PANDIT DEENDAYAL ENERGY UNIVERSITY

Lab Report

Natural Language Processing Lab

(21IC403P)

By,

Aashi Shah

Roll No: 21BIT269D

VII Semester

Submitted to,

Dr. Hiren Thakkar(For H1H2)

Information and Communication Technology (ICT) School of Technology

Pandit Deendayal Energy University
School of Technology

Certificate

This is to certify that **Aashi Shah**, Roll Number **21BIT269D**, has successfully completed the lab work for the course Natural Language Processing (21IC403P) as part of Semester 7, Batch H2, during the academic year 2024-2025.

This lab work is in partial fulfillment of the requirements for the Bachelor of Technology (B.Tech) program in Information and Communication Technology, graduating year 2025.

Date: 9/11/2024

[Signature]
Name: Dr. Hiren Thakkar

Index

10	Create a Python program to implement and compare various NLP algorithms for tasks such as classification, clustering, and sentiment analysis.	18	
11	Develop a Python script to perform sentiment analysis on text data using lexicon-based methods and machine learning models.	19	
12	Write a Python program to apply deep learning models such as RNNs, LSTMs, or Transformers for NLP tasks, including experimenting with pre-trained models like BERT or GPT.	20	

Experiment 1: Text and PDF File Manipulation with Python

i. Write a Python script to read, write, and manipulate text files.
ii. Extract and process text from PDF files using Python libraries like PyPDF2 and pdfminer.

Objective:

- Learn to read, write, and manipulate text files.

```
file = open("song.txt","r")
print(file.read())
```

```
bye bye bye - Deadpool wolverine
crazy frog - axel f
```

```
file = open("song.txt", "w")
file.write("Pitbull - Hotel Room Services")
file.close()
file = open("song.txt", "r")
print(file.read())
```

```
Pitbull - Hotel Room Services
```

```
file = open("song.txt", "a")
file.write("Heather - Conan Gray")
file.close()
file = open("song.txt", "r")
print(file.read())
```

```
Pitbull - Hotel Room ServicesHeather - Conan Gray
```

```
def search_and_replace(file_path, search_word, replace_word):
    with open(file_path, 'r') as file:
        file_contents = file.read()
        updated_contents = file_contents.replace(search_word, replace_word)

    with open(file_path, 'w') as file:
        file.write(updated_contents)

file_path = 'song.txt'
search_word = 'Pitbull'
replace_word = 'Max verstappan'
search_and_replace(file_path, search_word, replace_word)
```

```
file = open("song.txt", "r")
print(file.read())
```

```
Max verstappan - Hotel Room ServicesHeather - Conan Gray
```

Extract and process text from PDF files.

```
pip install PyPDF2
```

```
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl.metadata (6.8 kB)
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 8.0 MB/s eta 0:00:00
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
```

```
from PyPDF2 import PdfReader # Use PdfReader instead of PdfFileReader
def text_extractor(path):
  with open(path, 'rb') as f:
    pdf = PdfReader(f)
    page = pdf.pages[0] # Use index to get page, remember Python indexes start at 0
    print(page)
    print('Page type: {}'.format(str(type(page))))
    text = page.extract_text() # Use extract_text instead of extractText
    print(text)

if __name__ == '__main__':
  path = 'check.pdf'
  text_extractor(path)
```

```
{'/Type': '/Page', '/Parent': IndirectObject(2, 0, 135708460044960), '/Resources': {'/Font': {'/F1': IndirectObject(5, 0, 135708460044960), '/F2': IndirectObject(9, 0, 135708460044960), '/F3': I
Page type: <class 'PyPDF2._page.PageObject'>
Vaishnavi Savaliya 21BIT268D
Aashi Shah 21BI T269D
Riya Patel 21BIT282D

1. Sentiment Analysis and Opinion Mining:
Fake review detection to enhance the reliability of online platforms (4.4)
2. Speech Recognition and Synthesis:
Automatic speech recognition (ASR) for Live Captioning and Transcription of Movies,
Lectures, and Podcast (2.1)
3. Text Summarization:
Summarization of long documents to design news aggregation (3.1)

This are the preferences we have selected in the respective order.
```

Email Extraction:

```
import imaplib
import json
import mailparser

# Your Gmail credentials (should use environment variables or a secure
method)
username = "aashi12e@gmail.com"
password = secret-key

# Connect to the Gmail IMAP server
mail = imaplib.IMAP4_SSL("imap.gmail.com")
mail.login(username, password)

# Select the mailbox you want to use. Use 'INBOX' to read inbox emails.
mail.select("inbox")

# Search for all emails and get the latest one
status, messages = mail.search(None, "ALL")

if status == "OK":
    # Convert messages to a list of email IDs and get the last email ID
    email_ids = messages[0].split()
    latest_email_id = email_ids[-1]

    # Fetch the latest email by ID
    status, msg_data = mail.fetch(latest_email_id, "(RFC822)")

    if status == "OK":
        # Get the email content
        raw_email = msg_data[0][1]

        # Decode the email content
        for encoding in ['utf-8', 'latin-1', 'iso-8859-1']:
            try:
                raw_email_decoded = raw_email.decode(encoding)
                break
            except UnicodeDecodeError:
                raw_email_decoded = None

        if raw_email_decoded:
            # Parse the email using mailparser
            email_message = mailparser.parse_from_string(raw_email_decoded)

            # Store email details in a dictionary (only body is extracted)
            email_details = {
                "from": email_message.from_,
                "subject": email_message.subject,
```

```
                "date": email_message.date,
                "body": email_message.text_plain if
email_message.text_plain else email_message.text_html
            }

            # Convert the email details to JSON format
            email_json = json.dumps(email_details, indent=4, default=str)

            # Print the JSON output
            print(email_json)
        else:
            print("Failed to decode the email content.")
    else:
        print("Failed to fetch the latest email.")
else:
    print("Failed to search emails.")

# Close the mailbox and logout
mail.close()
mail.logout()
```

Output :


```
{
    "from": [
        [
            "Google",
            "no-reply@accounts.google.com"
        ]
    ],
    "subject": "Security alert",
    "date": "2024-08-15 12:47:38",
    "body": [
        "[image: Google]\r\nApp password created to sign in to your account\r\n\r\n\r\naashi12e@gmail.com\r\nIf you didn't generate this password for nlp, someone else might be using\r\nyour acc
    ]
}
('BYE', [b'LOGOUT Requested'])
```

Experiment 2: Pattern Searching in Text Using Regular Expressions

i. Implement a Python program to search for patterns in text utilizing regular expressions.

Objective:

- Utilize regular expressions to find patterns in text.

```
[ ] import re

    chat1='Zomato: Hello, I am having an issue with the pizza I ordered my orderID is # 5739275029'
    pattern = 'order[^\d]*(\d*)'
    matches = re.findall(pattern, chat1)
    matches

    ['5739275029']

[ ] chat2='I have a problem with my order number 2840285673'
    pattern = 'order[^\d]*(\d*)'
    matches = re.findall(pattern, chat2)
    matches

    ['2840285673']

    chat3='Nykaa: My order 137957305 is having an issue, I was charged 500$ when online it says 450$'
    pattern = 'order[^\d]*(\d*)'
    matches = re.findall(pattern, chat3)
    matches

    ['137957305']

[ ] def get_pattern_match(pattern, text):
        matches = re.findall(pattern, text)
        if matches:
            return matches[0]

[ ] get_pattern_match('order[^\d]*(\d*)', chat1)

    '5739275029'
```

```
[ ] chat1 = 'This is your order number 1235678912, aashi12e@gmail.com'
    chat2 = 'contact us: (61)-123-234563, aashi12e@gmail.com'
    chat3 = 'Hello, phone: 5368952167 email: aashi12e@gmail.com'

[ ] get_pattern_match('[a-zA-Z0-9_.]*@[a-z]*\.[a-zA-Z0-9]*',chat1)

    'aashi12e@gmail.com'

[ ] get_pattern_match('[a-zA-Z0-9_.]*@[a-z]*\.[a-zA-Z0-9]*',chat2)

    'aashi12e@gmail.com'

[ ] get_pattern_match('[a-zA-Z0-9_.]*@[a-z]*\.[a-zA-Z0-9]*',chat3)

    'aashi12e@gmail.com'

[ ] get_pattern_match('(\d{10})|(\(\d{3}\)-\d{3}-\d{4})',chat1)

    ('1235678912', '')

[ ] get_pattern_match('(\d{10})|(\(\d{2}\)-\d{3}-\d{4})', chat2)

    ('', '(61)-123-2345')

[ ] get_pattern_match('(\d{10})|(\(\d{5}\)-\d{3}-\d{4})', chat3)

    ('5368952167', '')

[ ] text = '''
```

```
[ ] F1 Team : Red Bull Racing
    They're so fast you'd think they installed a jet engine instead of a hybrid one.
    Their biggest challenge? Finding room for all those trophies.

    F1 Team : Mercedes-AMG Petronas
    Once the kings of F1, now they're just trying to figure out why their silver arrows aren't as sharp.
    Still, they make a pretty good backup plan for the podium.

    F1 Team : Scuderia Ferrari
    Ferrari continues to blend nostalgia with heartbreak.
    They might not win every race, but at least their fans have perfected the art of "maybe next year."
    '''
    pattern = 'F1 Team : ([^\n]*)'

    re.findall(pattern, text)
```

```
['Red Bull Racing', 'Mercedes-AMG Petronas', 'Scuderia Ferrari']
```

```
text = '''
As of the FY2024 Q2,
Formula 1 reported strong financial performance,
with a 20% increase in total revenue compared to the same period in FY2023.
This growth was driven by additional races, higher media rights fees,
and sponsorship revenues.
Despite missing earnings per share (EPS) estimates,
the overall revenue exceeded expectations, reaching nearly $988 million.
'''

pattern = 'FY(\d{4} (?:Q[1-4]))'
matches = re.findall(pattern, text)
matches
```

```
['2024 Q2']
```

Experiment 3: Ultra-fast Tokenization

i. Create a Python script to implement and compare ultra-fast tokenization techniques using libraries like NLTK, SpaCy, and FastText.

Objective:

- Implement and compare different tokenization methods.

Step 3: Tokenization using NLTK

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

    # Word tokenization using NLTK
    text = """There are multiple ways we can perform tokenization on given text data.
    We can choose any method based on language, library, and purpose of modeling."""
    tokens = word_tokenize(text)
    print("Word tokens with NLTK:", tokens)

    # Sentence tokenization using NLTK
    sentences = sent_tokenize(text)
    print("Sentence tokens with NLTK:", sentences)
```

```
Word tokens with NLTK: ['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'data', '.', 'We', 'can', 'choose', 'any', 'method', 'based',
Sentence tokens with NLTK: ['There are multiple ways we can perform tokenization on given text data.', 'We can choose any method based on language, library, and purpose of modeling.']
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\aashi\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

The NLTK word_tokenize function handles punctuation, making it more effective for NLP tasks. sent_tokenize is also used for sentence tokenization.

Step 4: Tokenization using SpaCy (English and Multilingual)

```
import spacy

# English Tokenization with SpaCy
nlp_en = spacy.blank("en")
text_en = """There are multiple ways we can perform tokenization on given text data."""
doc_en = nlp_en(text_en)
tokens_en = [token.text for token in doc_en]
print("Tokens with SpaCy (English):", tokens_en)

# Hindi Tokenization with SpaCy
nlp_hi = spacy.blank("hi")
text_hi = """ऐसे कई तरीके हैं जिनसे हम दिए गए टेक्स्ट डेटा पर टोकनाइजेशन कर सकते हैं।"""
doc_hi = nlp_hi(text_hi)
tokens_hi = [token.text for token in doc_hi]
print("Tokens with SpaCy (Hindi):", tokens_hi)

# Gujarati Tokenization with SpaCy
nlp_gu = spacy.blank("gu")
text_gu = """આપેલ ટેક્સ્ટ ડેટા પર આપણે ટોકનાઇઝેશન કરી શકીએ તે ઘણી રીતો છે."""
doc_gu = nlp_gu(text_gu)
tokens_gu = [token.text for token in doc_gu]
print("Tokens with SpaCy (Gujarati):", tokens_gu)
```

SpaCy is a powerful tool for tokenization, supporting multiple languages. Here we use it to tokenize English, Hindi, and Gujarati texts.

Step 5: Enhancements

```
from tokenizers import Tokenizer

# Load a pre-trained tokenizer
tokenizer = Tokenizer.from_pretrained("bert-base-uncased")

text = """There are multiple ways we can perform tokenization on given text data."""
output = tokenizer.encode(text)

print("Ultra-fast tokenization (Hugging Face):", output.tokens)
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
Ultra-fast tokenization (Hugging Face): ['[CLS]', 'there', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'token', '##ization', 'on', 'given', 'text', 'data', '.', '[SEP]']
C:\Users\aashi\AppData\Roaming\Python\Python310\site-packages\huggingface_hub\file_download.py:139: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate developer mode, see this article: https://docs.mic
  warnings.warn(message)
```

Experiment 4: Stemming and Lemmatization

i. Develop a Python program to apply stemming and lemmatization to normalize text data.

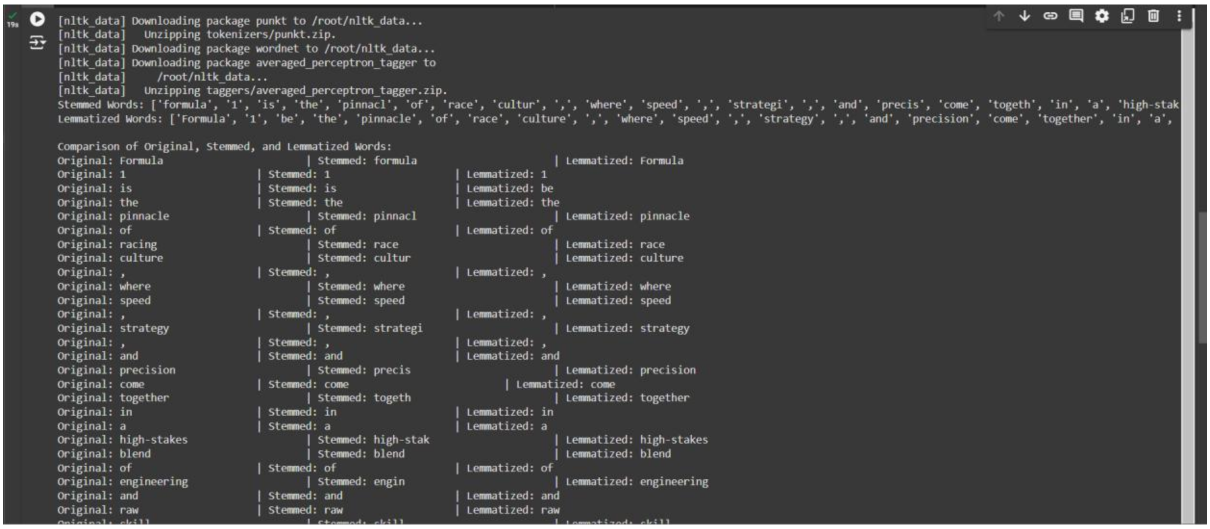
Objective:

- Apply stemming and lemmatization to normalize text.

```
# Stemming and Lemmatization Techniques in Text Preprocessing
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet

# Download required resources
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Function to get WordNet POS tags for more accurate lemmatization
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()  # Get the first
letter of POS tag
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)  # Default to NOUN if not
found

# Function for stemming
def stem_words(text):
    tokens = word_tokenize(text)
    return [stemmer.stem(word) for word in tokens]

# Function for lemmatization (with POS tagging)
def lemmatize_words(text):
    tokens = word_tokenize(text)
    return [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for
word in tokens]

# Example text
```

```
text = """Formula 1 is the pinnacle of racing culture, where speed,
strategy, and precision come together in a high-stakes blend of
engineering and raw skill.
 With cars that push the limits of physics and drivers that embody
relentless competitiveness, F1 captures the fascination of millions
worldwide."""

# Perform stemming
stemmed_words = stem_words(text)
print("Stemmed Words:", stemmed_words)

# Perform lemmatization
lemmatized_words = lemmatize_words(text)
print("Lemmatized Words:", lemmatized_words)

# BONUS: Compare stemmed and lemmatized results side by side
print("\nComparison of Original, Stemmed, and Lemmatized Words:")
tokens = word_tokenize(text)
for i, token in enumerate(tokens):
    print(f"Original: {token} \t\t\t| Stemmed: {stemmed_words[i]}
\t\t\t| Lemmatized: {lemmatized_words[i]}")
```

Output :



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
Stemmed Words: ['formula', '1', 'is', 'the', 'pinnacl', 'of', 'race', 'cultur', ',', 'where', 'speed', ',', 'strategi', ',', 'and', 'precis', 'come', 'togeth', 'in', 'a', 'high-stak
Lemmatized Words: ['Formula', '1', 'be', 'the', 'pinnacle', 'of', 'race', 'culture', ',', 'where', 'speed', ',', 'strategy', ',', 'and', 'precision', 'come', 'together', 'in', 'a',

Comparison of Original, Stemmed, and Lemmatized Words:
Original: Formula            | Stemmed: formula            | Lemmatized: Formula
Original: 1          | Stemmed: 1          | Lemmatized: 1
Original: is         | Stemmed: is         | Lemmatized: be
Original: the        | Stemmed: the        | Lemmatized: the
Original: pinnacle           | Stemmed: pinnacl            | Lemmatized: pinnacle
Original: of         | Stemmed: of         | Lemmatized: of
Original: racing             | Stemmed: race               | Lemmatized: race
Original: culture            | Stemmed: cultur             | Lemmatized: culture
Original: ,          | Stemmed: ,          | Lemmatized: ,
Original: where              | Stemmed: where              | Lemmatized: where
Original: speed              | Stemmed: speed              | Lemmatized: speed
Original: ,          | Stemmed: ,          | Lemmatized: ,
Original: strategy           | Stemmed: strategi           | Lemmatized: strategy
Original: ,          | Stemmed: ,          | Lemmatized: ,
Original: and        | Stemmed: and        | Lemmatized: and
Original: precision          | Stemmed: precis             | Lemmatized: precision
Original: come       | Stemmed: come               | Lemmatized: come
Original: together           | Stemmed: togeth             | Lemmatized: together
Original: in         | Stemmed: in         | Lemmatized: in
Original: a          | Stemmed: a          | Lemmatized: a
Original: high-stakes        | Stemmed: high-stak          | Lemmatized: high-stakes
Original: blend              | Stemmed: blend              | Lemmatized: blend
Original: of         | Stemmed: of         | Lemmatized: of
Original: engineering        | Stemmed: engin              | Lemmatized: engineering
Original: and        | Stemmed: and        | Lemmatized: and
Original: raw        | Stemmed: raw        | Lemmatized: raw
```

Experiment 5: Vocabulary Matching Techniques

i. Write a Python script to explore and implement various vocabulary matching techniques, including word embeddings and similarity measures.

Objective:

- Explore and implement various vocabulary matching techniques.

```
import Levenshtein
import gensim.downloader as api
from scipy.spatial.distance import cosine

# Load a pre-trained Word2Vec model
model = api.load("word2vec-ruscorpora-300")
def exact_match(word1, word2):
    """Check for exact match between two words."""
    return word1.lower() == word2.lower()  # Ignore case sensitivity

def levenshtein_distance(word1, word2):
    """Calculate Levenshtein distance between two words."""
    return Levenshtein.distance(word1, word2)

def jaccard_match(word1, word2):
    """Calculate Jaccard similarity between two words."""
    set1 = set(word1)
    set2 = set(word2)
    intersection = len(set1.intersection(set2))
    union = len(set1.union(set2))
    return intersection / union if union > 0 else 0  # Avoid division
by zero

def cosine_similarity(word1, word2):
    """Calculate cosine similarity between two words using their word
vectors."""
    try:
        # Get the word vectors
        vector1 = model[word1.lower()]  # Ensure lower case
        vector2 = model[word2.lower()]  # Ensure lower case

        # Calculate cosine similarity (1 - cosine distance)
        return 1 - cosine(vector1, vector2)
    except KeyError:
        return None  # Return None if the word is not in the model
# List of word pairs for comparison
word_pairs = [
    ("happy", "joyful"),         # Synonyms
    ("sad", "unhappy"),          # Related antonyms
    ("light", "dark"),           # Antonyms
```

```
    ("car", "automobile"),         # Synonyms
    ("teacher", "student"),        # Related roles
    ("fast", "quick"),             # Synonyms
    ("big", "large"),              # Synonyms
    ("cold", "ice"),               # Related concepts
    ("run", "running"),            # Different forms of the same word
    ("strong", "powerful"),        # Synonyms
    ("beautiful", "ugly"),         # Antonyms
    ("child", "adult"),            # Different life stages
    ("friend", "enemy"),           # Opposites
    ("smart", "intelligent"),      # Synonyms
    ("fish", "swim"),              # Related actions
    ("love", "hate")               # Opposites
]

# Evaluate each pair of words
for word1, word2 in word_pairs:
    print(f"Itr :: {word1} :: {word2}\n")

    print(f"Exact Match :: {exact_match(word1, word2)} ")
    print(f"Levenshtein Distance :: {levenshtein_distance(word1,
word2)}")
    print(f"Jaccard Similarity :: {jaccard_match(word1, word2)}")

    cosine_sim = cosine_similarity(word1, word2)
    if cosine_sim is not None:
        print(f"Cosine Similarity :: {cosine_sim}")
    else:
        print(f"Cosine Similarity :: One or both words not in the
model.")

    print("\n")
```

Output :

```
Itr :: happy :: joyful

Exact Match :: False
Levenshtein Distance :: 6
Jaccard Similarity :: 0.1111111111111111
Cosine Similarity :: One or both words not in the model.


Itr :: sad :: unhappy

Exact Match :: False
Levenshtein Distance :: 6
Jaccard Similarity :: 0.125
Cosine Similarity :: One or both words not in the model.


Itr :: light :: dark

Exact Match :: False
Levenshtein Distance :: 5
Jaccard Similarity :: 0.0
Cosine Similarity :: One or both words not in the model.


Itr :: car :: automobile

Exact Match :: False
Levenshtein Distance :: 10
Jaccard Similarity :: 0.09090909090909091
Cosine Similarity :: One or both words not in the model.


Itr :: teacher :: student

Exact Match :: False
Levenshtein Distance :: 7
Jaccard Similarity :: 0.2
```

Experiment 6: Part of Speech Tagging

i. Create a Python program to automatically tag parts of speech in raw text files using libraries like NLTK and SpaCy.

Objective:

- Automatically tag parts of speech in raw text files.

```
import nltk
import spacy
nltk.download('punkt', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)
def pos_tag_nltk(text):
    words = nltk.word_tokenize(text)
    pos_tags = nltk.pos_tag(words)
    return pos_tags

# Function to perform POS tagging using spaCy
def pos_tag_spacy(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc]
    return pos_tags
# Function to read text from a file
def read_text_file(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
        return None
# Function to display POS tags
def display_pos_tags(pos_tags, library_name):
    print(f"\nPOS tagging using {library_name}:\n")
    for word, tag in pos_tags:
        print(f"{word} -> {tag}")
# Path to the raw text file
file_path = 'sample.txt'  # Replace with your file path
# Read the text from the file
text_content = read_text_file(file_path)

if text_content:
    # Perform POS tagging using NLTK
    nltk_pos_tags = pos_tag_nltk(text_content)
    display_pos_tags(nltk_pos_tags, "NLTK")

    # Perform POS tagging using spaCy
    spacy_pos_tags = pos_tag_spacy(text_content)
    display_pos_tags(spacy_pos_tags, "spaCy")
```

Output :

```
POS tagging using NLTK:

Formula -> NNP
1 -> CD
is -> VBZ
the -> DT
pinnacle -> NN
of -> IN
racing -> VBG
culture -> NN
, -> ,
where -> WRB
speed -> NN
, -> ,
strategy -> NN
, -> ,
and -> CC
precision -> NN
come -> VBP
together -> RB
in -> IN
a -> DT
high-stakes -> JJ
blend -> NN
of -> IN
engineering -> NN
and -> CC
raw -> JJ
skill -> NN
. -> .
With -> IN
cars -> NNS
that -> WDT
push -> VBP
the -> DT
limits -> NNS
```

Experiment 7: Visualizing POS and Named Entity Recognition (NER)

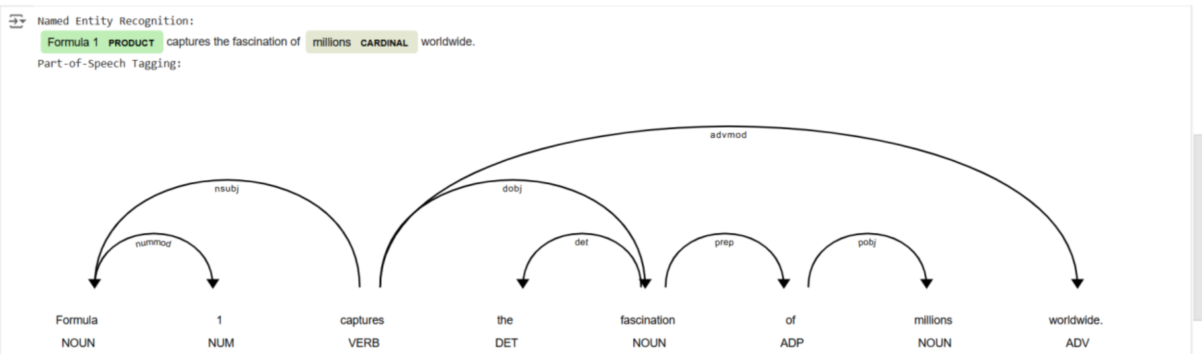
i. Implement a Python script to visualize POS tagging and NER using tools like SpaCy and Matplotlib.

Objective:

- Create visualizations for POS tagging and NER.

```
import spacy
from spacy import displacy

# Load the spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except Exception as e:
    print(f"Error loading spaCy model: {e}")
def visualize_pos_and_ner(text):
    # Process the text
    doc = nlp(text)

    # Display Named Entity Recognition (NER)
    print("Named Entity Recognition:")
    displacy.render(doc, style='ent', jupyter=True)  # Use jupyter=True for Jupyter Notebook

    # Display Part-of-Speech (POS) tagging (dependency parse)
    print("Part-of-Speech Tagging:")
    displacy.render(doc, style='dep', jupyter=True)  # Use jupyter=True for Jupyter Notebook
# Sample text
sample_text = "Formula 1 captures the fascination of millions worldwide."

# Visualize POS and NER
visualize_pos_and_ner(sample_text)
```

Output :



Experiment 8: Text Classification Algorithms

i. Develop a Python program to implement text classification algorithms such as Naive Bayes, SVM, and neural networks, and evaluate their performance.

Objective:

Implement text classification algorithms and evaluate their performance.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Load and split dataset
newsgroups = fetch_20newsgroups(subset='all')
X = newsgroups.data
y = newsgroups.target

# Convert text to TF-IDF features
tfidf = TfidfVectorizer(stop_words='english', max_features=10000)
X_tfidf = tfidf.fit_transform(X)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y,
test_size=0.2, random_state=42)

# Function to evaluate models
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    report = classification_report(y_test, predictions)
    return accuracy, report

# Model 1: Naive Bayes
print("Naive Bayes Model")
nb_model = MultinomialNB()
nb_accuracy, nb_report = evaluate_model(nb_model, X_train, X_test,
y_train, y_test)
print(f"Accuracy: {nb_accuracy}")
print(nb_report)
```

```
# Model 2: Support Vector Machine (SVM)
print("Support Vector Machine Model")
svm_model = SVC(kernel='linear', C=1)
svm_accuracy, svm_report = evaluate_model(svm_model, X_train, X_test,
y_train, y_test)
print(f"Accuracy: {svm_accuracy}")
print(svm_report)

# Model 3: Neural Network
print("Neural Network Model")
# Encode labels for neural network
encoder = LabelEncoder()
y_train_enc = to_categorical(encoder.fit_transform(y_train))
y_test_enc = to_categorical(encoder.transform(y_test))

# Define the neural network
nn_model = Sequential()
nn_model.add(Dense(512, input_shape=(X_train.shape[1],),
activation='relu'))
nn_model.add(Dense(256, activation='relu'))
nn_model.add(Dense(y_train_enc.shape[1], activation='softmax'))

# Compile the neural network
nn_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the neural network
nn_model.fit(X_train, y_train_enc, epochs=5, batch_size=128,
validation_split=0.1)

# Evaluate the neural network
nn_loss, nn_accuracy = nn_model.evaluate(X_test, y_test_enc)
print(f"Neural Network Accuracy: {nn_accuracy}")
```

Output :

```
Naive Bayes Model
Accuracy: 0.870291777188329
              precision    recall  f1-score   support

           0       0.83      0.88      0.86       151
           1       0.77      0.84      0.80       202
           2       0.81      0.82      0.81       195
           3       0.64      0.80      0.71       183
           4       0.90      0.85      0.88       205
           5       0.90      0.85      0.87       215
           6       0.85      0.76      0.81       193
           7       0.91      0.94      0.92       196
           8       0.89      0.94      0.92       168
           9       0.97      0.94      0.95       211
          10       0.90      0.97      0.94       198
          11       0.95      0.95      0.95       201
          12       0.92      0.77      0.84       202
          13       0.97      0.89      0.93       194
          14       0.90      0.97      0.93       189
          15       0.77      0.98      0.86       202
          16       0.85      0.96      0.90       188
          17       0.95      0.98      0.97       182
          18       0.96      0.79      0.87       159
          19       0.92      0.36      0.52       136

    accuracy                           0.87      3770
   macro avg       0.88      0.86      0.86      3770
weighted avg       0.88      0.87      0.87      3770
```

```
Support Vector Machine Model
Accuracy: 0.8986737400530505
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       151
           1       0.75      0.85      0.80       202
           2       0.85      0.84      0.84       195
           3       0.69      0.77      0.73       183
           4       0.88      0.84      0.86       205
           5       0.89      0.84      0.86       215
           6       0.81      0.82      0.81       193
           7       0.91      0.93      0.92       196
           8       0.99      0.94      0.97       168
           9       0.94      0.96      0.95       211
          10       0.97      0.95      0.96       198
          11       0.98      0.95      0.96       201
          12       0.85      0.85      0.85       202
          13       0.90      0.92      0.91       194
          14       0.96      0.96      0.96       189
          15       0.95      0.98      0.96       202
          16       0.96      0.95      0.95       188
          17       0.99      0.98      0.99       182
          18       0.94      0.91      0.92       159
          19       0.89      0.84      0.86       136

    accuracy                           0.90      3770
   macro avg       0.90      0.90      0.90      3770
weighted avg       0.90      0.90      0.90      3770
```

```
Neural Network Model
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pr
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
106/106 ━━━━━━━━━━━━━━━━━━━━ 12s 100ms/step - accuracy: 0.5408 - loss: 2.2147 - val_accuracy: 0.8912 - val_loss: 0.4387
Epoch 2/5
106/106 ━━━━━━━━━━━━━━━━━━━━ 10s 90ms/step - accuracy: 0.9604 - loss: 0.1778 - val_accuracy: 0.9098 - val_loss: 0.3374
Epoch 3/5
106/106 ━━━━━━━━━━━━━━━━━━━━ 9s 83ms/step - accuracy: 0.9948 - loss: 0.0376 - val_accuracy: 0.9151 - val_loss: 0.3355
Epoch 4/5
106/106 ━━━━━━━━━━━━━━━━━━━━ 11s 91ms/step - accuracy: 0.9975 - loss: 0.0156 - val_accuracy: 0.9131 - val_loss: 0.3360
Epoch 5/5
106/106 ━━━━━━━━━━━━━━━━━━━━ 10s 91ms/step - accuracy: 0.9983 - loss: 0.0107 - val_accuracy: 0.9145 - val_loss: 0.3473
118/118 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8971 - loss: 0.3767
Neural Network Accuracy: 0.9039787650108337
```

Experiment 9: Non-negative Matrix Factorization (NMF) for Topic Modeling

i. Write a Python script to apply Non-negative Matrix Factorization (NMF) for topic modeling and document clustering.

Objective:

- Apply NMF for topic modeling and document clustering.

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

# Sample data - more varied topics
documents = [
    "The person is in the house.",
    "The house is beautiful and big.",
    "Someone is in the house.",
    "The night sky is clear and full of stars.",
    "Stars are shining in the clear night.",
    "The night is quiet and calm.",
    "Today is a bright and sunny day.",
    "Someone likes sunny days in the house.",
]

# Vectorize text with TF-IDF, removing English stop words
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# Apply NMF with 2 topics
nmf = NMF(n_components=2, random_state=42)
nmf.fit(X)

# Display topics
print("Topics identified using NMF:")
for index, topic in enumerate(nmf.components_):
    print(f'\nTopic {index + 1}:')
    # Display top 5 words per topic
    top_words = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[-5:]]
    print(", ".join(top_words))
```

Output :

```
Topics identified using NMF:

Topic 1:
sunny, big, beautiful, person, house

Topic 2:
shining, sky, clear, stars, night
```

Experiment 10: Exploring Different Algorithms

i. Create a Python program to implement and compare various NLP algorithms for tasks such as classification, clustering, and sentiment analysis.

Objective:

- Implement and compare various NLP algorithms for tasks like classification, clustering, and sentiment analysis.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Sample data
texts = ["I love cars.", "Lewis Hamilton is GOAT!", "I like dumplings."]
labels = [1, 1, 0]

# Split data
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2)

# Vectorize text
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vec, y_train)
nb_predictions = nb_classifier.predict(X_test_vec)
nb_accuracy = accuracy_score(y_test, nb_predictions)

# SVM classifier
svm_classifier = SVC()
svm_classifier.fit(X_train_vec, y_train)
svm_predictions = svm_classifier.predict(X_test_vec)
svm_accuracy = accuracy_score(y_test, svm_predictions)

# Compare results
print(f'Naive Bayes Accuracy: {nb_accuracy}')
print(f'SVM Accuracy: {svm_accuracy}')
```

Output:

```
Naive Bayes Accuracy: 0.0
SVM Accuracy: 1.0
```

Experiment 11: Sentiment Analysis

i. Develop a Python script to perform sentiment analysis on text data using lexicon-based methods and machine learning models.

Objective:

- Perform sentiment analysis on text data using various methods.

```python
from textblob import TextBlob

# Sample F1-related texts
f1_texts = [
    "That was an incredible race! The strategy was perfect and the overtakes were amazing.",
    "I’m disappointed with the team’s performance today; the car setup just wasn’t right.",
    "What a historic win for the driver! Absolutely deserved and well-earned."
]

print("F1 Sentiment Analysis Results:\n")

for text in f1_texts:
    # Create TextBlob object and analyze sentiment
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    subjectivity = blob.sentiment.subjectivity

    # Classify sentiment based on polarity score
    if polarity > 0:
        sentiment_label = "Positive"
    elif polarity < 0:
        sentiment_label = "Negative"
    else:
        sentiment_label = "Neutral"

    # Display results
    print(f"Text: {text}")
    print(f"Polarity: {polarity}, Subjectivity: {subjectivity}")
    print(f"Sentiment: {sentiment_label}\n")
```

Output :

```
F1 Sentiment Analysis Results:

Text: That was an incredible race! The strategy was perfect and the overtakes were amazing.
Polarity: 0.8666666666666667, Subjectivity: 0.9333333333333332
Sentiment: Positive

Text: I’m disappointed with the team’s performance today; the car setup just wasn’t right.
Polarity: -0.23214285714285715, Subjectivity: 0.6428571428571428
Sentiment: Negative

Text: What a historic win for the driver! Absolutely deserved and well-earned.
Polarity: 0.39999999999999997, Subjectivity: 0.43333333333333335
Sentiment: Positive
```

Experiment 12: Deep Learning in NLP

i. Write a Python program to apply deep learning models such as RNNs, LSTMs, or Transformers for NLP tasks, including experimenting with pre-trained models like BERT or GPT.

Objective:

- Apply deep learning models for NLP tasks.

```python
import torch
from transformers import pipeline

# Load pre-trained model and tokenizer for sentiment analysis
classifier = pipeline('sentiment-analysis')

# Sample F1-related texts for sentiment analysis
f1_texts = [
    "The race was absolutely thrilling! Great strategy by the team.",
    "I'm disappointed with the driver's performance today.",
    "What an incredible win! He really deserved that podium finish."
]

# Perform sentiment analysis on each F1-related text
print("F1 Sentiment Analysis Results:\n")
for text in f1_texts:
    result = classifier(text)
    label = result[0]['label']
    score = result[0]['score']

    # Display each text and its corresponding sentiment
    print(f"Text: {text}")
    print(f"Sentiment: {label} (Confidence: {score:.2f})\n")
```

Output :

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggingface.co/distilbert/distilbert-base-uncased-finetunec
Using a pipeline without specifying a model name and revision in production is not recommended.
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 629/629 [00:00<00:00, 33.6kB/s]
model.safetensors: 100% 268M/268M [00:04<00:00, 120MB/s]
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.73kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 658kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. Th
  warnings.warn(
F1 Sentiment Analysis Results:

Text: The race was absolutely thrilling! Great strategy by the team.
Sentiment: POSITIVE (Confidence: 1.00)

Text: I'm disappointed with the driver's performance today.
Sentiment: NEGATIVE (Confidence: 1.00)

Text: What an incredible win! He really deserved that podium finish.
Sentiment: POSITIVE (Confidence: 1.00)
```