tree in nltk.corpus.treebank.parsed\_sents():

for t in tree.subtrees(give):

print\_node(t, 72)

**Output:**

A screenshot of a computer

Description automatically generated

**Source Code:**

1. **Probabilistic parser**

import nltk

from nltk import PCFG

grammar = PCFG.fromstring('''

NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]

NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4]

JJ -> "old" [0.4] | "young" [0.6]

CC -> "and" [0.9] | "or" [0.1]

''')

print(grammar)

viterbi\_parser = nltk.ViterbiParser(grammar)

token = "old men and women".split()

obj = viterbi\_parser.parse(token)

print("Output:")

for x in obj:

print(x)

**Output:**

A screenshot of a computer

Description automatically generated

**Practical no: 10C**

**Aim: Parse a sentence and draw a tree using malt parsing.**

**Source Code:**

1. Java should be installed.
2. [maltparser-1.7.2 zip](http://maltparser.org/dist/maltparser-1.7.2.zip) file should be copied in C:\Users\AppData\Local\Programs\Python\Python39 folder and should be extracted in the same folder.
3. [engmalt.linear-1.7.mco](https://www.maltparser.org/mco/english_parser/engmalt.linear-1.7.mco) & [engmalt.poly-1.7.mco](https://www.maltparser.org/mco/english_parser/engmalt.poly-1.7.mco) file should be copied to C:\Users\ AppData\Local\Programs\Python\Python39 folder

from nltk.parse import malt

mp = malt.MaltParser('maltparser-1.7.2', 'engmalt.linear-1.7.mco')#file

t = mp.parse\_one('I saw a bird from my window.'.split()).tree()

print(t)

t.draw()

**Output:**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Practical no: 11A**

**Aim: Multiword Expressions in NLP**

**Source Code:**

# Multiword Expressions in NLP

from nltk.tokenize import MWETokenizer

from nltk import sent\_tokenize, word\_tokenize

s = '''Good cake cost Rs.1500\kg in Hong Kong. Please buy me one of them.\n\nThanks.'''

mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator='-')

for sent in sent\_tokenize(s):

 print(mwe.tokenize(word\_tokenize(sent)))

**Output:**

A screenshot of a computer

Description automatically generated

**Practical no:11B**

**Aim: Normalized Web Distance and Word Similarity**

**Source Code:**

import numpy as np

import re

import textdistance # pip install textdistance

# we will need scikit-learn>=0.21

import sklearn # pip install sklearn

from sklearn.cluster import AgglomerativeClustering

texts = ['Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance', 'Reliance downtown', 'Reliance market','Mumbai', 'Mumbai Hyper', 'Mumbai dxb', 'mumbai airport','k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading', 'KM.Trading']

def normalize(text):

""" Keep only lower-cased text and numbers"""

return re.sub('[^a-z0-9]+', ' ', text.lower())

def group\_texts(texts, threshold=0.4):

""" Replace each text with the representative of its cluster"""

normalized\_texts = np.array([normalize(text) for text in texts])

distances = 1 - np.array([

[textdistance.jaro\_winkler(one, another) for one in normalized\_texts]

for another in normalized\_texts])

clustering = AgglomerativeClustering(

distance\_threshold=threshold,

# this parameter needs to be tuned carefully

affinity="precomputed", linkage="complete",n\_clusters=None

).fit(distances)

centers = dict()

for cluster\_id in set(clustering.labels\_):

index = clustering.labels\_ == cluster\_id

centrality = distances[:, index][index].sum(axis=1)

centers[cluster\_id] = normalized\_texts[index][centrality.argmin()]

return [centers[i] for i in clustering.labels\_]

print(group\_texts(texts))

**Output:**

A screenshot of a computer

Description automatically generated

**Practical no: 11C**

**Aim: Word Sense Disambiguation.**

**Source Code:**

#Word Sense Disambiguation

from nltk.corpus import wordnet as wn

def get\_first\_sense(word, pos=None):

if pos:

synsets = wn.synsets(word,pos)

else:

synsets = wn.synsets(word)

return synsets[0]

best\_synset = get\_first\_sense('bank')

print ('%s: %s' % (best\_synset.name, best\_synset.definition))

best\_synset = get\_first\_sense('set','n')

print ('%s: %s' % (best\_synset.name, best\_synset.definition))

best\_synset = get\_first\_sense('set','v')

print ('%s: %s' % (best\_synset.name, best\_synset.definition))

**Output:**

A screenshot of a computer

Description automatically generated