* Program 1 :text preprocessing

import re

import string

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from nltk import download

download('punkt')

download('stopwords')

download('wordnet')

import nltk

nltk.download('punkt')

import nltk

nltk.download('punkt\_tab')

# Sample text

text = """Hello! I'm new to NLP. Let's process this text. Visit https://www.example.com for details.

         Send your queries to help@nlp.com or call 123-456-7890. Text preprocessing is fun, especially when done correctly."""

# 1. Convert to lowercase

text\_lower = text.lower()

print("1. Lowercase Text:")

print(text\_lower)

# 2. Remove URLs and email addresses

text\_no\_urls\_emails = re.sub(r'https?://\S+|www\.\S+|[\w.-]+@[\w.-]+', '', text\_lower)

print("\n2. Remove URLs and Email Addresses:")

print(text\_no\_urls\_emails)

# 3. Remove numbers

text\_no\_numbers = re.sub(r'\d+', '', text\_no\_urls\_emails)

print("\n3. Remove Numbers:")

print(text\_no\_numbers)

# 4. Remove punctuation

text\_no\_punctuation = text\_no\_numbers.translate(str.maketrans('', '', string.punctuation))

print("\n4. Remove Punctuation:")

print(text\_no\_punctuation)

# 5. Tokenization

tokens = word\_tokenize(text\_no\_punctuation)

print("\n5. Tokenization:")

print(tokens)

# 6. Remove stopwords

stop\_words = set(stopwords.words('english'))

tokens\_no\_stopwords = [word for word in tokens if word not in stop\_words]

print("\n6. Remove Stopwords:")

print(tokens\_no\_stopwords)

# 7. Lemmatization

lemmatizer = WordNetLemmatizer()

tokens\_lemmatized = [lemmatizer.lemmatize(word) for word in tokens\_no\_stopwords]

print("\n7. Lemmatization:")

print(tokens\_lemmatized)

# 8. Stemming (Optional, use either stemming or lemmatization based on the task)

stemmer = PorterStemmer()

tokens\_stemmed = [stemmer.stem(word) for word in tokens\_no\_stopwords]

print("\n8. Stemming:")

print(tokens\_stemmed)

# 9. Remove short tokens (Optional, e.g., remove single-character tokens)

tokens\_filtered = [word for word in tokens\_stemmed if len(word) > 1]

print("\n9. Remove Short Tokens:")

print(tokens\_filtered)

# 10. Rejoin tokens into a clean text

cleaned\_text = ' '.join(tokens\_filtered)

print("\n10. Final Cleaned Text:")

print(cleaned\_text)

Output :-   
text\_no\_urls\_emails = re.sub(r'https?://\S+|www\.\S+|[\w.-]+@[\w.-]+', '', text\_lower) print("\n2. Remove URLs and Email Addresses:") print(text\_no\_urls\_emails) # 3. Remove numbers text\_no\_numbers = re.sub(r'\d+', '', text\_no\_urls\_emails) print("\n3. Remove Numbers:") print(text\_no\_numbers) # 4. Remove punctuation

1. Lowercase Text:

hello! i'm new to nlp. let's process this text. visit [https://www.example.com](https://www.example.com/) for details.

send your queries to [help@nlp.com](mailto:help@nlp.com) or call 123-456-7890. text preprocessing is fun, especially when done correctly.

2. Remove URLs and Email Addresses:

hello! i'm new to nlp. let's process this text. visit for details.

send your queries to or call 123-456-7890. text preprocessing is fun, especially when done correctly.

3. Remove Numbers:

hello! i'm new to nlp. let's process this text. visit for details.

send your queries to or call --. text preprocessing is fun, especially when done correctly.

4. Remove Punctuation:

hello im new to nlp lets process this text visit for details

send your queries to or call text preprocessing is fun especially when done correctly

5. Tokenization:

['hello', 'im', 'new', 'to', 'nlp', 'lets', 'process', 'this', 'text', 'visit', 'for', 'details', 'send', 'your', 'queries', 'to', 'or', 'call', 'text', 'preprocessing', 'is', 'fun', 'especially', 'when', 'done', 'correctly']

6. Remove Stopwords:

['hello', 'im', 'new', 'nlp', 'lets', 'process', 'text', 'visit', 'details', 'send', 'queries', 'call', 'text', 'preprocessing', 'fun', 'especially', 'done', 'correctly']

7. Lemmatization:

['hello', 'im', 'new', 'nlp', 'let', 'process', 'text', 'visit', 'detail', 'send', 'query', 'call', 'text', 'preprocessing', 'fun', 'especially', 'done', 'correctly']

8. Stemming:

['hello', 'im', 'new', 'nlp', 'let', 'process', 'text', 'visit', 'detail', 'send', 'queri', 'call', 'text', 'preprocess', 'fun', 'especi', 'done', 'correctli']

9. Remove Short Tokens:

['hello', 'im', 'new', 'nlp', 'let', 'process', 'text', 'visit', 'detail', 'send', 'queri', 'call', 'text', 'preprocess', 'fun', 'especi', 'done', 'correctli']

10. Final Cleaned Text:

hello im new nlp let process text visit detail send queri call text preprocess fun especi done correctli

* Program 2:text to speech sppech to text

pip install pyttsx3

pip install SpeechRecognition

pip install gTTS

import pyttsx3

import speech\_recognition as sr

from gtts import gTTS

import os

# Function for Text to Speech

def tts():

    text = input("Enter the text: ")

    lang = "en"

    speech = gTTS(text=text, lang=lang, slow=False)

    speech.save("output.mp3")

    print("Playing the converted text...")

    os.system("start output.mp3")  # Use 'start' for Windows, 'open' for macOS, or 'mpg321' for Linux

# Function for Speech to Text from a File (Audio File)

def att():

    file = "moon.wav"  # Make sure this file exists in your directory or change the path to your audio file

    s = sr.Recognizer()

    try:

        with sr.AudioFile(file) as source:

            audio\_data = s.record(source)

            text = s.recognize\_google(audio\_data)

            print("Text from audio file: ", text)

    except FileNotFoundError:

        print("The specified audio file does not exist.")

    except sr.UnknownValueError:

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

    except sr.RequestError as e:

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

# Main Menu Loop

while True:

    print("\nMENU:")

    print("Press 1 for Text to Speech")

    print("Press 2 for Speech to Text from File")

    print("Press 3 for Exit")

    # Get user choice

    choice = input("\nEnter your choice (1, 2, or 3): ")

    # Text to Speech

    if choice == '1':

        tts()

    # Speech to Text from an Audio File

    elif choice == '2':

        att()

    # Exit the program

    elif choice == '3':

        print("\nExiting program. Goodbye!")

        break

    # Invalid Input Handling

    else:

        print("Invalid choice. Please enter 1, 2, or 3.")

    print("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n")

Output :-

MENU:

Press 1 for Text to Speech

Press 2 for Speech to Text from File

Press 3 for Exit

Enter your choice (1, 2, or 3): 1

Enter the text: Hello, this is a text-to-speech conversion example!

Playing the converted text...

* Program 3:cfg

pip install nltk

from nltk import CFG, ChartParser

import re

# Step 1: Define the Grammar with Extended Rules

grammar = CFG.fromstring("""

S -> NP VP

NP -> Det N | Det Adj N | Det N PP

VP -> V NP | V PP | V

PP -> P NP

Det -> 'the' | 'a' | 'an'

N -> 'cat' | 'dog' | 'ball' | 'park' | 'game' | 'man' | 'woman'

Adj -> 'big' | 'small' | 'beautiful' | 'lazy' | 'fast'

V -> 'chased' | 'saw' | 'kicked' | 'played' | 'ran' | 'jumped'

P -> 'in' | 'on' | 'by' | 'with'

""")

# Step 2: Create the parser

parser = ChartParser(grammar)

# Step 3: Take user input

sentence = input("Enter a sentence: ").strip()

# Function to suggest basic corrections for common errors

def suggest\_corrections(sentence):

    suggestions = []

    # Basic checks for missing parts of speech

    words = sentence.split()

    if len(words) < 2:

        suggestions.append("The sentence seems too short. Try adding more words.")

    if re.search(r'\s(the|a|an)\s$', sentence):

        suggestions.append("The sentence ends with an article, but lacks a noun after it.")

    if re.search(r'[\w]+$', sentence) and not any(word in words for word in ['chased', 'saw', 'kicked', 'played', 'ran', 'jumped']):

        suggestions.append("The sentence seems to lack a verb. Try adding an action verb.")

    return suggestions

# Step 4: Parse the sentence

try:

    parses = list(parser.parse(sentence.split()))

    if parses:

        print("The sentence is grammatically correct!")

        print("Parse Trees:")

        for tree in parses:

            tree.pretty\_print()

    else:

        print("The sentence is not grammatically correct.")

        corrections = suggest\_corrections(sentence)

        if corrections:

            print("Suggestions for correction:")

            for suggestion in corrections:

                print(f"- {suggestion}")

        else:

            print("No suggestions available, please check the grammar.")

except ValueError as e:

    print("Error:", e)

OUTPUT:-

Enter a sentence: the dog chased the ball

The sentence is grammatically correct!

Parse Trees:

S

\_\_\_\_\_\_\_\_|\_\_\_\_\_

| VP

| \_\_\_\_\_|\_\_\_

NP | NP

\_\_\_|\_\_\_ | \_\_\_|\_\_\_

Det N V Det N

| | | | |

the dog chased the ball

* Program 4:-muphogical parsing using fst

def morphological\_parser(word):

    # Define a dictionary of suffixes and their meanings

    suffix\_rules = {

        "ing": "Present participle or gerund",

        "ed": "Past tense",

        "s": "Plural or third-person singular",

        "er": "Agent or comparative",

        "est": "Superlative",

        "ly": "Adverb",

        "ness": "State of being",

        "un": "Negation (prefix)",  # Exception: check for prefixes

        "re": "Repetition or backward (prefix)",  # Prefix example

    }

    # Results to store parsing information

    result = {"root": None, "affixes": []}

    # Check for prefixes

    for prefix in ["un", "re"]:

        if word.startswith(prefix):

            result["affixes"].append({"affix": prefix, "type": "prefix", "meaning": suffix\_rules[prefix]})

            word = word[len(prefix):]  # Remove prefix for further analysis

    # Check for suffixes

    for suffix in sorted(suffix\_rules.keys(), key=len, reverse=True):  # Longest suffix first

        if word.endswith(suffix) and suffix not in ["un", "re"]:  # Ignore prefixes here

            result["affixes"].append({"affix": suffix, "type": "suffix", "meaning": suffix\_rules[suffix]})

            word = word[:-len(suffix)]  # Remove suffix for further analysis

    # Assign the remaining part as the root

    result["root"] = word if word else "Unknown"

    return result

# Example Usage

words = ["unhappiest", "redoing", "walked", "teachers", "happily", "sadness"]

for word in words:

    print(f"Word: {word}")

    print(morphological\_parser(word))

    print()

OUTPUT:-

Word: unhappiest

{'root': 'happi', 'affixes': [{'affix': 'un', 'type': 'prefix', 'meaning': 'Negation (prefix)'}, {'affix': 'est', 'type': 'suffix', 'meaning': 'Superlative'}]}

Word: redoing

{'root': 'do', 'affixes': [{'affix': 're', 'type': 'prefix', 'meaning': 'Repetition or backward (prefix)'}, {'affix': 'ing', 'type': 'suffix', 'meaning': 'Present participle or gerund'}]}

Word: walked

{'root': 'walk', 'affixes': [{'affix': 'ed', 'type': 'suffix', 'meaning': 'Past tense'}]}

Word: teachers

{'root': 'teacher', 'affixes': [{'affix': 's', 'type': 'suffix', 'meaning': 'Plural or third-person singular'}]}

Word: happily

{'root': 'happi', 'affixes': [{'affix': 'ly', 'type': 'suffix', 'meaning': 'Adverb'}]}

Word: sadness

{'root': 'sad', 'affixes': [{'affix': 'ness', 'type': 'suffix', 'meaning': 'State of being'}]}

* Program 5:murphogical analze using porter stemmer

from nltk.stem import PorterStemmer

# Create a Porter Stemmer instance

porter\_stemmer = PorterStemmer()

# Example words for stemming

words = ["running", "jumps", "happily", "running", "happily"]

# Apply stemming to each word

stemmed\_words = [porter\_stemmer.stem(word) for word in words]

# Print the results

print("Original words:", words)

print("Stemmed words:", stemmed\_words)

OUTPUT:- Original words: ['running', 'jumps', 'happily', 'running', 'happily']

Stemmed words: ['run', 'jump', 'happili', 'run', 'happili']