

Parle Tilak Vidyalaya Associations
M. L. DAHANUKAR COLLEGE
Vile-Parle (East), Mumbai – 400 057.

Practical Journal
NATURAL LANGUAGE PROCESSING

Submitted by
Trupti Hiru Bhostekar

Seat No.:

M.Sc. [I.T.]-Information Technology Part II

2023 – 2024

INDEX

Sr no	AIM	Date	Sign
1	<ul style="list-style-type: none">a. Install NLTKb. Convert the given text to speechc. Convert audio file Speech to Text.		
2	<ul style="list-style-type: none">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 Study of tagged corpora with methods like tagged_sents, tagged_words.d. Write a program to find the most frequent noun tags.e. Map Words to Properties Using Python Dictionariesf. Study DefaultTagger, Regular expression tagger, UnigramTaggerg. 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.		
3	<ul style="list-style-type: none">a. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.b. Study lemmas, hyponyms, hypernyms, entailments,c. Write a program using python to find synonym and antonym of word "active" using Wordnetd. Compare two nounse. Handling stopword. Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List		
4	<p>Text Tokenization</p> <ul style="list-style-type: none">a. Tokenization using Python's split() functionb. Tokenization using Regular Expressions (RegEx)c. Tokenization using NLTKd. Tokenization using the spaCy librarye. Tokenization using Kerasf. Tokenization using Gensim		
5	Important NLP Libraries for Indian Languages and perform:		

	<ul style="list-style-type: none">a. word tokenization in Hindib. Generate similar sentences from a given Hindi text inputc. Identify the Indian language of a text		
6	<p>Illustrate part of speech tagging.</p> <ul style="list-style-type: none">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		
7	<ul style="list-style-type: none">a. Define grammer using nltk. Analyze a sentence using the same.b. Accept the input string with Regular expression of FA: 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 given sentence.		
8	Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer Study WordNetLemmatizer		
9	Implement Naive Bayes classifier		

Practical No 1

Aim :

- Install NLTK
- Convert the given text to speech
- Convert audio file Speech to Text.

a. Install NLTK

>pip install gtts

```
C:\Users\nishant\AppData\Local\Programs\Python\Python311\Scripts>pip install gtts
Collecting gtts
  Downloading gtts-2.5.1-py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: requests<3,>=2.27 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from gtts) (2.32.2)
Requirement already satisfied: click<8.2,>=7.1 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from gtts) (8.1.7)
Requirement already satisfied: colorama in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from click<8.2,>=7.1->gtts) (0.4.6)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests<3,>=2.27->gtts) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests<3,>=2.27->gtts) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests<3,>=2.27->gtts) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests<3,>=2.27->gtts) (2024.2.2)
Downloading gtts-2.5.1-py3-none-any.whl (29 kB)
Installing collected packages: gtts
Successfully installed gtts-2.5.1
```

>pip install nltk

```
C:\Users\nishant\AppData\Local\Programs\Python\Python311\Scripts>pip install nltk
Collecting nltk
  Using cached nltk-3.8.1-py3-none-any.whl.metadata (2.8 kB)
Requirement already satisfied: click in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from nltk) (1.4.2)
Collecting regex<=2021.8.3 (from nltk)
  Downloading regex-2024.5.15-cp311-cp311-win_amd64.whl.metadata (41 kB)
----- 42.0/42.0 kB 2.0 MB/s eta 0:00:00
Requirement already satisfied: tqdm in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from regex-2024.5.15-cp311-cp311-win_amd64.whl) (4.66.4)
Requirement already satisfied: colorama in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from click->nltk) (0.4.6)
Using cached nltk-3.8.1-py3-none-any.whl (1.5 MB)
Downloading regex-2024.5.15-cp311-cp311-win_amd64.whl (268 kB)
----- 269.0/269.0 kB 976.4 kB/s eta 0:00:00
Installing collected packages: regex, nltk
Successfully installed nltk-3.8.1 regex-2024.5.15
```

>pip install SpeechRecognition pydub

```
C:\Users\nishant\AppData\Local\Programs\Python\Python311\Scripts>pip install SpeechRecognition pydub
Collecting SpeechRecognition
  Downloading SpeechRecognition-3.10.4-py2.py3-none-any.whl.metadata (28 kB)
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: requests>=2.26.0 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from SpeechRecognition) (2.32.2)
Requirement already satisfied: typing-extensions in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from SpeechRecognition) (4.12.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests>=2.26.0->SpeechRecognition) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests>=2.26.0->SpeechRecognition) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests>=2.26.0->SpeechRecognition) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\nishant\appdata\local\programs\python\python311\site-packages (from requests>=2.26.0->SpeechRecognition) (2024.2.2)
Downloading SpeechRecognition-3.10.4-py2.py3-none-any.whl (32.8 MB)
----- 32.8/32.8 MB 2.4 MB/s eta 0:00:00
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, SpeechRecognition
Successfully installed SpeechRecognition-3.10.4 pydub-0.25.1
```

b. Convert the given text to speech

Steps:

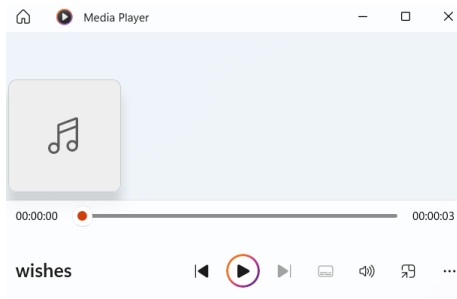
1. create file.txt in respective folder.
2. Enter some message in file.txt.
3. Save texttospeech.py file at same location.

Code :

```
from gtts import gTTS
import os
f = open('1.txt')
x=f.read()
language='en'
audio=gTTS(text=x,lang=language)
audio.save("wishes.wav")
os.system("wishes.wav")
print("program executed succesfully.")
```

Output :

```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract1b.py =====
program executed succesfully.
```



C. Convert audio file Speech to Text.

Code :

```
import speech_recognition as sr
filename = "Greetings.wav"
# initialize the recognizer
r = sr.Recognizer()
# open the file
with sr.AudioFile(filename) as source:
# listen for the data (load audio to memory)
    audio_data = r.record(source)
    # recognize (convert from speech to text)
    text = r.recognize_google(audio_data)
    print(text)
```

Output :

```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract1c.py =
good morning everyone
```

Practical No 2

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

As just mentioned, a text corpus is a large body of text. Many corpora are designed to contain a careful balance of material in one or more genres. We examined some small text collections, such as the speeches known as the US Presidential Inaugural Addresses.

This particular corpus actually contains dozens of individual texts — one per address — but for convenience we glued them end-to-end and treated them as a single text, also used various predefined texts that we accessed by typing from `nltk.book import *`. However, since we want to be able to work with other texts, this section examines a variety of text corpora. We'll see how to select individual texts, and how to work with them.

The Brown Corpus is a balanced corpus consisting of 500 text samples from 15 different categories.

Fields: Metadata about the corpus.

Raw Text: Full raw text of a specific file.

Words: Tokenized words in the corpus.

Sentences: Tokenized sentences in the corpus.

Categories: Accessing different categories of the corpus.

Code :

```
import nltk
nltk.download('brown')
from nltk.corpus import brown
brown.categories()
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Unzipping corpora/brown.zip.
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

`brown.fileids()`

```
➤ brown.fileids()
[ 'ca01',
  'ca02',
  'ca03',
  'ca04',
  'ca05',
  'ca06',
  'ca07',
  'ca08',
  'ca09',
  'ca10',
  'ca11',
```

`brown.words(categories = 'news')`

```
[6] new_var = brown.words(categories = 'news')
➤ ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

`brown.words(fileids='cg22')`

```
[8] brown.words(fileids='cg22')
➤ ['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
```

`brown.sents(categories=['news','editorial','reviews'])`

```
[9] brown.sents(categories=['news','editorial','reviews'])
➤ [['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', 'Atlanta's', 'recent', 'primary', 'election',
  'produced', 'no', 'evidence', 'that', 'any', 'irregularities', 'took', 'place', '.'], ['The', 'jury', 'further', 'said', 'in',
  'term-end', 'presentments', 'that', 'the', 'City', 'Executive', 'Committee', 'which', 'had', 'over-all', 'charge', 'of', 'the', 'election',
  'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of', 'Atlanta', 'for', 'the', 'manner', 'in', 'which',
  'the', 'election', 'was', 'conducted', '.'], ...]
```

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

Building a custom corpus can be useful for specific text analysis or natural language processing (NLP) tasks. In the following Python recipe, we are going to create a custom corpora which must be within one of the paths defined by NLTK. It is so because it can be found by NLTK. In order to avoid conflict with the official NLTK data package, let us create a custom `natural_language_toolkit_data` directory in our home directory.

Code :

```
import os, os.path
path = os.path.expanduser('~/.natural_language_toolkit_data')
if not os.path.exists(path):
    os.mkdir(path)
os.path.exists(path)
```


Output :

```
[10] import os, os.path
      path = os.path.expanduser('~natural_language_toolkit_data')
      if not os.path.exists(path):
          os.mkdir(path)
      os.path.exists(path)
```

⇒ True

Now we will make a wordlist file, named wordfile.txt and put it in nltk_data directory (content/wordfile.txt) and will load it by using nltk.data.load

Corpus readers

NLTK provides various CorpusReader classes.

Creating wordlist corpus

NLTK has WordListCorpusReader class that provides access to the file containing a list of words. For the following we need to create a wordlist file which can be CSV or normal text file. For example, we have created a file named 'list' that contains the following data .

Code :

```
from nltk.corpus.reader import WordListCorpusReader
reader_corpus = WordListCorpusReader('.', ['wordfile.txt'])
reader_corpus.words()
```

Output :

```
[11] from nltk.corpus.reader import WordListCorpusReader
      reader_corpus = WordListCorpusReader('.', ['wordfile.txt'])
      reader_corpus.words()
```

⇒ ['Hello', 'this', 'is', 'Practical', 'no', '2']

c. Study Conditional frequency distributions

Conditional frequency distributions (CFD) are a powerful tool in natural language processing and text analysis, allowing you to analyze the frequency of words or other features given specific conditions. The NLTK library in Python provides robust support for CFD through the `nltk.probability.ConditionalFreqDist` class.

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_Inuktitut', '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] Unzipping corpora/inaugural.zip.
[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>
['', '.', 'the', 'and', 'to', 'a', 'of', '``', ''''', 'was', 'I', 'in', 'he', 'had', '?', 'her', 'that', 'it
      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
```

Study of tagged corpora with methods like tagged_sents, tagged_words.

Studying tagged corpora involves analyzing text data where each word is associated with a part-of-speech (POS) tag. Parts of speech are also known as word classes or lexical categories. The collection of tags used for a particular task is known as a tagset. This tagging provides valuable syntactic information that can be used for various natural language processing (NLP) tasks. NLTK offers several tagged corpora, such as the Brown Corpus, and provides methods like `tagged_sents` and `tagged_words` to access this tagged data.

Tagged Words: Returns a list of (word, tag) tuples.

Tagged Sentences: Returns a list of sentences, where each sentence is a list of (word, tag) tuples.

Code :

```
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('words')

para = "And now we are going to learn something new"
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(nltk.pos_tag(words))
```

```
📄 [nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data] Unzipping corpora/words.zip.

sentence tokenization
=====
['And now we are going to learn something new']

word tokenization
=====

[('And', 'CC'), ('now', 'RB'), ('we', 'PRP'), ('are', 'VBP'), ('going', 'VBG'), ('to', 'TO'), ('learn', 'VB'), ('something', 'NN'), ('new', 'JJ')]
```

```
import nltk
nltk.corpus.brown.tagged_words()
```

```
▶ import nltk
nltk.corpus.brown.tagged_words()

📄 [('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

```
nltk.download('treebank')
nltk.corpus.treebank.tagged_words()
```

```
[19] nltk.download('treebank')
      nltk.corpus.treebank.tagged_words()
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ',', ' '), ...]
```

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

To find the most frequent noun tags in a tagged corpus using NLTK, you can follow these steps:

- Access the tagged words from a corpus.
- Filter out the noun tags.
- Compute the frequency distribution of these tags.
- Identify the most frequent noun tags.

Code :

```
nltk.download('universal_tagset')
from nltk.corpus import brown
noundist = nltk.FreqDist(w2 for ((w1, t1), (w2, t2)) in
    nltk.bigrams(brown.tagged_words(tagset="universal"))
    if w1.lower() == "the" and t2 == "NOUN")
noundist.most_common(10)
```

Output :

```
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
[('world', 346),
 ('time', 250),
 ('way', 236),
 ('end', 206),
 ('fact', 194),
 ('state', 190),
 ('man', 176),
 ('door', 172),
 ('house', 152),
 ('city', 127)]
```

e. Map Words to Properties Using Python Dictionaries

Mapping words to their properties using Python dictionaries is a common and powerful technique. You can create dictionaries where words are keys and their properties are values. These properties can include part-of-speech tags, frequencies, definitions, or any other relevant information.

Code :

```
#creating and printing a dictionary by mapping word with its properties
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
print(thisdict["brand"])
print('length:', len(thisdict))
print(type(thisdict))
```

Output :

```
[21] #creating and printing a dictionary by mapping word with its properties
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
print(thisdict["brand"])
print('length:', len(thisdict))
print(type(thisdict))
```

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

f. Study DefaultTagger, Regular expression tagger, UnigramTagger

These taggers can be used individually or in combination to build more sophisticated taggers or to handle different tagging tasks effectively.

A. DefaultTagger

The DefaultTagger assigns a specific tag to every word in the text. It's useful as a baseline tagger or when dealing with unknown words.

Code :

```
from nltk.corpus import brown
nltk.download('brown')
tags = [tag for (word, tag) in brown.tagged_words(categories='news')]
nltk.FreqDist(tags).max()
raw = 'I do not like green eggs and ham, I do not like them Sam I am!'
tokens = nltk.word_tokenize(raw)
default_tagger = nltk.DefaultTagger('NN')
default_tagger.tag(tokens)
```

Output :

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Package brown is already up-to-date!
[('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('green', 'NN'),
 ('eggs', 'NN'),
 ('and', 'NN'),
 ('ham', 'NN'),
 (',', 'NN'),
 ('I', 'NN'),
 ('do', 'NN'),
 ('not', 'NN'),
 ('like', 'NN'),
 ('them', 'NN'),
 ('Sam', 'NN'),
 ('I', 'NN'),
 ('am', 'NN'),
 ('!', 'NN')]
```

B. Regular Expression Tagger:

The Regular Expression Tagger assigns tags based on matching patterns defined using regular expressions. It's useful for tagging specific patterns of words.

Code :

```
import nltk
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import word_tokenize
from nltk import RegexpTagger
brown_sents = brown.sents(categories = 'news')
brown_tagged_sents = brown.tagged_sents(categories = 'news')
import nltk
```

```
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import word_tokenize
from nltk import RegexpTagger
brown_sents = brown.sents(categories = 'news')
brown_tagged_sents = brown.tagged_sents(categories = 'news')
patterns = [
(r'.*ing$', 'VBG'), # gerunds
(r'.*ed$', 'VBD'), # simple past
(r'.*es$', 'VBZ'), # 3rd singular present
(r'.*ould$', 'MD'), # modals
(r'.*\s$', 'NN$'), # possessive nouns
(r'.*s$', 'NNS'), # plural nouns
(r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
(r'.*', 'NN') # nouns (default)
]
regexp_tagger = nltk.RegexpTagger(patterns)
regexp_tagger.tag(brown_sents[3])
```

Output :

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[('...', 'NN'),
 ('Only', 'NN'),
 ('a', 'NN'),
 ('relative', 'NN'),
 ('handful', 'NN'),
 ('of', 'NN'),
 ('such', 'NN'),
 ('reports', 'NNS'),
 ('was', 'NNS'),
 ('received', 'VBD'),
 ('...', 'NN'),
 ('.', 'NN'),
 ('the', 'NN'),
 ('jury', 'NN'),
 ('said', 'NN'),
 ('.', 'NN'),
 ('...', 'NN'),
 ('considering', 'VBG'),
 ('the', 'NN'),
 ('widespread', 'NN'),
 ('interest', 'NN'),
 ('in', 'NN'),
 ('the', 'NN'),
 ('election', 'NN'),
 ('.', 'NN'),
 ('the', 'NN'),
 ('number', 'NN'),
 ('of', 'NN'),
 ('voters', 'NNS'),
 ('and', 'NN'),
 ('the', 'NN'),
 ('size', 'NN'),
 ('of', 'NN'),
 ('this', 'NNS'),
 ('city', 'NN'),
 ('...', 'NN'),
 ('.', 'NN')]
```

C. Unigram Tagger

The UnigramTagger assigns tags based on the most frequent tag for each word in the training data. It's a basic but effective tagger that often performs well, especially for languages with relatively free word order.

Code :

```
import nltk
nltk.download('brown')
nltk.download('punkt')
from nltk.corpus import brown
from nltk import UnigramTagger
brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
tags = unigram_tagger.tag(brown_sents[2007])
print("\nDisplaying the tags from brown sents : \n", tags)
evaluation = unigram_tagger.evaluate(brown_tagged_sents)
print("\nEvaluation of brown tagged sents : ", evaluation)
```

Output :

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

Displaying the tags from brown sents :
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments', 'NNS'), ('are', 'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'), ('type', 'NN'), ('', ' '),
<ipython-input-24-3468a80f160a>:11: DeprecationWarning:
  Function evaluate() has been deprecated. Use accuracy(gold)
  instead.
  evaluation = unigram_tagger.evaluate(brown_tagged_sents)

Evaluation of brown tagged sents : 0.9349006503968017
```

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

Code :

```
from __future__ import with_statement #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('wordfile.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
1
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
```

```
MENU
-----
1 . Hash tag segmentation
2 . URL segmentation
enter the input choice for performing word segmentation
2
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 3

Aim:

- a. Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.
- b. Study lemmas, hyponyms, hypernyms, entailments,
- c. Write a program using python to find synonym and antonym of word "active" using Wordnet
- d. Compare two nouns
- e. Handling stopword.

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

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

Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

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

WordNet is a large lexical database of English, developed at Princeton University. It groups English words into sets of synonyms called synsets, provides short definitions and usage examples, and records a number of relations among these synonym sets or their members.

Synsets : Synsets are sets of cognitive synonyms (synonyms that express the same concept). Each synset represents a distinct concept and contains a group of words that are interchangeable in some context.

Definitions : Each synset comes with a short definition that explains the concept it represents.

Examples : WordNet provides usage examples for synsets to illustrate how the words in the synset can be used in context.

Antonyms : WordNet includes antonyms for some words. Antonyms are words with opposite meanings.

Code :

```
import nltk
nltk.download('wordnet')

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 :

```
⇒ [nltk_data] Downloading package wordnet to /root/nltk_data...
[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, entailments

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

Hyponyms: More specific terms. Both come to picture as Synsets are organized in a structure similar to that of an inheritance tree. This tree can be traced all the way up to a root hypernym. Hypernyms provide a way to categorize and group words based on their similarity to each other.

Hypernym extraction is a crucial task for semantically motivated NLP tasks such as taxonomy and ontology learning, textual entailment or paraphrase identification. ... Our experiments confirm that both syntactic and definitional information play a crucial role in the hypernym extraction task.

Textual entailment (TE) in natural language processing is a directional relation between text fragments. The relation holds whenever the truth of one text fragment follows from another text. In the TE framework, the entailing and entailed texts are termed text (t) and hypothesis (h), respectively.

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_hyponyms(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', 'info
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('comp
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', 'par
[Synset('vehicle.n.01')]
```

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

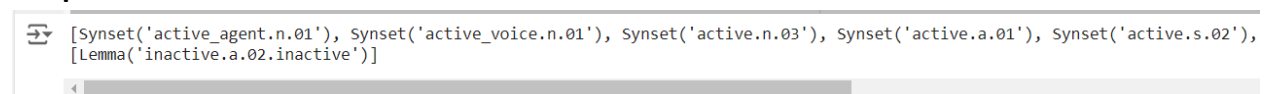
To find synonyms and antonyms of the word "active" using WordNet in Python, we can utilize the Natural Language Toolkit (nltk) library, which provides a simple interface to the WordNet lexical database.

Code :

```
from nltk.corpus import wordnet
# Retrieve synsets for the word "active"
# retrieves all the synsets (sets of cognitive synonyms) for the word "active". A synset contains a
group of synonyms that share a common meaning. The result is a list of synsets.
print( wordnet.synsets("active"))

# Find antonyms for a specific lemma
print(wordnet.lemma('active.a.01.active').antonyms()) #This retrieves a specific lemma (a word
form with a specific sense) from the synset active.a.01. Here, 'active' is the lemma name, 'a'
stands for adjective, and '01' is the sense number.
```

Output :



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

d. Compare two nouns

Code :

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet

# Get synsets for 'football' and 'soccer'
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(), ']') #Each synset has a name (e.g.,
Synset('football.n.01')), part of speech (e.g., noun 'n'), and a 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 into each other's goal ]
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

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

e. Handling stopwords.

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

NLTK provides a default set of stop words for English, which can be useful for various natural language processing tasks like text classification, sentiment analysis, and information retrieval. These stop words are common words that are often removed from text because they typically do not carry significant meaning.

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] Unzipping corpora/stopwords.zip.
['Yashesh', 'likes', 'play', 'football', ',', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'fond', 'tennis', '.']
['Yashesh', 'likes', 'football', ',', 'however', 'not', 'fond', 'tennis', '.']
```

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

Gensim, a popular library for topic modeling and natural language processing tasks, also provides a set of default stop words. Similar to NLTK, you can add or remove words from Gensim's default stop words list to customize it according to your needs.

Code :

```
import gensim
from gensim.parsing.preprocessing import remove_stopwords
text = "Yashesh likes to play football, however he is not too fond of tennis."
print('Original text: ', text)
filtered_sentence = remove_stopwords(text)
print('\n After removing Stop words: ', filtered_sentence)

#The below line retrieves the default set of stop words from Gensim and prints them.
all_stopwords = gensim.parsing.preprocessing.STOPWORDS
print('\n Stop words in Gensim: ', 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'])) # adding 'likes' and 'play' to stop words  
  
text = "Yashesh likes to play football, however he is not too fond of tennis."  
text_tokens = word_tokenize(text)  
  
# Filter out the tokens that are in the new stop words set (including 'likes' and 'play')  
tokens_without_sw = [word for word in text_tokens if not word in  
all_stopwords_gensim]  
print("\n After adding likes & play in stop word collection: ",tokens_without_sw)
```

Output :

```
Original text: Yashesh likes to play football, however he is not too fond of tennis.  
After removing Stop words: Yashesh likes play football, fond tennis.  
Stop words in Gensim: frozenset({'two', 'nobody', 'seemed', 'though', 'yourselves', 'still', 'another', 'into', 't  
After adding likes & play in stop word collection: ['Yashesh', 'football', ',', 'fond', 'tennis', '.']
```

```
# Remove the word 'not' from the existing set of Gensim stop words  
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)  
  
# Filter out the tokens again with 'not' removed from stop words set  
tokens_without_sw = [word for word in text_tokens if not word in  
all_stopwords_gensim]  
print(tokens_without_sw)
```

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

Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

Stop words are commonly used words in a language, such as "the," "is," "and," or "in," which do not contribute much to the overall meaning of the text. Stop word removal can improve the accuracy and efficiency of text analysis, text mining, and natural language processing tasks.

Code :

```
#python -m spacy download en_core_web_sm
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', '.']
```

Practical 04

Aim: Text Tokenization

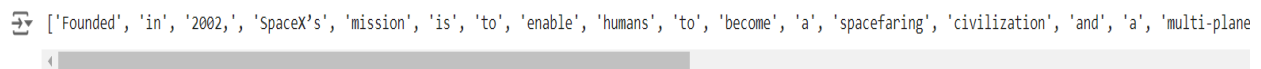
a. Tokenization using Python's split() function

Tokenization is the process of splitting text into smaller units called tokens, which can be words, phrases, or even characters. One of the simplest methods to perform tokenization in Python is by using the split() function. This function divides a string into a list of substrings based on a specified delimiter. By default, split() uses whitespace as the delimiter.

Code:

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
# Splits at space
a=text.split()
print(a)
```

Output :



```
['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-plane']
```

b. Tokenization using Regular Expressions (RegEx)

Tokenization using Regular Expressions (RegEx) is a powerful and flexible method for splitting text into tokens based on complex patterns. Regular expressions allow you to define patterns for identifying the tokens you want to extract from a string. This method is particularly useful for handling a variety of delimiters, removing unwanted characters, and performing more complex text processing tasks.

Code:

```
import re
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multiplanet species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed liquid-fuel launch vehicle to orbit the Earth."""
tokens = re.findall("[\w]+", text)
print(tokens)
```

Output :

```
>>> ===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract4b.py =====
['Founded', 'in', '2002', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'human
s', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multiplanet
', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city', 'on', 'Mars',
'In', '2008', 'SpaceX', 's', 'Falcon', '1', 'became', 'the', 'first', 'privatel
y', 'developed', 'liquid', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'E
arth']
>>>
```

c. Tokenization using NLTK

The Natural Language Toolkit (NLTK) is a comprehensive library for natural language processing in Python. It provides powerful tools for text processing, including tokenization, which is the process of splitting text into individual tokens (words, sentences, etc.). NLTK offers several built-in tokenizers that handle various aspects of tokenization.

Code :

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring
civilization and a multi-
planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first
privately developed
liquid-fuel launch vehicle to orbit the Earth."""
a=word_tokenize(text)
print(a)
```

Output :

```
>>> ===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract4c.py =====
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
['Founded', 'in', '2002', ',', 'SpaceX', ',', 's', 'mission', 'is', 'to', 'enabl
e', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'm
ultiplanet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on',
'Mars', '.', 'In', '2008', ',', 'SpaceX', ',', 's', 'Falcon', '1', 'became', 'th
e', 'first', 'privately', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to',
'orbit', 'the', 'Earth', '.']
>>>
```

d. Tokenization using the spaCy library

spaCy is a powerful and efficient library for natural language processing in Python. It provides a variety of advanced NLP capabilities, including tokenization, which is the process of breaking down text into individual tokens (such as words and punctuation marks). spaCy's tokenization is particularly robust, as it handles a wide range of linguistic features out-of-the-box.

Code :

```
#pip install -U spacy
#python -m spacy download en
from spacy.lang.en import English
# Load English tokenizer, tagger, parser, NER and word vectors
nlp = English()
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring
civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first
privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# "nlp" Object is used to create documents with linguistic annotations.
my_doc = nlp(text)
# Create list of word tokens
token_list = []
for token in my_doc:
    token_list.append(token.text)
token_list
print(token_list)
```

Output :

```
>>>
= RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract4d.py
['Founded', 'in', '2002', ',', 'SpaceX', 's', 'mission', 'is', 'to', 'enable',
'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi',
'-', 'planet', '\n', 'species', 'by', 'building', 'a', 'self', '-', 'sustaini
ng', 'city', 'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', 's', 'Falcon', '1',
'became', 'the', 'first', 'privately', 'developed', '\n', 'liquid', '-', 'fuel',
'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
>>>|
```

e. Tokenization using Keras.

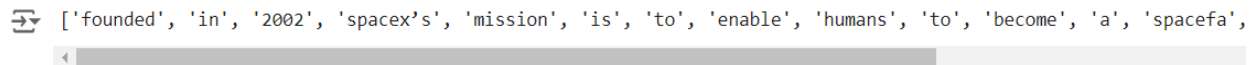
Keras, a popular deep learning library, includes utilities for text preprocessing, including tokenization. The `keras.preprocessing.text` module provides the `Tokenizer` class, which can convert text into sequences of tokens suitable for training machine learning models.

Code :

```
from keras.preprocessing.text import text_to_word_sequence
# define
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefa
ring civilization and a multi-planet
species by building a selfsustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first
privately deve
```

```
loped
liquid-fuel launch vehicle to orbit the Earth. """
# tokenize
result = text_to_word_sequence(text)
print(result)
```

Output :



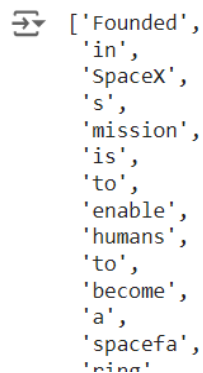
f. Tokenization using Gensim.

Gensim is a robust library for topic modeling and document similarity analysis in Python, and it includes utilities for text preprocessing, including tokenization. Gensim's gensim.utils module provides simple and effective functions for tokenizing text.

Code :

```
from gensim.utils import tokenize
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefa
ring civilization and a multi-planet
species by building a selfsustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first
privately deve
loped
liquid-fuel launch vehicle to orbit the Earth. """
list(tokenize(text))
```

Output :



Practical No 5

Aim: Illustrate part of speech tagging.

- Part of speech Tagging and chunking of user defined text.
- Named Entity recognition of user defined text.
- Named Entity recognition with diagram using NLTK corpus – treebank

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

Theory: POS Tagging (Parts of Speech Tagging) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called grammatical tagging.

Code :

```
import nltk
from nltk import pos_tag
from nltk import RegexpParser
nltk.download()
text="This is practical no 6".split()
print("After Split:",text)
nltk.download('averaged_perceptron_tagger')
# averaged_perceptron_tagger is used for tagging words with their parts of speech (POS)
tokens_tag = pos_tag(text)
print("After Token:",tokens_tag)
patterns= """"mychunk:{<NN.?>*<VBD.?>*<JJ.?>*<CC>?}""""
chunker = RegexpParser(patterns)
print("After Regex:",chunker)
output = chunker.parse(tokens_tag)
print("After Chunking",output)
```

Output:

```
>>>
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract6a.py =====
After Split: ['This', 'is', 'practical', 'no', '6']
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
After Token: [('This', 'DT'), ('is', 'VBZ'), ('practical', 'JJ'), ('no', 'DT'),
('6', 'CD')]
After Regex: chunk.RegexpParser with 1 stages:
RegexpChunkParser with 1 rules:
  <ChunkRule: '<NN.?>*<VBD.?>*<JJ.?>*<CC>?*>'>
After Chunking (S This/DT is/VBZ (mychunk practical/JJ) no/DT 6/CD)
>>>
```


b. Named Entity recognition of user defined text.

Theory: Named-entity recognition is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. European is NORD (nationalities or religious or political groups), Google is an organization, \$5.1 billion is monetary value and Wednesday is a date object. They are all correct.

Code :

```
import nltk
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()
ex = 'European authorities fined Google a record $5.1 billion on Wednesday for abusing its power
in the mobile phone market and ordered the company to alter its practices'
ex = nlp('European authorities fined Google a record $5.1 billion on Wednesday for abusing its
power in the mobile phone market and ordered the company to alter its practices')
print([(X.text, X.label_) for X in ex.ents])
```

Output :

```
→ [('European', 'NORP'), ('Google', 'ORG'), ('$5.1 billion', 'MONEY'), ('Wednesday', 'DATE')]
```

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

Theory: The maxent_ne_chunker contains two pre-trained English named entity chunkers trained on an ACE corpus.

Code :

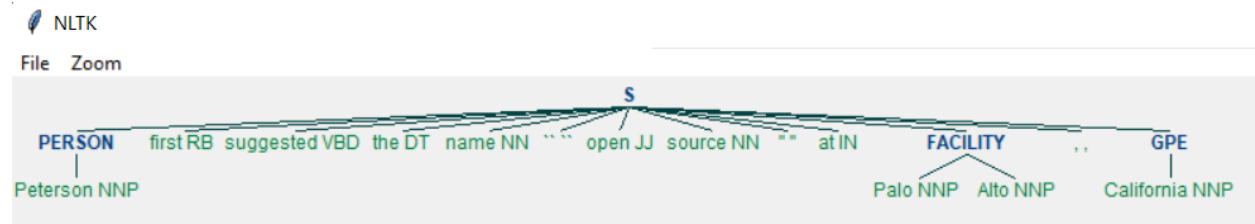
```
sentence = 'Peterson first suggested the name "open source" at Palo Alto, California'
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
words = nltk.word_tokenize(sentence)
pos_tagged = nltk.pos_tag(words)
nltk.download('maxent_ne_chunker')
```

```
nltk.download('words')
ne_tagged = nltk.ne_chunk(pos_tagged)
print("NE tagged text:")
print(ne_tagged)
print()
print("Recognized named entities:")
for ne in ne_tagged:
    if hasattr(ne, "label"):
        print(ne.label(), ne[0:])
ne_tagged.draw()
```

Output :

```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract6c.py
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package words is already up-to-date!
NE tagged text:
(S
  (PERSON Peterson/NNP)
  first/RB
  suggested/VBD
  the/DT
  name/NN
  ``/``
  open/JJ
  source/NN
  ''/'
  at/IN
  (FACILITY Palo/NNP Alto/NNP)
  ,/,
  (GPE California/NNP))

Recognized named entities:
PERSON [('Peterson', 'NNP')]
FACILITY [('Palo', 'NNP'), ('Alto', 'NNP')]
GPE [('California', 'NNP')]
```



Practical No 6

Aim:

- Define grammar using nltk. Analyze a sentence using the same.
- Accept the input string with Regular expression of FA: 101^+
- Accept the input string with Regular expression of FA: $(a+b)^*bba$
- Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

a. Define grammar using nltk. Analyze a sentence using the same.

To define grammar using NLTK (Natural Language Toolkit), we typically create a context-free grammar (CFG) using a syntax called Backus-Naur Form (BNF). Here's a simple example of defining a CFG for a basic English sentence:

Code :

```
# using Recursive Descent Parser
import nltk
grammar1 = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park"
P -> "in" | "on" | "by" | "with"
""")
sent = "Mary saw Bob".split()
rd_parser = nltk.RecursiveDescentParser(grammar1)
for tree in rd_parser.parse(sent):
    print(tree)
```



The diagram shows a parse tree for the sentence "Mary saw Bob". The root node is S, which branches into NP (Mary) and VP. VP branches into V (saw) and NP (Bob). The tree is displayed in a graphical format with nodes and edges.

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

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("Token: ",all_tokens)  
    parser = nltk.ChartParser(grammar1)  
    for tree in parser.parse(all_tokens):  
        print(tree)  
    #tree.draw()
```

```
➡ [nltk_data] Downloading package punkt to /root/nltk_data...  
[nltk_data] Unzipping tokenizers/punkt.zip.  
Token: ['Book', 'that', 'flight']  
(S (VP (VP Book) (NP (Det that) (NP flight))))
```

b. Accept the input string with Regular expression of FA: 101+

allows you to accept input strings that follow the regular expression pattern 101+

Code :

```
def FA(s):  
    # if the length is less than 3 then it can't be accepted, Therefore end the process.  
    if len(s) < 3:  
        return "Rejected"
```

```
# first three characters are fixed. Therefore checking them using index
if s[0] == '1': # Check if the first character is '1'
    if s[1] == '0': # Check if the second character is '0'
        if s[2] == '1': # Check if the third character is '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)): # Loop through the remaining characters
                if s[i] != '1': # If any character is not '1'
                    return "Rejected" # Reject the string
            return "Accepted" # If all characters after the first three are '1', accept the string

# If any of the conditions fail, return "Rejected"
return "Rejected"

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

# Loop through each input string and print whether it is accepted or rejected
# for i in inputs:
#     print(FA(i))

for i in inputs:
    print(f"Input: {i} - {FA(i)}")
```

Output :

```
➡ Input: 1 - Rejected
   Input: 10101 - Rejected
   Input: 101 - Accepted
   Input: 10111 - Accepted
   Input: 01010 - Rejected
   Input: 100 - Rejected
   Input:  - Rejected
   Input: 10111101 - Rejected
   Input: 1011111 - Accepted
```

c. Accept the input string with Regular expression of FA: (a+b)*bba.

code allows you to accept input strings that follow the regular expression pattern (a+b)*bba.

The regular expression (a+b)*bba can be broken down as follows:

(a+b)*: Zero or more occurrences of either 'a' or 'b'.

bba: The string must end with the sequence 'bba'.

Code :

def FA(s):

size = 0

Scan the 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 its length, it should be at least 3

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 conditions are 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

List of input strings to test

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

Iterate over the input strings and print the result of FA for each

for i in inputs:

print(f"Input: {i} - {FA(i)}")

Output :

```
↔ Input: bba - Accepted
   Input: ababbbba - Accepted
   Input: abba - Accepted
   Input: abb - Rejected
   Input: baba - Rejected
   Input: bbb - Rejected
   Input: - Rejected
```

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

Deductive Chart Parsing is a method for parsing sentences using a chart data structure, which stores partial parse trees as they are constructed.

Code :

```
import nltk
from nltk import CFG, ChartParser, word_tokenize
```

```
# Define the context-free grammar
```

```
grammar1 = 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'
```

```
""")
```

```
# Define the sentence and tokenize it
```

```
sentence = "I saw a bird in my balcony"
```

```
all_tokens = word_tokenize(sentence)
```

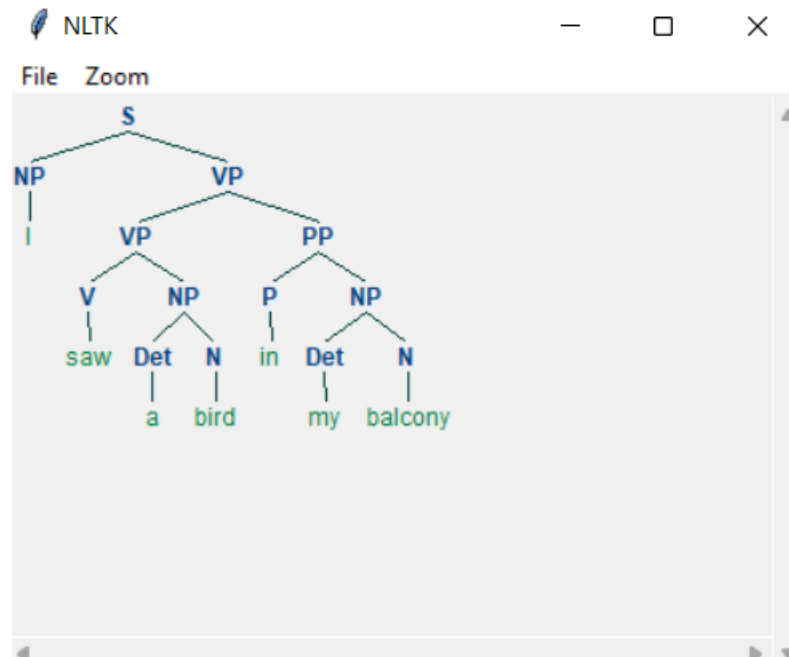
```
print(all_tokens)
```

```
# Initialize the chart parser with the grammar
parser = ChartParser(grammar1)
```

```
# Parse the tokens and print the parse trees
for tree in parser.parse(all_tokens):
    print(tree)
    #tree.draw()
```

Output :

```
↔ ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
(S
  (NP I)
  (VP
    (VP (V saw) (NP (Det a) (N bird)))
    (PP (P in) (NP (Det my) (N balcony)))))
(S
  (NP I)
  (VP
    (V saw)
    (NP (Det a) (N bird) (PP (P in) (NP (Det my) (N balcony))))))
```



Practical No 7

PART A : STUDY PORTER STEMMER, LANCASTER STEMMER, REGEXP STEMMER AND SNOWBALL STEMMER

a. Porter Stemmer

The Porter Stemmer, developed by Martin Porter in 1980, is a widely used algorithm in natural language processing (NLP) for reducing words to their root forms, or stems. This process helps in standardizing words and improving the performance of text-based applications, such as search engines and information retrieval systems.

Code :

```
# Program to implement Porter Stemmer
print("Name : Trupti Bhostekar \tRoll No : 2")
import nltk
nltk.download('punkt')
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
# Defining the stemmer
porter = PorterStemmer()
# Taking words which have a similar stem
terms = ["gene", "genes", "genesis", "genetic", "generic", "general"]

# Performing stemming using porter stemmer on words
print("\n1. Performing porter stemming on the words")
for each_term in terms:
    print(porter.stem(each_term))

# Taking a sentence
sentence = "Heya Reeba, do you know it is important to be pythononly while pythoning with
pythonlanguage. Stay being a pythoner"

# Performing stemming using porter stemmer on a sentence
print("\n2. Performing porter stemming on a sentence")
words = word_tokenize(sentence, language = 'english')
for each_word in words:
    print(porter.stem(each_word))
```

Output :

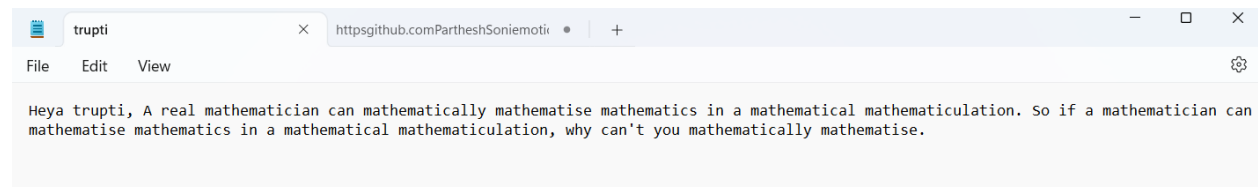
```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract8a.py
Name : Trupti Bhostekar      Roll No : 2
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!

1. Performing porter stemming on the words
gene
gene
genesi
genet
gener
gener

2. Performing porter stemming on a sentence
heya
reeba
'
do
you
know
it
is
import
to
be
pythonli
while
python
with
pythonlanguag
.
stay
be
a
python
>>>|
```

b. Lancaster Stemmer

The Lancaster Stemmer, also known as the Paice-Husk Stemmer, is another widely used stemming algorithm in natural language processing (NLP). It was developed by Chris Paice and Gareth Husk in 1990 and is known for its aggressive stemming approach, which often results in shorter stems compared to other algorithms like the Porter Stemmer.



Code :

```
# Program to implement Lancaster Stemmer
print("Name : Trupti Bhostekar \tRoll No : 2")
import nltk
nltk.download('punkt')
from nltk.stem import LancasterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
# Defining the stemmer
lancaster = LancasterStemmer()
# Taking words which have a similar stem
```

```
terms = ["enjoy", "enjoying", "enjoyed", "enjoyable", "enjoyment", "enjoyful"]
# Performing stemming using lancaster stemmer on words
print("\n1. Performing lancaster stemming on the words")
for each_term in terms:
    print(lancaster.stem(each_term))
# Taking a sentence
sentence = "Heya trupti, Why is it so with the dancers that when dancers dance, they dance as if they are dancing in the air?"
# Performing stemming using lancaster stemmer on a sentence
print("\n2. Performing lancaster stemming on a sentence")
words = word_tokenize(sentence, language = 'english')
for each_word in words:
    print(lancaster.stem(each_word))
# Performing stemming using lancaster stemmer on a text file
print("\n3. Performing lancaster stemming on a text file - one sentence at a time")
# Treating the text file as a collection of sentences
reeba_file = open("trupti.txt")
my_lines_list = reeba_file.readlines()
# Accessing one line at a time from the text file
words = word_tokenize(my_lines_list[0], language = 'english')
for each_word in words:
    print(lancaster.stem(each_word))
```

Output :

```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract8b.py =====
Name : Trupti Bhostekar      Roll No : 2
[nltk_data] Downloading package punkt to
[nltk_data]   C:/Users/nishant/AppData/Roaming/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

1. Performing lancaster stemming on the words
enjoy
enjoy
enjoy
enjoy
enjoy
enjoy
enjoy

2. Performing lancaster stemming on a sentence
hey
trupt
,
why
is
it
so
with
the
dant
that
when
dant
dant
,
they
dant
as
if
they
ar
in
the
air
?

3. Performing lancaster stemming on a text file - one sentence at a time
hey
trupt
,
a
real
mathem
can
mathem
mathem
mathem
mathem
in
a
mathem
mathematic
.
so
if
a
mathem
can
mathem
mathem
in
a
mathem
mathematic
,
why
ca
n't
you
mathem
mathem
.
```

c. Snowball Stemmer

The Snowball Stemmer, also known as the Porter2 Stemmer, is an advanced and more versatile version of the original Porter Stemmer, created by Martin Porter. Introduced in 2001, it provides more consistent and maintainable stemming rules and supports multiple languages, making it a robust tool in natural language processing (NLP).

Code :

```
# Program to implement Snowball Stemmer
print("Name : Trupti Bhostekar \tRoll No : 2")
import nltk
nltk.download('punkt')
from nltk.stem.snowball import SnowballStemmer
# Defining the stemmer
snowball_english = SnowballStemmer("english")
snowball_dutch = SnowballStemmer("dutch")
# Performing stemming on one word
print("\n1. Performing snowball stemming one word")
word = snowball_english.stem("Vibing")
print(word)
# Taking a list of english words
terms = ["trupti", "cheerful", "bravery", "drawing", "satisfactorily", "publisher", "painful",
"hardworking",
"keys"]
# Performing stemming using snowball stemmer on words
print("\n2. Performing snowball stemming on a set of english language words")
for each_term in terms:
    print(snowball_english.stem(each_term))
# Taking a list of dutch words
# trupti = trupti, bessen = berries, vriendelijkheid = friendliness, hobbelig = bumpy
terms2 = ["trupti", "bessen", "vriendelijkheid", "hobbelig"]
print("\n3. Performing snowball stemming on a set of dutch language words")
for each_term in terms2:
    print(snowball_dutch.stem(each_term))
```

Output :

```
///
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract8c.py =====
Name : Trupti Bhostekar      Roll No : 2
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

1. Performing snowball stemming one word
vibing

2. Performing snowball stemming on a set of english language words
trupti
cheerful
bravery
drawing
satisfactorily
publisher
painful
hardwork
key

3. Performing snowball stemming on a set of dutch language words
trupti
bess
vriendelijk
hobbel
>>>
```

d. RegExp Stemmer

The RegExp Stemmer, short for Regular Expression Stemmer, is a simpler and more flexible stemming approach that uses regular expressions (regex) to define rules for reducing words to their root forms. This method relies on pattern matching and replacement techniques, making it highly customizable for specific needs. Unlike more structured algorithms like Porter or Snowball, the RegExp Stemmer is straightforward but requires careful crafting of regex patterns to ensure effectiveness.

Code :

```
# Program to implement RegExp Stemmer
print("Name : Trupti Bhostekar \tRoll No : 2")
import nltk
nltk.download('punkt')
from nltk.stem import RegexpStemmer
# Defining the stemmer
regex = RegexpStemmer('ing$|s$|e$|able$|ment$|less$|ly$', min=4)
# Performing stemming one word
print("\n1. Performing regex stemming on one word at a time")
print(regex.stem('cars'))
print(regex.stem('bee'))
print(regex.stem('compute'))
# Taking a list of word
terms = ["truptis", "stemming", "mentally", "ease", "rockstar", "frictionless",
"management", "flowers",
"advisable"]
# Performing stemming using lancaster stemmer on words
```

```
print("\n2. Performing regexp stemming on a list of words")
for each_term in terms:
    print(regexp.stem(each_term))
```

Output :

```
>>> ===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract8d.py =====
Name : Trupti Bhostekar      Roll No : 2
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

1. Performing regexp stemming on one word at a time
car
bee
comput

2. Performing regexp stemming on a list of words
trupti
stemm
mental
eas
rockstar
friction
manage
flower
advis
...|
```

PART B : STUDY WORDNET LEMMATIZER

The WordNet Lemmatizer is a tool used in natural language processing (NLP) to reduce words to their base or dictionary form, known as the lemma. Unlike stemming, which often produces root forms by simply stripping suffixes, lemmatization considers the context and grammatical structure of the word to ensure the resulting lemma is a valid word.

Code :

```
# Program to implement WordNet Lemmatizer
print("Name : Trupti bhostekar \tRoll No : 2")
import nltk
nltk.download('wordnet')
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
# Initializing the Wordnet Lemmatizer
wordnet = WordNetLemmatizer()
# Performing WordNet lemmatization on single Words
print("\n1. Performing WordNet lemmatization on single Words")
print(wordnet.lemmatize("corpora"))
print(wordnet.lemmatize("best"))
```

```
print(wordnet.lemmatize("geese"))
print(wordnet.lemmatize("feet"))
print(wordnet.lemmatize("cacti"))
#Performing WordNet lemmatization on a sentence

print("\n2. Performing WordNet lemmatization on a sentence")
# Taking a sentence
sentence = "Heyaa trupti, how are you doing? Keep digging in for the sentences to observe lemmatization!"
# Tokenizing i.e. splitting the sentence into words
list_words = nltk.word_tokenize(sentence)
print("\nConverting the sentence into a list of words")
print(list_words)
# Lemmatize list of words and join
final = ' '.join([wordnet.lemmatize(each_word, pos = 'v') for each_word in list_words])
print("\nAfter applying wordnet lemmatizer, the result is....")
print(final)
```

Output :

```
===== RESTART: C:/Users/nishant/Documents/Trupti/sem4/NLP/pract8e.py =====
Name : Trupti bhostekar      Roll No : 2
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\nishant\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

1. Performing WordNet lemmatization on single Words
corpus
best
goose
foot
cactus

2. Performing WordNet lemmatization on a sentence

Converting the sentence into a list of words
['Heyaa', 'trupti', ',', 'how', 'are', 'you', 'doing', '?', 'Keep', 'digging', 'in', 'for', 'the', 'sentences',
'to', 'observe', 'lemmatization', '!']

After applying wordnet lemmatizer, the result is....
Heyaa trupti , how be you do ? Keep dig in for the sentence to observe lemmatization !
>>>
```

Practical no 8

Aim: Implement Naive Bayes classifier.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features. Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. We have explored the idea behind Gaussian Naive Bayes along with an example. Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem.

Code :

```
import sklearn
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
breastcancer = datasets.load_breast_cancer()
print("\nFeatures of breastcancer dataset : ", breastcancer.feature_names)
print("\nLabels of breastcancer dataset : ", breastcancer.target_names)
print("\nShape of breastcancer dataset : ", breastcancer.data.shape)
print("\n-----")
R = breastcancer.data
T = breastcancer.target

# Splitting the dataset into training set and testing set
Rtrain, Rtest, Ttrain, Ttest = train_test_split(R, T, test_size = 0.2, random_state = 0)

# 1. Using the Gaussian Naive Bayes Classifier
gauss = GaussianNB()

# Training the Gaussian Naive Bayes model using training set
```



```
gauss.fit(Rtrain,Ttrain)

# Making predictions using the test set
pred = gauss.predict(Rtest)

# Generating classification report of the Gaussian Naive Bayes Model
gcr = classification_report(Ttest,pred)
print("\nClassification Report gaussian : \n", gcr)

# Generating confusion matrix of the Gaussian Naive Bayes Model
gcm = confusion_matrix(Ttest, pred)
print("\nConfusion matrix gaussian : \n", gcm)

# Evaluating the naive bayes classifier on the basis of accuracy metric
accuracy = accuracy_score(Ttest, pred)
print("\nAccuracy : ", accuracy * 100)
```

Output :



```
Features of breastcancer dataset : ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
'mean smoothness' 'mean compactness' 'mean concavity'
'mean concave points' 'mean symmetry' 'mean fractal dimension'
'radius error' 'texture error' 'perimeter error' 'area error'
'smoothness error' 'compactness error' 'concavity error'
'concave points error' 'symmetry error' 'fractal dimension error'
'worst radius' 'worst texture' 'worst perimeter' 'worst area'
'worst smoothness' 'worst compactness' 'worst concavity'
'worst concave points' 'worst symmetry' 'worst fractal dimension']

Labels of breastcancer dataset : ['malignant' 'benign']

Shape of breastcancer dataset : (569, 30)
```



```
Classification Report gaussian :
              precision    recall  f1-score   support

         0               0.91       0.91       0.91         47
         1               0.94       0.94       0.94         67

 accuracy               0.93
 macro avg              0.93
weighted avg              0.93
```



```
Confusion matrix gaussian :
[[43  4]
 [ 4 63]]

Accuracy : 92.98245614035088
```