nlp-practice-1-fc

April 2, 2024

1 Practical 1

1.Write python script to convert given input text to speech.

[ ]:
```
# Check if the required module is installed.
!pip show gtts

# If the module is not installed, install it.
!pip install gtts

# Import the required modules.
from gtts import gTTS
import os

# Define the function to convert text to speech.
def text_to_speech(text, language='en', slow=False):
  speech = gTTS(text=text, lang=language, slow=slow)
  speech.save("output.mp3")
  os.system("mpg321 output.mp3")

# Define the text to be converted to speech.
text = "Hello, this is a sample text-to-speech conversion."

# Call the function to convert the text to speech.
text_to_speech(text)
```

2.Write python script to convert given input speech to text.

[ ]:
```
!pip install speech_recognition
```

[ ]:
```
import speech_recognition as sr
filename="C:/Users/tcsc/Downloads/Alice_Arnold_voice.ogg"
r=sr.Recognizer ()
with sr.AudioFile (filename) as source:
 audio_data=r.record (source)
 text=r.recognize_google (audio_data)
 print (text)
```

1.0.1 Work

[ ]:

2 PRACTICAL NO. 2

a) Study of various corpus-Brown, Inaugural, Reuters, UDHR with various methods like fields, raw, words, sents, categories

[ ]:
```
import nltk
from nltk.corpus import brown
nltk.download('brown')
```

[ ]:
```
brown.words()
```

[ ]:
```
brown.categories()
```

[ ]:
```
brown.words(categories='romance')
```

[ ]:
```
brown.words(categories='editorial')
```

[ ]:
```
brown.fileids()
```

[ ]:
```
brown.words(fileids=['cr07'])
```

[ ]:
```
brown.sents()
```

b) Study Conditional Frequency Distribution

[ ]:
```
news_text = brown.words(categories='news')
```

[ ]:
```
fdist=nltk.FreqDist([w.lower() for w in news_text])
```

[ ]:
```
modals=['can', 'could', 'may', 'might', 'must', 'will']
for m in modals:
  print(m,"",fdist[m])
```

ConditionalFreqDist

[ ]:
```
import nltk
from nltk.corpus import brown

# Create a ConditionalFreqDist
cfd = nltk.ConditionalFreqDist((genre, word)
                               for genre in brown.categories()
                               for word in brown.words(categories=genre))
```

```
# Define genres and modals
genres = ['news', 'religion', 'fiction', 'thriller', 'romance']
modals = ['can', 'could', 'may', 'might', 'must', 'will']

# Tabulate the data
cfd.tabulate(conditions=genres, samples=modals)
```

3 PRACTICAL NO. 3

a) Create and use your own corpora (plain text).

[ ]:
```
import nltk
from nltk.corpus import PlaintextCorpusReader

# Specify the corpus root directory
corpus_root = '/content/Msc.txt'

# Instantiate PlaintextCorpusReader
filelist = PlaintextCorpusReader(corpus_root, '.*')


print('\nFile list:\n', filelist.fileids())
print('\nFilelist Root:', filelist.root)


w = filelist.words('Msc.txt')
print('\nFirst 6 words:', w[:6])


w1 = filelist.sents('Msc.txt')
print('\nFirst sentence:', w1[0])



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 statistics
    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)
```

b) Study of tagged corpora with methods like tagged_sents, tagged_words.

[ ]:
```
import nltk

# Download the required datasets
nltk.download('conll2000')
nltk.download('treebank')

# Load the tagged words from the Brown, CoNLL 2000, and Treebank corpora
brown_tagged_words = nltk.corpus.brown.tagged_words()
conll2000_tagged_words = nltk.corpus.conll2000.tagged_words()
treebank_tagged_words = nltk.corpus.treebank.tagged_words()

# Display a few tagged words from each corpus
print("Brown Corpus Tagged Words:")
print(brown_tagged_words[:10])

print("\nCoNLL 2000 Corpus Tagged Words:")
print(conll2000_tagged_words[:10])

print("\nTreebank Corpus Tagged Words:")
print(treebank_tagged_words[:10])

# Load the tagged sentences from the Treebank corpus
treebank_tagged_sents = nltk.corpus.treebank.tagged_sents()

# Display a few tagged sentences from the Treebank corpus
print("\nTreebank Corpus Tagged Sentences:")
print(treebank_tagged_sents[:2])
```

3)WAP to find the most frequent noun tags.

[ ]:
```
import nltk
nltk.download('treebank')
```

[ ]:
```
from nltk.corpus import treebank
wsj =treebank.tagged_words()
word_tag = nltk.FreqDist(wsj)
[word for (word,tag) in word_tag if tag.startswith('N')]
```

4 PRACTICAL NO. 4

1)Map Words to Properties using Python Dictionaries

[1]:
```
# {'colorless': 'ADJ', 'ideas': 'N', 'sleep': 'V', 'furiously': 'ADJ'}
pos={}
pos['colorless'] ='ADJ'
pos['ideas'] = 'N'
pos['sleep'] ='V'
```

```
pos['furiously']='ADJ'
```

[ ]:
```
list(pos)
```

[ ]:
```
sorted(pos)
```

[ ]:
```
[w for w in pos if w.endswith('s')]
```

[ ]:
```
for word in sorted(pos):
  print(word, pos[word])
```

[ ]:
```
pos.keys()
```

[ ]:
```
pos.values()
```

[ ]:
```
pos.items()
```

[ ]:
```
pos['sleep']=['N','V']
pos
```

b) Study (i)DefaultTagger (ii) Regular Expression Tagger (iii) UnigramTagger.

[ ]:
```
import nltk
from nltk.corpus import brown
nltk.download('brown')
nltk.download('punkt')
# Get tags from the 'news' category in the Brown corpus
tags = [tag for (word, tag) in brown.tagged_words(categories='news')]

# Find the most common tag in the 'news' category
most_common_tag = nltk.FreqDist(tags).max()

# Print the most common tag
print(most_common_tag)  # This will output 'NN'

# Raw text for tagging
raw = 'I do not like green eggs and ham, I do not like them Sam I am!'
tokens = nltk.word_tokenize(raw)

# Create a DefaultTagger with the most common tag
default_tagger = nltk.DefaultTagger(most_common_tag)

# Tag the tokens with the DefaultTagger
tagged_tokens = default_tagger.tag(tokens)

# Print the tagged tokens
print(tagged_tokens)
```

[ ]:
```
# Evaluate the tagger using the 'news' category in the Brown corpus
gold_standard_tags = [tag for (word, tag) in brown.
 ↪tagged_words(categories='news')]
accuracy = default_tagger.evaluate(gold_standard_tags)

# Print the accuracy
print(accuracy)
```

4.0.1 PRACTICAL NO. 5

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

[ ]:
```
import nltk
from nltk.corpus import wordnet as wn

nltk.download('wordnet')

print(wn.synsets('motocar'))
print(wn.synsets('car'))

car_synset = wn.synset('car.n.01')
print(car_synset.lemma_names())
print(car_synset.examples())
print(car_synset.definition())
print(car_synset.lemmas())
print(car_synset.lemmas()[0].name())
print(wn.lemma('car.n.01.automobile').synset())
print(wn.lemma('car.n.01.automobile').name())
print(wn.lemmas('car'))

for synset in wn.synsets('car'):
    print(synset.lemma_names())
```

b) Study of lemmas, hyponyms, hypernyms, meronyms, entailments.

[ ]:
```
import nltk
from nltk.corpus import wordnet as wn

nltk.download('wordnet')

car = wn.synset('car.n.01')
print(car)

types_of_car = car.hyponyms()
print(types_of_car)

print(types_of_car[26])
```

```
print(sorted([lemma.name() for synset in types_of_car for lemma in synset.
 ↪lemmas()]))

print(car.hypernyms())
paths = car.hypernym_paths()
print(len(paths))
print([synset.name() for synset in paths[0]])
print(car.root_hypernyms())
print(wn.synset('tree.n.02').part_meronyms())
print(wn.synset('tree.n.01').substance_meronyms())
print(wn.synset('tree.n.01').member_holonyms())
print(wn.synset('walk.v.01').entailments())
print(wn.synset('eat.v.01').entailments())
print(wn.lemma('supply.n.02.supply').antonyms())
print(wn.lemma('rush.n.02.rush').antonyms())
print(wn.lemma('horizontal.a.01.horizontal').antonyms())
```

c) WAP to find synonym and antonym of word 'active' using Wordnet.

[ ]:
```
import nltk
from nltk.corpus import wordnet

nltk.download('wordnet')
print(wordnet.synsets("active"))
print(wordnet.synset('active.a.01').lemmas()[0].antonyms())
```

4.0.2 PRACTICAL NO. 6

a) Compare two nouns.
b) Handling stopword

[ ]:
```
import nltk

nltk.download('wordnet')
nltk.download('omw-1.4')

from nltk.corpus import wordnet

syn1 = wordnet.synsets("football")
syn2 = wordnet.synsets('soccer')

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()
```

b)Adding or Removing Stop Words in NLTK's Default Stop Word List

[ ]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('stopwords')
nltk.download('punkt')

print(stopwords.words())

text = "messi likes to play football, however he is not too fond of tennis"
text_tokens = word_tokenize(text)

token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪stopwords.words('english')]
print(token_without_sw)

all_stopwords = stopwords.words('english')
all_stopwords.append('play')

text_tokens = word_tokenize(text)
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords]
print(token_without_sw)

all_stopwords.remove('is')

text_tokens = word_tokenize(text)
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords]
print(token_without_sw)
```

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

[ ]:
```
import gensim
from gensim.parsing.preprocessing import remove_stopwords
from nltk.tokenize import word_tokenize

text = "messi 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)

from gensim.parsing.preprocessing import STOPWORDS

all_stopwords_gensim = STOPWORDS.union({'likes', 'play'})
text = "messi likes to play football, however he is not too fond of tennis"
text_tokens = word_tokenize(text)
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords_gensim]
print(token_without_sw)

all_stopwords_gensim = STOPWORDS
sw_list = {"not"}
all_stopwords_gensim = all_stopwords_gensim.difference(sw_list)
print(all_stopwords_gensim)

text = "messi likes to play football, however he is not too fond of tennis"
text_tokens = word_tokenize(text)
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords_gensim]
print(token_without_sw)
```

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

```
[ ]: import spacy
import nltk
from nltk.tokenize import word_tokenize

sp = spacy.load('en_core_web_sm')
all_stopwords = sp.Defaults.stop_words
all_stopwords.add("play")

text = "messi likes to play football, however he is not too fond of tennis"
text_tokens = word_tokenize(text)
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords]
print(token_without_sw)

all_stopwords.remove('is')
token_without_sw = [word for word in text_tokens if word.lower() not in␣
 ↪all_stopwords]
print(token_without_sw)
```

4.0.3 PRACTICAL NO. 7

a)Tokenization using Python's split() function

[ ]:
```
text="this tool is on a beta stage, alexa developers can use get metrics"
data=text.split()
for i in data:
    print(i)
```

b) Tokenization using Regular Expressions (Regfx)

[ ]:
```
import nltk
from nltk.tokenize import RegexpTokenizer
tk=RegexpTokenizer('s+',gaps=True)
str="i love to study NLP in Python"
tokens=tk.tokenize(str)
print(tokens)
['i love to ', 'tudy NLP in Python']
```

c) Tokenization using NLTK

[ ]:
```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

str = "i love to study DL"
print(word_tokenize(str))
```

d) Tokenization using the spaCy library

[ ]:
```
import spacy
nlp=spacy.blank("en")
str="i love nlp"
doc=nlp(str)
words=[word.text for word in doc]
print(words)
```

e) Tokenization using Keras

[ ]:
```
import keras
from keras.preprocessing.text import text_to_word_sequence
str="i love to study NLP"
tokens=text_to_word_sequence(str)
print(tokens)
```

f) Tokenization using Gensim

[ ]:
```
! pip install gensim
```

[31]:
```
from gensim.utils import tokenize
str="i love to study nlp"
list(tokenize(str))
```

[31]: ['i', 'love', 'to', 'study', 'nlp']

4.0.4 PRACTICAL NO. 8

Import NLP Libraries for Indian Languages and perform:

a) Word tokenization in Hindi.

[ ]:
```
!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
```

[3]:
```
!pip install typing-extensions
```

```
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (4.10.0)
```

[ ]:
```
from inltk.inltk import setup
setup('hi')
from inltk.inltk import tokenize
hindi_text = """                        """
# tokenize(input text, language code)
tokenize(hindi_text, "hi")
```

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

[ ]:
```
!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('        ', 5, 'hi')
print(output)
```

c) Identify the Indian language of a text.

[ ]:
```
!pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/torch_stable.
↪html
!pip install inltk
!pip install tornado==4.5.3
from inltk.inltk import setup
setup('gu')
from inltk.inltk import identify_language
```

```
#Identify the Lnaguage of given text
identify_language('       ')
```

4.0.5 PRACTICAL NO. 9

AIM ->Illustrate POS tagging:

a) Sentence tokenization, word tokenization, part of speech tagging and chunking of user define text.

[ ]:
```
import nltk
from nltk import tokenize
from nltk import tag
from nltk import chunk
nltk.download('maxent_ne_chunker')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('words')

para = "Today we will be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n======\n", sents)
print("\nword tokenization\n=========\n")
for index in range(len(sents)):
    words = tokenize.word_tokenize(sents[index])
    print(words)

# POS Tagging
print("\nPOS tagging\n==========\n")
tagged_words = []
for index in range(len(sents)):
    tagged_words.append(tag.pos_tag(tokenize.word_tokenize(sents[index])))
print(tagged_words)

# chunking
print("\nChunking\n==========\n")
tree = []
for index in range(len(sents)):
    tree.append(chunk.ne_chunk(tag.pos_tag(tokenize.
 ↪word_tokenize(sents[index]))))
print(tree)
```

b) Name Entity Recognition of user defined text

[ ]:
```
import spacy

nlp = spacy.load("en_core_web_sm")
```

```
text = """Apple Inc., originally named Apple Computer, Inc.,
is a multinational corporation that creates and markets consumer electronics␣
 ↪and attendant computer software, and is a digital distributor of media␣
 ↪content. Apple's core product lines are the iPhone smartphone, iPad tablet␣
 ↪computer, and the Macintosh personal computer.
The company offers its products online and has a chain of retail stores known␣
 ↪as Apple Stores. Founders Steve Jobs, Steve Wozniak, and Ronald Wayne␣
 ↪created Apple Computer Co. on April 1, 1976, to market Wozniak's Apple I␣
 ↪desktop computer, [2] and Jobs and Wozniak incorporated the company on␣
 ↪January 3, 1977, [3] in Cupertino, California."""

doc = nlp(text)

print("Noun phrases: ", [chunk.text for chunk in doc.noun_chunks])
print("Verbs: ", [token.lemma_ for token in doc if token.pos_ == "VERB"])
```

c) Name Entity Recognition with diagram using NLTK corpus-treebank.

[ ]:
```
import nltk

nltk.download("treebank")

from nltk.corpus import treebank_chunk


treebank_chunk.tagged_sents()[0]
```

[ ]:
```
treebank_chunk.chunked_paras()[0]
```

[ ]:
```
treebank_chunk.chunked_words()
```