Group 44

Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

CERTIFICATE

Certify that Mr./Miss Arnav Malvia of Computer Engg Department, Semester VII with Roll No. 2103109 has completed a course of the necessary experiments in the subject Natural Language Processing under my supervision in the **Thadomal Shahani Engineering College** Laboratory in the year 2024 - 2025

Teacher In-Charge

Head of the Department

Date 15/10/2024

Principal

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1.	Study various applications of NLP and NLP tools			
2.	Study various applications of NLP and formulate the Problem Statement for Mini Project.			
3.	To implement NLP Pre-processing tasks.			
4.	To implement Advanced text pre-processing techniques.			
5.	To implement shallow and deep parsing.			
6.	To implement Text processing models.			
7.	To study and implement Morphological Analysis of English sentences.			
8.	To implement Part of Speech-tagging using HMM.			
9.	To implement Named Entity Recognition for given real world application.			
10.	Experimenting with an Advanced NLP problem of your choice using hugging face transformers, spaCy, NLTK, etc for fine tuning LLMs for standard NLP problems			
11.	Mini Project			
12.	Assignment			

Experiment 1

Aim: Study various applications of NLP and NLP Tools

Theory:

Natural Language Processing (NLP) is a field of artificial intelligence that deals with the interaction between computers and human (natural) languages. It aims to enable computers to understand, interpret, and generate human language. NLP tasks can range from simple tasks like text classification to more complex ones like machine translation and question answering.

NLP Libraries

1. NLTK:
   NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces and libraries for tasks such as tokenization, stemming, lemmatization, part-of-speech tagging, and parsing. NLTK is widely used in natural language processing (NLP) research and education.
   The role of NLTK (Natural Language Toolkit) in text analysis are as follows:
   - Tokenization: NLTK offers functions to split text into tokens, such as words or sentences, facilitating further analysis by breaking down the text into manageable units.
   - Stemming and Lemmatization: NLTK provides algorithms for reducing words to their root forms (stemming) or canonical forms (lemmatization), aiding in text normalization and improving analysis accuracy.
   - Part-of-Speech Tagging: NLTK includes tools for assigning grammatical tags to words in a text corpus, enabling syntactic analysis and understanding of sentence structures.
   - Parsing: Parsing is the process of analyzing the structure of sentences to understand how words relate to each other grammatically. NLTK supports parsing techniques for analyzing the grammatical structure of sentences, facilitating deeper linguistic analysis and parsing tasks.
   - Named Entity Recognition (NER): NLTK offers functionality for identifying and classifying named entities (such as names of persons, organizations, or locations) within text data, enabling entity extraction and information retrieval tasks.

2. spaCy:
   spaCy is a fast and efficient NLP library designed for production use. It offers pre-trained models and robust features for tasks like tokenization, named entity recognition (NER), dependency parsing, and word vectors. spaCy's performance and usability make it a popular choice for building NLP applications.
   The role of spaCy in text analysis are as follows:

- Tokenization: spaCy provides efficient tokenization algorithms to split text into individual tokens (words or subwords), facilitating subsequent analysis by breaking down text into manageable units.
- Named Entity Recognition (NER): spaCy offers built-in models for identifying and classifying named entities (such as names of persons, organizations, or locations) within text data, enabling extraction of relevant information and entity-level analysis.
- Dependency Parsing: spaCy includes advanced algorithms for dependency parsing, which analyze the syntactic structure of sentences to determine the relationships between words and their dependencies, aiding in understanding sentence semantics and structure.
- Part-of-Speech (POS) Tagging: spaCy's models assign part-of-speech tags to words in a sentence, providing information about their syntactic roles and grammatical properties, which is useful for various NLP tasks such as syntactic analysis and semantic understanding.
- Word Vectors: spaCy offers pre-trained word vectors (word embeddings) that capture semantic similarities and relationships between words in a text corpus, enabling tasks such as similarity matching, document classification, and language modeling.

3. Hugging Face Transformers:
   Hugging Face Transformer is a library built on top of PyTorch and TensorFlow for working with transformer-based models, such as BERT, GPT, and RoBERTa. It provides pre-trained models and tools for fine-tuning, inference, and generation tasks in NLP, including text classification, question answering, and text generation.
   The role of PyNLPl in text analysis are as follows:
   - Pre-Trained Models: Hugging Face Transformers provides access to a vast repository of pre-trained transformer-based models, including BERT, GPT, and RoBERTa, for various natural language processing (NLP) tasks.
   - Fine-Tuning Capabilities: The library offers tools and utilities for fine-tuning pre-trained models on specific tasks or datasets, enabling users to customize models for their specific applications and improve performance.
   - Inference Support: Hugging Face Transformers supports inference with pre-trained models, allowing users to make predictions or generate text using the models without the need for additional training, facilitating quick deployment in production environments.
   - Wide Range of NLP Tasks: Users can leverage Hugging Face Transformers for a diverse set of NLP tasks, including text classification, question answering, named entity recognition, machine translation, and text generation.
   - Compatibility and Flexibility: Built on top of PyTorch and TensorFlow, Hugging Face Transformers is compatible with both deep learning frameworks, providing flexibility for users to choose their preferred backend and integrate seamlessly into their existing workflows.

4. AllenNLP:
   AllenNLP is a deep learning library built on top of PyTorch designed for NLP research and development. It provides pre-built models and components for tasks like text classification, named entity recognition, semantic role labeling, and machine reading comprehension.
   ELMo (Embeddings from Language Models) is a deep contextualized word representation technique that captures word meaning by considering the entire sentence context, enhancing NLP tasks' accuracy and performance, is also developed by AllenNLP.
   The role of Gensim in text analysis are as follows:
   - Pre-built Models: AllenNLP offers a collection of pre-trained deep learning models for a variety of natural language processing (NLP) tasks such as text classification, named entity recognition (NER), semantic role labeling (SRL), and machine reading comprehension (MRC). ELMo
   - PyTorch Integration: AllenNLP is built on top of PyTorch, a popular deep learning framework, allowing users to leverage PyTorch's flexibility and efficiency for building and training custom NLP models.
   - Modular Components: AllenNLP provides modular components and abstractions, allowing users to easily build and customize their own NLP models by combining different modules, such as embedding layers, recurrent neural networks (RNNs), and attention mechanisms.

<u>Cloud-Based NLP Platforms</u>

1. Google Cloud NLP:
   The Google Natural Language API is an easy to use interface to a set of powerful NLP models which have been pre-trained by Google to perform various tasks. As these models have been trained on enormously large document corpuses, their performance is usually quite good as long as they are used on datasets that do not make use of a very idiosyncratic language.The biggest advantage of using these pre-trained models via the API is, that no training dataset is needed. The API allows the user to immediately start making predictions, which can be very valuable in situations where little labeled data is available.

   The Natural Language API comprises five different services:

   - Syntax Analysis - For a given text, Google's syntax analysis will return a breakdown of all words with a rich set of linguistic information for each token.

- Sentiment Analysis - Google's sentiment analysis will provide the prevailing emotional opinion within a provided text. The API returns two values: The "score" describes the emotional leaning of the text from -1 (negative) to +1 (positive), with 0 being neutral.
- Entity Analysis - Entity Analysis is the process of detecting known entities like public figures or landmarks from a given text. Entity detection is very helpful for all kinds of classification and topic modeling tasks.The Google Natural Language API provides some basic information about each detected entity and even provides a link to the respective Wikipedia article if it exists.
- Entity Sentiment Analysis - If there are models for entity detection and sentiment analysis, it's only natural to go a step further and combine them to detect the prevailing emotions towards the different entities in a text.While the Sentiment Analysis API finds all displays of emotion in the document and aggregates them, the Entity Sentiment Analysis tries to find the dependencies between different parts of the document and the identified entities and then attributes the emotions in these text segments to the respective entities.
- Text Classification - The Google Natural language API comes with a plug-and-play text classification model.The model is trained to classify the input documents into a large set of categories. The categories are structured hierarchical, e.g. the Category "Hobbies & Leisure" has several sub-categories, one of which would be "Hobbies & Leisure/Outdoors" which itself has sub-categories like "Hobbies & Leisure/Outdoors/Fishing."

2. AWS Comprehend:

Amazon Comprehend is a natural language processing (NLP) service that uses machine learning (ML) to analyze text and extract insights. It can identify elements like entities, key phrases, sentiments, and language.

Here are some benefits of using Amazon Comprehend:

- Discover relationships in text
- Amazon Comprehend can find relationships in text from customer support incidents, product opinions, social media feeds, and more.
- Easy to use
- Amazon Comprehend has an easy-to-use interface that allows developers and data scientists to quickly incorporate text analysis into their applications.
- No machine learning experience required
- Users don't need to provision any servers or have machine learning experience to use Amazon Comprehend.

Some of the features of Amazon Comprehend include:

- Custom Entity Recognition
- Custom Classification
- Keyphrase Extraction
- Sentiment Analysis
- Entity Recognition
- PII Identification and Redaction
- Toxicity Detection
- Prompt Safety Classification
- Events Detection
- Language Detection

3. Azure Text Analytics:
   Azure Text Analytics is a cloud-based AI service that uses Natural Language Processing (NLP) to analyze and extract information from text data. It's part of the Azure Cognitive Service for Language.
   Here are some of the capabilities of Azure Text Analytics:
   - Sentiment analysis: Detects sentiment labels at the sentence or document level
   - Key phrase extraction: Extracts key phrases from text
   - Entity recognition: Identifies different entities in text and categorizes them into predefined classes or types
   - Language detection: Identifies the language for a given text input
   - Personally identifiable information (PII) entity recognition: Identifies PII
   - Text analytics for health: Provides text analytics for health
   - Custom named entity recognition: Provides custom named entity recognition
   - Custom text classification: Provides custom text classification
   - Extractive text summarization: Provides extractive text summarization
   - Abstractive text summarization: Provides abstractive text summarization

Experiment 2

Aim: Study various applications of NLP and formulate the Problem Statement for Mini Project

Theory:

Natural Language Processing (NLP) is a field of artificial intelligence that deals with the interaction between computers and human (natural) languages. It aims to enable computers to understand, interpret, and generate human language. NLP tasks can range from simple tasks like text classification to more complex ones like machine translation and question answering.

Applications of NLP

1. Introduction to Natural Language Processing Applications
Natural Language Processing (NLP) has revolutionized how computers understand and interact with human language. This study explores various applications of NLP, with a specific focus on implementing a Smart Resume Analyser as our practical mini-project.

2. Survey of NLP Applications

2.1 Common NLP Applications

1. **Document Analysis**
    - Resume parsing and evaluation
    - Contract analysis
    - Medical record processing
2. **Text Classification**
    - Email categorization
    - Sentiment analysis
    - Resume skill categorization
3. **Information Extraction**
    - Named Entity Recognition (NER)
    - Key phrase extraction
    - Relationship extraction
4. **Recommendation Systems**
    - Course recommendations
    - Career path suggestions
    - Skill development guidance

3. Smart Resume Analyser: Our Mini-Project Implementation

3.1 Problem Statement

To develop a comprehensive resume analysis system using NLP techniques that can:

- Automatically extract and evaluate information from resumes
- Provide career recommendations and resume improvement suggestions
- Offer a user-friendly interface for both applicants and administrators

3.2 Technical Implementation

3.2.1 Key Features

1. **User and Admin Sections**
    - Separate interfaces for applicants and administrators

    - Secure admin authentication
2. **Resume Analysis**
    - Resume scoring mechanism
    - Extraction of key information
    - Skills identification and categorization
3. **Recommendation Engine**
    - Career recommendations
    - Course suggestions
    - Skill improvement recommendations
    - YouTube video recommendations for skill development

3.2.2 Technical Stack

1. **Programming Language**
    - Python 3.6+
2. **Key Libraries**
    - PyResparser for information extraction
    - PDFMiner for PDF text extraction
    - Streamlit for web application development
3. **Database**
    - SQL database for storing user data and analysis results
4. **Machine Learning**
    - KNN Algorithm for classification tasks

3.3 Methodology

3.3.1 Data Processing Pipeline

1. **Resume Upload and Extraction**
    - Support for PDF format
    - Text extraction using PDFMiner
    - Information parsing using PyResparser
2. **Analysis Process**
    - Skill identification and categorization
    - Experience evaluation
    - Education assessment
3. **Scoring System**
    - Weighted evaluation of different resume components
    - Comparative analysis against industry standards

3.3.2 Recommendation System

1. **Career Recommendations**
    - Based on extracted skills and experience
    - Industry trend analysis
2. **Course Recommendations**
    - Identification of skill gaps
    - Suggestion of relevant courses
3. **Resume Enhancement**
    - Tips for improving resume content
    - Format and presentation suggestions

4. Technical Architecture

4.1 System Components

1. **Frontend**
    - Streamlit-based user interface
    - Separate user and admin dashboards
2. **Backend**
    - Python-based processing engine
    - Machine learning classifier
    - Database integration
3. **Storage**
    - Local storage for uploaded resumes
    - SQL database for analysis results

4.2 Implementation Details

1. **File Structure**
    - App.py: Main application file
    - Courses.py: Course and video recommendations
    - Classifier.py: Machine learning implementation
2. **Security Features**
    - Admin authentication (Username: machine_learning_hub)
    - Secure file handling

5. Results and Evaluation

5.1 System Capabilities

1. **Accurate Information Extraction**
    - Successful parsing of various resume formats
    - Reliable skill and experience identification
2. **Effective Recommendations**
    - Personalized career guidance
    - Relevant course suggestions

5.2 Performance Metrics

1. Accuracy of Information Extraction
2. User Satisfaction with Recommendations
3. System Response Time

6. Literature Review

1. Jobscan

- ATS compatibility check
- Keyword optimization
- Job description matching
- Resume formatting suggestions

2. Resume Worded

- AI-powered resume scoring
- LinkedIn profile optimization
- Industry-specific feedback
- Instant resume review

3. VMock
- Real-time resume evaluation
- Personalized improvement suggestions
- Career path guidance
- Skills gap analysis
4. Skillroads
- AI resume builder
- Cover letter generator
- LinkedIn profile optimization
- Career advice
5. RezRunner
- ATS simulation
- Keyword optimization
- Format checking
- Industry-specific suggestions
6. Rezi AI Resume Builder
- AI-powered content generation
- ATS-friendly templates
- Skills suggestions
- Job description keyword matching

Existing Research:
- Traditional resume screening is time-consuming and subjective
- AI-powered resume analysis is gaining traction in HR tech
- Previous works focus on:
    - Automated resume parsing
    - Candidate ranking systems
    - Skill extraction using NLP

Gaps Addressed:
- Real-time feedback for job seekers
- Personalized skill recommendations
- Integration of multiple NLP techniques

7. Conclusion
The Smart Resume Analyser project demonstrates the practical application of NLP techniques in solving real-world problems. By successfully implementing features like resume parsing, scoring, and providing recommendations, the project showcases the potential of NLP in revolutionizing the recruitment process.

8. References
1. PyResparser Documentation
2. PDFMiner Documentation
3. Streamlit Documentation

Related research papers and implementations

Experiment 3

Aim: To implement NLP Pre-processing Tasks

Theory:

Introduction

Natural Language Processing (NLP) involves a series of steps to prepare textual data for analysis. Effective preprocessing is crucial for enhancing the quality of input data, thereby improving the performance of subsequent NLP models. This document outlines the key techniques used in the provided code to clean and preprocess text data.

Objectives

The primary goal is to teach students the fundamental techniques of text cleaning and preprocessing, focusing on various tasks that streamline textual data for analysis.

Preprocessing Techniques

1. **Removing HTML Tags**
   - **Purpose**: Many textual datasets contain HTML markup, which needs to be stripped away to isolate the actual content.
   - **Method**: The code uses BeautifulSoup, a Python library, to parse the HTML and extract text, ensuring that only the relevant information remains.
2. **Cleaning Text**
   - **Purpose**: Text often contains extraneous characters, excessive whitespace, and symbols that do not contribute to meaningful analysis.
   - **Method**: Regular expressions (regex) are employed to:
     - Remove extra whitespaces.
     - Eliminate special characters, ensuring only alphanumeric characters and spaces are retained.
3. **Expanding Contractions**
   - **Purpose**: Contractions can hinder natural language understanding. Expanding them makes the text clearer and more formal.
   - **Method**: The `contractions` library is utilized to replace contractions (e.g., "isn't" to "is not") with their full forms, facilitating better comprehension.
4. **Lowercasing Text**
   - **Purpose**: Normalizing text to lowercase prevents discrepancies arising from case sensitivity, ensuring uniformity in data representation.
   - **Method**: The code converts all characters in the text to lowercase.
5. **Removing Stopwords and Numbers**
   - **Purpose**: Stopwords (common words like "and," "the," etc.) and numbers often do not provide significant meaning in the context of NLP tasks. Their removal can enhance the quality of the text for analysis.

- **Method**: NLTK's list of English stopwords is utilized to filter out these words, and a function is implemented to eliminate numerical values.

6. **Tokenization**
   - **Purpose**: Tokenization is the process of breaking down text into individual words or phrases (tokens). This is essential for various NLP tasks, such as sentiment analysis and topic modeling.
   - **Method**: The code employs NLTK's `word_tokenize` function to tokenize the cleaned text, preparing it for further processing.

Sample Data

The code processes a diverse set of sample text data, including sentences with HTML markup, contractions, and numerical values. This variety showcases the effectiveness of the preprocessing techniques in handling different types of textual data.

Results

After executing the preprocessing steps, the code outputs two key results:

- **Cleaned Texts**: The transformed texts, free from HTML tags, extra spaces, special characters, contractions, stopwords, and numbers.
- **Tokenized Texts**: The tokenized form of the cleaned texts, ready for further analysis.

<u>Code:</u>

Importing nltk

```
[ ] import pandas as pd
    import re
    import nltk
    from nltk.corpus import stopwords
    from nltk.tokenize import word_tokenize
    from bs4 import BeautifulSoup
    # downloading NLTK resources
    nltk.download('punkt')
    nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Initializing sample data

```
[ ] # sample data
    sample_data = [
        "<p>Hello, world! NLP Experiment 3. Isn't it great?</p>",
        "<div>Welcome to the NLP course! Let's make things easier.</div>",
        "Email us at: https://example.com & learn more about <b>NLP</b>!",
        "The weather is 75 degrees today! How lovely, isn't it?",
        "Here's a sentence with numbers: 1234 and a lot of   spaces."
    ]
```

Removing HTML Tags, extra whitespaces, and special characters

```
[ ] # remove HTML tags using BeautifulSoup
    def remove_html_tags(text):
        return BeautifulSoup(text, "html.parser").get_text()
    cleaned_text = [remove_html_tags(text) for text in sample_data]
```

```
[ ] # remove extra whitespaces and special characters
    # r'[^\w\s]' matches any character that is not a word character (\w) and is not a whitespace character (\s).
    def clean_text(text):
        text = re.sub(r'\s+', ' ', text)  #extra whitespace
        text = re.sub(r'[^\w\s]', '', text)  #special characters
        return text.strip()
    cleaned_text = [clean_text(text) for text in cleaned_text]
```

Sentence boundary detection, expanding contractions

```
] !pip install contractions
  import contractions
  # expand contractions using the contractions library
  def expand_contractions(text):
      return contractions.fix(text)
  cleaned_text = [expand_contractions(text) for text in cleaned_text]
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl.metadata (1.5 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl.metadata (13 kB)
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 14.3 MB/s eta 0:00:00
Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (110 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.7/110.7 kB 9.9 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.1.0 textsearch-0.0.24
```

Lowercasing text, converting number words to numeric form, removing numbers and stopwords

```
[ ] #lowercasing text
    cleaned_text = [text.lower() for text in cleaned_text]
```

```
[ ] #remove stopwords and numbers using NLTK
    stop_words = set(stopwords.words('english'))

    def remove_stopwords_and_numbers(text):
        return ' '.join([word for word in word_tokenize(text) if word not in stop_words and not word.isdigit()])

    cleaned_text = [remove_stopwords_and_numbers(text) for text in cleaned_text]
```

Tokenization

```
[ ] #tokenization using NLTK
    tokenized_text = [word_tokenize(text) for text in cleaned_text]
```

Final Results

```
[ ] #results
    print("Cleaned Texts:")
    for text in cleaned_text:
        print(text)
    print("\nTokenized Texts:")
    for tokens in tokenized_text:
        print(tokens)
```

```
Cleaned Texts:
hello world nlp experiment great
welcome nlp course let us make things easier
email us httpsexamplecom learn nlp
weather degrees today lovely
sentence numbers lot spaces

Tokenized Texts:
['hello', 'world', 'nlp', 'experiment', 'great']
['welcome', 'nlp', 'course', 'let', 'us', 'make', 'things', 'easier']
['email', 'us', 'httpsexamplecom', 'learn', 'nlp']
['weather', 'degrees', 'today', 'lovely']
['sentence', 'numbers', 'lot', 'spaces']
```

Experiment 4

Aim: To implement Advanced Text Pre-processing Techniques

Theory

Text preprocessing is a crucial step in Natural Language Processing (NLP) to transform raw text into a format suitable for further analysis. It involves several techniques such as stop word removal, lemmatization, stemming, noise handling, multilingual processing, and entity anonymization. The following implementation demonstrates how to clean and prepare text data for NLP tasks using these advanced preprocessing techniques.

a. Stop Word Removal, Lemmatization, Stemming

1. **Stop Word Removal**: Stop words are common words (e.g., "is", "the", "and") that do not contribute much meaning to the text. By removing them, we can focus on more important words that carry significant information.
   In this implementation, stop words are removed using both **English** and **Hindi** stop word lists. We use nltk.stopwords for English stop words and a custom list for Hindi stop words. The function remove_stopwords eliminates these words from the input text.
2. **Stemming**: Stemming reduces words to their root form by removing suffixes. It helps normalize variations of a word, such as converting "playing" to "play". The PorterStemmer from the nltk library is employed here to stem the words in the text. In the stem_text function, stemming is applied to each word after tokenizing the text.
3. **Lemmatization**: Lemmatization is another text normalization technique similar to stemming, but it ensures that the transformed word is a valid word in the language. It considers the context of the word and reduces it to its base or dictionary form (lemma).
   The WordNetLemmatizer from nltk is used in the lemmatize_text function to perform lemmatization on the tokenized words.

b. Handling Noise in Text Data, Multilingual Text Processing, and Entity Anonymization

1. **Handling Noise in Text Data**: Noise in text data refers to irrelevant information such as numbers, extra whitespaces, and special characters. By cleaning the text from such noise, we ensure that the data becomes more structured and meaningful for analysis. In the clean_text function, URLs and special characters are removed using regular expressions, and the text is converted to lowercase. Additionally, the handle_noise function removes numbers and extra whitespace to make the text more uniform.
2. **Multilingual Text Processing**: In real-world applications, text data often comes from different languages. To handle multilingual data, we can apply preprocessing specific to each language, or conditionally handle content based on the language detected.
   In this example, the multilingual_processed list processes Hindi and English text

differently by applying custom logic for texts containing Hindi words like "नम ते", and converting them to lowercase.

3. **Entity Anonymization**: Entity anonymization is a technique used to protect sensitive information by replacing entities (like names, locations, or organizations) with placeholders. This ensures privacy while still allowing the text to be processed. The anonymize_entities function replaces specific entities (e.g., "world" and "परk ण") with a generic placeholder "ENTITY". This approach is particularly useful when working with sensitive or private data.

<u>Code</u>

Importing NLTK

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import re

#download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

Initializing sample data

```
[ ] # Sample data with one line in Hindi and the rest in English
    sample_data = [
        "Hello, world! This is NLP Experiment-4. Isn't it great?",
        "नमस्ते, यह एक परीक्षण है।",
        "Welcome to the NLP course! Let's make things easier.",
        "Visit us at: https://example.com & learn more about NLP!",
        "The weather is 75 degrees today! How lovely, isn't it?",
        "Data processing requires cleaning: remove, transform, and load."
    ]
```

Remove special characters and convert the text to lower case

```
[ ] #remove special characters and lower the text
    def clean_text(text):
        #remove URL and special characters
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
        text = re.sub(r'[^\w\s]', '', text)   # remove punctuation
        return text.lower()  # convert to lowercase

    cleaned_text = [clean_text(text) for text in sample_data]
```

Stop word removal, lemmatization, stemming

```
[ ] #remove stopwords
    # Define a list of Hindi stop words
    hindi_stop_words = set([
        "और", "को", "में", "है", "यह", "से", "कि", "के", "करे", "जैसे", "नहीं",
        "जो", "हमें", "वह", "तथा", "उन्हें", "आप", "किसी", "क्योंकि", "जब",
        "जहाँ", "कब", "फिर", "करना", "ध्यान", "केवल", "बनाना"
    ])
    stop_words_english = set(stopwords.words('english'))
    #stop_words_hindi = set(stopwords.words('hindi'))
    def remove_stopwords(text):
        return ' '.join([word for word in word_tokenize(text) if word not in stop_words_english and word not in hindi_stop_words])

    cleaned_text = [remove_stopwords(text) for text in cleaned_text]
```

```
[ ] #lemmatization
    lemmatizer = WordNetLemmatizer()

    def lemmatize_text(text):
        return ' '.join([lemmatizer.lemmatize(word) for word in word_tokenize(text)])

    lemmatized_text = [lemmatize_text(text) for text in cleaned_text]
```

```
[ ] #stemming
    stemmer = PorterStemmer()

    def stem_text(text):
        return ' '.join([stemmer.stem(word) for word in word_tokenize(text)])

    stemmed_text = [stem_text(text) for text in cleaned_text]
```

Handling noise in text data, multilingual text processing, and entity anonymization

```
[ ] #handling noise in text data
    def handle_noise(text):
        # Remove numbers and extra whitespace
        text = re.sub(r'\d+', '', text)
        text = ' '.join(text.split())
        return text
    noisy_handled_text = [handle_noise(text) for text in cleaned_text]
```

```
[ ] #entity Anonymization
    def anonymize_entities(text):
        # Replace specific entities with placeholders
        text = text.replace("world", "ENTITY").replace("परीक्षण", "ENTITY")
        return text

    anonymized_text = [anonymize_entities(text) for text in noisy_handled_text]
```

```
[ ] #multilingual Processing Example
    multilingual_processed = [text.lower() if 'नमस्ते' in text else text for text in sample_data]
```

Final Results

```
[ ] #results
    print("Cleaned Texts:")
    for text in cleaned_text:
        print(text)

    print("\nStemmed Texts:")
    for text in stemmed_text:
        print(text)

    print("\nLemmatized Texts:")
    for text in lemmatized_text:
        print(text)

    print("\nHandled Noise Texts:")
    for text in noisy_handled_text:
        print(text)

    print("\nAnonymized Texts:")
    for text in anonymized_text:
        print(text)

    print("\nMultilingual Processed Texts:")
    for text in multilingual_processed:
        print(text)
```

```
Cleaned Texts:
hello world nlp experiment4 isnt great
नमसत एक परकषण ह
welcome nlp course lets make things easier
visit us learn nlp
weather 75 degrees today lovely isnt
data processing requires cleaning remove transform load

Stemmed Texts:
hello world nlp experiment4 isnt great
नमसत एक परकषण ह
welcom nlp cours let make thing easier
visit us learn nlp
weather 75 degre today love isnt
data process requir clean remov transform load

Lemmatized Texts:
hello world nlp experiment4 isnt great
नमसत एक परकषण ह
welcome nlp course let make thing easier
visit u learn nlp
weather 75 degree today lovely isnt
data processing requires cleaning remove transform load

Handled Noise Texts:
hello world nlp experiment isnt great
नमसत एक परकषण ह
welcome nlp course lets make things easier
visit us learn nlp
weather degrees today lovely isnt
data processing requires cleaning remove transform load
```

```
Anonymized Texts:
hello ENTITY nlp experiment isnt great
नमसत एक परकषण ह
welcome nlp course lets make things easier
visit us learn nlp
weather degrees today lovely isnt
data processing requires cleaning remove transform load

Multilingual Processed Texts:
Hello, world! This is NLP Experiment-4. Isn't it great?
नमस्ते, यह एक परीक्षण है।
Welcome to the NLP course! Let's make things easier.
Visit us at: https://example.com & learn more about NLP!
The weather is 75 degrees today! How lovely, isn't it?
Data processing requires cleaning: remove, transform, and load.
```

Experiment 5

Aim: To implement shallow and deep Parsing

Theory

Part-Of-Speech (POS) Tagging

POS tagging is the process of labeling each word in a sentence with its corresponding part of speech, such as noun, verb, adjective, etc. It helps in understanding the grammatical structure of a sentence, which is foundational for parsing. POS tagging uses models like Hidden Markov Models (HMM) or more advanced approaches such as neural networks. Common tags include:

- **NN**: Noun
- **VB**: Verb
- **JJ**: Adjective
- **DT**: Determiner

Example:

- Input: "The quick brown fox jumps over the lazy dog."
- POS Output: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]

Shallow Parsing (Chunking)

Shallow parsing, or **chunking**, breaks down a sentence into non-overlapping, meaningful chunks, often focusing on noun phrases (NPs) or verb phrases (VPs). The key difference between shallow and deep parsing is that chunking does not represent the full syntactic structure of a sentence; it only identifies smaller chunks of related words.

Chunking is typically done by defining patterns (regular expressions) based on POS tags. A chunk grammar might define patterns for noun phrases as a determiner followed by adjectives and a noun.

Example:

- Chunk Grammar: "NP: {<DT>?<JJ>*<NN>}" (A noun phrase contains an optional determiner, any number of adjectives, and a noun)
- Input: "The quick brown fox jumps over the lazy dog."
- Chunk Output: `(NP The/DT quick/JJ brown/JJ fox/NN) (VP jumps/VBZ) (PP over/IN (NP the/DT lazy/JJ dog/NN))`

Purpose: Chunking is useful for recognizing phrases in text without needing the full sentence structure, helping in applications like Named Entity Recognition (NER).

Deep Parsing

Deep parsing (also called full parsing or syntactic parsing) involves understanding the complete syntactic structure of a sentence. This is done by breaking down the sentence into its components based on a grammar, usually a **Context-Free Grammar (CFG)**. Deep parsing generates a parse tree that shows how each word in the sentence relates to the others syntactically, creating a hierarchical structure.

Context-Free Grammar (CFG): A CFG consists of a set of production rules, where each rule defines how a symbol can be broken down into other symbols or terminal tokens (like words). CFGs can be used to model a wide range of natural language structures.

Example:

- Input Sentence: "The quick brown fox jumps over the lazy dog."
- Parse Tree: This tree shows the syntactic structure of the sentence, with rules like S -> NP VP, NP -> DT JJ NN, and so on.

Purpose: Deep parsing is used in tasks requiring syntactic understanding, such as machine translation, question answering, or any application where understanding the full grammatical structure is crucial.

CFG-based Parse Tree Generation

In **CFG-based parsing**, the grammar specifies a set of rules to describe the structure of a sentence. A parse tree is then generated by applying these rules. The tree visually represents the syntactic structure, showing how a sentence breaks down into sub-phrases like noun phrases (NP) and verb phrases (VP).

For example:

- The sentence "The quick brown fox jumps over the lazy dog" can be parsed using the following CFG rules:
  - `S -> NP VP` (Sentence consists of a noun phrase and a verb phrase)
  - `NP -> DT JJ NN` (Noun phrase consists of determiner, adjectives, and a noun)
  - `VP -> VBZ PP` (Verb phrase consists of a verb and a prepositional phrase)
  - `PP -> IN NP` (Prepositional phrase consists of preposition and noun phrase)

Parsing Applications:

- **Shallow Parsing** is useful in Named Entity Recognition (NER), where chunks like "Barack Obama" or "New York" are identified as entities. Chunking also helps with extracting simple phrase structures for further processing.

- **Deep Parsing** is essential for more complex applications like **machine translation** or **natural language understanding**, where a full understanding of the sentence's syntactic structure is needed.

Named Entity Recognition (NER)

NER is an application of chunking that classifies elements in text into predefined categories, such as names of people, organizations, locations, dates, etc. It's commonly used in information extraction tasks.

Example:

- Input: "Apple Inc. is a technology company in California."
- NER Output: `[('Apple Inc.', 'ORGANIZATION'), ('California', 'LOCATION')]`

<u>Code</u>

Importing NLTK

```
[2] # Install NLTK and download resources
    import nltk
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')
    nltk.download('maxent_ne_chunker')
    nltk.download('words')
    nltk.download('treebank')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
True
```

Initializing sample data

```
[3] #sample text
    text = "The quick brown fox jumps over the lazy dog."
    #tokenize the text into words
    tokens = nltk.word_tokenize(text)
    #perform POS tagging
    pos_tags = nltk.pos_tag(tokens)
    #display the tokens with POS tags
    print("POS Tags:", pos_tags)
```

```
POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
```

Defining chunk grammar

```
[4] #define a chunk grammar for noun phrases (NP)
    chunk_grammar = "NP: {<DT>?<JJ>*<NN>}"
    #create a chunk parser
    chunk_parser = nltk.RegexpParser(chunk_grammar)
    #parse the POS tags into chunks
    chunked = chunk_parser.parse(pos_tags)
    #display the chunk tree
    # Print the chunk tree instead of drawing it
    print(chunked)
```

```
(S
  (NP The/DT quick/JJ brown/NN)
  (NP fox/NN)
  jumps/VBZ
  over/IN
  (NP the/DT lazy/JJ dog/NN)
  ./.)
```

Defining CFG grammar

```
[6] # Define a simple CFG grammar
    cfg_grammar = nltk.CFG.fromstring("""
        S -> NP VP
        NP -> DT JJ NN | DT NN
        VP -> VBZ PP
        PP -> IN NP
        DT -> 'the'
        JJ -> 'quick' | 'brown' | 'lazy'
        NN -> 'fox' | 'dog'
        VBZ -> 'jumps'
        IN -> 'over'
    """)

    # Create a parser for the CFG grammar
    cfg_parser = nltk.ChartParser(cfg_grammar)

    # Tokenize the sentence and make all tokens lowercase to match the grammar
    tokens = ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

    # Parse the sentence and print the CFG parse tree in a readable format
    for tree in cfg_parser.parse(tokens):
        print(tree)
        tree.pretty_print()  # Prints the tree in a more visual format in text
```

Final Results

```
# Assume pos_tags is already defined
# Perform Named Entity Recognition (NER) on POS tagged words
ner_tree = nltk.ne_chunk(pos_tags)

# Print the named entity tree in a text format
print(ner_tree)
ner_tree.pretty_print()
```

```
(S
  The/DT
  quick/JJ
  brown/NN
  fox/NN
  jumps/VBZ
  over/IN
  the/DT
  lazy/JJ
  dog/NN
  ./.)
                                    S
  ___________________________________|_____________________________________
The/DT quick/JJ brown/NN fox/NN jumps/VBZ over/IN the/DT lazy/JJ dog/NN ./.
```

Experiment 6

Aim: To implement Text Processing Models

Theory

N-Gram Model

Definition:

An **N-Gram model** is a type of probabilistic language model used to predict the next item in a sequence of text. It breaks down a sentence into contiguous sequences of `N` words or tokens. These can be **unigrams (N=1)**, **bigrams (N=2)**, **trigrams (N=3)**, and so on.

Purpose:

- N-Gram models are used to analyze the frequency of word sequences in a text.
- They can capture local word dependencies and are used in tasks like text generation, text completion, and language modeling.

How It Works:

- The code tokenizes the input text into lowercase tokens using NLTK's word tokenizer.
- It then constructs **bigrams (N=2)** by pairing every two consecutive words.
- Finally, it counts the frequency of each bigram in the text using Python's `Counter`.

Example:

For the input sentence:

- "I like natural language processing."
- Bigrams: `[('i', 'like'), ('like', 'natural'), ('natural', 'language'), ('language', 'processing')]`

Applications:

- Autocomplete and spell check
- Text classification
- Simple text generation tasks

TF-IDF (Term Frequency - Inverse Document Frequency)

Definition:

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It combines two metrics:

1. **Term Frequency (TF):** The number of times a term appears in a document.
2. **Inverse Document Frequency (IDF):** Measures how much information the word provides, based on how rare it is across all documents in the corpus.

Formula:

- **TF** = (Number of times term `t` appears in a document) / (Total number of terms in the document)
- **IDF** = log(Total number of documents / Number of documents containing term `t`)

Purpose:

- TF-IDF is used to weigh the significance of terms in a document. Words that frequently appear in a single document but are rare across the entire corpus are given a high weight.
- It is commonly used in text mining, search engines, and information retrieval.

How It Works:

- In the code, a TF-IDF vectorizer is applied to a set of documents. It converts each document into a vector where each dimension represents the importance of a word in that document.
- The output is a sparse matrix representing the TF-IDF values of terms across the documents.

Applications:

- Document similarity
- Text classification
- Keyword extraction

Word Embeddings

1. Word2Vec Model

Definition:

Word2Vec is a popular word embedding technique that learns vector representations of words from large corpora. The idea is that words with similar meanings will have similar vector representations.

- **CBOW (Continuous Bag of Words)**: Predicts the current word based on the context.
- **Skip-gram**: Predicts the surrounding context based on the current word.

How It Works:

- Each word in a sentence is represented as a multi-dimensional vector (typically 100-300 dimensions).
- Word2Vec captures semantic relationships between words by analyzing word co-occurrences in a given context.
- For example, in the sentence "natural language processing is fascinating," Word2Vec will learn that "natural" and "language" are semantically related.

In the Code:

- The Word2Vec model is trained on the sample sentences, and a vector is retrieved for the word "natural."
- This vector is a dense representation of the word in a 100-dimensional space.

Applications:

- Word similarity and analogy tasks
- Text classification
- Sentiment analysis

---

2. GloVe (Global Vectors for Word Representation)

(Not in the code, but related to Word2Vec)

Definition:

GloVe is another popular word embedding model that generates word vectors by factorizing the word co-occurrence matrix. It captures both local (Word2Vec-like) and global statistics of the text corpus.

Difference from Word2Vec:

- Word2Vec is based on a predictive approach (predicting a word given context), while GloVe is based on matrix factorization of word co-occurrence counts.

Applications:

- Similar to Word2Vec, used in tasks like document similarity, word analogies, and NLP applications.

Contextual Embeddings: BERT (Bidirectional Encoder Representations from Transformers)

Definition:

BERT is a state-of-the-art **contextual word embedding** model that understands the meaning of words in context. Unlike static embeddings like Word2Vec and GloVe, BERT creates dynamic word representations, meaning the embedding of a word changes depending on the sentence it appears in.

BERT is based on the **Transformer architecture** and uses a **bidirectional approach**. It learns context from both left-to-right and right-to-left (unlike Word2Vec which only considers context in one direction).

How It Works:

- **Tokenization**: The text is first tokenized using BERT's tokenizer, which converts words into tokens that can be understood by the BERT model.
- **Embeddings**: The BERT model generates embeddings for each token in the text.
- **Outputs**: BERT returns a series of embeddings (or hidden states) for each token in the text. The embeddings capture the contextual meaning of the word.

In the Code:

- The code uses Hugging Face's `transformers` library to load a pre-trained BERT model and tokenizer.
- The input text is tokenized, and the BERT model is used to generate contextual embeddings.
- The last hidden states, which represent the token embeddings, are printed.

Applications:

- Question answering (QA)
- Text classification and sentiment analysis
- Named Entity Recognition (NER)
- Machine translation

<u>Code</u>

Importing NLTK and it’s libraries

```
[ ] #libraries
    import nltk
    from nltk import ngrams
    from collections import Counter
    from sklearn.feature_extraction.text import TfidfVectorizer
    from gensim.models import Word2Vec
    from gensim.models import KeyedVectors
    from transformers import BertTokenizer, BertModel
    import torch

    nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

N-Gram Model

```
[ ] #N-Gram Model
    #sample text
    text = "I like natural language processing. Natural language processing is a sem-7 Subject."
    #tokenization
    tokens = nltk.word_tokenize(text.lower())
    #create N-Grams
    n = 2
    bigrams = list(ngrams(tokens, n))
    # Count frequency of N-Grams
    bigram_freq = Counter(bigrams)
    print(bigram_freq)
```

```
Counter({('natural', 'language'): 2, ('language', 'processing'): 2, ('i', 'like'): 1, ('like', 'natural'): 1, ('processing', '.'): 1, ('.', 'natural'): 1, ('processing', 'is'): 1,
```

TF-IDF Model

```
[ ] #TF-IDF Model
    #sample documents
    documents = [
        "I love natural language processing.",
        "Natural language processing is fascinating.",
        "I enjoy programming."
    ]

    #TF-IDF Vectorizer
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(documents)

    #Display TF-IDF matrix
    print(tfidf_matrix.toarray())
    print(vectorizer.get_feature_names_out())
```

```
[[0.         0.         0.         0.45985353 0.60465213 0.45985353
  0.45985353 0.        ]
 [0.         0.51741994 0.51741994 0.3935112  0.         0.3935112
  0.3935112  0.        ]
 [0.70710678 0.         0.         0.         0.         0.
  0.         0.70710678]]
['enjoy' 'fascinating' 'is' 'language' 'love' 'natural' 'processing'
 'programming']
```

Word2Vec Model

```
#Word2Vec Model
#sample sentences
sentences = [
    ['i', 'love', 'natural', 'language', 'processing'],
    ['natural', 'language', 'processing', 'is', 'fascinating'],
    ['i', 'enjoy', 'programming']
]

#train Word2Vec model
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)

# Get vector for a word
vector = word2vec_model.wv['natural']
print(vector)
```

```
[ 9.4563962e-05  3.0773198e-03 -6.8126451e-03 -1.3754654e-03
  7.6685809e-03  7.3464094e-03 -3.6732971e-03  2.6427018e-03
 -8.3171297e-03  6.2054861e-03 -4.6373224e-03 -3.1641065e-03
  9.3113566e-03  8.7338570e-04  7.4907029e-03 -6.0740625e-03
  5.1605068e-03  9.9228229e-03 -8.4573915e-03 -5.1356913e-03
 -7.0648370e-03 -4.8626517e-03 -3.7785638e-03 -8.5361991e-03
  7.9556061e-03 -4.8439382e-03  8.4236134e-03  5.2625705e-03
 -6.5500261e-03  3.9578713e-03  5.4701497e-03 -7.4265362e-03
 -7.4057197e-03 -2.4752307e-03 -8.6257253e-03 -1.5815723e-03
 -4.0343284e-04  3.2996845e-03  1.4418805e-03 -8.8142155e-04
 -5.5940580e-03  1.7303658e-03 -8.9737179e-04  6.7936908e-03
  3.9735902e-03  4.5294715e-03  1.4343059e-03 -2.6998555e-03
 -4.3668128e-03 -1.0320747e-03  1.4370275e-03 -2.6460087e-03
 -7.0737829e-03 -7.8053069e-03 -9.1217868e-03 -5.9351693e-03
 -1.8474245e-03 -4.3238713e-03 -6.4606704e-03 -3.7173224e-03
  4.2891586e-03 -3.7390434e-03  8.3781751e-03  1.5339935e-03
 -7.2423196e-03  9.4337985e-03  7.6312125e-03  5.4932819e-03
 -6.8488456e-03  5.8226790e-03  4.0090932e-03  5.1853694e-03
  4.2559016e-03  1.9397545e-03 -3.1701624e-03  8.3538452e-03
  9.6121803e-03  3.7926030e-03 -2.8369951e-03  7.1275235e-06
  1.2188185e-03 -8.4583247e-03 -8.2239453e-03 -2.3101569e-04
  1.2372875e-03 -5.7433806e-03 -4.7252737e-03 -7.3460746e-03
  8.3286157e-03  1.2129784e-04 -4.5093987e-03  5.7017053e-03
  9.1800150e-03 -4.0998720e-03  7.9646818e-03  5.3754342e-03
  5.8791232e-03  5.1259040e-04  8.2130842e-03 -7.0190406e-03]
```

BERT Model

```python
#BERT Model
# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
# Sample text
text = "I like natural language processing."
# Tokenization
inputs = tokenizer(text, return_tensors='pt')
# Get embeddings
with torch.no_grad():
    outputs = model(**inputs)
# The last hidden states
last_hidden_states = outputs.last_hidden_state
print(last_hidden_states)
```

```
/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(
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 1.11kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 3.21MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 5.61MB/s]
config.json: 100%  570/570 [00:00<00:00, 10.1kB/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(
model.safetensors: 100%  440M/440M [00:05<00:00, 168MB/s]
tensor([[[-0.0629, -0.0945, -0.3407,  ..., -0.2161,  0.2816,  0.6494],
         [ 0.6210,  0.0370, -0.5664,  ..., -0.0484,  0.6805,  0.5347],
         [ 0.0380,  0.7285,  0.7982,  ..., -0.0221,  0.1011,  0.2799],
         ...,
         [ 0.7782,  0.3084, -0.5455,  ..., -0.9033, -0.2293, -0.3651],
         [ 0.5053,  0.1037, -0.3244,  ...,  0.4139, -0.3787, -0.4161],
         [ 0.8558,  0.0791, -0.1600,  ...,  0.3292, -0.6768, -0.2697]]])
```

Experiment 7

Aim: To study and implement Morphological Analysis of English sentences

Theory

Morphological Analysis refers to the process of analyzing the structure of words by breaking them down into their constituent parts, called **morphemes**. Morphemes are the smallest units of meaning or grammatical function in a language. Morphological analysis is an essential task in **Natural Language Processing (NLP)**, especially for understanding word formation, derivation, and grammatical relationships within sentences.

The provided code demonstrates various aspects of morphological analysis using two popular NLP libraries: **NLTK** and **spaCy**.

1. Tokenization and Part-of-Speech (POS) Tagging with NLTK

Tokenization:

- Tokenization is the process of splitting a text into its constituent words or tokens.
- NLTK's `word_tokenize` function is used to tokenize the input sentence into individual words.

Part-of-Speech (POS) Tagging:

- POS tagging involves assigning grammatical categories (such as noun, verb, adjective) to each word in the sentence.
- NLTK provides the `pos_tag` function, which uses a pre-trained model to assign POS tags to the tokens.
- Each token is paired with its respective POS tag, providing insight into its grammatical role in the sentence.

POS Tags:

- Example tags include:
  - **NN**: Noun
  - **VB**: Verb
  - **JJ**: Adjective
  - **RB**: Adverb
  - **DT**: Determiner

Example:

For the sentence, *"I am studying morphological analysis in natural language processing"*, NLTK breaks down the sentence into tokens and assigns POS tags like `NN` (noun), `VB` (verb), etc.

I: PRP

am: VBP

studying: VBG

morphological: JJ

analysis: NN

...

2. Morphological Analysis with spaCy

spaCy's Model:

- **spaCy** provides powerful tools for tokenization, POS tagging, dependency parsing, and morphological analysis through pre-trained language models. The model used here is **'en_core_web_sm'** for English.

Morphological Features:

- **Lemma**: The base or dictionary form of a word (e.g., "studying" → "study").
- **POS**: The part-of-speech tag assigned by spaCy (e.g., `VERB`, `NOUN`, etc.).
- **Tag**: Detailed POS tag (similar to NLTK).
- **Dependency**: The syntactic relationship between words (e.g., subject, object).

spaCy not only provides morphological information but also establishes **dependency relationships** between words in the sentence, which helps in deeper syntactic parsing.

Example:

For the word *"studying"*, spaCy outputs:

- **Lemma**: "study"
- **POS**: VERB
- **Tag**: VBG (Verb, gerund)
- **Dependency**: ROOT (the main verb in the sentence)

Word: studying, Lemma: study, POS: VERB, Tag: VBG, Dependency: ROOT
Word: analysis, Lemma: analysis, POS: NOUN, Tag: NN, Dependency: nsubj
...

3. Morpheme Breakdown

Definition:

A **morpheme** is the smallest grammatical or meaningful unit of language. Morphological analysis breaks down words into:

- **Roots/Stems**: The main part of a word that carries meaning.
- **Prefixes**: Added to the beginning of a word to modify its meaning (e.g., "un-", "re-").
- **Suffixes**: Added to the end of a word to indicate grammatical features such as tense, plurality, or derivation (e.g., "-ing", "-ed").

Manual Morpheme Breakdown:

- The code includes a basic function, `split_morphemes`, that identifies common prefixes and suffixes in words and attempts to split the word into its stem and affix.
- This is a simplified approach and does not cover all possible morphological variations.

Example:

For the word "studying":

- **Prefix**: None
- **Stem**: "study"
- **Suffix**: "ing"

For the word "nonexistent":

- **Prefix**: "non"
- **Stem**: "existent"

This approach demonstrates how words can be broken down into their morphemes using basic rules.

<u>Code</u>

Importing NLTK

```
[ ] !pip install nltk spacy
    !python -m spacy download en_core_web_sm
    import nltk
    import spacy
    #NLTK resources
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: spacy in /usr/local/lib/python3.10/dist-packages (3.7.6)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.5)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python3.10/dist-packages (from spacy) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from spacy) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (0.12.5)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (from spacy) (2.9.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy) (3.1.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy) (71.0.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (24.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (3.4.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/dist-packages (from spacy) (1.26.4)
```

Tokenization and POS-Tagging

```
[ ] #Tokenization and POS Tagging with NLTK
    #Sample text
    text = "I am studying morphological analysis in natural language processing."
    # Tokenization
    tokens = nltk.word_tokenize(text)
    # Part-of-Speech Tagging
    pos_tags = nltk.pos_tag(tokens)
    print("Tokens and POS tags:")
    for token, tag in pos_tags:
        print(f"{token}: {tag}")
```

```
Tokens and POS tags:
I: PRP
am: VBP
studying: VBG
morphological: JJ
analysis: NN
in: IN
natural: JJ
language: NN
processing: NN
.: .
```

Morphological Analysis with spaCy

```
[ ] #Morphological Analysis with spaCy
    # Load spaCy's English model
    nlp = spacy.load('en_core_web_sm')
    # Process the text
    doc = nlp(text)
    # Extracting morphological features
    print("\nMorphological features:")
    for token in doc:
        print(f"Word: {token.text}, Lemma: {token.lemma_}, POS: {token.pos_}, Tag: {token.tag_}, Dependency: {token.dep_}")
```

```
Morphological features:
Word: I, Lemma: I, POS: PRON, Tag: PRP, Dependency: nsubj
Word: am, Lemma: be, POS: AUX, Tag: VBP, Dependency: aux
Word: studying, Lemma: study, POS: VERB, Tag: VBG, Dependency: ROOT
Word: morphological, Lemma: morphological, POS: ADJ, Tag: JJ, Dependency: amod
Word: analysis, Lemma: analysis, POS: NOUN, Tag: NN, Dependency: dobj
Word: in, Lemma: in, POS: ADP, Tag: IN, Dependency: prep
Word: natural, Lemma: natural, POS: ADJ, Tag: JJ, Dependency: amod
Word: language, Lemma: language, POS: NOUN, Tag: NN, Dependency: compound
Word: processing, Lemma: processing, POS: NOUN, Tag: NN, Dependency: pobj
Word: ., Lemma: ., POS: PUNCT, Tag: ., Dependency: punct
```

Morpheme breakdown

```
#Morpheme Breakdown
#splitting words into morphemes
def split_morphemes(word):
    # Basic rules for demonstration
    prefixes = ['un', 're', 'in', 'dis', 'pre', 'mis', 'non']
    suffixes = ['ing', 'ed', 's', 'ly', 'er', 'ness']
    for prefix in prefixes:
        if word.startswith(prefix):
            stem = word[len(prefix):]
            return prefix, stem

    for suffix in suffixes:
        if word.endswith(suffix):
            stem = word[:-len(suffix)]
            return stem, suffix

    return word, None
print("\nMorpheme Breakdown:")
for token in tokens:
    prefix, morpheme = split_morphemes(token)
    print(f"Word: {token}, Prefix: {prefix}, Morpheme: {morpheme}")
```

```
Morpheme Breakdown:
Word: I, Prefix: I, Morpheme: None
Word: am, Prefix: am, Morpheme: None
Word: studying, Prefix: study, Morpheme: ing
Word: morphological, Prefix: morphological, Morpheme: None
Word: analysis, Prefix: analysi, Morpheme: s
Word: in, Prefix: in, Morpheme:
Word: natural, Prefix: natural, Morpheme: None
Word: language, Prefix: language, Morpheme: None
Word: processing, Prefix: process, Morpheme: ing
Word: ., Prefix: ., Morpheme: None
```

Experiment 8

Aim: To implement Part of Speech- tagging using HMM

Theory

Part-of-Speech (POS) Tagging

POS Tagging is the process of assigning parts of speech (such as noun, verb, adjective, etc.) to each word in a given sentence. It is an essential task in Natural Language Processing (NLP) for syntactic analysis of a sentence.

Hidden Markov Model (HMM)

A Hidden Markov Model (HMM) is a statistical model used to represent systems where the underlying states (POS tags in this case) are hidden, but the observable outcomes (words in a sentence) are visible. It models the probability of sequences based on the relationship between observable events and hidden states.

In the context of POS tagging:

- Hidden States: These are the POS tags (e.g., `NOUN`, `VERB`, `ADJ`).
- Observed Sequence: The sequence of words in the sentence.

HMM assumes that:

1. The probability of each POS tag depends only on the previous tag (Markov assumption).
2. The probability of observing a word depends only on the current POS tag (Emission assumption).

Components of HMM:

1. **States (S):** These represent the possible POS tags.
2. **Observations (O):** These represent the words in the sentence.
3. **Transition Probabilities (A):** These are the probabilities of moving from one state (POS tag) to another. Aij=P(ti∣tj)A_{ij} = P(t_{i} | t_{j})Aij=P(ti∣tj) where `t_i` is the current POS tag and `t_j` is the previous POS tag.
4. **Emission Probabilities (B):** These are the probabilities of emitting a word given a specific POS tag. Bij=P(wi∣tj)B_{ij} = P(w_{i} | t_{j})Bij=P(wi∣tj) where `w_i` is the word, and `t_j` is the POS tag.
5. **Initial Probabilities (π):** These represent the probability distribution over the initial POS tags.

Steps in HMM POS Tagging

1. **Training:**
   - During the training phase, HMM learns the transition probabilities (probability of a POS tag given the previous tag) and emission probabilities (probability of a word given a POS tag) from a large annotated corpus (like the Brown corpus).
   - The HMM uses frequency counts from the corpus to estimate these probabilities.
2. **Decoding with the Viterbi Algorithm:**
   - The Viterbi Algorithm is a dynamic programming approach used to find the most probable sequence of POS tags given a sequence of words.
   - It computes the highest probability path through the model (best sequence of POS tags), given the observed sequence (words in the sentence).

Mathematical Model of HMM for POS Tagging

Given a sentence W={w1,w2,…,wn}W = \{w_1, w_2, \dots, w_n\}W={w1,w2,…,wn} (the sequence of words), HMM finds the sequence of POS tags T={t1,t2,…,tn}T = \{t_1, t_2, \dots, t_n\}T={t1,t2,…,tn} that maximizes the joint probability P(T,W)P(T, W)P(T,W). Using Bayes' rule, this can be factored as:

P(T,W)=P(W∣T)⋅P(T)P(T, W) = P(W | T) \cdot P(T)P(T,W)=P(W∣T)⋅P(T)

Where:

- P(W∣T)P(W | T)P(W∣T) is the emission probability (probability of generating the word sequence given the tag sequence).
- P(T)P(T)P(T) is the transition probability (probability of the tag sequence).

The goal of POS tagging is to find the tag sequence T∗T^*T∗ that maximizes the product:

T∗=argmaxTP(T,W)=argmaxTP(W∣T)⋅P(T)T^* = \arg\max_{T} P(T, W) = \arg\max_{T} P(W | T) \cdot P(T)T∗=argTmaxP(T,W)=argTmaxP(W∣T)⋅P(T)

The Viterbi algorithm is used to efficiently compute this maximization.

Viterbi Algorithm: Decoding the Best Tag Sequence

The Viterbi algorithm finds the most probable sequence of hidden states (POS tags) given an observed sequence of words. It uses dynamic programming to compute probabilities at each step, maintaining the highest probability of reaching each state.

- Initialization: Start with the initial probabilities of each tag.
- Recursion: For each word in the sentence, compute the highest probability of reaching each tag given the previous tag and the transition/emission probabilities.

- Termination: Identify the tag sequence with the maximum probability at the end.

Benefits of HMM POS Tagging

1. **Probabilistic Nature:** Unlike deterministic approaches (like rule-based systems), HMM-based POS tagging is probabilistic, meaning it can handle ambiguity and uncertainty in tagging.
2. **Efficient Decoding:** The Viterbi algorithm makes it feasible to find the most likely tag sequence in a sentence with many words.
3. **Generative Model:** HMM models both the tag sequences and the observed words, which can be beneficial when the dataset is sparse or noisy.

Applications of HMM POS Tagging

1. **Syntactic Parsing:** POS tags are useful for further syntactic analysis of sentences (like parsing dependency trees).
2. **Information Retrieval:** POS tagging helps in understanding sentence structures for better search engine performance.
3. **Named Entity Recognition (NER):** POS tags help in identifying named entities like persons, organizations, and locations.

<u>Code</u>

Importing NLTK and necessary libraries

```
[1] # Import necessary libraries
    import nltk
    from nltk.corpus import brown
    from nltk.tag import hmm
```

```
[2] # Download necessary resources
    nltk.download('brown')
    nltk.download('universal_tagset')
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
True
```

Loading the brown corpus

```
# Load the Brown corpus with universal tagset
corpus = brown.tagged_sents(tagset='universal')
```

Splitting data into training and testing data

```
[4] # Split data into training and testing sets
    train_data = corpus[:int(len(corpus) * 0.9)]  # 90% for training
    test_data = corpus[int(len(corpus) * 0.9):]    # 10% for testing
```

Training HMM POS tagger

```
# Train an HMM POS Tagger
trainer = hmm.HiddenMarkovModelTrainer()
hmm_tagger = trainer.train(train_data)
```

Testing HMM POS tagger

```
[6] # Test the HMM POS tagger on a sample sentence
    sentence = "I am learning natural language processing".split()
    print("Sentence:", sentence)

    Sentence: ['I', 'am', 'learning', 'natural', 'language', 'processing']
```

POS tagging with HMM

```
[7] # POS Tagging with HMM
    tagged_sentence = hmm_tagger.tag(sentence)
    print("Tagged Sentence (HMM):")
    for word, tag in tagged_sentence:
        print(f"{word}: {tag}")

    /usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:336: RuntimeWarning: overflow encountered in cast
      O[i, k] = self._output_logprob(si, self._symbols[k])
    /usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:334: RuntimeWarning: overflow encountered in cast
      X[i, j] = self._transitions[si].logprob(self._states[j])
    Tagged Sentence (HMM):
    I: PRON
    am: VERB
    learning: VERB
    natural: ADJ
    language: NOUN
    processing: VERB
```

Calculating accuracy

```
[8] # Evaluate the HMM tagger on the test set
    accuracy = hmm_tagger.evaluate(test_data)
    print(f"\nHMM POS Tagger Accuracy: {accuracy:.4f}")

    <ipython-input-8-15f1b55a2d95>:2: DeprecationWarning:
      Function evaluate() has been deprecated.  Use accuracy(gold)
      instead.
      accuracy = hmm_tagger.evaluate(test_data)
    /usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:364: RuntimeWarning: overflow encountered in cast
      O[i, k] = self._output_logprob(si, self._symbols[k])

    HMM POS Tagger Accuracy: 0.7011
```

Experiment 9

Aim:

To implement Named Entity Recognition for given real world applications

Theory:

Named Entity Recognition (NER)

NER is a crucial task in Natural Language Processing (NLP) that involves identifying and classifying named entities in text into predefined categories such as names of people, organizations, locations, dates, and more. It plays a significant role in various NLP applications, including information extraction, question answering, and content classification.

NER is the process of detecting and categorizing named entities in a text. Named entities are typically proper nouns or phrases that refer to specific items, which can include:

- Person Names: Names of individuals (e.g., "Albert Einstein").
- Organizations: Names of companies, institutions, or groups (e.g., "OpenAI").
- Locations: Geographic locations like cities, countries, and landmarks (e.g., "New York").
- Dates and Times: Specific dates, times, or durations (e.g., "January 1, 2023").
- Monetary Values: Quantities of money (e.g., "$100").
- Percentages: Values expressed as percentages (e.g., "50%").

Importance of NER:

NER is crucial because it helps in understanding the context and meaning of text. It enables systems to interpret content more effectively and extract valuable insights. NER has applications in various fields, such as:

- Information Retrieval: Enhances search engines by improving the understanding of user queries.
- Content Classification: Helps categorize documents based on the entities they contain.
- Sentiment Analysis: Analyzes opinions about specific entities (e.g., products, brands).
- Customer Support: Automatically categorizes support tickets based on entity recognition.

How NER Works

NER typically involves several steps, which may include:

1. Text Preprocessing:

Before applying NER, the text often undergoes preprocessing steps such as tokenization (splitting text into words or phrases), normalization (converting to a standard format), and part-of-speech tagging (assigning word types).

2. Model Training:

NER models can be trained using supervised learning techniques on annotated datasets. These datasets contain text with labeled entities indicating their categories. Common algorithms used for training NER models include:

- Rule-based Systems: These systems use handcrafted rules based on linguistic patterns to identify entities.
- Machine Learning Models: Traditional models like Conditional Random Fields (CRFs) or Hidden Markov Models (HMMs) can be trained on labeled data.

- Deep Learning Approaches: Modern NER systems often use neural networks, particularly architectures like Bi-directional Long Short-Term Memory (Bi-LSTM) networks and Transformers (e.g., BERT, SpaCy, or Hugging Face's Transformers library).

3. Entity Recognition:
The trained model processes the text to identify and classify entities. It assigns entity tags to words or phrases, which indicate their category (e.g., PERSON, ORGANIZATION, LOCATION).

4. Post-processing:
After entity recognition, the results may undergo additional processing to refine or validate entity classifications. This can involve linking entities to external knowledge bases or databases for disambiguation.

Challenges in NER
While NER has made significant advancements, several challenges still exist:

1. Ambiguity: Some words or phrases can refer to multiple entities depending on the context (e.g., "Apple" could refer to the fruit or the technology company).

2. Variability: Different ways to mention the same entity (e.g., "USA," "United States," or "America") can lead to inconsistencies.

3. Domain-specific Entities: NER systems trained on general datasets may struggle with domain-specific entities (e.g., medical terms, legal jargon).

4. Language Variability: Recognizing entities in different languages or dialects poses challenges due to variations in naming conventions.

5. Nested Entities: Some entities can contain other entities (e.g., "Barack Obama, the former president of the United States"), which complicates recognition.

Named Entity Recognition (NER) is a vital NLP task that identifies and categorizes named entities in text, enhancing text understanding and facilitating information extraction. Despite its challenges, NER is implemented through various techniques, including rule-based systems, traditional machine learning models, and modern deep learning approaches, allowing it to be utilized in a wide range of applications, from search engines to customer support systems. Libraries like SpaCy and Hugging Face Transformers have made NER more accessible and efficient for developers and researchers alike.

Code:

```
import torch
from transformers import BertTokenizer, BertForTokenClassification
from transformers import pipeline
import requests
```

```
# Load the pre-trained BERT model for NER
tokenizer = BertTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model = BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-
english")

# Create a pipeline for NER
ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer, aggregation_strategy="simple")
import torch
from transformers import BertTokenizer, BertForTokenClassification
from transformers import pipeline
import requests

# Load the pre-trained BERT model for NER
tokenizer = BertTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model = BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-
english")

# Create a pipeline for NER
ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer, aggregation_strategy="simple")

# Sample news articles (Replace this with actual API data)
news_articles = [
    {
        "title": "Taylor Swift wins big at the awards",
        "content": "Taylor Swift performed at the awards. Taylor Swift also won two awards and gave a
speech."
    },
    {
        "title": "Shah Rukh Khan's new movie creates waves",
        "content": "Shah Rukh Khan's latest movie is making headlines. SRK was seen promoting his film
across various cities."
    },
    {
        "title": "Deepika Padukone shares fitness tips",
        "content": "Deepika Padukone, known for her fitness regime, shared her tips on staying healthy
during a live session."
    },
    {
        "title": "Priyanka Chopra attends charity event",
        "content": "Priyanka Chopra attended a charity gala in New York. Chopra spoke about her initiatives
in global education."
    },
    {
        "title": "Ranveer Singh joins upcoming sports biopic",
        "content": "Ranveer Singh will portray a famous Indian cricketer in his next biopic. Singh was
spotted at the training camp."
    },
     {
        "title": "Alia Bhatt wins the best actress award",
        "content": "Alia Bhatt received the best actress award for her performance. Bhatt dedicated her win
to her fans."
```

},
{
"title": "Keanu Reeves returns to action movies",
"content": "Keanu Reeves is set to return to action-packed roles in his upcoming movie. Reeves is beloved for his iconic roles."
},
{
"title": "New Charity Event Scheduled",
"content": "Last year, Emma Watson attended a charity gala to support education for girls around the world. The event raised significant funds and awareness."
},
{
"title": "Upcoming Film Festivals Announced",
"content": "This year's film festival will showcase several films starring Leonardo DiCaprio. His recent projects have garnered critical acclaim."
},
{
"title": "Innovative Technology in Film Production",
"content": "Recent advancements in technology have allowed directors to create stunning visual effects. Actors like Tom Holland are now utilizing these techniques in their performances."
},
{
"title": "Climate Change Report Reveals Alarming Trends",
"content": "A new report from climate scientists highlights the rapid changes in weather patterns across the globe, warning of severe consequences."
},
{
"title": "New Technology Promises to Revolutionize Renewable Energy",
"content": "Innovative solar panels are set to change the landscape of renewable energy, offering higher efficiency and lower costs."
},
{
"title": "Local Community Comes Together for Charity Event",
"content": "Residents organized a charity run to support local families in need, raising thousands of dollars for the cause."
},
{
"title": "Breakthrough in Cancer Research Offers New Hope",
"content": "Researchers have discovered a new treatment that could significantly improve survival rates for certain types of cancer."
},
{
"title": "Global Economy Faces Uncertainty Amid Rising Inflation",
"content": "Economists are expressing concerns over rising inflation rates, which could impact global markets and consumer spending."
},
{
"title": "Scientists Discover New Species in Amazon Rainforest",
"content": "A team of biologists has identified a new species of frog in the Amazon rainforest, highlighting the region's rich biodiversity."
},

```
    {
        "title": "Study Shows Link Between Sleep and Mental Health",
        "content": "A recent study finds that poor sleep quality is associated with increased symptoms of
anxiety and depression among adults."
    },
    {
        "title": "New Transportation Initiative Aims to Reduce Traffic Congestion",
        "content": "City officials have launched a new initiative to improve public transportation and
encourage carpooling to ease traffic in downtown areas."
    },
    {
        "title": "Education Reform Bill Passes in Local Legislature",
        "content": "A comprehensive education reform bill aimed at improving school funding and resources
has successfully passed in the state legislature."
    },
    {
        "title": "Wildfires Continue to Spread in Western States",
        "content": "Firefighters are battling multiple wildfires across the western United States, with
containment efforts hampered by high winds."
    }
]

# Function to check if a person is a celebrity using Wikipedia
def is_celebrity(name):
    search_url =
f"https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch={name}&format=json"
    response = requests.get(search_url)

    if response.status_code == 200:
        search_results = response.json().get("query", {}).get("search", [])
        if search_results:
            return True
    return False

# Function to detect celebrity names using BERT NER
def detect_celebrities_bert(article_content):
    ner_results = ner_pipeline(article_content)

    celebrities = set()

    # Extract person entities and verify celebrity status via Wikipedia
    for entity in ner_results:
        if entity["entity_group"] == "PER":  # Person entity
            person_name = entity["word"]
            if is_celebrity(person_name):
                celebrities.add(person_name)

    return celebrities

# Processing each article
for article in news_articles:
```

```
print(f"Title: {article['title']}")

# Detect celebrity names in the article content using BERT
celebrities = detect_celebrities_bert(article['content'])

if celebrities:
    print("Celebrities mentioned: ", ", ".join(celebrities))
else:
    print("No celebrities mentioned in this article.")

print("-" * 50)
```

Output:

```
Title: Taylor Swift wins big at the awards
Celebrities mentioned:  Taylor Swift
--------------------------------------------------
Title: Shah Rukh Khan's new movie creates waves
Celebrities mentioned:  Shah Rukh Khan, SRK
--------------------------------------------------
Title: Deepika Padukone shares fitness tips
Celebrities mentioned:  Deepika Padukone
--------------------------------------------------
Title: Priyanka Chopra attends charity event
Celebrities mentioned:  Priyanka Chopra, Chopra
--------------------------------------------------
Title: Ranveer Singh joins upcoming sports biopic
Celebrities mentioned:  Singh, Ranveer Singh
--------------------------------------------------
Title: Alia Bhatt wins the best actress award
Celebrities mentioned:  Alia Bhatt, Bhatt
--------------------------------------------------
Title: Keanu Reeves returns to action movies
Celebrities mentioned:  Reeves, Keanu Reeves
--------------------------------------------------
Title: New Charity Event Scheduled
Celebrities mentioned:  Emma Watson
--------------------------------------------------
```

```
--------------------------------------------------
Title: Upcoming Film Festivals Announced
Celebrities mentioned:  Leonardo DiCaprio
--------------------------------------------------
Title: Innovative Technology in Film Production
Celebrities mentioned:  Tom Holland
--------------------------------------------------
Title: Climate Change Report Reveals Alarming Trends
No celebrities mentioned in this article.
--------------------------------------------------
Title: New Technology Promises to Revolutionize Renewable Energy
No celebrities mentioned in this article.
--------------------------------------------------
Title: Local Community Comes Together for Charity Event
No celebrities mentioned in this article.
--------------------------------------------------
Title: Breakthrough in Cancer Research Offers New Hope
No celebrities mentioned in this article.
--------------------------------------------------
Title: Global Economy Faces Uncertainty Amid Rising Inflation
No celebrities mentioned in this article.
--------------------------------------------------
Title: Scientists Discover New Species in Amazon Rainforest
No celebrities mentioned in this article.
--------------------------------------------------
Title: Study Shows Link Between Sleep and Mental Health
No celebrities mentioned in this article.
--------------------------------------------------
Title: New Transportation Initiative Aims to Reduce Traffic Congestion
No celebrities mentioned in this article.
--------------------------------------------------
Title: Education Reform Bill Passes in Local Legislature
No celebrities mentioned in this article.
--------------------------------------------------
Title: Wildfires Continue to Spread in Western States
No celebrities mentioned in this article.
```

Experiment -10

Aim: Implementing text classification using BART model (Hugging Face transformation)

Theory:

BART model (Bidirectional and Auto-Regressive Transformers)
Text summarization is the process of distilling the essential information from a piece of text, creating a shorter version while retaining its core meaning. There are two main types: extractive, which selects and stitches together existing sentences, and abstractive, which generates new sentences to convey the main points.
BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model pre-trained on a denoising autoencoder objective, meaning it's great at reconstructing clean sentences from noisy ones. BART (Bidirectional and Auto-Regressive Transformers) is a transformer-based model that has shown strong performance in various natural language processing tasks, including abstractive text summarization. The Hugging Face Transformers library provides a variety of pre-trained models, each with its strengths and use cases. BART is one of the models.

Architecture
BART combines elements from the encoder-decoder structure of models like T5 and the bidirectional context of models like BERT. It consists of two main components:

a. Encoder
- Bidirectional Context: The encoder processes the input sequence in a bidirectional manner, allowing it to attend to the entire context of the input simultaneously. This is similar to BERT, enabling the model to understand relationships between words regardless of their position in the text.
- Multi-Head Self-Attention: Each encoder layer consists of self-attention heads that learn to weigh the importance of different tokens relative to each other.
- Feed-Forward Layers: After the self-attention mechanism, the output goes through a feed-forward neural network, allowing for non-linear transformations.

b. Decoder
- Auto-Regressive Generation: The decoder generates the output sequence one token at a time in a left-to-right manner, similar to models like GPT. This means that when predicting a token, it only has access to previously generated tokens.
- Masked Self-Attention: The decoder employs masked self-attention to prevent it from seeing future tokens during training.
- Encoder-Decoder Attention: The decoder has an additional attention mechanism that allows it to attend to the encoder's output, ensuring that the generation process is informed by the entire input context.

Pre-training
BART is pre-trained using a denoising autoencoder approach. This involves corrupting the input text in various ways (e.g., masking, shuffling, deleting) and training the model to reconstruct the original text. The corruption methods may include:

- Token Masking: Randomly replacing some tokens with a mask token.
- Token Deletion: Randomly removing tokens from the input.
- Sentence Permutation: Shuffling sentences in the input.

The model learns to generate the original text from these corrupted inputs, developing a strong understanding of language.

Fine-tuning of BART model
After pre-training, BART can be fine-tuned on specific tasks, such as summarization or translation, using supervised datasets. Fine-tuning adapts the pre-trained weights to the specific characteristics and requirements of the target task, allowing BART to generate more relevant and contextually appropriate outputs.

Applications

1. Text Summarization: Both extractive and abstractive summarization. BART can generate concise summaries of long texts by understanding their context.
2. Machine Translation: Translating text from one language to another by generating the translated output based on the source text.
3. Text Generation: Creating coherent and contextually relevant text based on prompts, such as in creative writing or dialogue generation.
4. Question Answering: Generating answers based on input questions and context.

Advantages

- State-of-the-Art Performance: BART has achieved impressive results on various NLP benchmarks, particularly in summarization and translation tasks.
- Flexibility: The model can be used for multiple tasks without the need for significant architectural changes.
- Robustness: The denoising autoencoder pre-training enables BART to handle noisy and imperfect data effectively.

Limitations

- Computationally Intensive: Like other transformer-based models, BART requires substantial computational resources for both training and inference, particularly for larger versions of the model.
- Dependency on Large Datasets: Effective pre-training requires access to large and diverse datasets, which may not be available for all languages or domains.

Code:

Step 1: Install Required Libraries

Install the necessary libraries, including Hugging Face Transformers, `peft` (for LoRA), spaCy, NLTK, AllenNLP, and BERTScore.

```
!pip install transformers datasets peft bert-score spacy nltk allennlp
allennlp-models

# Install spaCy model
!python -m spacy download en_core_web_sm

# NLTK data (stopwords)
import nltk
nltk.download('stopwords')
```

```
Requirement already satisfied: transformers in
/usr/local/lib/python3.10/dist-packages (4.44.2)
Collecting datasets
  Downloading datasets-3.0.1-py3-none-any.whl.metadata (20 kB)
Collecting peft
  Downloading peft-0.13.1-py3-none-any.whl.metadata (13 kB)
Collecting bert-score
  Downloading bert_score-0.3.13-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: spacy in
/usr/local/lib/python3.10/dist-packages (3.7.5)
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-
packages (3.8.1)
Collecting allennlp
  Downloading allennlp-2.10.1-py3-none-any.whl.metadata (21 kB)
Collecting allennlp-models
  Downloading allennlp_models-2.10.1-py3-none-any.whl.metadata (23 kB)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.24.7)
```

```
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2024.9.11)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.66.5)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (16.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.17-py310-none-any.whl.metadata (7.2
kB)
Requirement already satisfied: fsspec<=2024.6.1,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from
fsspec[http]<=2024.6.1,>=2023.1.0->datasets) (2024.6.1)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.10.8)
Requirement already satisfied: psutil in
/usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: torch>=1.13.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (2.4.1+cu121)
Requirement already satisfied: accelerate>=0.21.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (0.34.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from bert-score) (3.7.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in
/usr/local/lib/python3.10/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.10/dist-packages (from spacy) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
```

```
/usr/local/lib/python3.10/dist-packages (from spacy) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.10/dist-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in
/usr/local/lib/python3.10/dist-packages (from spacy) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in
/usr/local/lib/python3.10/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in
/usr/local/lib/python3.10/dist-packages (from spacy) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in
/usr/local/lib/python3.10/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in
/usr/local/lib/python3.10/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in
/usr/local/lib/python3.10/dist-packages (from spacy) (0.12.5)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in
/usr/local/lib/python3.10/dist-packages (from spacy) (2.9.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from spacy) (3.1.4)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from spacy) (71.0.4)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in
/usr/local/lib/python3.10/dist-packages (from spacy) (3.4.1)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
INFO: pip is looking at multiple versions of allennlp to determine
which version is compatible with other requirements. This could take a
while.
Collecting allennlp
  Downloading allennlp-2.10.0-py3-none-any.whl.metadata (20 kB)
  Downloading allennlp-2.9.3-py3-none-any.whl.metadata (19 kB)
  Downloading allennlp-2.9.2-py3-none-any.whl.metadata (19 kB)
  Downloading allennlp-2.9.1-py3-none-any.whl.metadata (19 kB)
  Downloading allennlp-2.9.0-py3-none-any.whl.metadata (18 kB)
  Downloading allennlp-2.8.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.7.0-py3-none-any.whl.metadata (17 kB)
INFO: pip is still looking at multiple versions of allennlp to
determine which version is compatible with other requirements. This
could take a while.
  Downloading allennlp-2.6.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.5.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.4.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.3.1-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.3.0-py3-none-any.whl.metadata (17 kB)
INFO: This is taking longer than usual. You might need to provide the
dependency resolver with stricter constraints to reduce runtime. See
https://pip.pypa.io/warnings/backtracking for guidance. If you want to
```

```
abort this run, press Ctrl + C.
  Downloading allennlp-2.2.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.1.0-py3-none-any.whl.metadata (17 kB)
  Downloading allennlp-2.0.1-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-2.0.0-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.5.0-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.4.1-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.4.0-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.3.0-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.2.2-py3-none-any.whl.metadata (15 kB)
  Downloading allennlp-1.2.1-py3-none-any.whl.metadata (14 kB)
  Downloading allennlp-1.2.0-py3-none-any.whl.metadata (13 kB)
  Downloading allennlp-1.1.0-py3-none-any.whl.metadata (13 kB)
  Downloading allennlp-1.0.0-py3-none-any.whl.metadata (11 kB)
  Downloading allennlp-0.9.0-py3-none-any.whl.metadata (11 kB)
Collecting overrides (from allennlp)
  Downloading overrides-7.7.0-py3-none-any.whl.metadata (5.8 kB)
Collecting spacy
  Downloading spacy-2.1.9.tar.gz (30.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.7/30.7 MB 26.2 MB/s eta
0:00:00
ERROR: Operation cancelled by user
Collecting en-core-web-sm==3.7.1
  Downloading
https://github.com/explosion/spacy-models/releases/download/en_core_we
b_sm-3.7.1/en_core_web_sm-3.7.1-py3-none-any.whl (12.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 45.0 MB/s eta
0:00:00
ent already satisfied: spacy<3.8.0,>=3.7.2 in
/usr/local/lib/python3.10/dist-packages (from en-core-web-sm==3.7.1)
(3.7.5)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (8.2.5)
```

```
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (0.12.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (4.66.5)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (2.9.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.1.4)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (71.0.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (24.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (3.4.1)
Requirement already satisfied: numpy>=1.19.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.26.4)
Requirement already satisfied: language-data>=1.2 in
/usr/local/lib/python3.10/dist-packages (from langcodes<4.0.0,>=3.2.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.2.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!
=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in
/usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!
=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
```

```
(2.23.4)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!
=1.8.1,<3.0.0,>=1.7.4->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2024.8.30)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in
/usr/local/lib/python3.10/dist-packages (from thinc<8.3.0,>=8.2.2-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in
/usr/local/lib/python3.10/dist-packages (from thinc<8.3.0,>=8.2.2-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.1.5)
Requirement already satisfied: click>=8.0.0 in
/usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (8.1.7)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in
/usr/local/lib/python3.10/dist-packages (from typer<1.0.0,>=0.3.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (13.9.1)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (0.19.0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in
/usr/local/lib/python3.10/dist-packages (from weasel<0.5.0,>=0.1.0-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (7.0.4)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2-
>spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1) (2.1.5)
Requirement already satisfied: marisa-trie>=0.7.7 in
/usr/local/lib/python3.10/dist-packages (from language-data>=1.2-
>langcodes<4.0.0,>=3.2.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(1.2.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich>=10.11.0-
>typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(3.0.0)
```

```
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich>=10.11.0-
>typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.2->en-core-web-sm==3.7.1)
(2.18.0)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.10/dist-packages (from smart-
open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->spacy<3.8.0,>=3.7.2->en-
core-web-sm==3.7.1) (1.16.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0-
>rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy<3.8.0,>=3.7.2->en-core-web-
sm==3.7.1) (0.1.2)
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart
Python in
order to load all the package's dependencies. You can do this by
selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
True
```

Step 2: Step 2: Load Dataset and Preprocess Using spaCy and NLTK

We will use the IMDb dataset for binary text classification (positive/negative reviews). Let's load the dataset and preprocess it using **spaCy** and **NLTK**.

```
!pip install datasets
```

```
Collecting datasets
  Using cached datasets-3.0.1-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.16.1)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (16.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Using cached dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in
```

```
/usr/local/lib/python3.10/dist-packages (from datasets) (4.66.5)
Collecting xxhash (from datasets)
  Using cached xxhash-3.5.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess (from datasets)
  Using cached multiprocess-0.70.17-py310-none-any.whl.metadata (7.2
kB)
Requirement already satisfied: fsspec<=2024.6.1,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from
fsspec[http]<=2024.6.1,>=2023.1.0->datasets) (2024.6.1)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.10.8)
Requirement already satisfied: huggingface-hub>=0.22.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.24.7)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from datasets) (24.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.1.0)
Requirement already satisfied: yarl<2.0,>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.13.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.22.0-
>datasets) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.32.2-
>datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.32.2-
>datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests>=2.32.2-
>datasets) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.32.2-
>datasets) (2024.8.30)
INFO: pip is looking at multiple versions of multiprocess to determine
which version is compatible with other requirements. This could take a
while.
  Downloading multiprocess-0.70.16-py310-none-any.whl.metadata (7.2
kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2024.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas->datasets) (1.16.0)
Downloading datasets-3.0.1-py3-none-any.whl (471 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 471.6/471.6 kB 6.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 9.4 MB/s eta
0:00:00
ultiprocess-0.70.16-py310-none-any.whl (134 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 13.2 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 14.2 MB/s eta
0:00:00
ultiprocess, datasets
Successfully installed datasets-3.0.1 dill-0.3.8 multiprocess-0.70.16
xxhash-3.5.0
```

```python
from datasets import load_dataset
import spacy
from nltk.corpus import stopwords

# Load the IMDb dataset
dataset = load_dataset("imdb")

# Load spaCy and stopwords
nlp = spacy.load("en_core_web_sm")
stop_words = set(stopwords.words('english'))

# Preprocessing function using spaCy and NLTK
def preprocess_text(text):
```

```
    doc = nlp(text)
    tokens = [token.text.lower() for token in doc if not token.is_stop
and not token.is_punct]
    return " ".join(tokens)

# Sample 1,000 examples from both the train and test sets
train_dataset = dataset['train'].shuffle(seed=42).select(range(500))
# 1,000 samples for training
test_dataset = dataset['test'].shuffle(seed=42).select(range(500))
# 1,000 samples for testing

# Apply preprocessing to both subsets
train_dataset = train_dataset.map(lambda x: {'text':
preprocess_text(x['text'])})
test_dataset = test_dataset.map(lambda x: {'text':
preprocess_text(x['text'])})
```

{"model_id":"7499e4579f2d43e5b132889cc08cdd27","version_major":2,"version_minor":0}

{"model_id":"323c6c783fee4c8aa4da280119d11503","version_major":2,"version_minor":0}

```
print(train_dataset[0])
```

```
{'text': 'relation fortier profiler fact police series violent crimes
profiler looks crispy fortier looks classic profiler plots simple
fortier plot far complicated fortier looks like prime suspect spot
similarities main character weak weirdo clairvoyance people like
compare judge evaluate enjoying funny thing people writing fortier
looks american hand arguing prefer american series maybe language
spirit think series english american way actors good funny acting
superficial', 'label': 1}
```

Step 3: Load a Pre-Trained BERT Model from Hugging Face Hub and Download Files

```
from huggingface_hub import hf_hub_download
from transformers import AutoTokenizer, BertForSequenceClassification,
BertConfig

# Load model and tokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name)

# Tokenize a sample input and run inference
sample_text = "This movie was amazing!"
inputs = tokenizer(sample_text, return_tensors="pt")
outputs = model(**inputs)
```

```
# Print the output logits
print(f"Logits: {outputs.logits}")
```

```
/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. This behavior will be depracted in transformers v4.45, and
will be then set to `False` by default. For more details check this
issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
```

{"model_id":"6f339e6c6292471387aca2719999234f","version_major":2,"version_minor":0}

```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
Logits: tensor([[-0.3845, -0.3864]], grad_fn=<AddmmBackward0>)
```

Step 4: Fine-Tuning BERT with LoRA

```
!pip install peft
```

```
Collecting peft
  Using cached peft-0.13.1-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from peft) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (24.1)
Requirement already satisfied: psutil in
/usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: pyyaml in
/usr/local/lib/python3.10/dist-packages (from peft) (6.0.2)
Requirement already satisfied: torch>=1.13.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (2.4.1+cu121)
Requirement already satisfied: transformers in
/usr/local/lib/python3.10/dist-packages (from peft) (4.44.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from peft) (4.66.5)
Requirement already satisfied: accelerate>=0.21.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (0.34.2)
Requirement already satisfied: safetensors in
/usr/local/lib/python3.10/dist-packages (from peft) (0.4.5)
Requirement already satisfied: huggingface-hub>=0.17.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (0.24.7)
Requirement already satisfied: filelock in
```

```
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0-
>peft) (3.16.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0-
>peft) (2024.6.1)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0-
>peft) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0-
>peft) (4.12.2)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(1.13.3)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(3.3)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(3.1.4)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers->peft)
(2024.9.11)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers->peft)
(0.19.1)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.13.0-
>peft) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.17.0->peft) (2024.8.30)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.13.0-
>peft) (1.3.0)
Downloading peft-0.13.1-py3-none-any.whl (320 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 320.7/320.7 kB 4.4 MB/s eta
0:00:00
```

```python
from peft import get_peft_model, LoraConfig, TaskType
```

```python
# Configure LoRA for sequence classification
lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    inference_mode=False,
    r=8,                # Rank of the adaptation matrices
    lora_alpha=16,      # Scaling factor
    lora_dropout=0.1    # Dropout rate
)

# Apply LoRA configuration to the model
model = get_peft_model(model, lora_config)

# Display LoRA configuration
print(model)
```

```
PeftModelForSequenceClassification(
  (base_model): LoraModel(
    (model): BertForSequenceClassification(
      (bert): BertModel(
        (embeddings): BertEmbeddings(
          (word_embeddings): Embedding(30522, 768, padding_idx=0)
          (position_embeddings): Embedding(512, 768)
          (token_type_embeddings): Embedding(2, 768)
          (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder): BertEncoder(
          (layer): ModuleList(
            (0-11): 12 x BertLayer(
              (attention): BertAttention(
                (self): BertSdpaSelfAttention(
                  (query): lora.Linear(
                    (base_layer): Linear(in_features=768,
out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.1, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768,
out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8,
out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                    (lora_magnitude_vector): ModuleDict()
                  )
```

```
                    (key): Linear(in_features=768, out_features=768,
bias=True)
                    (value): lora.Linear(
                      (base_layer): Linear(in_features=768,
out_features=768, bias=True)
                      (lora_dropout): ModuleDict(
                        (default): Dropout(p=0.1, inplace=False)
                      )
                      (lora_A): ModuleDict(
                        (default): Linear(in_features=768,
out_features=8, bias=False)
                      )
                      (lora_B): ModuleDict(
                        (default): Linear(in_features=8,
out_features=768, bias=False)
                      )
                      (lora_embedding_A): ParameterDict()
                      (lora_embedding_B): ParameterDict()
                      (lora_magnitude_vector): ModuleDict()
                    )
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                  (output): BertSelfOutput(
                    (dense): Linear(in_features=768, out_features=768,
bias=True)
                    (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                )
                (intermediate): BertIntermediate(
                  (dense): Linear(in_features=768, out_features=3072,
bias=True)
                  (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                  (dense): Linear(in_features=3072, out_features=768,
bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
            )
          )
          (pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768,
bias=True)
            (activation): Tanh()
```

```
        )
      )
      (dropout): Dropout(p=0.1, inplace=False)
      (classifier): ModulesToSaveWrapper(
        (original_module): Linear(in_features=768, out_features=2,
bias=True)
        (modules_to_save): ModuleDict(
          (default): Linear(in_features=768, out_features=2,
bias=True)
        )
      )
    )
  )
)
```

###Step 5: Tokenization and Training

```
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length",
truncation=True, max_length=512)

# Tokenize the datasets
tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_test = test_dataset.map(tokenize_function, batched=True)

# Set format for PyTorch
tokenized_train = tokenized_train.rename_column("label", "labels")
tokenized_test = tokenized_test.rename_column("label", "labels")

tokenized_train.set_format(type="torch", columns=["input_ids",
"attention_mask", "labels"])
tokenized_test.set_format(type="torch", columns=["input_ids",
"attention_mask", "labels"])

# Define training arguments
from transformers import Trainer, TrainingArguments

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=1,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=4,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    logging_dir='./logs',
    logging_steps=100,
    load_best_model_at_end=True,
```

```
    metric_for_best_model="accuracy",
    fp16=True,
    max_steps=50
)
```

```
!pip install evaluate  # Install the 'evaluate' library
```

```
Collecting evaluate
  Downloading evaluate-0.4.3-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: datasets>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from evaluate) (3.0.1)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from evaluate) (1.26.4)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-
packages (from evaluate) (0.3.8)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from evaluate) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from evaluate) (2.32.3)
Requirement already satisfied: tqdm>=4.62.1 in
/usr/local/lib/python3.10/dist-packages (from evaluate) (4.66.5)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.10/dist-packages (from evaluate) (3.5.0)
Requirement already satisfied: multiprocess in
/usr/local/lib/python3.10/dist-packages (from evaluate) (0.70.16)
Requirement already satisfied: fsspec>=2021.05.0 in
/usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2021.05.0-
>evaluate) (2024.6.1)
Requirement already satisfied: huggingface-hub>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from evaluate) (0.24.7)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from evaluate) (24.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0-
>evaluate) (3.16.1)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0-
>evaluate) (16.1.0)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0-
>evaluate) (3.10.8)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0-
>evaluate) (6.0.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.7.0-
>evaluate) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>evaluate) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>evaluate) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>evaluate) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>evaluate) (2024.8.30)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->evaluate)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->evaluate)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas->evaluate)
(2024.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (6.1.0)
Requirement already satisfied: yarl<2.0,>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (1.13.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>datasets>=2.0.0->evaluate) (4.0.3)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas->evaluate) (1.16.0)
Downloading evaluate-0.4.3-py3-none-any.whl (84 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.0/84.0 kB 2.2 MB/s eta
0:00:00
```

```
import evaluate
```

```
# Load the accuracy metric
metric = evaluate.load("accuracy")

# Define the compute_metrics function
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

{"model_id":"8f3c005ed9e647ceb71ddcbfe83cd0ce","version_major":2,"version_minor":0}

```
# Initialize Trainer

# Custom Trainer class to ensure contiguous parameters
class ContiguousParameterTrainer(Trainer):
    def save_model(self, output_dir=None, _internal_call=False):
        # Make sure all parameters are contiguous before saving
        for name, param in self.model.named_parameters():
            if param.is_leaf and not param.data.is_contiguous():
                param.data = param.data.contiguous()
        super().save_model(output_dir, _internal_call)

trainer = ContiguousParameterTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    compute_metrics=compute_metrics,
)

# Start training
trainer.train()
model.save_pretrained('./results')
tokenizer.save_pretrained('./results')
```

```
max_steps is given, it will override any value given in
num_train_epochs
```

```
<IPython.core.display.HTML object>
```

```
('./results/tokenizer_config.json',
 './results/special_tokens_map.json',
 './results/vocab.txt',
 './results/added_tokens.json',
 './results/tokenizer.json')
```

###Step 6: Analyze with AllenNLP

```
!pip install allennlp
```

```
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
Collecting allennlp
  Using cached allennlp-2.10.1-py3-none-any.whl.metadata (21 kB)
Collecting torch<1.13.0,>=1.10.0 (from allennlp)
  Using cached torch-1.12.1-cp310-cp310-manylinux1_x86_64.whl.metadata
(22 kB)
Collecting torchvision<0.14.0,>=0.8.1 (from allennlp)
  Using cached torchvision-0.13.1-cp310-cp310-
manylinux1_x86_64.whl.metadata (10 kB)
Collecting cached-path<1.2.0,>=1.1.3 (from allennlp)
  Using cached cached_path-1.1.6-py3-none-any.whl.metadata (6.0 kB)
Collecting fairscale==0.4.6 (from allennlp)
  Using cached fairscale-0.4.6-py3-none-any.whl
Requirement already satisfied: nltk>=3.6.5 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (3.8.1)
Collecting spacy<3.4,>=2.1.0 (from allennlp)
  Using cached spacy-3.3.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (23 kB)
Requirement already satisfied: numpy>=1.21.4 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (1.26.4)
Requirement already satisfied: tensorboardX>=1.2 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (2.6.2.2)
Requirement already satisfied: requests>=2.28 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (2.32.3)
Requirement already satisfied: tqdm>=4.62 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (4.66.5)
Requirement already satisfied: h5py>=3.6.0 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (3.11.0)
Requirement already satisfied: scikit-learn>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (1.5.2)
Requirement already satisfied: scipy>=1.7.3 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (1.13.1)
Requirement already satisfied: pytest>=6.2.5 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (7.4.4)
Collecting transformers<4.21,>=4.1 (from allennlp)
  Using cached transformers-4.20.1-py3-none-any.whl.metadata (77 kB)
Requirement already satisfied: sentencepiece>=0.1.96 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.2.0)
Requirement already satisfied: filelock<3.8,>=3.3 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (3.7.1)
Requirement already satisfied: lmdb>=1.2.1 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (1.5.1)
Requirement already satisfied: more-itertools>=8.12.0 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (10.5.0)
Requirement already satisfied: termcolor==1.1.0 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (1.1.0)
Collecting wandb<0.13.0,>=0.10.0 (from allennlp)
  Using cached wandb-0.12.21-py2.py3-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: huggingface-hub>=0.0.16 in
```

```
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.24.7)
Requirement already satisfied: dill>=0.3.4 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.3.8)
Requirement already satisfied: base58>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (2.1.1)
Requirement already satisfied: sacremoses in
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.1.1)
Requirement already satisfied: typer>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.4.2)
Requirement already satisfied: protobuf<4.0.0,>=3.12.0 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (3.20.3)
Requirement already satisfied: traitlets>5.1.1 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (5.7.1)
Requirement already satisfied: jsonnet>=0.10.0 in
/usr/local/lib/python3.10/dist-packages (from allennlp) (0.20.0)
Requirement already satisfied: rich<13.0,>=12.1 in
/usr/local/lib/python3.10/dist-packages (from cached-
path<1.2.0,>=1.1.3->allennlp) (12.6.0)
Collecting boto3<2.0,>=1.0 (from cached-path<1.2.0,>=1.1.3->allennlp)
  Using cached boto3-1.35.37-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: google-cloud-storage<3.0,>=1.32.0 in
/usr/local/lib/python3.10/dist-packages (from cached-
path<1.2.0,>=1.1.3->allennlp) (2.8.0)
Collecting huggingface-hub>=0.0.16 (from allennlp)
  Using cached huggingface_hub-0.10.1-py3-none-any.whl.metadata (6.1
kB)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.0.16-
>allennlp) (6.0.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.0.16-
>allennlp) (4.5.0)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.0.16-
>allennlp) (24.1)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.6.5->allennlp)
(8.1.7)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.6.5->allennlp)
(1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.6.5->allennlp)
(2024.9.11)
Requirement already satisfied: iniconfig in
/usr/local/lib/python3.10/dist-packages (from pytest>=6.2.5->allennlp)
(2.0.0)
Requirement already satisfied: pluggy<2.0,>=0.12 in
/usr/local/lib/python3.10/dist-packages (from pytest>=6.2.5->allennlp)
(1.5.0)
```

```
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in
/usr/local/lib/python3.10/dist-packages (from pytest>=6.2.5->allennlp)
(1.2.2)
Requirement already satisfied: tomli>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from pytest>=6.2.5->allennlp)
(2.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.28-
>allennlp) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.28-
>allennlp) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.28-
>allennlp) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.28-
>allennlp) (2024.8.30)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.1-
>allennlp) (3.5.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.9 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (3.0.9)
Collecting thinc<8.1.0,>=8.0.14 (from spacy<3.4,>=2.1.0->allennlp)
  Using cached thinc-8.0.17-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (14 kB)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (0.7.11)
Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (0.10.1)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
```

```
>allennlp) (2.0.10)
Collecting pathy>=0.3.5 (from spacy<3.4,>=2.1.0->allennlp)
  Using cached pathy-0.11.0-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (6.4.0)
Collecting pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 (from
spacy<3.4,>=2.1.0->allennlp)
  Using cached pydantic-1.8.2-py3-none-any.whl.metadata (103 kB)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (3.1.4)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (71.0.4)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in
/usr/local/lib/python3.10/dist-packages (from spacy<3.4,>=2.1.0-
>allennlp) (3.4.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.10/dist-packages (from
torchvision<0.14.0,>=0.8.1->allennlp) (10.4.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.13,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from transformers<4.21,>=4.1-
>allennlp) (0.12.1)
Collecting GitPython>=1.0.0 (from wandb<0.13.0,>=0.10.0->allennlp)
  Using cached GitPython-3.1.43-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: promise<3,>=2.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (2.3)
Requirement already satisfied: shortuuid>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (1.0.13)
Requirement already satisfied: psutil>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (5.9.5)
Requirement already satisfied: sentry-sdk>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (2.16.0)
Requirement already satisfied: six>=1.13.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (1.16.0)
Requirement already satisfied: docker-pycreds>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (0.4.0)
Requirement already satisfied: pathtools in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (0.1.2)
Requirement already satisfied: setproctitle in
/usr/local/lib/python3.10/dist-packages (from wandb<0.13.0,>=0.10.0-
>allennlp) (1.3.3)
```

```
Collecting botocore<1.36.0,>=1.35.37 (from boto3<2.0,>=1.0->cached-
path<1.2.0,>=1.1.3->allennlp)
  Using cached botocore-1.35.37-py3-none-any.whl.metadata (5.7 kB)
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from boto3<2.0,>=1.0->cached-
path<1.2.0,>=1.1.3->allennlp) (1.0.1)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3<2.0,>=1.0->cached-
path<1.2.0,>=1.1.3->allennlp)
  Using cached s3transfer-0.10.3-py3-none-any.whl.metadata (1.7 kB)
Collecting gitdb<5,>=4.0.1 (from GitPython>=1.0.0-
>wandb<0.13.0,>=0.10.0->allennlp)
  Using cached gitdb-4.0.11-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: google-auth<3.0dev,>=1.25.0 in
/usr/local/lib/python3.10/dist-packages (from google-cloud-
storage<3.0,>=1.32.0->cached-path<1.2.0,>=1.1.3->allennlp) (2.27.0)
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!
=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5 in /usr/local/lib/python3.10/dist-
packages (from google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (2.19.2)
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from google-cloud-
storage<3.0,>=1.32.0->cached-path<1.2.0,>=1.1.3->allennlp) (2.4.1)
Requirement already satisfied: google-resumable-media>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from google-cloud-
storage<3.0,>=1.32.0->cached-path<1.2.0,>=1.1.3->allennlp) (2.7.2)
Requirement already satisfied: language-data>=1.2 in
/usr/local/lib/python3.10/dist-packages (from langcodes<4.0.0,>=3.2.0-
>spacy<3.4,>=2.1.0->allennlp) (1.2.0)
Requirement already satisfied: pathlib-abc==0.1.1 in
/usr/local/lib/python3.10/dist-packages (from pathy>=0.3.5-
>spacy<3.4,>=2.1.0->allennlp) (0.1.1)
Requirement already satisfied: commonmark<0.10.0,>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from rich<13.0,>=12.1-
>cached-path<1.2.0,>=1.1.3->allennlp) (0.9.1)
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in
/usr/local/lib/python3.10/dist-packages (from rich<13.0,>=12.1-
>cached-path<1.2.0,>=1.1.3->allennlp) (2.18.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2-
>spacy<3.4,>=2.1.0->allennlp) (2.1.5)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/usr/local/lib/python3.10/dist-packages (from
botocore<1.36.0,>=1.35.37->boto3<2.0,>=1.0->cached-path<1.2.0,>=1.1.3-
>allennlp) (2.8.2)
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1-
>GitPython>=1.0.0->wandb<0.13.0,>=0.10.0->allennlp) (5.0.1)
Requirement already satisfied: googleapis-common-
protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages
(from google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!
```

```
=2.3.0,<3.0.0dev,>=1.31.5->google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (1.65.0)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in
/usr/local/lib/python3.10/dist-packages (from google-api-core!=2.0.*,!
=2.1.*,!=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5->google-cloud-
storage<3.0,>=1.32.0->cached-path<1.2.0,>=1.1.3->allennlp) (1.24.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3.0dev,>=1.25.0->google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (5.5.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3.0dev,>=1.25.0->google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (0.4.1)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3.0dev,>=1.25.0->google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (4.9)
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from google-resumable-
media>=2.3.2->google-cloud-storage<3.0,>=1.32.0->cached-
path<1.2.0,>=1.1.3->allennlp) (1.6.0)
Requirement already satisfied: marisa-trie>=0.7.7 in
/usr/local/lib/python3.10/dist-packages (from language-data>=1.2-
>langcodes<4.0.0,>=3.2.0->spacy<3.4,>=2.1.0->allennlp) (1.2.0)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3.0dev,>=1.25.0->google-cloud-storage<3.0,>=1.32.0-
>cached-path<1.2.0,>=1.1.3->allennlp) (0.6.1)
Using cached allennlp-2.10.1-py3-none-any.whl (730 kB)
Using cached cached_path-1.1.6-py3-none-any.whl (26 kB)
Using cached huggingface_hub-0.10.1-py3-none-any.whl (163 kB)
Using cached spacy-3.3.3-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (6.3 MB)
Using cached torch-1.12.1-cp310-cp310-manylinux1_x86_64.whl (776.3 MB)
Using cached torchvision-0.13.1-cp310-cp310-manylinux1_x86_64.whl
(19.1 MB)
Using cached transformers-4.20.1-py3-none-any.whl (4.4 MB)
Using cached wandb-0.12.21-py2.py3-none-any.whl (1.8 MB)
Using cached boto3-1.35.37-py3-none-any.whl (139 kB)
Using cached GitPython-3.1.43-py3-none-any.whl (207 kB)
Using cached pathy-0.11.0-py3-none-any.whl (47 kB)
Using cached pydantic-1.8.2-py3-none-any.whl (126 kB)
Using cached thinc-8.0.17-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (659 kB)
Using cached botocore-1.35.37-py3-none-any.whl (12.6 MB)
Using cached gitdb-4.0.11-py3-none-any.whl (62 kB)
Using cached s3transfer-0.10.3-py3-none-any.whl (82 kB)
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
```

```
Installing collected packages: torch, pydantic, gitdb, torchvision,
thinc, pathy, huggingface-hub, GitPython, fairscale, botocore, wandb,
transformers, s3transfer, spacy, boto3, cached-path, allennlp
  Attempting uninstall: pydantic
    Found existing installation: pydantic 2.9.2
    Uninstalling pydantic-2.9.2:
      Successfully uninstalled pydantic-2.9.2
  Attempting uninstall: torchvision
    Found existing installation: torchvision 0.19.1+cu121
    Uninstalling torchvision-0.19.1+cu121:
      Successfully uninstalled torchvision-0.19.1+cu121
  Attempting uninstall: thinc
    Found existing installation: thinc 8.2.5
    Uninstalling thinc-8.2.5:
      Successfully uninstalled thinc-8.2.5
  Attempting uninstall: huggingface-hub
    Found existing installation: huggingface-hub 0.24.7
    Uninstalling huggingface-hub-0.24.7:
      Successfully uninstalled huggingface-hub-0.24.7
  Attempting uninstall: transformers
    Found existing installation: transformers 4.44.2
    Uninstalling transformers-4.44.2:
      Successfully uninstalled transformers-4.44.2
  Attempting uninstall: spacy
    Found existing installation: spacy 3.7.5
    Uninstalling spacy-3.7.5:
      Successfully uninstalled spacy-3.7.5
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
accelerate 0.34.2 requires huggingface-hub>=0.21.0, but you have
huggingface-hub 0.10.1 which is incompatible.
albumentations 1.4.15 requires pydantic>=2.7.0, but you have pydantic
1.8.2 which is incompatible.
datasets 3.0.1 requires huggingface-hub>=0.22.0, but you have
huggingface-hub 0.10.1 which is incompatible.
en-core-web-sm 3.7.1 requires spacy<3.8.0,>=3.7.2, but you have spacy
3.3.3 which is incompatible.
peft 0.13.1 requires huggingface-hub>=0.17.0, but you have
huggingface-hub 0.10.1 which is incompatible.
peft 0.13.1 requires torch>=1.13.0, but you have torch 1.12.1 which is
incompatible.
torchaudio 2.4.1+cu121 requires torch==2.4.1, but you have torch
1.12.1 which is incompatible.
Successfully installed GitPython-3.1.43 allennlp-2.10.1 boto3-1.35.37
botocore-1.35.37 cached-path-1.1.6 fairscale-0.4.6 gitdb-4.0.11
huggingface-hub-0.10.1 pathy-0.11.0 pydantic-1.8.2 s3transfer-0.10.3
spacy-3.3.3 thinc-8.0.17 torch-1.12.1 torchvision-0.13.1 transformers-
4.20.1 wandb-0.12.21
```

{"id":"bd25f3758e6e40be9fa60d56d1a85d81","pip_warning":{"packages":["huggingface_hub","pydantic","spacy","thinc","torch","torchgen","torchvision","transformers"]}}

```
from allennlp.predictors.predictor import Predictor
import allennlp_models.tagging

# Load AllenNLP's NER model
predictor =
Predictor.from_path("https://storage.googleapis.com/allennlp-public-models/ner-model-2020.02.10.tar.gz")

# Example sentence for NER
example_sentence = "Hugging Face Inc. is based in New York City."

# Perform NER
ner_result = predictor.predict(sentence=example_sentence)

# Display NER results
print(ner_result)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-54-9c64a514d34c> in <cell line: 2>()
      1 from allennlp.predictors.predictor import Predictor
----> 2 import allennlp_models.tagging
      3
      4 # Load AllenNLP's NER model
      5 predictor =
Predictor.from_path("https://storage.googleapis.com/allennlp-public-models/ner-model-2020.02.10.tar.gz")

ModuleNotFoundError: No module named 'allennlp_models'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

###Step 7: Evaluation with BERTScore

```
!pip install bert_score
```

```
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
Collecting bert_score
  Using cached bert_score-0.3.13-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: torch>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from bert_score) (1.12.1)
Requirement already satisfied: pandas>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from bert_score) (2.2.2)
Requirement already satisfied: transformers>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from bert_score) (4.20.1)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from bert_score) (1.26.4)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from bert_score) (2.32.3)
Requirement already satisfied: tqdm>=4.31.1 in
/usr/local/lib/python3.10/dist-packages (from bert_score) (4.66.5)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from bert_score) (3.7.1)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from bert_score) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.1-
>bert_score) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.1-
>bert_score) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.1-
>bert_score) (2024.2)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch>=1.0.0-
>bert_score) (4.5.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers>=3.0.0-
>bert_score) (3.7.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.1.0 in
/usr/local/lib/python3.10/dist-packages (from transformers>=3.0.0-
>bert_score) (0.10.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers>=3.0.0-
>bert_score) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers>=3.0.0-
>bert_score) (2024.9.11)
Requirement already satisfied: tokenizers!=0.11.3,<0.13,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from transformers>=3.0.0-
>bert_score) (0.12.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(1.3.0)
```

```
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(1.4.7)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->bert_score)
(3.1.4)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->bert_score)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->bert_score)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->bert_score)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->bert_score)
(2024.8.30)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.0.1->bert_score) (1.16.0)
Downloading bert_score-0.3.13-py3-none-any.whl (61 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 2.2 MB/s eta
0:00:00
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
Installing collected packages: bert_score
Successfully installed bert_score-0.3.13
```

```
!pip install transformers
```

```
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
Requirement already satisfied: transformers in
/usr/local/lib/python3.10/dist-packages (4.20.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers) (3.7.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.1.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.10.1)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
```

```
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2024.9.11)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers!=0.11.3,<0.13,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.12.1)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.66.5)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.1.0->transformers) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2024.8.30)
WARNING: Ignoring invalid distribution -orch
(/usr/local/lib/python3.10/dist-packages)
```

```python
from sklearn.metrics import f1_score

# Example references and candidates (tokenized)
references = ["The movie was fantastic and gripping."]
candidates = ["The film was incredible and engaging."]

# Tokenize and flatten the lists
refs_tokens = [ref.split() for ref in references]
cands_tokens = [cand.split() for cand in candidates]

# Calculate F1 score
flat_refs = [item for sublist in refs_tokens for item in sublist]
flat_cands = [item for sublist in cands_tokens for item in sublist]
f1 = f1_score(flat_refs, flat_cands, average='weighted')

print(f"F1 Score: {f1:.4f}")
```

```
F1 Score: 0.5000
```

Mini Project :- Resume Analyser

Theory:

The Resume Analyzer project is designed to leverage **Natural Language Processing (NLP)** techniques to extract, analyze, and enhance resumes, particularly in PDF format. The tool simplifies the job application process by automatically pulling out key details from resumes, scoring them based on various parameters, and providing tailored suggestions. It utilizes advanced text extraction techniques to handle structured and unstructured data commonly found in resumes, ensuring that essential information is captured and presented in a clear, useful manner. The system's primary goal is to help users improve their resumes, making them more appealing to potential employers and aligning them with job opportunities.

NLP plays a crucial role in this project as it enables the automated extraction of text from **PDF resumes**, ensuring the accurate identification of key sections such as work experience, skills, education, and achievements. By analyzing this data, the tool can offer personalized feedback and recommendations, making it easier for candidates to enhance their resumes and increase their job prospects.

Features:

1. Resume Score

One of the key features of the Resume Analyzer is its ability to evaluate resumes and provide a resume score. This score reflects how well-structured and well-written the resume is, based on several criteria including clarity, relevance of content, keyword usage, and formatting. The tool assesses whether the resume meets industry standards and highlights areas for improvement. This score provides an immediate overview of the resume's quality and effectiveness, giving users insights into how they can optimize it for job applications.

2. Text Extraction from PDF Resumes Using NLP

The project uses NLP algorithms to extract important text from PDF resumes, which are often challenging due to formatting inconsistencies. The text extraction component reads through the PDF, identifies important sections like personal information, education, experience, and skills, and organizes them for further analysis. This feature allows the Resume Analyzer to work with a wide range of resumes, regardless of the document structure or formatting style. By accurately extracting this data, the tool can then analyze and provide recommendations specific to each resume.

3. Extracting Important Information

Once the text is extracted from the PDF, the Resume Analyzer processes it to identify the most relevant information. The system looks for crucial sections such as:

- **Work Experience**: Job roles, companies, and duration of employment.
- **Skills**: Both technical and soft skills mentioned in the resume.
- **Education**: Degrees, certifications, and relevant academic achievements.

- **Achievements and Projects**: Significant accomplishments and completed projects that are directly related to the job being applied for.

This feature helps users focus on key aspects of their resume, making sure the most important information is highlighted.

4. Job Suggestions

Based on the extracted content, the Resume Analyzer suggests relevant job roles that align with the user's qualifications, skills, and experience. The system compares the resume data with available job listings or predefined job roles, using machine learning algorithms to match users with potential opportunities. This feature reduces the time spent manually searching for jobs and increases the likelihood of finding a suitable position.

5. Skill Recommendations

The tool analyzes the resume to identify gaps in skills or qualifications that might improve the user's employability. It recommends additional skills that the candidate could acquire to make their resume more competitive. These recommendations are based on the job roles the user is interested in or the industry they are targeting, ensuring that the suggested skills are relevant to their career path.

6. Course Recommendations

To help users upskill and stay competitive, the Resume Analyzer provides personalized course recommendations. These courses are selected based on the user's current skills and the job roles they are aiming for. The courses focus on improving skills that may be underrepresented in the resume, offering a clear path for professional development.

7. Resume Score Analysis

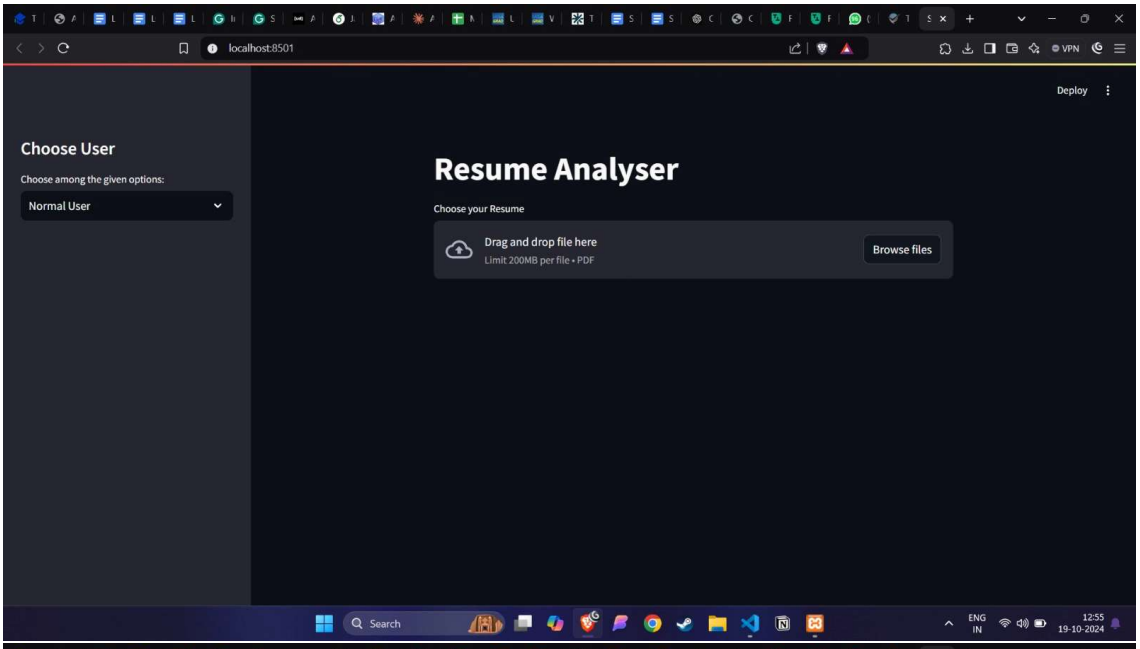
In addition to providing a resume score, the system offers a detailed breakdown of the score, explaining why certain sections scored higher or lower. This feedback helps users understand the strengths and weaknesses of their resumes, encouraging targeted improvements. For example, a low score in the "Experience" section might prompt users to elaborate on previous roles or include more detailed descriptions of their responsibilities and achievements.

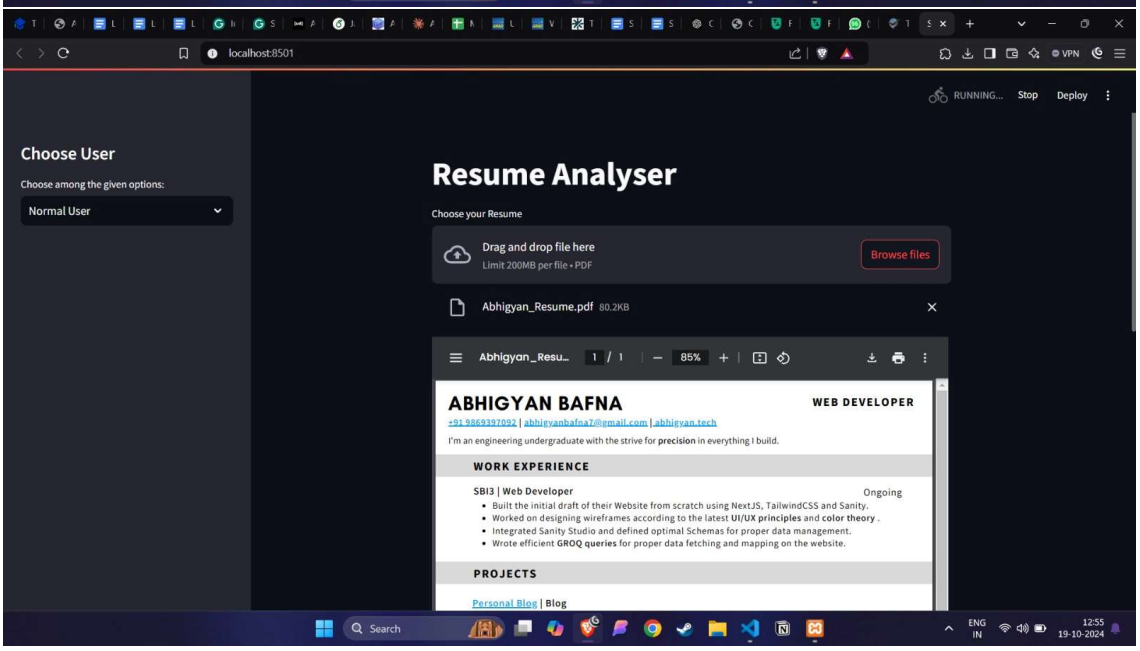
Advantages:

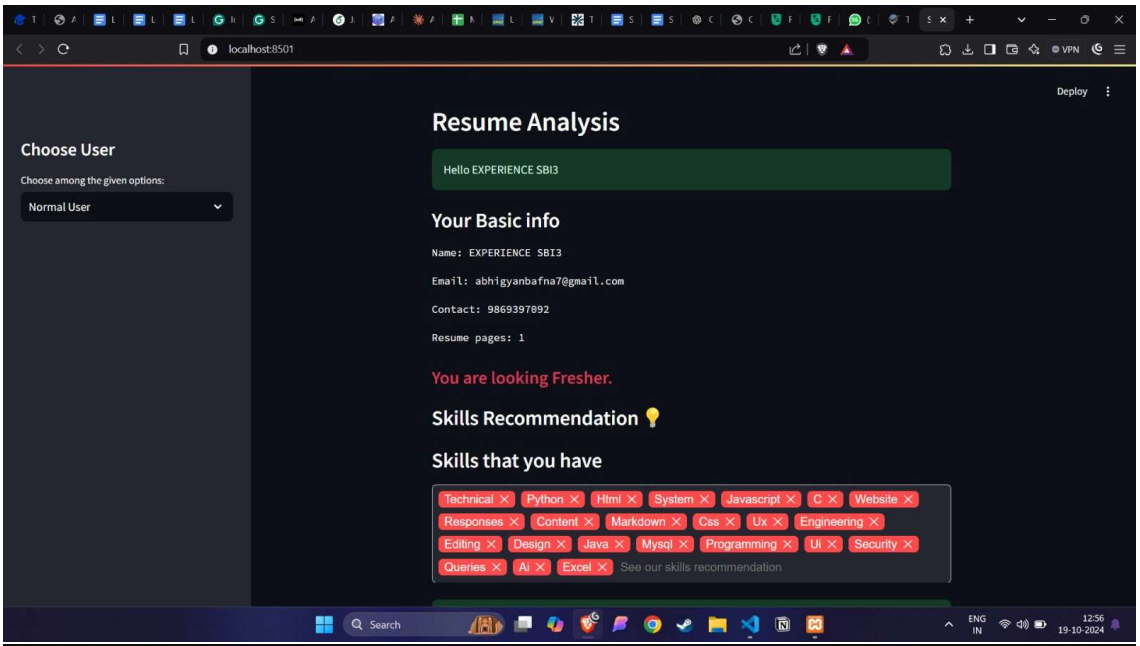
- **Automated Text Extraction**: The ability to extract text from PDF resumes, regardless of formatting, ensures that the tool can handle a wide variety of document types.
- **Personalized Feedback**: By analyzing the resume content, the tool provides tailored feedback that helps users improve specific sections.
- **Efficient Job Search**: The job recommendation feature saves users time by suggesting roles that match their qualifications.
- **Skill Enhancement**: The Resume Analyzer doesn't just evaluate resumes but also suggests skills and courses to help users advance in their careers.

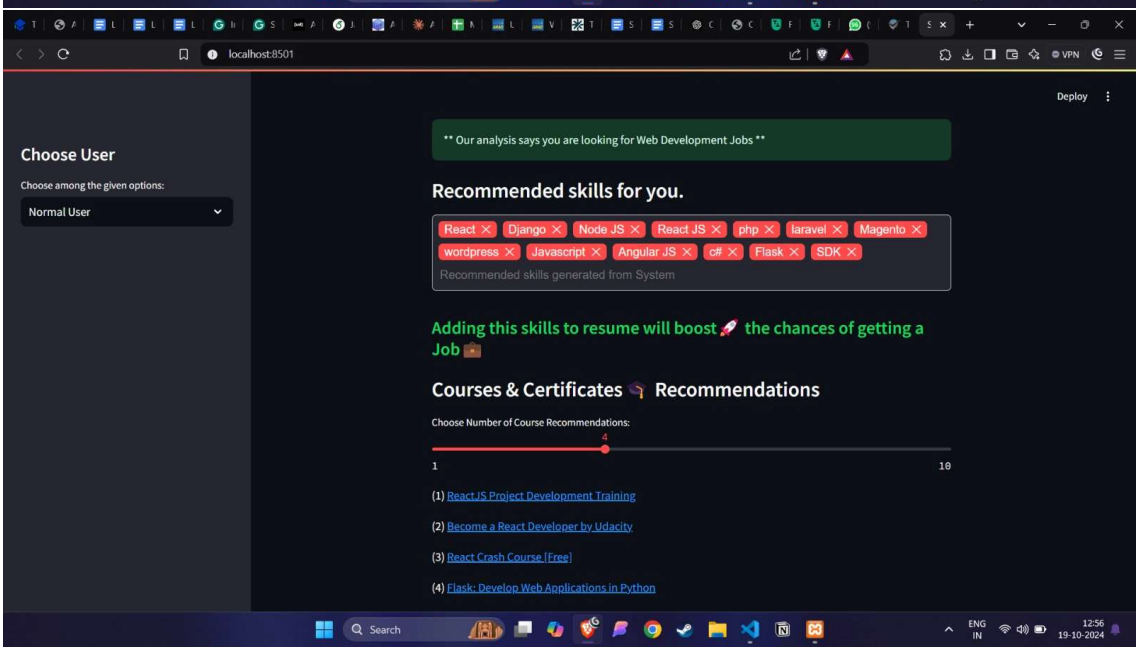
- **Improved Job Prospects**: By optimizing resumes for specific roles and industries, the tool increases the chances of a candidate landing a job interview.

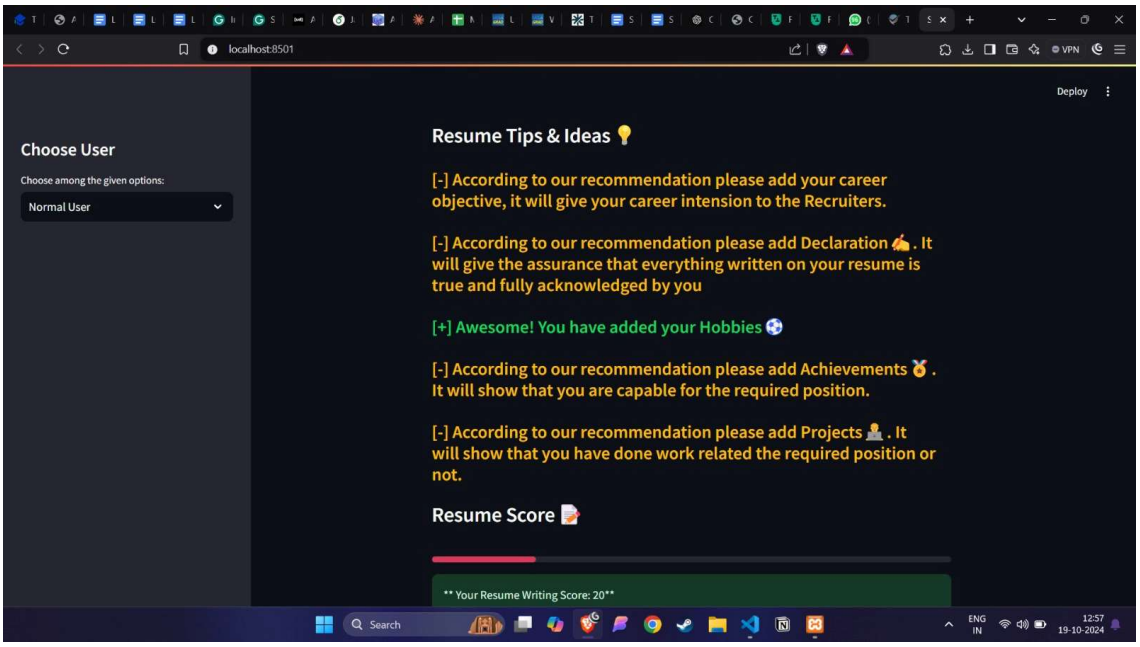
Output:






localhost:8501
Deploy
Choose User
Choose among the given options:
Normal User
Resume Analysis
Hello EXPERIENCE SBI3
Your Basic info
Name: EXPERIENCE SBI3
Email: abhigyanbafna7@gmail.com
Contact: 9869397092
Resume pages: 1
You are looking Fresher.
Skills Recommendation
Skills that you have
Technical Python Html System Javascript C Website
Responses Content Markdown Css Ux Engineering
Editing Design Java Mysql Programming Ui Security
Queries Ai Excel See our skills recommendation
Search
ENG IN
12:56 19-10-2024


localhost:8501
Deploy
Choose User
Choose among the given options:
Normal User
** Our analysis says you are looking for Web Development Jobs **
Recommended skills for you.
React Django Node JS React JS php laravel Magento
wordpress Javascript Angular JS c# Flask SDK
Recommended skills generated from System
Adding this skills to resume will boost the chances of getting a Job
Courses & Certificates Recommendations
Choose Number of Course Recommendations:
4
1
10
(1) React.JS Project Development Training
(2) Become a React Developer by Udacity
(3) React Crash Course [Free]
(4) Flask: Develop Web Applications in Python
Search
ENG IN
12:56 19-10-2024


localhost:8501
Deploy
Choose User
Choose among the given options:
Normal User
Resume Tips & Ideas
[-] According to our recommendation please add your career objective, it will give your career intension to the Recruiters.
[-] According to our recommendation please add Declaration . It will give the assurance that everything written on your resume is true and fully acknowledged by you
[+] Awesome! You have added your Hobbies
[-] According to our recommendation please add Achievements . It will show that you are capable for the required position.
[-] According to our recommendation please add Projects . It will show that you have done work related the required position or not.
Resume Score
** Your Resume Writing Score: 20**
Search
ENG
IN
12:57
19-10-2024

Arnav Malvia
2103109
C32
NLP

NLP

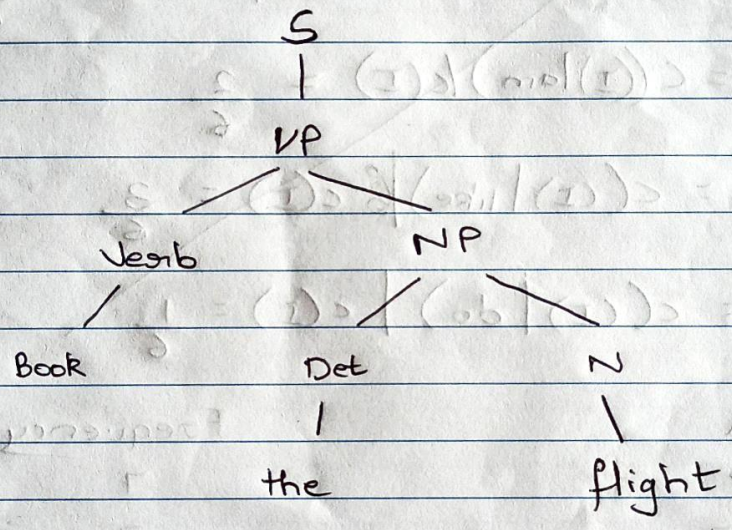
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Assignment

Q1 Using the following CFG, Parse the sentence "Book the flight"

S → NP | VP
S → VP
NP → Det | N
NP → Det | Adj | N
VP → V
VP → V | NP

Ans 1



Q2 Explain the n-gram language model. Give the following corpus.

<S> I am Henry </S>
<S> I like college </S>
<S> Do Henry like college </S>
<S> Henry I am </S>
<S> Do I like Henry </S>
<S> Do I like college </S>
<S> I do like Henry </S>

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

- Find which statement is more probable using a bigram model? Apply Laplace smoothing if required.
  i) I like college
  ii) do I like Henry.
- Predict the next probable word for "I like."

Ans 2 $P(I|<s>) = c(<s>|I) / c(<s>) = \frac{3}{7}$

$P(Do|<s>) = c(<s>|Do) / c(<s>) = \frac{3}{7}$

$P(Henry|<s>) = c(<s>|Henry) / c(<s>) = \frac{1}{7}$

$P(am|I) = c((I)|am) / c(I) = \frac{2}{6}$

$P(like|I) = c((I)|like) / c(I) = \frac{3}{6}$

$P(do|I) = c((I)|do) / c(I) = \frac{1}{6}$

Ans 2

Word	Frequency
<s>	7
</s>	7
I	6
am	2
Henry	5
like	5
college	3
do	4

Next word prediction probability w_{i-1} = do

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Next word	Probability Next word
P(<IS>|do)	0/4
P(<I>|do)	2/4
P(<am>|do)	0/4
P(<Henry>|do)	1/4
P(<like>|do)	1/4
P(<college>|do)	0/4
P(do|do)	0/4

Next word prediction probability w_{i-1} = Henry

Next Word	Probability Next Word = N/D
P(<IS>|Henry)	3/5
P(<I>|Henry)	1/5
P(<am>|Henry)	0
P(<Henry>|Henry)	0
P(<like>|Henry)	1/5
P(<college>|Henry)	0
P(<do>|do)	0

<IS> Do I like

Use Trigram

P<I like> = S

Next work w_{i-2} = I and w_{i-1} = like

Next word	Probability next word $= \frac{count(w_{i-2}, w_{i-1}, w)}{count(w_{i-2}, w_{i-1})}$
P(<s> | I like)	0/3
P(<I> | I like)	0/3
P(<am> | I like)	0/3
P(<Henry> | I like)	1/3
P(<like> | like)	0/3
P(<college> | I like)	2/3
P(do | I like)	0/3

4) <s> Do I like college?

Use Fourgram

$w_{i-3} = I$ $w_{i-2} = like$ $w_{i-1} = college$

Next Word	Probability Next Word $= \frac{count(w_{i-3}, w_{i-2}, w_{i-1}, w_i)}{count(w_{i-3}, w_{i-2}, w_{i-1})}$
P(<s> | I like college)	2/2
P(<I> | I like college)	0/2
P(<am> | I like college)	0/2
P(<Henry> | I like college)	0/2
P(<like> | I like college)	0/2
P(<college> | I like college)	0/2
P(do | I like college)	0/2

1) <s> I like college <s>

$= P(I \mid \langle b \rangle) \times P(like \mid I) \times P(college \mid like) \times P(\langle s \rangle \mid college)$

$= \frac{3}{7} \times \frac{3}{6} \times \frac{3}{5} \times \frac{3}{3} = \frac{9}{70} = 0.13$

2) <s> Do I like Henry </s>

$= P(do|<s>) \times P((I)|do) \times P(like|I) \times P(Henry|like) \times P(</s>|Henry)$

$= \frac{3}{7} \times \frac{2}{4} \times \frac{3}{6} \times \frac{2}{5} \times \frac{3}{5} = \frac{9}{350} = 0.0257$

Q3 List all bigrams and compute first 5 bigram probabilities for the same sentence given below:

<s> the students pass the test </s>

<s> the students wait for the pass </s>

<s> teachers test students </s>

Explain how spelling correction is done using N-grams.

Ans 3 Bigrams are

(the|<s>), (the|student), (student|pass), (pass|the), (the|test), (test|<s>), (<s>, the), (the|student), (student|wait) (wait|for), (for|the), (the|pass) (pass|</s>), (<s>|teachers) (teachers|test) (test|students), (students|</s>)

$P(the|student) = P(student|the) \,|\, P(student)$

$= 0$

$P(the|<s>) = P(<s>|the) \,|\, P(<s>)$

$= 2/3$

$P(student|the) = P(the|student) \,|\, P(the)$

$= 2/4$

$P(</s>|test) = P(test|</s>) \,|\, P(test)$

$= 1/2$

$P(student|test) = count(test, student) \,|\, count(test)$

$= 1/2$

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Spelling correction using N-gram is a simple and effective approach to address misspelled words in a text. N-grams are used to identify and suggest corrections for misspelled words by comparing them to correctly spelled words for language model.

i) Building an n-gram model : First, we need to arrange a corpus of text with correctly spelled words to build an n-gram language model.

ii) Tokenization into N-gram : The text in the corpus is tokenized into N-grams (typically bigrams or bigram).

iii) Counting N-grams: For each N-gram, you count the number of times it appears in the corpus.

iv) Spelling correction: To correct a mispelled word, you tokenize the misspelled words into N-gram and compare these N-grams to N-grams in your language model and calculate the likelihood probability of misspelled n-gram for each n-gram from misspelled, you consider possible correct alternatives.

Q4 The following POS tagged Corpus is given

<S> the/DT students/NN pass/V the/DT test/NN <\S>

<S> the/DT students/NN wait/V for/P the/DT pass/NN <\S>

<S> teachers/N test/V students/NN <\S>

Write down the transition and emission probabilities for a bigram HMM.

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Emission Probability

Words	Noun	Verb	Model	Preposition	DT
The	0	0	4/4 = 1	0	
students	3/5 = 0.6	0	0	0	
pass	0	1/3 = 0.3	0	0	
wait	0	1/3 = 0.3	0	0	
for	0	0	0	1/1 = 1	
teachers	1/5 = 0.2	0	0	0	
~~wait~~ test	1/5 = 0.2	1/3 = 0.3	0	0	

	N	V	M	P	<\s>
<s>	1	0	2	0	0
N	0	3	0	0	3
V	1	0	1	1	0
M	4	0	0	0	0
P	0	0	1	0	0

	N	V	M	P	<\s>
<s>	1/3 = 0.3	0	2/3 = 0.6	0	0
N	0	3/6 = 0.5	0	0	3/6 = 0.5
V	1/3 = 0.3	0	1/3 = 0.3	1/3 = 0.3	0
M	1	0	0	0	0
P	0	0	1	0	0

Q5 For each sentence, identify whether the different meanings arise from structural ambiguity, semantic ambiguity or pragmatic ambiguity?

a) Time flies like an arrow

b) He crushed the key to my heart.

Ans 5 In the two sentences, different meanings ~~wise~~ arise from different type ambiguity.

a) Time flies like an arrow

The ambiguity in this sentence is an example of structural ambiguity. It can be represented in two different ways based on structural grouping of data.

i) "Time flies" [i.e insects known as flies that exist over time] are similar to an arrow. In this interpretation, like an arrow is a simile indicating comparison between time flies and arrow.

ii) "Time" the concept of time passes quickly in a manner similar to an arrow. "Like an arrow" is a similie indicating comparision between speed of time and arrow.

b) He crushed the key to my heart

Semantic ambiguity - The word crushed is the source of semantic ambiguity. It can be interpreted in two different ways:

- Literally as in physically crushing into pieces
- Figuratively emotional impact (crushed) it is in emotions.

Q6 Identify the morphological type (Noun phrase, Verb phrase, Adjective phrase) of following sentence segments.
a) important to Bill
b) looked up the tree

Ans 6 a) Important to Bill

Important is an Adjective.
to Bill is prepositional phrase.

The entire phrase is therefore an adjective phrase.

b) Looked up the tree

"looked" is a verb.
"up the tree" is a prepositional phrase.

The entire segment is a verb phrase.

Q7 For the given corpus
N: Noun [Martin, Justin, Will, Spot, Pat]
M: Modal [can, will]
V: Verb [watch, spot, pat]

Statement: Justin will spot will.

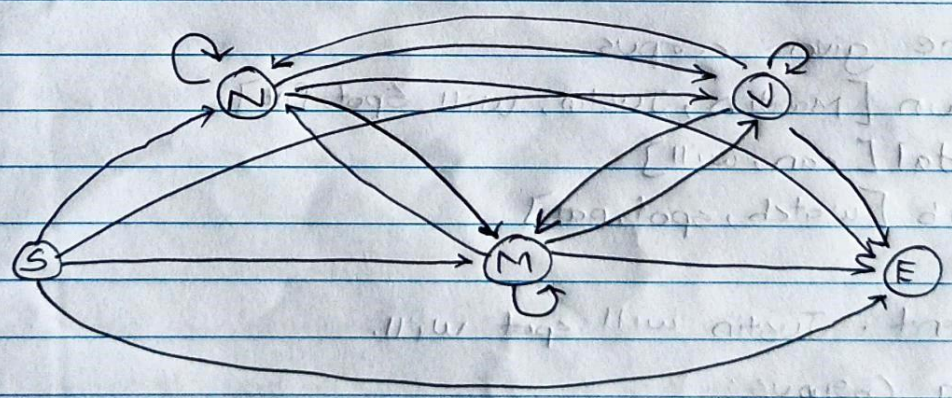
Training Corpus:
<S> Martin Justin can watch will <E>
<S> Spot will watch Martin <E>
<S> Will Justin spot Martin <E>
<S> Martin will pat spot <E>

Emmission Probability Matrix

	N	M	V
Martin	4/10	0	0
Justin	2/10	0	0
can	0	1/3	0
watch	0	0	2/4
will	2/10	2/3	0
spot	2/10	0	1/4
pat	0	0	1/4

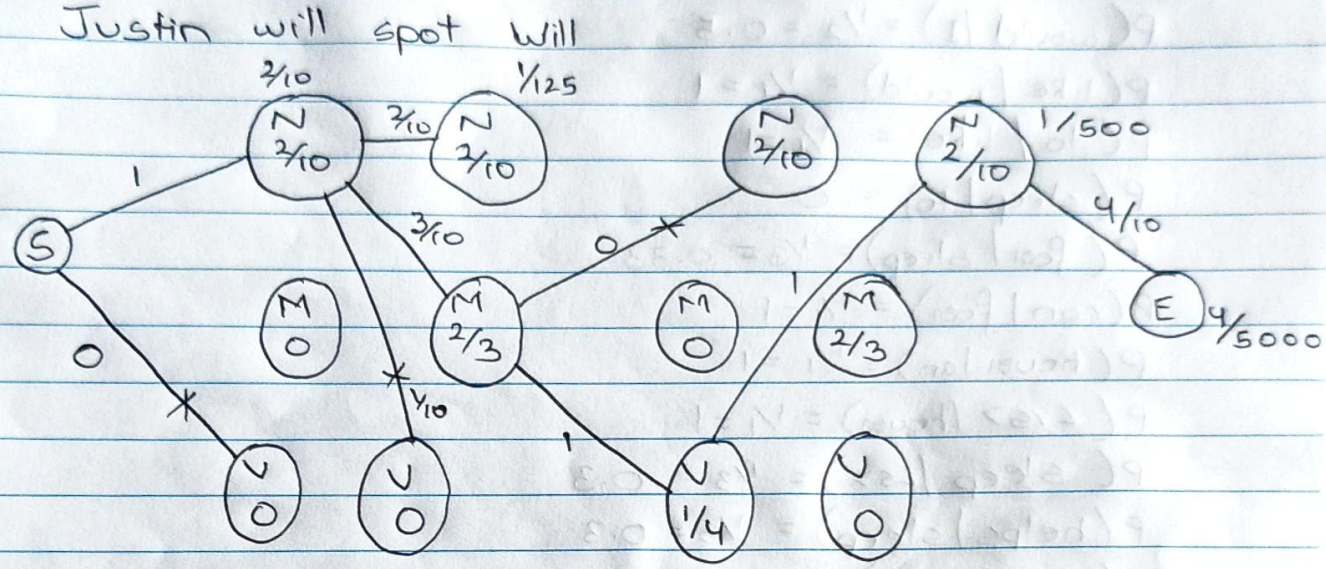
Transition Probability Matrix

	N	M	V	E
S	4/4	0	0	0
N	2/10	3/10	1/10	4/10
M	0	0	3/3	0
V	4/4	0	0	0



Martin Justin can watch will spot pat.

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE



S → N → M → V → N → E

Q8 List all possible bigrams. Compute conditional probabilities and predict the next word for the word "to".
Consider the following corpus:

<S> I tell you to sleep and rest <\S>
<S> I would like to sleep for an hour <\S>
<S> Sleep helps one to relax <\S>

P(I/<S>) = count(<S> I) /count <S> = 2/3 = 0.66
P(tell/I) = 1/2 = 0.5
P(you/tell) = 1/1 = 1
P(to/you) = 1/1 = 1
P(sleep/to) = 2/3 = 0.66
P(and/sleep) = 1/3 = 0.33
P(rest/and) = 1/1 = 1
P(<\S>/rest) = 1/1 = 1

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

P(would / I) = 1/2 = 0.5
P(like / would) = 1/1 = 1
P(to / like) = 1/1 = 1
P(sleep / to) = 0
P(for / sleep) = 1/3 = 0.33
P(can / for) = 1/1 = 1
P(hour / an) = 1/1 = 1
P(</s> / hour) = 1/1 = 1
P(sleep / <s>) = 1/3 = 0.3
P(helps / sleep) = 1/3 = 0.3
P(one / helps) = 1/1 = 1
P(to / one) = 1/1 = 1
P(relax / to) = 1/3 = 0.33
P(</s> / relax) = 1/1 = 1

Highest ⇒ P(sleep / to) = 0.66
∴ word is "sleep"

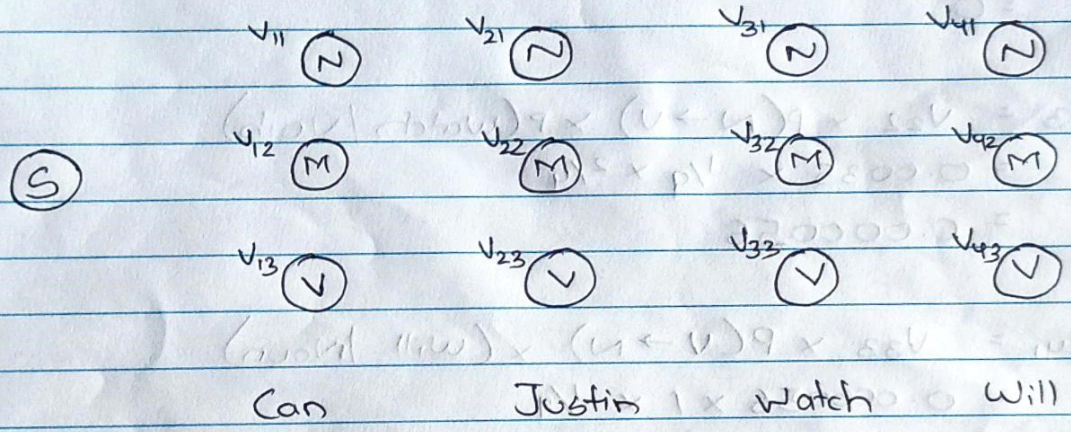
Q9 Emission Probability

Words	Noun	Model	Verb
Martin	4/9	0	0
Justin	2/9	0	0
an	0	1/4	0
watch	0	0	3/4
will	1/9	3/4	0
spot	2/9	0	1/4
pat	0	0	1/4

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Transition Probability

	N	M	V	</S>
<S>	3/4	1/4	0	0
N	1/9	3/9	1/9	4/9
M	1/4	0	3/4	0
V	4/4	0	0	0



i) $S \rightarrow N$

$V_{11} = P(S \rightarrow N) \times P(\text{can}/\text{Noun})$

$= 3/4 \times 0 = 0$

$V_{12} = P(S \rightarrow N) \times P(\text{can}/\text{Model})$

$= 0.062$

$V_{13} = P(S \rightarrow V) \times P(\text{can}/\text{Verb})$

$= 0$

ii) $V_{21} = V_{12} \times P(M \rightarrow N) \times (\text{Justin}/\text{Noun})$

$= 1/16 \times 1/4 \times 2/9$

$= 0.0034$

$V_{22} = V_{12} \times P(N \rightarrow M) \times (\text{Justin}/\text{Model})$

$= 1/16 \times 3/9 \times 0$

$= 0$

$V_{23} = V_{12} \times P(N \to V) \times (\text{Justin}/\text{Verb})$
$= 0$

iii) $V_{31} = V_{22} \times P(N \to N) \times P(\text{Watch}/\text{Noun})$
$= 0.0035 \times 1/9 \times 0$
$= 0$

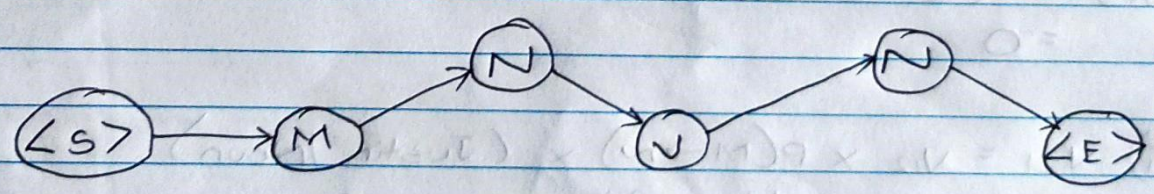
$V_{32} = V_{22} \times P(N \to M) \times P(\text{Watch}/\text{Model})$
$= 0$

$V_{33} = V_{22} \times P(N \to V) \times P(\text{watch}/\text{Verb})$
$= 0.0035 \times 1/9 \times 2/4$
$= 0.000056$

iv) $V_{41} = V_{33} \times P(V \to N) \times (\text{Will}/\text{Noun})$
$= 0.000056 \times 1 \times 1/9$
$= 9.55 \times 10^{-6}$

$V_{42} = 0$
$V_{43} = 0$

$\therefore$ Final probability is $9.55 \times 10^{-6} \times 4/9 = 4.24 \times 10^{-6}$



Can Justin watch Will

Q10 <S> Book a car </S>
<S> Park the car </S>
<S> The book is in the car </S>
<S> The car is in a park </S>

"The park is a book"

	N	V	Det	Prep
Book	1/6	1/4	0	0
a	0	0	2/6	0
car	4/6	0	0	0
the	0	0	4/6	0
is	0	2/4	0	0
in	0	0	0	2/2
park	1/6	1/4	0	0

	N	V	Det	Prep	</S>
<S>	0	2/4	2/4	0	0
N	0	2/6	0	0	4/6
V	0	0	2/4	2/4	0
Det	6/6	0	0	0	0
Prep	0	0	2/2	0	0

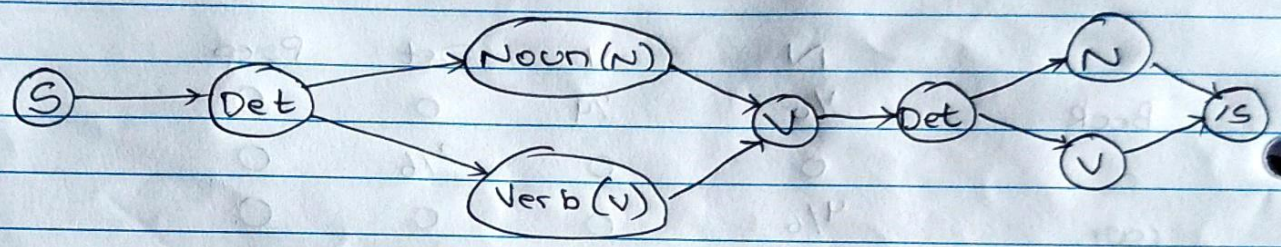
Transition Probability Matrix

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Data : <s> The park is a book </s>

The	Park	is	a	book
Det	Noun Verb	verb	Det	Noun Verb

$1 \times 2 \times 1 \times 1 \times 2 = 4$ ways



$P(\text{Det} / \text{The}, \langle s \rangle) = \frac{4}{6} \times \frac{2}{4} = \frac{1}{3}$

$P(\text{Noun} / \text{Park}, \text{Det}) = \frac{1}{6} \times \frac{1}{3} = \frac{1}{18}$

$P(\text{Verb} / \text{Park}, \text{Det}) = \frac{1}{400} \times \frac{1}{3} = \frac{1}{1200}$

$\because \frac{1}{18} > \frac{1}{1200}$

$\therefore$ Park is a noun.

$P(\text{Verb} / \text{is}, \text{Noun}) = \frac{1}{16} \times \frac{1}{18} = \frac{1}{108}$

$P(\text{Det} / \text{a}, \text{verb}) = \frac{1}{6} \times \frac{1}{108} = \frac{1}{648}$

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

book as noun

$P(\text{Noun} / \text{book}, \text{Det}) = \frac{1}{6} \times \frac{1}{648} = \frac{1}{3888}$

~~book as verb~~

~~$P(\text{verb} / \text{book}, \text{Det}) = \frac{1}{6} \times \frac{1}{648} = \frac{1}{}$~~

book as verb

$P(\text{verb} / \text{book}, \text{Det}) = \frac{1}{4} \times 0 = 0$

∴ Book is a noun

Result after applying HMM model

<s>	The	Park	is	a	book	</s>
	Det	Noun	verb	Det	Noun	

Q11 Calculate the transmission and emission probabilities for the set of sentences below:

<s> Mary Jane can see will </s>
<s> Spot will see Mary </s>
<s> Will Jane spot Mary </s>
<s> Mary will ~~spot~~ pat spot </s>

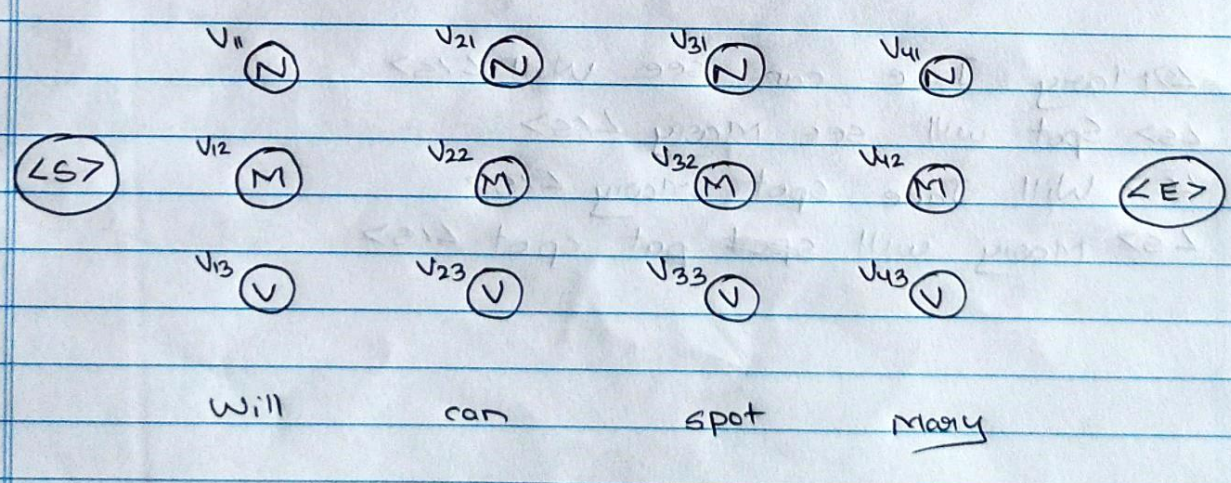
THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Emission probability Matrix

	N	M	V
Mary	4/9	0	0
Jane	2/9	0	0
can	0	1/4	0
see	0	0	2/4
will	1/9	3/4	0
spot	2/9	0	1/4
pat	0	0	1/4

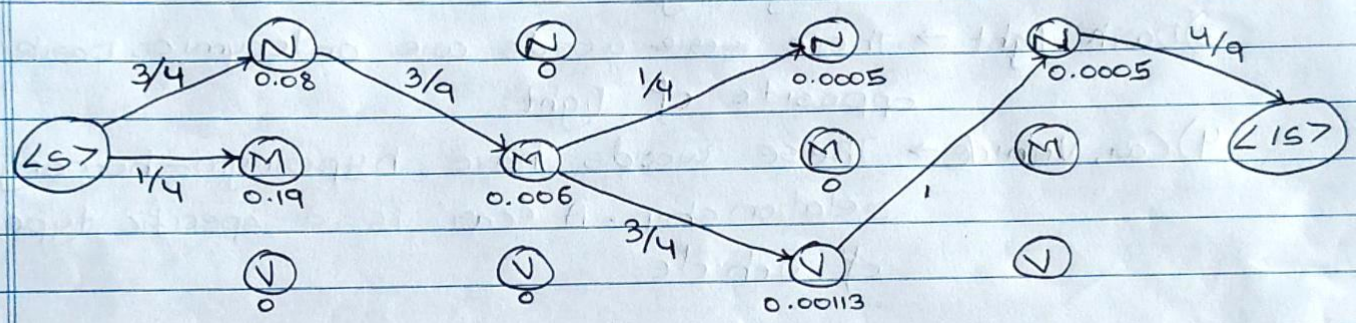
Transmission probability Matrix

	N	M	V	<S>
<S>	3/4	1/4	0	0
N	1/9	3/9	1/9	4/9
M	1/4	0	3/4	0
V	4/4	0	0	0

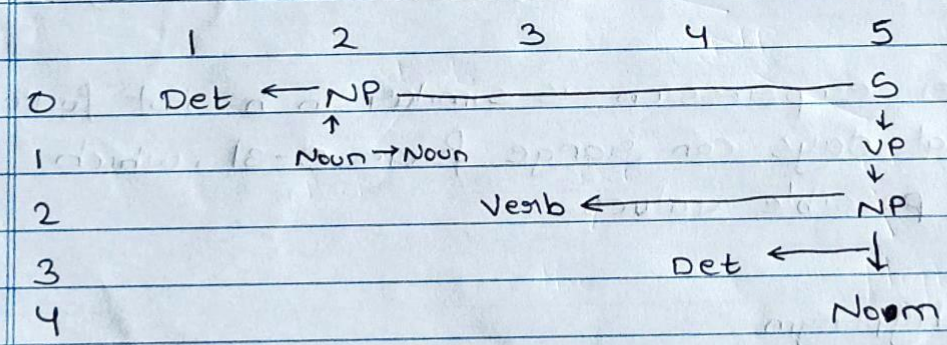


Q12 Tag the sentence "Will can spot Mary" using Viterbi algorithm.

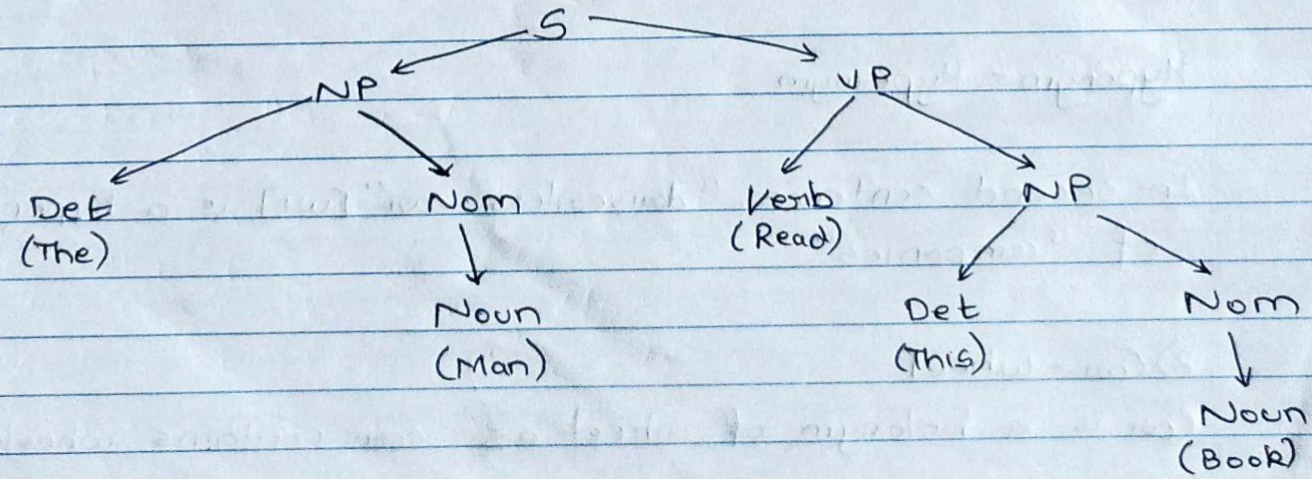
Ans 12 Using the transmission and emission probabilities from the previous numericals



Q13 0 The 1 man 2 read 3 this 4 book 5



This can be connected to tree as follow



Q14 1) Write, Right → They are homophones, which means they sound the same but have unrelated meanings and spelling.

2) Big, Large → These words are synonic. They both mean 'having a greater size'.

3) Dark, light → These ~~were~~ words are antonymic. Dark is opposite of light.

4) Car, Vehicle → These words have hyperinym-homonym relationship. A 'car' is a specific type of vehicle.

Q15 Ans is c

c) (19|20)\d{2}-(0[1-9]|1[0-2])-([0-2][1-9]|3[0-1])

This regular expression is similar in b but further specifies that days can range from 0-31, which covers all possible values.

Q16 Holonym → Hyponym

"Invention" [w1] is a Holonym of "the wheel" in sentence 1.

Hyponym - Hyperinym

In second sentence, "driverless car" [w1] is a hyponym of "companies"

2) Car - wheel

Car is a holonym of wheel as car contains wheels.

3) daffodils - flower
daffodils is a hyponym of flower as daffodils are a type of flower.

4) flower - daffodils
flower is the hyperonym of daffodils as it is the broader category that includes daffodils.

Thus only option '4' is correct.

Q17 Consider the corpus
<s> I tell you to sleep and rest </s>
<s> I would like to sleep for an hour </s>
<s> Sleep helps one to relax </s>
List all possible bigrams. Compute conditional probabilities and predict the next word for the word "to".

Ans 17 (<s>, I), (I, tell), (tell, you), (you, to), (to, sleep), (stop, and), (and, rest), (rest, </s>)

(<s>, I), (I, would), (would, like), (like, to), (to, sleep), (sleep, for), (for, an), (an, hour), (hour, </s>)

(<s>, sleep), (sleep, helps), (helps, one), (one, to), (to, relax), (relax, </s>)

i) For sentence 1:

Conditional Probability

$$P(A|B) = \frac{count(B,A)}{count(B)}$$

$$P(I|<s>) = \frac{count(<s>, I)}{count <s>} = \frac{2}{3}$$

$$P(tell|I) = \frac{count(I, tell)}{count(I)} = \frac{1}{2}$$

$$P(you|tell) = \frac{count(tell, you)}{count(tell)} = \frac{1}{1}$$

$$P(to|you) = \frac{count(you, to)}{count(you)} = \frac{1}{1}$$

$$P(sleep|to) = \frac{count(to, sleep)}{count(to)} = \frac{2}{3}$$

$$P(and|sleep) = \frac{count(sleep, and)}{count(sleep)} = \frac{1}{3}$$

$$P(rest|and) = \frac{count(and, rest)}{count(and)} = \frac{1}{1}$$

$$P(</s>|rest) = \frac{count(rest, </s>)}{count(rest)} = \frac{1}{1}$$

ii) $$P(would|I) = \frac{count(I|would)}{count(I)} = \frac{1}{2}$$

$$P(like|would) = \frac{count(would|like)}{count(would)} = \frac{1}{1}$$

$$P(to|like) = \frac{count(like, to)}{count(like)} = \frac{1}{1}$$

$$P(for|sleep) = \frac{count(sleep, for)}{count(sleep)} = \frac{1}{3}$$

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

$$P(an \mid for) = \frac{count(for, an)}{count(for)} = \frac{1}{1}$$

$$P(hour \mid an) = \frac{count(an, hour)}{count(an)} = \frac{1}{1}$$

$$P(\langle /s \rangle \mid hour) = \frac{count(hour, \langle /s \rangle)}{count(hour)} = \frac{1}{1}$$

iii) $$P(sleep, \mid \langle s \rangle) = \frac{count(\langle s \rangle, sleep)}{count(\langle s \rangle)} = \frac{1}{3}$$

$$P(helps, sleep) = \frac{count(sleep, helps)}{count(sleep)} = \frac{1}{3}$$

$$P(one, helps) = \frac{count(helps, one)}{count(helps)} = \frac{1}{1}$$

$$P(to, one) = \frac{count(one, to)}{count(one)} = \frac{1}{1}$$

$$P(relax, to) = \frac{count(to, relax)}{count(to)} = \frac{1}{3}$$

$$P(\langle /s \rangle \mid relax) = \frac{count(relax, \langle /s \rangle)}{count(relax)} = \frac{1}{1}$$

Since the $P(sleep \mid to) > P(relax \mid to)$

$$\frac{2}{3} > \frac{1}{3}$$

$\therefore$ The next predicted word for 'to' is 'sleep'.

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

17b Explain CFG with suitable example : Discuss the following potential problems in CFG as: 1) Agreement 2) Sub categorization 3) Movement.

A CFG or Context Free Grammar is a formal grammar that consits of a set of production rules used to generate strings in given formal language defined as

$G = (V, \Sigma, R, S)$ where

V - Finite set of variables (NT)

Σ - Finite set of terminals (T)

R - Finite set of production rules that maps a NT to another string of T or NT.

S - Start symbol

Thus, CFG contains a list of rules that define the set of all well formed sentences. Each rule has LHS. which identifies syntactic category and RHS that defines alternative component parts, reading from left to right.

Example:

Production rules

S → NP VP	Det → 'a' | 'the'
NP → Det Noun | NP PP	Noun → 'cake' | 'child' | 'fork'
PP → Pre NP	Pre → 'with'
VP → Verb NP	Verb → 'ate'

Here all above rules are belonging to set R.

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

$V = \{S, NP, VP, PP, Det, Noun, Verb, Aux, Pre\}$
$\Sigma = \{'a', 'ate', 'cake', 'child', 'fork', 'the', 'with'\}$
$S = S$

Problems :-

1) Agreement
In NLP agreement between subjects and verbs (or nouns and adjectives) is crucial.
For eg 'The dog barks' vs 'The dogs bark' cannot be captured by a CFG as it cannot distinguish between singular and plural subjects.

2) Sub categorisation
Some verbs need specific types of complements.
Eg- 'give' needs a direct and indirect object which must be specified by CFG as to which verbs can take which phrases as arguments.

3) Movement
NLP allow for movement like wh questions or topicalization. For eg in question 'What did you eat'. Object moves to front which is challenging for CFG that follow fixed hierarchical structure.

Q18 Name the ambiguity in the following sentences

1) The boy saw the girl with the telescope.
Syntactic Ambiguity
Can mean boy used telescope to see the girl or girl had telescope when the boy saw her.

THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

2) The horse ran up the hill. It was very steep. It soon got tired.

Pronoun Reference Ambiguity

The first 'it' refers to the hill and the second 'it' refers to the horse.

3) The car hit the pole while it was moving.

Syntactic Ambiguity

It can refer both the car or pole. ie. the car was moving when it hit the pole or the pole was moving when it was hit by the car.