nlp-lab

December 26, 2023

[4]:
```python
#1a
import pandas as pd
import os
import docx
import PyPDF2
dir_path='C:\\Users\\ayush\\OneDrive\\Desktop\\DSCE\\7\\nl\\1'
files=[f for f in os.listdir(dir_path) if (f.endswith('.txt') or f.endswith('.
↪docx') or f.endswith('.pdf'))]
data = []
for txt_file in files:
    if(txt_file.endswith('.txt')):
        with open(os.path.join(dir_path, txt_file), 'r') as file:
            content = file.read()
            data.append({'filename': txt_file, 'content': content })
    elif(txt_file.endswith('.docx')):
        docx_path = os.path.join(dir_path, txt_file)
        doc = docx.Document(docx_path)
        content = "\n".join([paragraph.text for paragraph in doc.paragraphs])
        data.append({'filename': txt_file, 'content': content })
    elif(txt_file.endswith('.pdf')):
        with open(os.path.join(dir_path, txt_file), 'rb') as f:
            pdf_reader = PyPDF2.PdfReader(f)
            num_pages = len(pdf_reader.pages)
            for page in range(num_pages):
                content=pdf_reader.pages[page].extract_text()
            data.append({'filename': txt_file, 'content': content})
df = pd.DataFrame(data)
print(df)
```

```
                  filename                                            content
0              aditya.docx                                            aditya
1  Ayush Aditya_Resume.pdf  EDUCATION\n•BE in Artificial intelligence and …
2               sample.txt                                             ayush
```

[ ]:
```python
#1b
import os

dir_path = 'C:\\Users\\ayush\\OneDrive\\Desktop\\DSCE\\7\\nl\\1'
```

```
files = [f for f in os.listdir(dir_path) if f.endswith('.txt')]
data = []

for txt_file in files:
    with open(os.path.join(dir_path, txt_file), 'r') as file:
        content = file.read()
        data.append({'filename': txt_file, 'content': content})

# Print the content of text files
for item in data:
    print("Filename:", item['filename'])
    print("Content:")
    print(item['content'])
    print("\n")
```

[5]:

```
#2a
import pandas as pd
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
ps = PorterStemmer()
def clean_text(text):
    text = re.sub(r'[^A-Za-z\s]', '', text)
    text = text.lower()
    tokens = word_tokenize(text)
    tokens = [ps.stem(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)
    df['cleaned_content'] = df['content'].apply(clean_text)
    lemmatized_output = ' '.join([lemmatizer.lemmatize(w) for w in tokens])
    print(lemmatized_output)
df['cleaned_content'] = df['content'].apply(clean_text)
print(df)
```

```
                  filename                                           content  \
0              aditya.docx                                            aditya
1  Ayush Aditya_Resume.pdf  EDUCATION\n•BE in Artificial intelligence and …
2               sample.txt                                             ayush

                                cleaned_content
0                                        aditya
1  educ artifici intellig machin learn dayananda …
2                                         ayush
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
#2b
import pandas as pd
import re

# Sample data creation (replace this with your actual data loading logic)
data = {'content': ["This is an example sentence.", "Another example sentence.
 ↪"]}
df = pd.DataFrame(data)

def clean_text(text):
    # Remove non-alphabetic characters
    text = re.sub(r'[^A-Za-z\s]', '', text)

    # Convert to lowercase
    text = text.lower()

    # Tokenize
    tokens = text.split()

    # Remove stopwords
    stop_words = set(["is", "an", "the", "this", "another"])  # Add more␣
 ↪stopwords as needed
    tokens = [word for word in tokens if word not in stop_words]

    # Stemming (using a simple example)
    tokens = [word[:-1] if word.endswith('s') else word for word in tokens]

    # Lemmatization (using a simple example)
    tokens = [word[:-1] if word.endswith('s') else word for word in tokens]

    return ' '.join(tokens)

df['cleaned_content'] = df['content'].apply(clean_text)
print(df)
```

[6]:
```
#3a
from nltk.util import ngrams
def generate_ngrams(text, n):
    """Generate n-grams from the given text"""
    tokens = text.split()
    return [' '.join(gram) for gram in ngrams(tokens, n)]
df['trigram'] = df['cleaned_content'].apply(generate_ngrams, n=2)
print(df)
```

```
                filename                                               content  \
0            aditya.docx                                                aditya
1  Ayush Aditya_Resume.pdf  EDUCATION\n•BE in Artificial intelligence and …
2             sample.txt                                                 ayush

                                     cleaned_content  \
0                                             aditya
1  educ artifici intellig machin learn dayananda …
2                                              ayush

                                            trigram
0                                                []
1  [educ artifici, artifici intellig, intellig ma…
2                                                []
```

[ ]:
```
#3b
import pandas as pd

# Sample data creation
data = {'cleaned_content': ["this is an example sentence", "another example␣
 ↪sentence"]}
df = pd.DataFrame(data)

# Function to generate bigrams
def generate_ngrams(text, n):
    tokens = text.split()
    ngrams_list = [' '.join(tokens[i:i+n]) for i in range(len(tokens)-n+1)]
    return ngrams_list

# Apply the function to generate bigrams
df['bigrams'] = df['cleaned_content'].apply(generate_ngrams, n=2)

# Print the dataframe
print(df)
```

[7]:
```
#4a
import nltk
nltk.download('averaged_perceptron_tagger')
def pos_tagging(text):
    """Generate POS tags for the given text"""
    tokens = nltk.word_tokenize(text)
    return nltk.pos_tag(tokens)
# Apply POS tagging to the cleaned_content
```

```
df['POS_tags'] = df['cleaned_content'].apply(pos_tagging)
print(df)
```

```
                filename                                          content  \
0            aditya.docx                                           aditya   
1  Ayush Aditya_Resume.pdf  EDUCATION\n•BE in Artificial intelligence and …
2             sample.txt                                            ayush   

                                   cleaned_content  \
0                                           aditya   
1  educ artifici intellig machin learn dayananda …   
2                                            ayush   

                                           trigram  \
0                                               []   
1  [educ artifici, artifici intellig, intellig ma…   
2                                               []   

                                          POS_tags  
0                                 [(aditya, NN)]  
1  [(educ, NN), (artifici, NN), (intellig, NN), (…  
2                                  [(ayush, NN)]  
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

[ ]:
```
#4b
import pandas as pd

# Sample data creation
data = {'cleaned_content': ["this is an example sentence", "another example␣
↪sentence"]}
df = pd.DataFrame(data)

# Function to generate simple POS tags (Noun, Verb, Adjective)
def simple_pos_tagging(text):
    tokens = text.split()
    pos_tags = []
    for token in tokens:
        if token.endswith('ing'):
            pos_tags.append((token, 'Verb'))
        elif token.endswith('ly'):
            pos_tags.append((token, 'Adverb'))
        else:
            pos_tags.append((token, 'Noun'))
    return pos_tags
```

```
# Apply the function to generate POS tags
df['POS_tags'] = df['cleaned_content'].apply(simple_pos_tagging)

# Print the dataframe
print(df)
```

[8]:
```
#5a
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')
def noun_phrase_chunking(text_with_tags):
    """ Extract noun phrases using chunking """
    grammar = "NP: {<DT>?<JJ>*<NN>}"
    cp = nltk.RegexpParser(grammar)
    tree = cp.parse(text_with_tags)
    noun_phrases = []
    for subtree in tree.subtrees():
        if subtree.label() == 'NP':
            noun_phrases.append(' '.join(word for word, tag in subtree.
↪leaves()))
    return noun_phrases
df['noun_phrases'] = df['POS_tags'].apply(noun_phrase_chunking)
print(df)
```

```
                  filename                                            content  \
0              aditya.docx                                             aditya
1  Ayush Aditya_Resume.pdf  EDUCATION\n•BE in Artificial intelligence and …
2               sample.txt                                              ayush

                                 cleaned_content  \
0                                         aditya
1  educ artifici intellig machin learn dayananda …
2                                          ayush

                                         trigram  \
0                                             []
1  [educ artifici, artifici intellig, intellig ma…
2                                             []

                                        POS_tags  \
0                                  [(aditya, NN)]
1  [(educ, NN), (artifici, NN), (intellig, NN), (…
2                                   [(ayush, NN)]

                                    noun_phrases
0                                       [aditya]
```

```
1  [educ, artifici, intellig, machin, dayananda, …
2                                            [ayush]
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\ayush\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\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[ ]:
```python
#5b
import pandas as pd

# Sample data creation
data = {'POS_tags': [
    [('this', 'Noun'), ('is', 'Noun'), ('an', 'Noun'), ('example', 'Noun'),␣
  ↪('sentence', 'Noun')],
    [('another', 'Noun'), ('example', 'Noun'), ('sentence', 'Noun')]
]}
df = pd.DataFrame(data)

# Function to perform simple noun phrase chunking
def simple_noun_phrase_chunking(pos_tags):
    noun_phrases = []
    current_phrase = []

    for token, tag in pos_tags:
        if tag in ['Noun', 'Adjective']:
            current_phrase.append(token)
        elif current_phrase:
            noun_phrases.append(' '.join(current_phrase))
            current_phrase = []

    if current_phrase:
        noun_phrases.append(' '.join(current_phrase))

    return noun_phrases

# Apply the function to generate noun phrases
df['noun_phrases'] = df['POS_tags'].apply(simple_noun_phrase_chunking)

# Print the dataframe
print(df)
```

[9]:
```python
#6a
import spacy
import random
```

```
# Load the spaCy model
nlp = spacy.load("en_core_web_sm")

# Sentence prompts dictionary
sentence_prompts = {
    "She opened the door and saw a": ["beautiful garden", "mysterious figure",␣
  ↪"bright light"],
    "After a long day at work, I like to relax by": ["watching my favorite TV␣
  ↪show", "going for a walk", "reading a book"]
}

# Input prompt
input_prompt = "After a long day at work, I like to relax by"

# Check if the input prompt is in the dictionary
if input_prompt in sentence_prompts:
    possible_completions = sentence_prompts[input_prompt]
    print("Possible Completions:")
    for completion in possible_completions:
        print(f"- {input_prompt} {completion}")
else:
    print("Prompt not found in the dictionary.")
    # Use spaCy to generate a random sentence completion
    doc = nlp(input_prompt)
    random_completion = " ".join([token.text for token in doc] + [random.
  ↪choice(["enjoying", "listening", "playing"])])
    print(f"- {random_completion}")
```

```
Possible Completions:
- After a long day at work, I like to relax by watching my favorite TV show
- After a long day at work, I like to relax by going for a walk
- After a long day at work, I like to relax by reading a book
```

```
[ ]: #6b

sentence_prompts = {
    "She opened the door and saw a": ["beautiful garden", "mysterious figure",␣
  ↪"bright light"],
    "After a long day at work, I like to relax by": ["watching my favorite TV␣
  ↪show", "going for a walk", "reading a book"]
}

input_prompt = "After a long day at work, I like to relax by"

if input_prompt in sentence_prompts:
    possible_completions = sentence_prompts[input_prompt]
    print("Possible Completions:")
    for completion in possible_completions:
        print(f"- {input_prompt} {completion}")
else:
    print("Prompt not found in the dictionary.")
    # Use random to create a random sentence completion
    random_completion = random.choice(["enjoying a cup of tea", "listening to␣
  ↪music", "playing video games"])
    print(f"- {input_prompt} {random_completion}")
```

```
[10]: #7a
from textblob import TextBlob
data = ["I love this product!", "It's terrible.", "Neutral statement."]
sentiments = [TextBlob(text).sentiment.polarity for text in data]
labels = ['positive' if score > 0 else 'negative' if score < 0 else'neutral'␣
  ↪for score in sentiments]
result_df = pd.DataFrame({'text': data, 'sentiment_score': sentiments,'label':␣
  ↪labels})
print(result_df)
```

```
                   text  sentiment_score     label
0  I love this product!            0.625  positive
1        It's terrible.           -1.000  negative
2    Neutral statement.            0.000   neutral
```

```
[ ]: #7b

data = ["I love this product!", "It's terrible.", "Neutral statement."]

def determine_sentiment_label(text):
    if "love" in text.lower():
        return 'positive'
    elif "terrible" in text.lower():
        return 'negative'
    else:
        return 'neutral'

result_dict = {'text': data, 'label': [determine_sentiment_label(text) for text␣
  ↪in data]}

for text, label in zip(result_dict['text'], result_dict['label']):
    print(f"Text: {text}")
    print(f"Label: {label}")
    print()
```

```
[2]: pip install transformers
pip install sentence-transformers
```

```python
from transformers import GPT2Tokenizer, GPT2LMHeadModel, BartTokenizer,␣
↪BartForConditionalGeneration
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

def abstractive_summarization(text):
    # GPT-2 model for abstractive summarization
    tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
    model = GPT2LMHeadModel.from_pretrained("gpt2")

    # Tokenize and generate summary
    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt",␣
↪max_length=1024, truncation=True)
    summary_ids = model.generate(inputs, max_length=150, length_penalty=2.0,␣
↪num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    return summary



def extractive_summarization_sentence_transformers(text, num_sentences=3):
    # Sentence Transformers for extractive summarization
    model = SentenceTransformer("bert-base-nli-mean-tokens")

    # Split text into sentences
    sentences = text.split('. ')

    # Compute sentence embeddings
    embeddings = model.encode(sentences)

    # Calculate pairwise cosine similarity between embeddings
    similarity_matrix = cosine_similarity(embeddings, embeddings)

    # Get indices of top-ranked sentences based on similarity
    top_sentence_indices = np.argsort(similarity_matrix.
↪sum(axis=1))[-num_sentences:]

    # Sort sentences based on their original order
    top_sentence_indices = sorted(top_sentence_indices)

    # Generate extractive summary
    extractive_summary = '. '.join(sentences[i] for i in top_sentence_indices)

    return extractive_summary
```

```
# Example usage
text = """In the heart of the bustling city, there stood an old bookstore with␣
 ↪creaky woodenfloors and shelves that seemed to lean under the weight of␣
 ↪countless stories. The air wasfilled with the comforting scent of aged paper␣
 ↪and the soft murmur of people lost in the  worlds between the covers. A the␣
 ↪afternoon sun streamed through dusty windows, casting a
warm glow on antique book covers, occasionally knocking over a book or two. The␣
 ↪bookstore, with its charm and character, was a haven for book lovers seeking␣
 ↪solace and adventure within the pages of both old classics and new releases.
"""
abstractive_summary = abstractive_summarization(text)

extractive_summary_sentence_transformers =␣
 ↪extractive_summarization_sentence_transformers(text)

print("Abstractive Summary:", abstractive_summary)

print("\nExtractive Summary:", extractive_summary_sentence_transformers)
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision
a4f8f3e (https://huggingface.co/sshleifer/distilbart-cnn-12-6).
Using a pipeline without specifying a model name and revision in production is
not recommended.
```

```
Requirement already satisfied: transformers in
c:\users\ayush\anaconda3\lib\site-packages (4.35.2)
Requirement already satisfied: numpy>=1.17 in c:\users\ayush\anaconda3\lib\site-
packages (from transformers) (1.20.3)
Requirement already satisfied: packaging>=20.0 in
c:\users\ayush\anaconda3\lib\site-packages (from transformers) (21.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in
c:\users\ayush\anaconda3\lib\site-packages (from transformers) (0.19.4)
Requirement already satisfied: regex!=2019.12.17 in
c:\users\ayush\anaconda3\lib\site-packages (from transformers) (2021.8.3)
Requirement already satisfied: requests in c:\users\ayush\anaconda3\lib\site-
packages (from transformers) (2.26.0)
Requirement already satisfied: tqdm>=4.27 in c:\users\ayush\anaconda3\lib\site-
packages (from transformers) (4.65.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in
c:\users\ayush\anaconda3\lib\site-packages (from transformers) (0.15.0)
Requirement already satisfied: pyyaml>=5.1 in c:\users\ayush\anaconda3\lib\site-
packages (from transformers) (6.0)
Requirement already satisfied: safetensors>=0.3.1 in
c:\users\ayush\anaconda3\lib\site-packages (from transformers) (0.4.0)
Requirement already satisfied: filelock in c:\users\ayush\anaconda3\lib\site-
packages (from transformers) (3.3.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
c:\users\ayush\anaconda3\lib\site-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (4.8.0)
Requirement already satisfied: fsspec>=2023.5.0 in
c:\users\ayush\anaconda3\lib\site-packages (from huggingface-
hub<1.0,>=0.16.4->transformers) (2023.10.0)
Requirement already satisfied: pyparsing>=2.0.2 in
c:\users\ayush\anaconda3\lib\site-packages (from packaging>=20.0->transformers)
(3.0.4)
Requirement already satisfied: colorama in c:\users\ayush\anaconda3\lib\site-
packages (from tqdm>=4.27->transformers) (0.4.6)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\ayush\anaconda3\lib\site-packages (from requests->transformers)
(2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\ayush\anaconda3\lib\site-packages (from requests->transformers)
(1.26.7)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\ayush\anaconda3\lib\site-packages (from requests->transformers) (3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in
c:\users\ayush\anaconda3\lib\site-packages (from requests->transformers) (2.0.4)
```

[2]:
```
[{'summary_text': ' President Trump ordered the military to start withdrawing
roughly 7,000 troops from Afghanistan in the coming months, two defense
officials said Thursday . The move is an abrupt shift in the 17-year-old war
there and a decision that stunned Afghan officials, who said they had not been
briefed on the plans . The announcement came hours after Jim Mattis, the
secretary of defense, said that he would resign from his position .'}]
```

[ ]:
```
#8b
def simple_summarization(article, num_sentences=3):
    sentences = article.split(".")
    # Remove empty strings from the list
    sentences = [s.strip() for s in sentences if s.strip()]

    # Calculate the importance score for each sentence (based on sentence␣
 ↪length)
    scores = [len(sentence) for sentence in sentences]

    # Select the top N sentences with the highest importance scores
    selected_sentences = sorted(zip(sentences, scores), key=lambda x: x[1],␣
 ↪reverse=True)[:num_sentences]

    # Extract the selected sentences
    summary = [sentence for sentence, _ in selected_sentences]

    return '. '.join(summary)

# Article text
```

```
article = """
  WASHINGTON - The Trump administration has ordered the military to start␣
 ↪withdrawing roughly 7,000 troops from Afghanistan in
  the coming months, two defense officials said Thursday, an abrupt shift in␣
 ↪the 17-year-old war there and a decision that stunned Afghan officials, who␣
 ↪said they had not been briefed on the plans.
  President Trump made the decision to pull the troops - about half the␣
 ↪number the United States has in Afghanistan now - at the same time he␣
 ↪decided to pull American forces out of Syria, one official said.
  The announcement came hours after Jim Mattis, the secretary of defense,␣
 ↪said that he would resign from his position at the end of February after␣
 ↪disagreeing with the president over his approach to policy in the Middle␣
 ↪East.
  The whirlwind of troop withdrawals and the resignation of Mr. Mattis leave␣
 ↪a murky picture for what is next in the United States' longest war, and they␣
 ↪come as Afghanistan has been troubled by spasms of violence afflicting the␣
 ↪capital, Kabul, and other important areas.
  The United States has also been conducting talks with representatives
  of the Taliban, in what officials have described as discussions that could␣
 ↪lead to formal talks to end the conflict.
  Senior Afghan officials and Western diplomats in Kabul woke up to the shock␣
 ↪of the news on Friday morning, and many of them braced for chaos ahead.
  Several Afghan officials, often in the loop on security planning and␣
 ↪decision-making, said they had received no indication in recent days that␣
 ↪the Americans would pull troops out.
  The fear that Mr. Trump might take impulsive actions, however, often loomed␣
 ↪in the background of discussions with the United States, they said.
  They saw the abrupt decision as a further sign that voices from the ground␣
 ↪were lacking in the debate over the war and that with Mr. Mattis's␣
 ↪resignation, Afghanistan had lost one of the last influential
  voices in Washington who channeled the reality of the conflict into the␣
 ↪White House's deliberations.
  The president long campaigned on bringing troops home, but in 2017, at
  the request of Mr. Mattis, he begrudgingly pledged an additional 4,000
  troops to the Afghan campaign to try to hasten an end to the conflict.
  Though Pentagon officials have said the influx of forces - coupled with a␣
 ↪more aggressive air campaign - was helping the war effort, Afghan forces␣
 ↪continued to take nearly unsustainable levels of casualties and lose ground␣
 ↪to the Taliban.
  The renewed American effort in 2017 was the first step in ensuring Afghan␣
 ↪forces could become more independent without a set timeline for
  a withdrawal.
  But with plans to quickly reduce the number of American troops in the␣
 ↪country, it is unclear if the Afghans can hold their own against an␣
 ↪increasingly aggressive Taliban.
  Currently, American airstrikes are at levels not seen since the height
```

```
    of the war, when tens of thousands of American troops were spread␣
  ↪throughout the country.
    That air support, officials say, consists mostly of propping up Afghan
    troops while they try to hold territory from a resurgent Taliban.
"""

# Perform summarization
summary = simple_summarization(article)

# Print the summary
print(summary)
```

[11]:
```
#9a
import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.chunk import ne_chunk
nltk.download('punkt')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
nltk.download('words')
text = "Barack Obama was born in Hawaii and served as the 44th President of the␣
  ↪United States."
words = word_tokenize(text)
pos_tags = pos_tag(words)
named_entities = ne_chunk(pos_tags)
print(named_entities)
```

```
(S
  (PERSON Barack/NNP)
  (PERSON Obama/NNP)
  was/VBD
  born/VBN
  in/IN
  (GPE Hawaii/NNP)
  and/CC
  served/VBD
  as/IN
  the/DT
  44th/CD
  President/NNP
  of/IN
  the/DT
  (GPE United/NNP States/NNPS)
  ./.)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
```

```
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[ ]:

```
#9b
# Given sentence
sentence = "Barack Obama was born in Hawaii and served as the 44th President of␣
 ↪the United States"

# Initialize lists
person_list = []
place_list = []

# Extract entities and populate lists
entities = sentence.split()
for entity in entities:
    if entity in ["Barack", "Obama"]:
        person_list.append(entity)
    elif entity in ["Hawaii", "United", "States"]:
        place_list.append(entity)

# Print the lists
print("Person List:", person_list)
print("Place List:", place_list)
```

[12]:

```
#10
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet
nltk.download('punkt')
nltk.download('wordnet')
text = "The quick brown foxes are jumping over the lazy dogs."
words = word_tokenize(text)
porter_stemmer = PorterStemmer()
stemmed_words = [porter_stemmer.stem(word) for word in words]
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word, pos=wordnet.VERB) for
word in words]
```

```
print("Original words:", words)
print("Stemmed words:", stemmed_words)
print("Lemmatized words:", lemmatized_words)
print("\n")
word = "misunderstanding"
prefixes = ["mis"]
root = "understand"
suffixes = ["ing"]
morphemes = []
for prefix in prefixes:
    if word.startswith(prefix):
        morphemes.append(prefix)
        word = word[len(prefix):]
morphemes.append(word)
print("Word:", word)
print("Morphemes:", morphemes)
```

```
Original words: ['The', 'quick', 'brown', 'foxes', 'are', 'jumping', 'over',
'the', 'lazy', 'dogs', '.']
Stemmed words: ['the', 'quick', 'brown', 'fox', 'are', 'jump', 'over', 'the',
'lazi', 'dog', '.']
Lemmatized words: ['The', 'quick', 'brown', 'fox', 'be', 'jump', 'over', 'the',
'lazy', 'dog', '.']


Word: understanding
Morphemes: ['mis', 'understanding']
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ayush\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[ ]:
```
#10b
def simple_tokenizer(text):
    return text.split()

def simple_porter_stemmer(word):
    # A simple stemming function (for illustration purposes)
    if word.endswith("es"):
        return word[:-2]
    elif word.endswith("s"):
        return word[:-1]
    elif word.endswith("ing"):
        return word[:-3]
    return word
```

```
def simple_wordnet_lemmatizer(word):
    # A simple lemmatization function (for illustration purposes)
    if word.endswith("es"):
        return word[:-2]
    elif word.endswith("s"):
        return word[:-1]
    elif word.endswith("ing"):
        return word[:-3]
    return word

def analyze_morphemes(word, prefixes, root, suffixes):
    morphemes = []
    for prefix in prefixes:
        if word.startswith(prefix):
            morphemes.append(prefix)
            word = word[len(prefix):]
    morphemes.append(root)
    for suffix in suffixes:
        if word.endswith(suffix):
            morphemes.append(suffix)
            word = word[:-len(suffix)]
    return morphemes

text = "The quick brown foxes are jumping over the lazy dogs"
words = simple_tokenizer(text)

stemmed_words = [simple_porter_stemmer(word) for word in words]
lemmatized_words = [simple_wordnet_lemmatizer(word) for word in words]

print("Original words:", words)
print("Stemmed words:", stemmed_words)
print("Lemmatized words:", lemmatized_words)
print("\n")

word = "misunderstanding"
prefixes = ["mis"]
root = "understand"
suffixes = ["ing"]
morphemes = analyze_morphemes(word, prefixes, root, suffixes)

print("Word:", word)
print("Morphemes:", morphemes)
```

[ ]:

```
```