**NLP Lab Manual**

**Practical No. 1:**

1. **Install NLTK**

**Python 3.9.2 Installation on Windows**

Step 1) **Go to link**<https://www.python.org/downloads/>, **and select the latest version for windows.**

A screenshot of a computer

Description automatically generated with low confidence

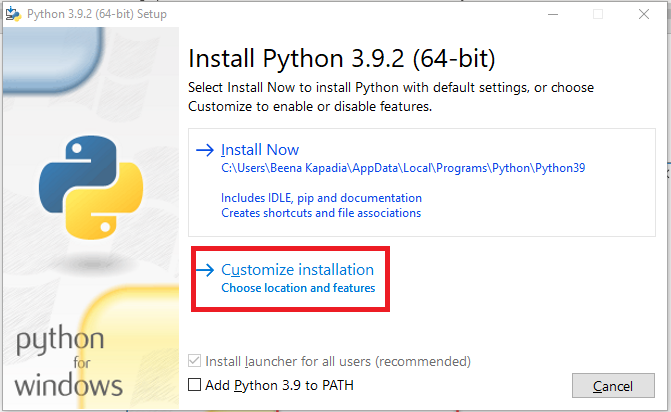
**Note**: If you don't want to download the latest version, you can visit the download tab and see all releases.

Graphical user interface, text, application

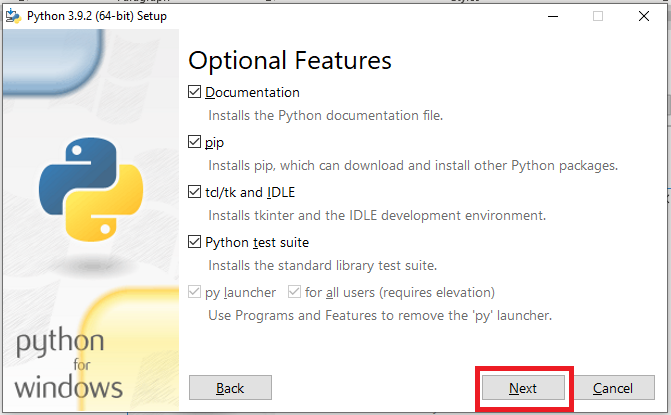
Description automatically generated

**Step 2)**Click on the Windows installer (64 bit)

**Step 3)**Select Customize Installation

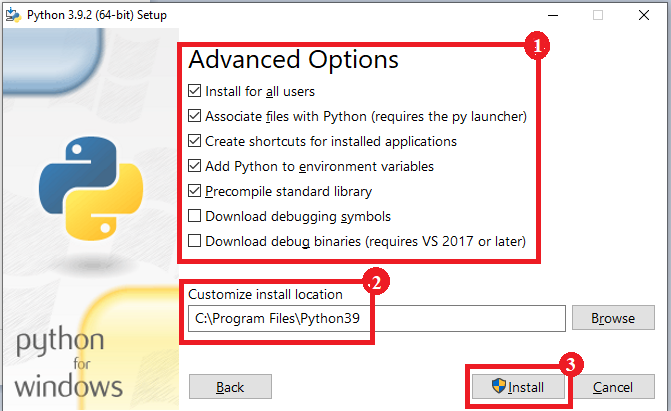


**Step 4)**Click NEXT



**Step 5)**In next screen

1. Select the advanced options
2. Give a Custom install location. Keep the default folder as c:\Program files\Python39
3. Click Install



**Step 6)**Click Close button once install is done.

**Step 7) open command prompt window and run the following commands:**

**C:\Users\Beena Kapadia>pip install --upgrade pip**

**C:\Users\Beena Kapadia>pip install --user -U nltk**

**C:\Users\Beena Kapadia>>pip install --user -U numpy**

**C:\Users\Beena Kapadia>python**

**>>> import nltk**

**>>>**

Text

Description automatically generated

(Browse https://www.nltk.org/install.html for more details)

**b) Convert the given text to speech.**

**Source code:**

# text to speech

# for google colab put ! at the start of pip.

# pip install gtts

# pip install playsound

from playsound import playsound

# import required for textto speech conversion

from gtts import gTTS

mytext = "Welcome to Natural Language programming"

language = "en"

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

myobj.save("myfile.mp3")

playsound("myfile.mp3")

**Output:**

welcomeNLP.mp3 audio file is getting created and it plays the file with playsound() method, while running the program.

**c) Convert audio file Speech to Text.**

**Source code:**

Note: required to store the input file "male.wav" in the current folder before running the program.

#pip3 install SpeechRecognitionpydub

# for google colab put ! at the start of pip.

import speech\_recognition as sr

filename = "male.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)

Input:

male.wav (any wav file)

**Output:**

Text

Description automatically generated

**Practical No. 2:**

**a. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like filelds, raw, words, sents, categories.**

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

**c. Study Conditional frequency distributions**

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

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

**f. Map Words to Properties Using Python Dictionaries**

**g. Study DefaultTagger, Regular expression tagger, UnigramTagger**

**h. 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.**

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

**source code:**

'''NLTK includes a small selection of texts from the Project brown electronic text archive, which contains some 25,000 free electronic books, hosted at http://www.brown.org/. We begin by getting the Python interpreter to load the NLTK package, then ask to see nltk.corpus.brown.fileids(), the file identifiers in this corpus:'''

import nltk

from nltk.corpus import brown

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

'''Let’s pick out the first of these texts — Emma by Jane Austen — and give it a short name, emma, then find out how many words it contains:'''

ca01 = brown.words('ca01')

# display first few words

print('\nca01 has following words:\n',ca01)

# total number of words in ca01

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

#categories or files

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

print (brown.categories())

'''display other information about each text, by looping over all the values of fileid corresponding to the brown file identifiers listed earlier and then computing statistics for each text.'''

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

print ('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\t\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 (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents),'\t\t\t', int(num\_words/num\_vocab),'\t\t\t', fileid)

**output:**

Text

Description automatically generated

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

**source code:**

'''NLTK includes a small selection of texts from the Project filelist electronic text archive, which contains some 25,000 free electronic books, hosted at http://www.filelist.org/. We begin by getting the Python interpreter to load the NLTK package, then ask to see nltk.corpus.filelist.fileids(), the file identifiers in this corpus:'''

**Plaintext Corpora:**

import nltk

from nltk.corpus import PlaintextCorpusReader

corpus\_root = 'D:/2020/NLP/Practical/uni'

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

print ('\n File list: \n')

print (filelist.fileids())

print (filelist.root)

'''display other information about each text, by looping over all the values of fileid corresponding to the filelist file identifiers listed earlier and then computing statistics for each text.'''

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

print ('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\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 (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents),'\t\t\t', int(num\_words/num\_vocab),'\t\t', fileid)

**output:**

Graphical user interface, text, application

Description automatically generated

**Categorial Corpara:**

from nltk.corpus.reader import CategorizedPlaintextCorpusReader

mycat = CategorizedPlaintextCorpusReader(

'C:\\MYCorpus\_cat', r'sample.\*\.txt', cat\_pattern = r'.\*?\_(one|two).\*')

print ("Categorize : ", mycat.categories())

print ("\nOne : ", mycat.fileids(categories =['one']))

print ("\nTwo : ", mycat.fileids(categories =['two']))

mycat.words(categories='one')

print ('Avg Word Len\tAvg Sentence Len\t No of Times Each Word Appears On Avg\t FileName')

for fileid in mycat.fileids():

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

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

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

num\_vocab = len(set([w.lower() for w in mycat.words(fileid)]))

print (int(num\_chars/num\_words),'\t\t\t', int(num\_words/num\_sents), '\t\t\t', int(num\_words/num\_vocab), '\t\t\t\t', fileid)

**c. Study Conditional frequency distributions**

**source 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

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:**

Graphical user interface, text

Description automatically generated

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

**Source code**:

#study of tagged corpora – tagged words

nltk.corpus.brown.tagged\_words()

from nltk.corpus import brown

brown\_news\_tagged = brown.tagged\_words(categories='news', tagset='universal')

tag\_fd = nltk.FreqDist(tag for (word, tag) in brown\_news\_tagged)

tag\_fd.keys()

#Tagged Sentences

brown\_tagged\_sents = brown.tagged\_sents(categories='news')

brown\_sents = brown.sents(categories='news')

print(brown\_sents)

print(brown\_tagged\_sents)

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

**Code:**

import nltk

from collections import defaultdict

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

tagged = nltk.pos\_tag(text)

print(tagged)

# checking if it is a noun or not

addNounWords = []

count=0

for words in tagged:

val = tagged[count][1]

if(val == 'NN' or val == 'NNS' or val == 'NNPS' or val == '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:

Text

Description automatically generated

**f. Map Words to Properties Using Python Dictionaries**

**code:**

#creating and printing a dictionay 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 picture containing text

Description automatically generated

**g. Study i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger**

**i) DefaultTagger**

**code:**

import nltk

from nltk.tag import DefaultTagger

exptagger = DefaultTagger('NN')

from nltk.corpus import treebank

testsentences = treebank.tagged\_sents() [1000:]

print(exptagger.evaluate (testsentences))

#Tagging a list of sentences

import nltk

from nltk.tag import DefaultTagger

exptagger = DefaultTagger('NN')

print(exptagger.tag\_sents([['Hi', ','], ['How', 'are', 'you', '?']]))

**output**

A picture containing text

Description automatically generated

**ii) Regular expression tagger,**

**code:**

from nltk.corpus import brown

from nltk.tag import RegexpTagger

test\_sent = brown.sents(categories='news')[0]

regexp\_tagger = RegexpTagger(

[(r'^-?[0-9]+(.[0-9]+)?$', 'CD'), # cardinal numbers

(r'(The|the|A|a|An|an)$', 'AT'), # articles

(r'.\*able$', 'JJ'), # adjectives

(r'.\*ness$', 'NN'), # nouns formed from adjectives

(r'.\*ly$', 'RB'), # adverbs

(r'.\*s$', 'NNS'), # plural nouns

(r'.\*ing$', 'VBG'), # gerunds

(r'.\*ed$', 'VBD'), # past tense verbs

(r'.\*', 'NN') # nouns (default)

])

print(regexp\_tagger)

print(regexp\_tagger.tag(test\_sent))

**output:**

Text, letter

Description automatically generated

**iii) UnigramTagger**

**code:**

# Loading Libraries

from nltk.tag import UnigramTagger

from nltk.corpus import treebank

# Training using first 10 tagged sentences of the treebank corpus as data.

# Using data

train\_sents = treebank.tagged\_sents()[:10]

# Initializing

tagger = UnigramTagger(train\_sents)

# Lets see the first sentence

# (of the treebank corpus) as list

print(treebank.sents()[0])

print('\n',tagger.tag(treebank.sents()[0]))

#Finding the tagged results after training.

tagger.tag(treebank.sents()[0])

#Overriding the context model

tagger = UnigramTagger(model ={'Pierre': 'NN'})

print('\n',tagger.tag(treebank.sents()[0]))

**output:**

Text, letter

Description automatically generated

**h. 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.**

**Question:**

Initialize the hash tag test data or URL test data and convert to plain text without any space.. Read a text file of different words and compare the plain text data with the words exist in that text file and find out different words available in that plain text. Also find out how many words could be found. (for example, text = "#whatismyname" or text = [www.whatismyname.com](http://www.whatismyname.com). Convert that to plain text without space as: whatismyname and read text file as words.txt. Now compare plain text with words given in a file and find the words form the plain text and the count of words which could be found)

**Source code:**

from \_\_future\_\_ import with\_statement #with statement for reading file

import re # Regular expression

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 containg in the list

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

else:

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

print(a)

for each in a:

testword.append(each) #test word

test\_lenth = len(testword) # lenth 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

test\_tot\_itr = 0 #each iteration value

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

Score = 0 # initial value for score

N = 37 # total no of corpus

M = 0

C = 0

while test\_tot\_itr<test\_lenth:

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

if ans\_words != 0:

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)

**Input:**

**Words.txt**

--------------

check

domain

big

rocks

name

cheap

being

human

current

rates

ought

to

go

down

apple

domains

honesty

hour

follow

back

social

media

30

seconds

earth

this

is

insane

it

time

what

is

my

name

let

us

go

**Output:**

Text

Description automatically generated

3. a. **Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms**

**Source code:**

'''WordNet provides synsets which is the collection of synonym words also called “lemmas”'''

import nltk

from nltk.corpus import wordnet

print(wordnet.synsets("computer"))

# definition and example of the word ‘computer’

print(wordnet.synset("computer.n.01").definition())

#examples

print("Examples:", wordnet.synset("computer.n.01").examples())

#get Antonyms

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

**output:**

Text

Description automatically generated

**b. Study lemmas, hyponyms, hypernyms.**

**Source code:**

import nltk

from nltk.corpus import wordnet

print(wordnet.synsets("computer"))

print(wordnet.synset("computer.n.01").lemma\_names())

#all lemmas for each synset.

for e in wordnet.synsets("computer"):

print(f'{e} --> {e.lemma\_names()}')

#print all lemmas for a given synset

print(wordnet.synset('computer.n.01').lemmas())

#get the synset corresponding to lemma

print(wordnet.lemma('computer.n.01.computing\_device').synset())

#Get the name of the lemma

print(wordnet.lemma('computer.n.01.computing\_device').name())

#Hyponyms give abstract concepts of the word that are much more specific

#the list of hyponyms words of the computer

syn = wordnet.synset('computer.n.01')

print(syn.hyponyms)

print([lemma.name() for synset in syn.hyponyms() for lemma in synset.lemmas()])

#the semantic similarity in WordNet

vehicle = wordnet.synset('vehicle.n.01')

car = wordnet.synset('car.n.01')

print(car.lowest\_common\_hypernyms(vehicle))

**Output:**

Text

Description automatically generated

**c. Write a program using python to find synonym and antonym of word "active" using Wordnet.**

**Source code:**

from nltk.corpus import wordnet

synonyms = []

antonyms = []

for syn in wordnet.synsets("active"):

for l in syn.lemmas():

synonyms.append(l.name())

if l.antonyms():

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

print(set(synonyms))

print(set(antonyms))

**d. Compare two nouns**

**source 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()

**output:**

Graphical user interface, text, application

Description automatically generated

**e. Handling stopword:**

**i) Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List**

**code:**

import nltk

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk.tokenize import word\_tokenize

text = "Yashesh likes to play football, however he is not too fond of tennis."

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()]

print(tokens\_without\_sw)

#add the word play to the NLTK stop word collection

all\_stopwords = stopwords.words('english')

all\_stopwords.append('play')

text\_tokens = word\_tokenize(text)

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

print(tokens\_without\_sw)

#remove ‘not’ from stop word collection

all\_stopwords.remove('not')

text\_tokens = word\_tokenize(text)

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

print(tokens\_without\_sw)

**output**

Text

Description automatically generated

**ii) Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List**

**code:**

#pip install gensim

import gensim

from gensim.parsing.preprocessing import remove\_stopwords

text = "Yashesh likes to play football, however he is not too fond of tennis."

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:'''

from gensim.parsing.preprocessing import STOPWORDS

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

text = "Yashesh likes to play football, however he is not too fond of tennis."

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:

['Yashesh', 'football', ',', 'fond', 'tennis', '.']

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

from gensim.parsing.preprocessing import STOPWORDS

all\_stopwords\_gensim = STOPWORDS

sw\_list = {"not"}

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

text = "Yashesh likes to play football, however he is not too fond of tennis."

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**

Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio": <https://visualstudio.microsoft.com/downloads/>

**iii) Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List**

**code:**

#pip install spacy

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

#python -m spacy download en

import spacy

import nltk

from nltk.tokenize import word\_tokenize

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

#add the word play to the NLTK stop word collection

all\_stopwords = sp.Defaults.stop\_words

all\_stopwords.add("play")

text = "Yashesh likes to play football, however he is not too fond of tennis."

text\_tokens = word\_tokenize(text)

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

print(tokens\_without\_sw)

#remove 'not' from stop word collection

all\_stopwords.remove('not')

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

print(tokens\_without\_sw)

**output:**

Graphical user interface, text

Description automatically generated

1. **Text Tokenization**

**a. Tokenization using Python’s split() function**

**code:**

text = """ This tool is an a beta stage. Alexa developers can use Get Metrics API to seamlessly analyse metric. It also supports custom skill model, prebuilt Flash Briefing model, and the Smart Home Skill API. You can use this tool for creation of monitors, alarms, and dashboards that spotlight changes. The release of these three tools will enable developers to create visual rich skills for Alexa devices with screens. Amazon describes these tools as the collection of tech and tools for creating visually rich and interactive voice experiences. """

data = text.split('.')

for i in data:

print (i)

**output:**

Text

Description automatically generated

**b. Tokenization using Regular Expressions (RegEx)**

**code:**

import nltk

# import RegexpTokenizer() method from nltk

from nltk.tokenize import RegexpTokenizer

# Create a reference variable for Class RegexpTokenizer

tk = RegexpTokenizer('\s+', gaps = True)

# Create a string input

str = "I love to study Natural Language Processing in Python"

# Use tokenize method

tokens = tk.tokenize(str)

print(tokens)

**output:**

Text

Description automatically generated with medium confidence

**c. Tokenization using NLTK**

**code:**

import nltk

from nltk.tokenize import word\_tokenize

# Create a string input

str = "I love to study Natural Language Processing in Python"

# Use tokenize method

print(word\_tokenize(str))

**output:**

Text

Description automatically generated with low confidence

**d. Tokenization using the spaCylibrary**

**code:**

import spacy

nlp = spacy.blank("en")

# Create a string input

str = "I love to study Natural Language Processing in Python"

# Create an instance of document;

# doc object is a container for a sequence of Token objects.

doc = nlp(str)

# Read the words; Print the words

#

words = [word.text for word in doc]

print(words)

**output:**

Text

Description automatically generated with low confidence

**e. Tokenization using Keras**

**code:**

#pip install keras

#pip install tensorflow

import keras

from keras.preprocessing.text import text\_to\_word\_sequence

# Create a string input

str = "I love to study Natural Language Processing in Python"

# tokenizing the text

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

print(tokens)

**output:**

Text

Description automatically generated

**f. Tokenization using Gensim**

**code:**

#pip install gensim

from gensim.utils import tokenize

# Create a string input

str = "I love to study Natural Language Processing in Python"

# tokenizing the text

list(tokenize(str))

**output:**

Microsoft Visual C++ 14.0 is required. Get it with "Build Tools for Visual Studio": <https://visualstudio.microsoft.com/downloads/>

1. **Import NLP Libraries for Indian Languages and perform:**

Note: Execute this practical in https://colab.research.google.com/

**a) word tokenization in Hindi**

**Source code:**

!pip install torch==1.3.1+cpu -f <https://download.pytorch.org/whl/torch_stable.html>

pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch\_stable.html

pip install inltk

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup('hi')

from inltk.inltk import tokenize

hindi\_text = """प्राकृतिकभाषासीखनाबहुतदिलचस्पहै।"""

# tokenize(input text, language code)

tokenize(hindi\_text, "hi")

**output**

['▁प्राकृतिक', '▁भाषा', '▁सीखना', '▁बहुत', '▁दिलचस्प', '▁है', '।']

**b) Generate similar sentences from a given Hindi text input**

**Source code:**

!pip install torch==1.3.1+cpu -f <https://download.pytorch.org/whl/torch_stable.html>

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup('hi')

from inltk.inltk import get\_similar\_sentences

# get similar sentences to the one given in hindi

output = get\_similar\_sentences('मैंआजबहुतखुशहूं', 5, 'hi')

print(output)

**Output:**

['मैंआजकलबहुतखुशहूं', 'मैंआजअत्यधिकखुशहूं', 'मैंअभीबहुतखुशहूं', 'मैंवर्तमानबहुतखुशहूं', 'मैंवर्तमानबहुतखुशहूं']

**c) Identify the Indian language of a text**

**Source code:**

!pip install torch==1.3.1+cpu -f <https://download.pytorch.org/whl/torch_stable.html>

!pip install inltk

!pip install tornado==4.5.3

from inltk.inltk import setup

setup('gu')

from inltk.inltk import identify\_language

#Identify the Lnaguage of given text

identify\_language('બીનાકાપડિયા')

**Output:**

gujarati

1. **Illustrate part of speech tagging.**

**a. Part of speech Tagging and chunking of user defined text.**

**b. Named Entity recognition of user defined text.**

**c. Named Entity recognition with diagram using NLTK corpus – treebank**

**POS Tagging, chunking and NER:**

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

**Source code:**

import nltk

from nltk import tokenize

nltk.download('punkt')

from nltk import tag

from nltk import chunk

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

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

nltk.download('words')

para = "Hello! My name is Beena Kapadia. 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)

# 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")

print(tree)

**Output:**

sentence tokenization

===================

['Hello!', 'My name is Beena Kapadia.', "Today you'll be learning NLTK."]

word tokenization

===================

['Hello', '!']

['My', 'name', 'is', 'Beena', 'Kapadia', '.']

['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']

POS Tagging

===========

[[('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')]]

chunking

========

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')])]

**b) Named Entity recognition using user defined text.**

**Source code:**

!pip install -U spacy

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

import spacy

# Load English tokenizer, tagger, parser and NER

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:**

Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', 'major American car companies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']

Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'be', 'talk', 'say']

**c) Named Entity recognition with diagram using NLTK corpus – treebank.**

**Source code:**

Note: It runs on Python IDLE

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()

**Output:**

A picture containing timeline

Description automatically generated

1. **Finite state automata**
2. **Define grammar using nltk. Analyze 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 -> singular Noun

NP -> 'flight'

VP -> 'Book'

""")

sentence = "Book that flight"

for index in range(len(sentence)):

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:

Graphical user interface

Description automatically generated with low confidence

**b) Accept the input string with Regular expression of Finite Automaton: 101+.**

**Source 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"

#first three characters are fixed. Therefore, checking them using index

if s[0]=='1':

if s[1]=='0':

if s[2]=='1':

# After index 2 only "1" can appear. Therefore break the process if any other character is detected

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:**

Rejected

Rejected

Accepted

Accepted

Rejected

Rejected

Rejected

Rejected

Accepted

1. **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:**

Rejected

Rejected

Accepted

Accepted

Rejected

Rejected

Rejected

Rejected

Accepted

1. **Implementation of Deductive Chart Parsing using context free grammar and a givensentence.**

**Source code:**

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.draw()

**output:**

Graphical user interface

Description automatically generatedGraphical user interface, diagram

Description automatically generated

**8.** **Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer**

**Study WordNetLemmatizer**

**Code:**

#**PorterStemmer**

import nltk

from nltk.stem import PorterStemmer

word\_stemmer = PorterStemmer()

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

**Output:**

 **#LancasterStemmer**

import nltk

from nltk.stem import LancasterStemmer

Lanc\_stemmer = LancasterStemmer()

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

**Output:**



**#RegexpStemmer**

import nltk

from nltk.stem import RegexpStemmer

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

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

**output**



**#SnowballStemmer**

import nltk

from nltk.stem import SnowballStemmer

english\_stemmer = SnowballStemmer('english')

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

**output**



**#WordNetLemmatizer**

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("word :\tlemma")

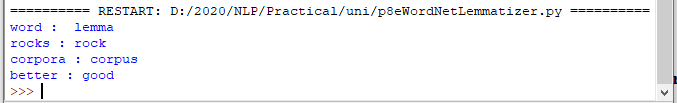
print("rocks :", lemmatizer.lemmatize("rocks"))

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

# a denotes adjective in "pos"

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

**Output:**



**9. Implement Naive Bayes classifier**

**Code:**

#pip install pandas

#pip install sklearn

import pandas as pd

import numpy as np

sms\_data = pd.read\_csv("spam.csv", encoding='latin-1')

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

stemming = PorterStemmer()

corpus = []

for i in range (0,len(sms\_data)):

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

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)

from sklearn.feature\_extraction.text import CountVectorizer

countvectorizer =CountVectorizer()

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

print(x)

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

print(y)

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

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

#Multinomial Naïve Bayes.

from sklearn.naive\_bayes import MultinomialNB

multinomialnb = MultinomialNB()

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

# Predicting on test data:

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

print(y\_pred)

#Results of our Models

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

from sklearn.metrics import accuracy\_score

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

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

**input:**

spam.csv file from github

**output:**

Background pattern

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

**10. a. Speech Tagging:**

**i.Speech tagging using spacy**

**code**

import spacy

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

sen = sp(u"I like to play football. 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)

num\_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**:

Text

Description automatically generated with medium confidence

To view the dependency tree, type the following address in your browser: http://127.0.0.1:5000/. You will see the following dependency tree:

**ii. Speech tagging using nktl**

**code:**

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 like:

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()

output:

Text, letter

Description automatically generated

**b. Statistical parsing:**

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

**Source code:**

#probabilitistic parser

#Usage of Give and Gave in the Penn Treebank sample

import nltk

import nltk.parse.viterbi

import nltk.parse.pchart

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:**

Text, letter

Description automatically generated

**ii. probabilistic parser**

**Source code:**

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:**

Graphical user interface, text, application, email

Description automatically generated

**c. Malt parsing:**

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

Note: 1) Java should be installed.

2) maltparser-1.7.2 zip file should be copied in C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder and should be extracted in the same folder.

3) engmalt.linear-1.7.mco file should be copied to C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder

**Source code:**

# copy maltparser-1.7.2(unzipped version) and engmalt.linear-1.7.mco files to C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39 folder

# java should be installed

# environment variables should be set - MALT\_PARSER - C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39\maltparser-1.7.2 and MALT\_MODEL - C:\Users\Beena Kapadia\AppData\Local\Programs\Python\Python39\engmalt.linear-1.7.mco

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:**

(saw I (bird a (from (window. my))))

Diagram

Description automatically generated with medium confidence

1. **a) 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 Mumbai. 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:**

Graphical user interface, text

Description automatically generated with medium confidence

**b) Normalized Web Distance and Word Similarity**

**Source code:**

# Normalized Web Distance and Word Similarity

#convert

#Reliance supermarket

#Reliance hypermarket

#Reliance

#Reliance

#Reliance downtown

#Relianc market

#Mumbai

#Mumbai Hyper

#Mumbai dxb

#mumbai airport

#k.m trading

#KM Trading

#KM trade

#K.M. Trading

#KM.Trading

#into

#Reliance

#Reliance

#Reliance

#Reliance

#Reliance

#Reliance

#Mumbai

#Mumbai

#Mumbai

#Mumbai

#KM Trading

#KM Trading

#KM Trading

#KM Trading

#KM Trading

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', 'Relianc 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:**

Text

Description automatically generated

**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:**

Text

Description automatically generated

**c. Word Sense Disambiguation**

import codecs

from nltk.tokenize import PunktSentenceTokenizer,sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords, wordnet

from nltk.stem import WordNetLemmatizer, PorterStemmer

def simpleFilter(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

# Remove Stop Words . Word Stemming . Return new tokenised list.

def filteredSentence(sentence):

filtered\_sent = []

lemmatizer = WordNetLemmatizer() #lemmatizes the words

ps = PorterStemmer() #stemmer stems the root of the word.

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(ps.stem(w)))

for i in synonymsCreator(w):

filtered\_sent.append(i)

return filtered\_sent

if \_\_name\_\_ == '\_\_main\_\_':

cricfile = codecs.open("E:\\Department\\MscIT\_Sem\_III\_and sem I nov 2020\\NLP\\cricketbat.txt", 'r', "utf-8")

sent2 = cricfile.read().lower()

vampirefile = codecs.open("E:\\Department\\MscIT\_Sem\_III\_and sem I nov 2020\\NLP\\vampirebat.txt", 'r', 'utf-8')

sent1 = vampirefile.read().lower()

sent3 = "start"

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 animals come and drink water from it.

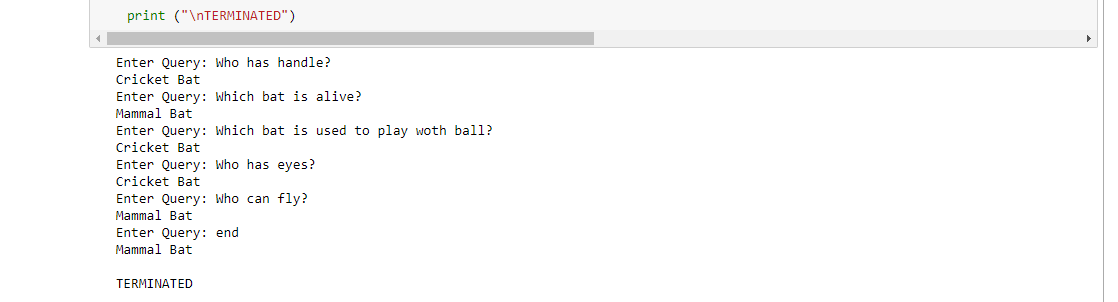
#sentence2: the commercial banks are used for finance. all the financial matters are managed by financial banks and they have lots of money, user accounts like salary account and savings account, current account. money can also be withdrawn from this bank.

#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 the hind limbs or tail and anatomical adaptations for echolocation, by which they navigate and hunt prey.

#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 and is used by cricket players.

print ("\nTERMINATED")

****