RAMNIRANJAN JHUNJHUNWALA COLLEGE

GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF INFORMATION TECHNOLOGY

2021 - 2022

M.Sc.(I.T.) SEMESTER IV

Natural Language Processing

Name: Preeti Mishra

Roll No: 05

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Ms.**Preeti Mishra** with Roll No. **05** has successfully completed the necessary course of experiments in the subject of **NLP** during the academic year **2021 – 2022** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -IV.

_____ _____

Internal Examiner External Examiner

_____ _____

Head of Department College Seal

INDEX

Practical No	Practical
01	a) Install NLTK. b) Convert the given text to speech.
02	a) Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, 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 i) DefaultTagger, ii) Regular expression tagger, iii) UnigramTagger
03	a) Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms b) Study lemmas, hyponyms, hypernyms.
04	Text Tokenization
05	Import NLP Libraries for Indian Languages and perform a) Word tokenization in Hindi b) Generate similar sentences from a given Hindi text input.
06	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
07	Finite state automata a) Define grammar using nltk. Analyze a sentence using the same. b) Accept the input string with Regular expression of Finite Automaton: 101+. c) Accept the input string with Regular expression of FA: (a+b)*bba. d) Implementation of Deductive Chart Parsing using context free grammar and a
08	Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer Study WordNetLemmatizer
09	Try to get the scatter plot for PCA projection of Word2Vec model for the given text
10	a. Speech Tagging: b. Speech tagging using nktl c. Malt parsing:
11	a) Multiword Expressions in NLP b) Normalized Web Distance and Word Similarity c) Word Sense Disambiguation

Practical No. 1:

a) Install NLTK.
b) Convert the given text to speech.

Code:-

```
# text to speech
# pip install gtts
# pip install playsound
from playsound import playsound
# import required for text to 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 SpeechRecognition pydub
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)

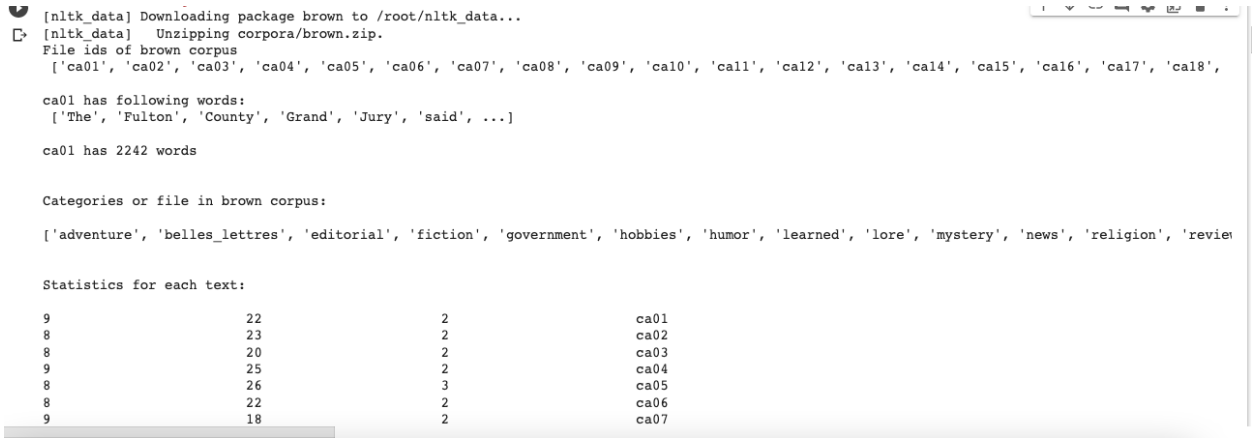
Output:

Practical No. 2:
a. Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories.

```
import nltk
%pip install brown
import nltk
nltk.download('brown')
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:-



```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
File ids of brown corpus
 ['ca01', 'ca02', 'ca03', 'ca04', 'ca05', 'ca06', 'ca07', 'ca08', 'ca09', 'ca10', 'ca11', 'ca12', 'ca13', 'ca14', 'ca15', 'ca16', 'ca17', 'ca18',

ca01 has following words:
 ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]

ca01 has 2242 words


Categories or file in brown corpus:

['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'revie

Statistics for each text:

9                       22                      2                       ca01
8                       23                      2                       ca02
8                       20                      2                       ca03
9                       25                      2                       ca04
8                       26                      3                       ca05
8                       22                      2                       ca06
9                       18                      2                       ca07
```

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

source code:

```
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)
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:-

c. Study Conditional frequency distributions

source code:

```
#process a sequence of pairs
import nltk
nltk.download('inaugural')
nltk.download('udhr')
```

```
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:-

```
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package udhr to /root/nltk_data...
[nltk_data]   Unzipping corpora/udhr.zip.
170576
[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]
[('romance', 'afraid'), ('romance', 'not'), ('romance', "''"), ('romance', '.')]
<ConditionalFreqDist with 2 conditions>
['news', 'romance']
<FreqDist with 14394 samples and 100554 outcomes>
<FreqDist with 8452 samples and 70022 outcomes>
['They', 'neither', 'liked', 'nor', 'disliked', 'the', 'Old', 'Man', '.', 'To', 'them', 'he', 'could', 'have', 'been', 'broken', 'bell', 'in', 'ch
                  0    1    2    3    4    5    6    7    8    9
       English    0  185  525  883  997 1166 1283 1440 1558 1638
German_Deutsch    0  171  263  614  717  894 1013 1110 1213 1275
```

d. Study of tagged corpora with methods like tagged_sents, tagged_words.

Source code:

```
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('words')
para = "Hello! My name is Preeti Mishra. 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:-

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!

sentence tokenization
==================
 ['Hello!', 'My name is Preeti Mishra.', "Today you'll be learning NLTK."]

word tokenization
==================

['Hello', '!']
['My', 'name', 'is', 'Preeti', 'Mishra', '.']
['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']
```

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

Source Code:

```
from collections import defaultdict
text = nltk.word_tokenize("Preeti is a Application Developer at TCS.Preeti
Working in TCS BANCS")
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:-

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[('Preeti', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('Application', 'NNP'), ('Developer', 'NNP'), ('at', 'IN'), ('TCS.Preeti', 'NNP'), ('Working', 'NNP'
['Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti', 'Preeti']
Word with maximum frequency : Preeti
```

f. Map Words to Properties Using Python Dictionaries

Source 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:-

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
3
<class 'dict'>
```

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

i) DefaultTagger

Source code:

```
import nltk
nltk.download('treebank')
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:-

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
0.13198749536374715
[[('Hi', 'NN'), (',', 'NN')], [('How', 'NN'), ('are', 'NN'), ('you', 'NN'), ('?', 'NN')]]
```

---

ii) Regular expression tagger,

Source 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:-

```
<Regexp Tagger: size=9>
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'), ('Jury', 'NN'), ('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'), ('investigation
```

iii) UnigramTagger

Source code:

```
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:-

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director', 'Nov.', '29', '.']

 [('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('th

 [('Pierre', 'NN'), ('Vinken', None), (',', None), ('61', None), ('years', None), ('old', None), (',', None), ('will', None), ('join', None), ('the
```

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:

```
rom __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)
```

Output:-

```
...  MENU
     -----------
      1 . Hash tag segmentation
      2 . URL segmentation
     enter the input choice for performing word segmentation
     [                    ]
```

```
input with HashTag #whatismyname
whatismyname
what -appears in the corpus
is -appears in the corpus
my -appears in the corpus
name -appears in the corpus
output
---------
what is my name
Score 4.0
```

input with URL www.whatismyname.com
whatismyname
what -appears in the corpus
is -appears in the corpus
my -appears in the corpus
name -appears in the corpus
output
---
what is my name
Score 4.0

Practical No : 03

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())
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[Synset('computer.n.01'), Synset('calculator.n.01')]
a machine for performing calculations automatically
Examples: []
[Lemma('sell.v.01.sell')]
```

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:-

```
[Synset('computer.n.01'), Synset('calculator.n.01')]
['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'information_processing_system']
Synset('computer.n.01') --> ['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'information_processing_s
Synset('calculator.n.01') --> ['calculator', 'reckoner', 'figurer', 'estimator', 'computer']
[Lemma('computer.n.01.computer'), Lemma('computer.n.01.computing_machine'), Lemma('computer.n.01.computing_device'), Lemma('computer.n.01.data_proce
Synset('computer.n.01')
computing_device
<bound method _WordNetObject.hyponyms of Synset('computer.n.01')>
['analog_computer', 'analogue_computer', 'digital_computer', 'home_computer', 'node', 'client', 'guest', 'number_cruncher', 'pari-mutuel_machine', '
[Synset('vehicle.n.01')]
```

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

Source code:

```
from nltk.corpus import wordnet
print( wordnet.synsets("active"))
```

```
print(wordnet.lemma('active.a.01.active').antonyms())
```

Output:-

```
[Synset('active_agent.n.01'), Synset('active_voice.n.01'), Synset('active.n.03'), Synset('active.a.01'), Synset('active.s.02'), Synset('active.a.03'
[Lemma('inactive.a.02.inactive')]
```

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:-

```
Path similarity of:
Synset('football.n.01') ( n ) [ any of various games played with a ball (round or oval) in which two teams try to kick or carry or propel the ball i
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to kick or head a ball into the opponents' goal ]
 is 0.5

Path similarity of:
Synset('football.n.02') ( n ) [ the inflated oblong ball used in playing American football ]
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to kick or head a ball into the opponents' goal ]
 is 0.05
```

e. Handling stopword:

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

code:

```
import nltk
nltk.download('punkt')
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:-

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['Yashesh', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'not', 'fond', 'tennis', '.']
```

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

code:

```
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:-

```
Yashesh likes play football, fond tennis.
frozenset({'am', 'around', 'than', 'third', 'six', 'those', 'nobody', 'often', 'sincere', 'whereas', 'become', 'do', 'whereupon', 'did', 'else', 'le
['Yashesh', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'play', 'football', ',', 'not', 'fond', 'tennis', '.']
```

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:-

```
['Yashesh', 'likes', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'not', 'fond', 'tennis', '.']
```

4. 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:-

```
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
```

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:-

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

---

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:-

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

---

d. Tokenization using the spaCy library
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:-

```
['I', 'love', 'to', 'study', 'Natural', 'Language', 'Processing', 'in', 'Python']
```

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:-

```
['i', 'love', 'to', 'study', 'natural', 'language', 'processing', 'in', 'python']
```

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:-

```
['I',
 'love',
 'to',
 'study',
 'Natural',
 'Language',
 'Processing',
 'in',
 'Python']
```

5. 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 inltk
!pip install tornado==4.5.3
from inltk.inltk import setup
setup('hi')
from inltk.inltk import tokenize
hindi_text = """प्राकृ तिक भाषा सीखना बहि तिलचस्प है।"""
# 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('मैंआज बहि खुश हूं', 5, 'hi')
print(output)
```

Output:-

['मैं आजकल बहुत खुश हूं', 'मैं आज अत्यधिक खुश हूं', 'मैं अभी बहुत खुश हूं', 'मैं वर्तमान बहुत खुश हूं', 'मैं वर्तमान बहुत खुश हूं']

c) Identify the Indian language of a text

Source code:

```
!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('બીના કાપડિયા')
```

Output:-

gujarati

Practical No : 6

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'), ('.', '.')]]

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'), ('.', '.')]]

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')]), ('.', '.')])]

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')]), ('.', '.')])]

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', 'talk',
'say']
```

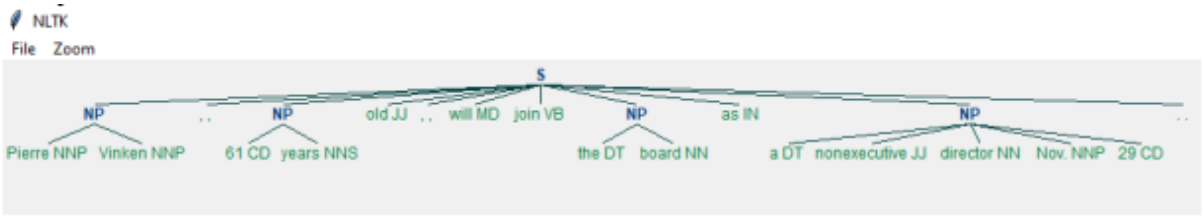
c) Named Entity recognition with diagram using NLTK corpus – treebank.

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:


NLTK
File Zoom
S
NP
Pierre NNP
Vinken NNP
,,
NP
61 CD
years NNS
old JJ
,,
will MD
join VB
NP
the DT
board NN
as IN
NP
a DT
nonexecutive JJ
director NN
Nov. NNP
29 CD
..

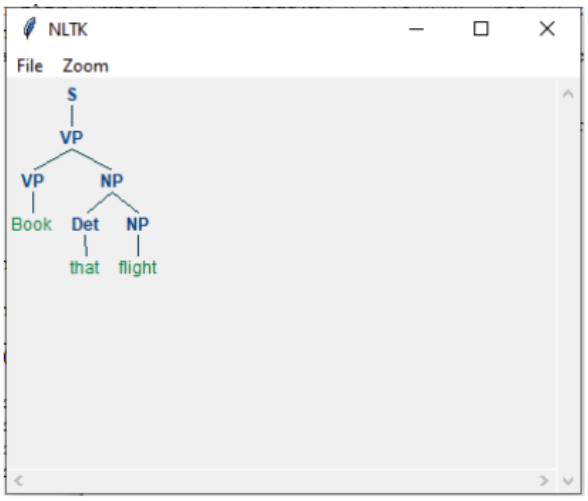
Practical No : 07
Finite state automata

a) 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 :



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
```

c) 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:

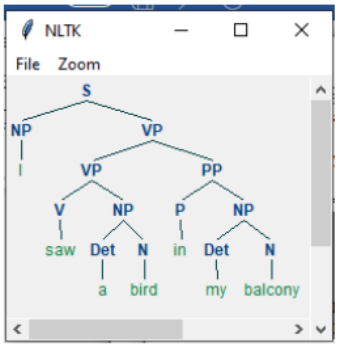
```
Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
Accepted
```

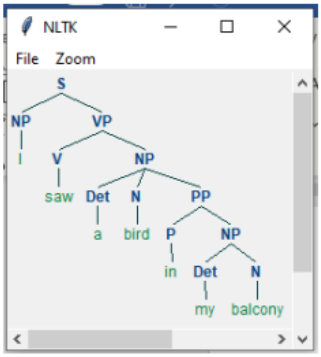
d) Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

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





Practical No : 08
Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer
Study WordNetLemmatizer

Code:

```
# PorterStemmer
import nltk
from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
print(word_stemmer.stem('writing'))

#LancasterStemmer
import nltk
from nltk.stem import LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print(Lanc_stemmer.stem('writing'))


#RegexpStemmer
import nltk
from nltk.stem import RegexpStemmer
Reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)
print(Reg_stemmer.stem('writing'))


#SnowballStemmer
import nltk
from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
print(english_stemmer.stem ('writing'))


#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"))
```

```
# PorterStemmer
import nltk
from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
print(word_stemmer.stem('writing'))
```

```
write
```

```
#LancasterStemmer
import nltk
from nltk.stem import LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print(Lanc_stemmer.stem('writing'))
```

```
writ
```

```
#RegexpStemmer
import nltk
from nltk.stem import RegexpStemmer
Reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)
print(Reg_stemmer.stem('writing'))
```

```
writ
```

```
#SnowballStemmer
import nltk
from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
print(english_stemmer.stem ('writing'))
```

```
write
```

```python
#WordNetLemmatizer
import nltk
nltk.download('wordnet')
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"))
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
word :  lemma
rocks : rock
corpora : corpus
better : good
```

Practical No : 09

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

```
[ ] import nltk
    nltk.download('stopwords')
    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()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
[ ] 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)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

```
[ ] #Multinomial Naïve Bayes.
    from sklearn.naive_bayes import MultinomialNB
    multinomialnb = MultinomialNB()
    multinomialnb.fit(x_train,y_train)
```

```
MultinomialNB()
```

```
[ ]  # Predicting on test data:
     y_pred = multinomialnb.predict(x_test)
     print(y_pred)
```

```
['ham' 'ham' 'ham' ... 'ham' 'ham' 'ham']
```

```
[ ]  #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))
```

```
              precision    recall  f1-score   support

         ham       0.99      0.99      0.99      1448
        spam       0.92      0.93      0.92       224

    accuracy                           0.98      1672
   macro avg       0.95      0.96      0.96      1672
weighted avg       0.98      0.98      0.98      1672

accuracy_score:  0.979066985645933
```

Practical No : 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:

```
I like to play football. I hated it in my childhood though
VERB
VBD
verb, past tense
I            PRON       PRP      pronoun, personal
like         VERB       VBP      verb, non-3rd person singular present
to           PART       TO       infinitival "to"
play         VERB       VB       verb, base form
football     NOUN       NN       noun, singular or mass
.            PUNCT      .        punctuation mark, sentence closer
I            PRON       PRP      pronoun, personal
hated        VERB       VBD      verb, past tense
it           PRON       PRP      pronoun, personal
in           ADP        IN       conjunction, subordinating or preposition
my           DET        PRP$     pronoun, possessive
childhood    NOUN       NN       noun, singular or mass
though       SCONJ      IN       conjunction, subordinating or preposition
```

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

```
[nltk_data] Downloading package state_union to /root/nltk_data...
[nltk_data]   Unzipping corpora/state_union.zip.

**********************************************************************
  Resource punkt not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt')

  Searched in:
    - '/root/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
    - '/usr/nltk_data'
    - '/usr/lib/nltk_data'
    -
```

Y

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:

```
================== RESTART: D:/2020/NLP/Practical/uni/p10b1.py ==============
gave NP: the chefs / NP: a standing ovation
give NP: advertisers / NP: discounts for maintaining or increasing ad sp...
give NP: it / PP-DTV: to the politicians
gave NP: them / NP: similar help
give NP: them / NP:
give NP: only French history questions / PP-DTV: to students in a Europe...
give NP: federal judges / NP: a raise
give NP: consumers / NP: the straight scoop on the U.S. waste crisis
gave NP: Mitsui / NP: access to a high-tech medical product
give NP: Mitsubishi / NP: a window on the U.S. glass industry
give NP: much thought / PP-DTV: to the rates she was receiving , nor to ...
give NP: your Foster Savings Institution / NP: the gift of hope and free...
give NP: market operators / NP: the authority to suspend trading in futu...
gave NP: quick approval / PP-DTV: to $ 3.18 billion in supplemental appr...
give NP: the Transportation Department / NP: up to 50 days to review any...
give NP: the president / NP: such power
give NP: me / NP: the heebie-jeebies
give NP: holders / NP: the right , but not the obligation , to buy a cal...
gave NP: Mr. Thomas / NP: only a `` qualified '' rating , rather than ``...
give NP: the president / NP: line-item veto power
>>>
```

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 :

Practical No : 11

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['Good', 'cake', 'cost', 'Rs.1500\\kg', 'in', 'Mumbai', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
```

b) Normalized Web Distance and Word Similarity
Source code:

```
# Normalized Web Distance and Word Similarity
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:

```
========== RESTART: D:/2020/NLP/Practical/uni/p11b.py ==========
['reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'mumbai
', 'mumbai', 'mumbai', 'mumbai', 'km trading', 'km trading', 'km trading', 'km t
rading', 'km trading']
>>>
```

c) 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:

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
<bound method Synset.name of Synset('bank.n.01')>: <bound method
Synset.definition of Synset('bank.n.01')>
<bound method Synset.name of Synset('set.n.01')>: <bound method
Synset.definition of Synset('set.n.01')>
<bound method Synset.name of Synset('put.v.01')>: <bound method
Synset.definition of Synset('put.v.01')>
```