### **INDEX**

| No. | Practical | Date | Sign |
| --- | --- | --- | --- |
| 1 (a) | Install gtts, pyttsx3, speake3, pyaudio & speech\_recognition |  |  |
| 1 (b) | Convert the given text to speech |  |  |
| 1 (c) | Convert audio file speech to text |  |  |
| 2 (a) | Text tokenization using python split functions |  |  |
| 2 (b) | Tokenization using regex (regular expression) |  |  |
| 2 (c) | Tokenization using nltk |  |  |
| 2 (d) | Tokenization using SpaCy library |  |  |
| 2 (e) | To implement and demonstrate the process of tokenization using the Gensim |  |  |
| 3 | Implement a sentiment analysis |  |  |
| 4 (a) | Word tokenization in Hindi |  |  |
| 4 (b) | Identify the Indian language of a text |  |  |
| 5 (a) | Part of speech tagging and chunking of user defined text |  |  |
| 5 (b) | Named entity recognition of user defined text |  |  |
| 5 (c) | Named entity recognition with diagram using nltk corpus - treebank |  |  |
| 6 | Wordnet, synset, hypernyos (abstract), hyponyms (specific), pos tag in wordnet |  |  |
| 7 (a) | Python Program to Find Synonyms and Antonyms of Active |  |  |
| 7 (b) | Comparing Two Nouns Using Wu-Palmer Similarity |  |  |
| 7 (c) | Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words |  |  |
| 7 (d) | Using Gensim Adding and Removing Stop Words using custom Stop Words |  |  |
| 8 | Python program to find & match patterns using Regex |  |  |
| 9 (a) | Define grammer using nltk. Analyze a sentence using the using pretty tree |  |  |
| 9 (b) | Accept the input string with Regular expression of FA: 101+ |  |  |
| 9 (c) | Accept the input string with Regular expression of FA: (a|b)\*bba |  |  |
| 10 (a) | Implementing the program of Multiword |  |  |
| 10 (b) | Implementing the program of Multiword using NER |  |  |

### 

### 1 (b) - 1: convert the given text to speech (gTTs)

from gtts import gTTS

import os

mytext = 'This is MSC DS Part 2!'

language = 'en'

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

myobj.save("welcome.mp3")

os.system("start welcome.mp3")

### 1 (b) - 2: convert the given text to speech (pyttsx)

import pyttsx3

def onStart():

print('starting')

def onWord(name, location, length):

print('word', name, location, length)

def onEnd(name, completed):

print('finishing', name, completed)

engine = pyttsx3.init()

engine.setProperty('rate', 50)

engine.setProperty('volume', 0.9)

engine.connect('started-utterance', onStart)

engine.connect('started-word', onWord)

engine.connect('finished-utterance', onEnd)

sen = 'This is a class of MSCDS PART 2'

engine.say(sen)

engine.runAndWait()

### 1 (c): convert audio file speech to text (speech\_recognition)

import speech\_recognition as sr

def live\_speech\_to\_text():

# Initialize the recognizer

recognizer = sr.Recognizer()

microphone = sr.Microphone()

try:

print("Adjusting for ambient noise... Please wait.")

with microphone as source:

# Adjust for background noise

recognizer.adjust\_for\_ambient\_noise(source, duration=2)

print("Listening... Speak into the microphone.")

while True:

try:

# Capture audio

print("\nSay something:")

audio = recognizer.listen(source)

# Recognize speech using Google Web Speech API

print("Processing...")

text = recognizer.recognize\_google(audio)

print(f"You said: {text}")

except sr.UnknownValueError:

print("Sorry, I could not understand the audio.")

except sr.RequestError as e:

print(f"Could not request results; {e}")

except KeyboardInterrupt:

print("\nProgram terminated by user.")

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

live\_speech\_to\_text()

### 2 (a): text tokenization using python split functions

****

### 2 (b): tokenization using regex (regular expression)

****

### 2 (c): tokenization using nltk

****

### 2 (d): tokenization using SpaCy library

****

### 3a: Simple sentiment Analysis

**Code:**

import nltk

nltk.download('vader\_lexicon')

from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

sentiment = sia.polarity\_scores("I absolutely donot love this product! It's not amazing.")

print(sentiment)

**Output:**

{'neg': 0.222, 'neu': 0.433, 'pos': 0.345, 'compound': 0.4016}

### 3b: Sentiment Analysis using some of the Model

**Code:**

import nltk

import random

from nltk.corpus import movie\_reviews

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

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import make\_pipeline

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

nltk.download('movie\_reviews')

document = [(list(movie\_reviews.words(fileid)), category)

for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

random.shuffle(document)

text = [" ".join(words) for words, label in document]

labels = [label for words, label in document]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(text, labels, test\_size=0.2, random\_state=42)

model = make\_pipeline(CountVectorizer(), TfidfTransformer(), MultinomialNB())

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

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

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

**Output:**

Accuracy: 0.85

precision recall f1-score support

neg 0.88 0.83 0.85 213

pos 0.82 0.88 0.85 187

accuracy 0.85 400

macro avg 0.85 0.85 0.85 400

weighted avg 0.85 0.85 0.85 400

****

### 5 (a):Part of speech tagging and chunking of user defined text

**Code:**

import nltk

from nltk.tokenize import word\_tokenize

from nltk import pos\_tag

from nltk.chunk import ne\_chunk

nltk.download('punkt\_tab')

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

sentence = "Apple is looking at buying U.K. startup for $1 billion."

tokens = word\_tokenize(sentence)

tagged = pos\_tag(tokens)

print("POS Tags:", tagged)

grammar = "NP: {<DT>?<JJ>\*<NN>+}"

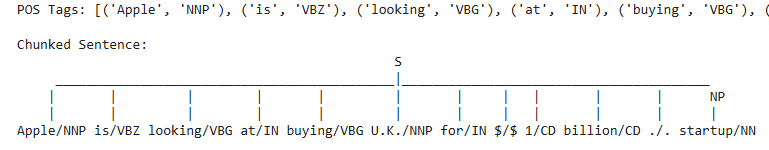
chunk\_parser = nltk.RegexpParser(grammar)

tree = chunk\_parser.parse(tagged)

print("\nChunked Sentence:")

tree.pretty\_print()

**Output:**

****

### 5 (b)**:** Named entity recognition of user defined text

**Code:**

from nltk import ne\_chunk

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

nltk.download('words')

sentence = "Barack Obama was born in Hawaii on August 4, 1961."

tokens = word\_tokenize(sentence)

tagged = pos\_tag(tokens)

named\_entities = ne\_chunk(tagged)

print("\nNamed Entities:", named\_entities)

**Output:**

Named Entities: (S (PERSON Barack/NNP) (PERSON Obama/NNP) was/VBD born/VBN in/IN (GPE Hawaii/NNP) on/IN August/NNP 4/CD ,/, 1961/CD ./.)

### 5 (c): named entity recognition with diagram using nltk corpus - treebank

**Code:**

from nltk.corpus import treebank

nltk.download('treebank')

sentence = treebank.sents()[0]

print("Sample Sentence from Treebank:", ' '.join(sentence))

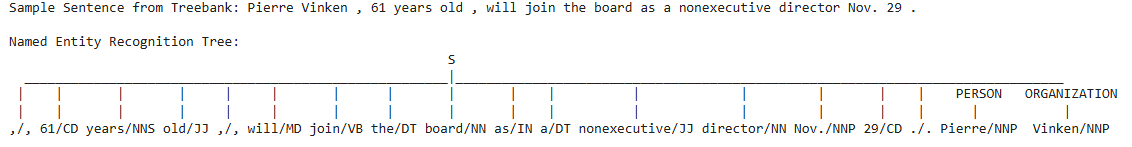
tagged = pos\_tag(sentence)

named\_entities = ne\_chunk(tagged)

print("\nNamed Entity Recognition Tree:")

named\_entities.pretty\_print()

**Output:**



### 6 (a)

**Code:**

import nltk

nltk.download('wordnet')

from nltk.corpus import wordnet

syn = wordnet.synsets('night')[0]

print ("\nSynset name : ", syn.name())

# Defining the word

print ("\nSynset meaning : ", syn.definition())

# list of phrases that use the word in context

print ("\nSynset example : ",syn.examples())

**Output:**

Synset name : night.n.01 Synset meaning : the time after sunset and before sunrise while it is dark outside Synset example : [] [nltk\_data] Downloading package wordnet to /root/nltk\_data… [nltk\_data] Package wordnet is already up-to-date!

### 6 (b)

**Code:**

import nltk

nltk.download('wordnet')

from nltk.corpus import wordnet

syn = wordnet.synsets('night')[0]

print ("\nSynset name : ", syn.name())

print ("\nSynset abstract term : ", syn.hypernyms())

print ("\nSynset root hypernerm : ", syn.root\_hypernyms())

**Output:**

Synset name : night.n.01 Synset abstract term : [Synset('time\_period.n.01')] Synset root hypernerm : [Synset('entity.n.01')]

### 

### 6 (c)

**Code:**

import nltk

nltk.download('wordnet')

from nltk.corpus import wordnet

syn = wordnet.synsets('night')[0]

print ("\nSyn tag : ", syn.pos())

syn = wordnet.synsets('doing')[0]

print ("Syn tag : ", syn.pos())

syn = wordnet.synsets('beautiful')[0]

print ("Syn tag : ", syn.pos())

syn = wordnet.synsets('quickly')[0]

print ("Syn tag : ",syn.pos())

**Output:**

Syn tag : n Syn tag : v Syn tag : a Syn tag : r

### 6 (d)

**Code:**

iimport nltk

nltk.download('wordnet')

from nltk.corpus import wordnet

words = ["how", "doing", "beautiful", "quickly"]

for word in words:

synsets = wordnet.synsets(word)

if synsets:

print(f"Word: {word}, Syn tag: {synsets[0].pos()}")

else:

print(f"Word: {word}, No synset found")

**Output:**

Word: how, No synset found Word: doing, Syn tag: v Word: beautiful, Syn tag: a Word: quickly, Syn tag: r

****

### 

### 

****

### 8: Python program to find & match patterns using Regex

****

**Code:**

import re

# Example text

text = """Hello! My name is griffith.

I was born on 1-11-3025 and my email is berserk@casca.com.

Visit my website at http://www.bandofhawk.com.

You can call me at +1-487-565-9634 or 923-426-5106.

Follow us on Twitter @OpenAI and use #MachineLearning for updates!

Elon Musk founded Tesla and SpaceX.

Contact us at support@example.com or sales@mywebsite.org"""

# Extract words

words = re.findall(r'\b\w+\b', text)

print("Words:", words)

# Extract emails

emails = re.findall(r'[\w.\_%+-]+@[\w.-]+\.[a-zA-Z]{2,}', text)

print("Emails:", emails)

# Extract usernames from emails

usernames = re.findall(r'\b(\w+)@\w+\.\w+\b', text)

print("Usernames:", usernames)

# Normalize text by replacing multiple spaces/newlines with a single space

normalized\_text = re.sub(r'\s+', ' ', text)

print("Normalized Text:", normalized\_text)

# Extract numbers (including phone numbers)

numbers = re.findall(r'\d+', text)

phone\_numbers = re.findall(r'\+?\d{1,2}[-.\s]?\d{3}[-.\s]?\d{3}[-.\s]?\d{4}', text)

print("Numbers:", numbers)

print("Phone Numbers:", phone\_numbers)

# Extract words before an exclamation mark

before\_exclamation = re.findall(r'.\*!', text)

print("Before Exclamation:", before\_exclamation)

# Extract vowels

vowels = re.findall(r'[aeiou]', text, re.IGNORECASE)

print("Vowels:", vowels)

# Extract dates (MM-DD-YYYY format)

dates = re.findall(r'\d{1,2}-\d{1,2}-\d{4}', text)

print("Dates:", dates)

# Check if text starts with 'Hello' and ends with a phone number

starts\_hello = re.match(r'^Hello', text.strip()) is not None

ends\_with\_phone = re.search(r'\d{3}-\d{3}-\d{4}$', text.strip()) is not None

print("Starts with 'Hello':", starts\_hello)

print("Ends with phone number:", ends\_with\_phone)

# Extract domain names from URLs

domains = re.findall(r'http://(?:www\.)?([\w.-]+)', text)

print("Domains:", domains)

# Extract hashtags and mentions

hashtags = re.findall(r'#\w+', text)

mentions = re.findall(r'@\w+', text)

print("Hashtags:", hashtags)

print("Mentions:", mentions)

# Extract capitalized words (proper nouns, organizations, names)

capitalized\_words = re.findall(r'\b[A-Z][A-Za-z]+\b', text)

print("Capitalized Words:", capitalized\_words)

****

****