Grp no: 38

Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./Miss Parth Sandeep Dabholkar of Computer Department, Semester VII with Roll No. 2103032 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 15/10/24

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

Grade

EXPERIMENT NO: 1

AIM: Study Various applications of NLP and NLP Tools. Familiarise students with various tools and libraries available for NLP tasks.

THEORY: Natural Language Processing (NLP) libraries are essential collections of tools, functions, and pre-built algorithms that assist developers and researchers in processing and analyzing human language data. These libraries offer the building blocks needed to create applications capable of understanding, interpreting, and generating human language, making them invaluable for tasks such as text analysis, sentiment detection, translation, and more.

Some of the key functionalities that NLP libraries typically offer:

1. **Text Preprocessing**: Tokenization, stopword removal, stemming, and lemmatization.

2. **Text Analysis**: Part-of-speech tagging, named entity recognition (NER), sentiment analysis, and dependency parsing.

3. **Text Classification**: Topic modelling, text categorization, and spam detection.

4. **Text Generation**: Language modelling, machine translation, and text summarization.

5. **Advanced Techniques**: Word embeddings, transformer models (like BERT, GPT), and sequence-to-sequence models.

Popular NLP libraries

Popular NLP libraries include NLTK (for comprehensive text processing), spaCy (for fast, production-ready tasks), Hugging Face Transformers (for advanced transformer models), and AllenNLP (for research-focused custom model development). These libraries simplify the implementation of complex NLP tasks.

1. NLTK (Natural Language Toolkit)

a. Description:

NLTK is a long-established and popular library in the NLP field, offering a wide range of tools for processing human language, such as tokenization, parsing, and classification. It's especially useful for educational purposes and for beginners in NLP.

b. Key Features:

Provides a rich collection of text datasets and linguistic resources.

Offers comprehensive tools for analyzing and processing language data.

Includes a wide range of algorithms for text classification, tokenization, stemming, tagging, parsing, and semantic reasoning, all with an easy-to-use interface.

c. Use Cases:

NLTK can tokenize text, remove stop words, and perform stemming or lemmatization, making it essential for preparing text data for machine learning models.

2. spaCy

a. Description:

spaCy is a powerful NLP library built for high performance and production environments. It is recognized for its speed, accuracy, and user-friendly interface. With pre-trained models for multiple languages, spaCy is extensively used in industry to develop NLP applications.

b. Key Features:

Quick and efficient tokenization.

Pre-trained word embeddings and deep learning models.

Compatibility with deep learning frameworks like TensorFlow and PyTorch.

Support for NER (Named Entity Recognition), part-of-speech tagging, and syntactic dependency parsing.

c. Use Cases:

Production-level NLP applications, information extraction, text analysis, and building NLP pipelines.

3. Hugging Face Transformers

a. Description:

Hugging Face Transformers is a popular library for working with transformer models like BERT, GPT, and T5. It offers easy-to-use APIs for training, fine-tuning, and deploying transformer-based models for various NLP tasks. The library supports a wide range of pre-trained models and is widely adopted in both research and industry.

b. Key Features:

Access to a vast collection of pre-trained models like BERT, GPT, RoBERTa, and T5, which can be used for various NLP tasks without the need for extensive training.

Easily fine-tune pre-trained models on specific datasets for tasks like text classification, question answering, and summarization.

Compatible with major deep learning frameworks, including TensorFlow and PyTorch, allowing flexibility in model deployment.

Provides efficient tokenizers that are optimized for different transformer models, ensuring quick and accurate text preprocessing.

Simplifies common NLP tasks (e.g., sentiment analysis, translation, and text generation) with ready-to-use pipelines that handle all steps from tokenization to inference.

c. Use Cases:

Advanced NLP tasks, research, chatbot development, sentiment analysis, and language translation.

4. AllenNLP

a. Description:

AllenNLP is an open-source NLP research library built on top of PyTorch. It is designed to facilitate the development of NLP models and experiments, particularly for academic and research purposes. AllenNLP provides easy-to-use abstractions and components for building custom models and experiments.

b. Key Features:

Modular and extensible architecture for building custom NLP models.

Pre-built models for various NLP tasks like reading comprehension, text classification, and semantic role labeling.

Support for deep learning models with PyTorch integration.

Tools for managing datasets, training, and evaluation.

c. Use Cases:

NLP research, academic projects, and experimentation with custom models.

Introduction to Cloud-Based NLP Platforms

1. Google Cloud NLP

a. Description:

Google Cloud Natural Language API provides powerful tools for analyzing and understanding text. It offers pre-trained models that can perform tasks like sentiment analysis, entity recognition, syntax analysis, and language detection. The API is fully managed and scalable, making it suitable for integrating NLP into various applications.

b. Key Features:

Sentiment analysis, entity recognition, syntax analysis, and content classification.

Support for multiple languages.

Integration with other Google Cloud services like BigQuery and AutoML.

Scalable and easy to use through RESTful APIs.

c. Use Cases:

Sentiment analysis for social media monitoring, customer feedback analysis, document classification, and automated content tagging.

2. AWS Comprehend

a. Description:

AWS Comprehend is a fully managed NLP service by Amazon Web Services. It uses machine learning to find insights and relationships in text. The service can identify the language, extract key phrases, entities, sentiment, and more from unstructured text.

b. Key Features:

Language detection, entity recognition, key phrase extraction, and sentiment analysis.

Custom entity recognition using AutoML.

Integration with the AWS ecosystem (S3, Lambda, etc.).

Real-time processing and batch processing capabilities.

c. Use Cases:

Analyzing customer reviews, understanding product feedback, content analysis, and knowledge extraction from documents.

3. Azure Text Analytics

a. Description:

Azure Text Analytics is a cloud-based service provided by Microsoft Azure that offers advanced NLP capabilities. It provides tools for sentiment analysis, key phrase extraction, language detection, and named entity recognition. Azure Text Analytics can be integrated into applications to gain insights from unstructured text data.

b. Key Features:

Sentiment analysis, key phrase extraction, entity recognition, and language detection.

Scalable and integrated with other Azure services like Azure Cognitive Services.

Easy-to-use REST APIs.

Support for multiple languages.

c. Use Cases:

Sentiment analysis for customer feedback, text analytics for business intelligence, automating document processing, and enhancing customer support systems.

Conclusion:

Students have developed a strong understanding of essential NLP libraries and cloud-based platforms, equipping them with the skills to apply these tools in practical, real-world applications.

EXPERIMENT NO: 2

MINI-PROJECT: Document Summarization Tool

OBJECTIVE: Expose to wide range of NLP applications and enable them to select a real-world application for Mini-Project.

Applications covered:

1. Sentiment Analysis:
- Understanding Opinions: Analyzes text to determine the sentiment or emotional tone, such as positive, negative, or neutral.
- Business Insights: Helps companies gauge customer satisfaction and opinions about products or services.

2. Machine Translation:
- Language Conversion: Translates text from one language to another, enabling cross-language communication.
- Global Accessibility: Facilitates access to content and services in multiple languages, breaking down language barriers.

3. Chatbots and Virtual Assistants:
- Automated Responses: Provides instant replies and assistance to user queries through conversational interfaces.
- Customer Support: Enhances customer service by handling routine inquiries and tasks efficiently.

4. Text Summarization:
- Information Extraction: Condenses long documents into shorter summaries while retaining key information.
- Efficiency: Saves time for users by providing concise overviews of extensive content.

5. Named Entity Recognition (NER):
- Entity Identification: Identifies and classifies entities like names of people, organizations, and locations in text.
- Data Structuring: Assists in organizing and extracting structured data from unstructured text sources.

6. Speech Recognition:
- Voice-to-Text Conversion: Converts spoken language into written text, enabling hands-free interaction.
- Accessibility: Supports users with disabilities and provides an alternative input method for various applications.

7. Text Classification:
- Categorization: Automatically assigns predefined categories or tags to text documents based on their content.
- Content Organization: Enhances search and retrieval processes by organizing information into relevant categories.

8. Information Retrieval:

- Search Optimization: Improves the accuracy and relevance of search engine results based on natural language queries.
- Data Extraction: Helps users find specific information quickly from large datasets or document collections.

9. Language Generation:
- Content Creation: Generates human-like text for applications such as content writing, storytelling, and report generation.
- Personalization: Creates tailored content based on user preferences and context.

10. Question Answering Systems:
- Direct Responses: Provides accurate answers to user questions posed in natural language, often using a knowledge base.
- Enhanced Interaction: Improves user experience by offering precise and relevant information quickly.

Literature Review of Recent NLP Advancement and Applications:

1. Document Summarization Tool Overview:

The document text summarization project aims to develop an advanced algorithm that can automatically generate concise summaries of lengthy texts. Leveraging natural language processing and machine learning techniques, the system will analyze the content to identify key information and themes. It will then produce a coherent and informative summary, preserving the original context and relevance. This tool will be beneficial for quickly grasping the essence of extensive documents, enhancing productivity and decision-making across various domains.

2. Recent Advancements:

a. Transformers and Pre-trained Models: Models like GPT-4, BERT, and T5 have improved summarization accuracy by leveraging deep learning to understand context and nuances in text. These models are pre-trained on vast datasets and fine-tuned for specific summarization tasks.

b. Extractive and Abstractive Summarization: Modern tools combine both approaches. Extractive methods identify and extract key sentences, while abstractive methods generate new summaries that may rephrase or synthesize the original content. The integration of these methods offers more comprehensive summaries.

c. Fine-tuning and Transfer Learning: Techniques such as fine-tuning pre-trained models on domain-specific data have enhanced summarization for specialized fields, resulting in summaries that are more relevant and accurate for particular industries.

d. Multi-document Summarization: Advances in algorithms enable the aggregation and summarization of information from multiple documents, providing a more holistic view of a topic by consolidating diverse sources.

e. Interactive Summarization: Some systems now offer interactive features, allowing users to refine and adjust summaries based on their needs, which improves the relevance and utility of the generated content.

3. Applications of Document Summarization Tool:

- **Healthcare**: Condensing patient records, research papers, and medical reports to streamline clinical decision-making and facilitate more efficient medical research.
- **Business and Finance**: Generating summaries of financial reports, market analyses, and business documents to support quick decision-making and strategy development.
- **Education**: Assisting students and educators by summarizing academic papers, textbooks, and study materials for easier comprehension and review.
- **News and Media**: Providing concise summaries of news articles and reports to help readers stay informed without having to read lengthy texts.
- **Customer Support**: Summarizing customer feedback, support tickets, and interaction logs to identify key issues and trends, improving service and response strategies.

Problem Statement for Mini-Project:

Title: Document Summarization Tool using Natural Language Processing

Problem Statement:

In today's information-rich environment, professionals across various domains face the challenge of managing and extracting key insights from extensive textual documents. The goal of this mini-project is to develop a document summarization tool that leverages Natural Language Processing (NLP) techniques to automatically generate concise, coherent, and contextually accurate summaries of lengthy texts. The tool should be able to handle diverse types of documents, including news articles, research papers, and business reports, providing summaries that retain essential information while reducing the overall length. The objective is to enhance information accessibility, streamline decision-making processes, and improve productivity by offering an efficient and effective summarization solution.

Problem Objective:

- **Develop a Summarization Model**: Create a model using advanced NLP techniques, such as transformers, to generate high-quality summaries that accurately reflect the main ideas and essential details of various types of documents.
- **Ensure Contextual Accuracy**: Implement mechanisms to preserve the context and meaning of the original text, ensuring that the summaries are coherent and relevant.
- **Handle Diverse Document Types**: Build a tool capable of processing and summarizing different kinds of documents, including news articles, academic papers, and business reports.
- **Optimize Performance**: Evaluate and refine the model to achieve a balance between summary length and information retention, ensuring efficiency and effectiveness in generating concise summaries.

- **Provide User-Friendly Output**: Design an intuitive interface that allows users to easily input documents and obtain summaries, enhancing the tool's accessibility and usability for diverse applications.

Development Process:

1. Define Requirements and Scope:

- Identify the types of documents to be summarized (e.g., news articles, academic papers, business reports).
- Determine specific requirements for summary length, accuracy, and coherence.
- Establish performance metrics for evaluation (e.g., ROUGE score, readability).

2. Data Collection and Preparation:

- Gather a diverse dataset of documents relevant to the chosen types.
- Annotate or acquire pre-summarized versions of these documents for training and evaluation.
- Preprocess the data by cleaning text, tokenizing, and handling any specific formatting issues.

3. Model Selection and Training:

- Choose an appropriate NLP model architecture (e.g., BERT, GPT-4, T5) based on the project requirements.
- Fine-tune the pre-trained model on the dataset, using techniques such as transfer learning to adapt it for document summarization.
- Implement both extractive and abstractive summarization methods if needed, combining them for improved results.

4. Tool Development and Integration:

- Develop the core functionality of the summarization tool, integrating the trained model with a user interface or API.
- Implement features for document input, summary generation, and output presentation.
- Ensure the tool handles various document formats and sizes effectively.

5. Evaluation and Optimization:

- Test the tool using a separate validation dataset to assess its performance against predefined metrics.
- Collect feedback from users or conduct user studies to identify areas for improvement.
- Refine the model and tool based on evaluation results and feedback to enhance accuracy, coherence, and usability.

6. Deployment and Documentation:

- Deploy the tool for use, ensuring it is accessible to the intended users (e.g., through a web interface or software application).
- Document the development process, including setup instructions, usage guidelines, and technical details.
- Provide support and maintenance plans to address any issues and updates.

7. Post-Deployment Monitoring:

- Monitor the tool's performance in real-world scenarios to ensure it meets user needs and expectations.
- Gather ongoing feedback and make iterative improvements based on actual usage and emerging requirements.

Project Outcomes:

Effective Summarization Tool:

- A functional document summarization tool that generates concise, accurate summaries of lengthy texts across various types of documents, including news articles, academic papers, and business reports.

Improved Efficiency:

- Enhanced productivity for users by significantly reducing the time needed to extract key information from extensive documents, facilitating quicker decision-making and information retrieval.

In conclusion, the successful implementation of the document summarization tool demonstrates our proficiency in applying advanced NLP techniques to practical challenges. We have effectively developed a system that delivers accurate and coherent summaries, significantly enhancing information accessibility. This project has deepened our understanding of model training, data processing, and user interface design, highlighting our capability to tackle complex NLP tasks.

NLP Experiment 3

Aim: To implement NLP Pre-processing Tasks

Theory:

Preprocessing steps:
Removing inconsistent data, in the process of web scraping a few longer synopses had hyperlinks to expand their content, the dataset has text content of those hyperlinks as well. We had to remove the inconsistent part of the data.

Example:

Removing rows that were outside of the default English character set, there were multiple instances where a foreign script was scraped but since the system did not support the same, there was a loss of data and random characters added noise in those rows.
This was done using a regular expression: re.compile(r'^[a-zA-Z\s]*$')

The above regex matches with all English characters, those unmatched with the regex are replaced with a white space (‘ ’) and all blank/white space data (that is all cells of the data frame that are empty) are removed from the data frame.

Example:
We remove all the punctuations in each synopsis using a regex:
re.sub(r'[^\w\s]', '', sentence)
The above regex substitutes each character that is not a word character /w or a /s a white space character which effectively removes punctuations from a sentence.
We convert all of the text data into lowercase.
We split all sentences into tokens and each unique token is assigned a unique number representing the token.
We represent each sentence into a sequence of those numbers in this method.

Libraries and Tools Used:

- Pandas (used for manipulating data)
- re (for matching and removing noisy data from the dataset)

Preprocessing Data

```
import pandas as pd
import re
```

```
df = pd.read_csv('/content/train.csv')
```

```
df.head(5)
```

	id	movie_name	synopsis	genre
0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy
1	50185	Entity Project	A director and her friends renting a haunted h...	horror
2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family
3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi
4	2206	Apat na anino	Buy Day - Four Men Widely - Apart in Life - By...	action

Next steps: Generate code with df View recommended plots New interactive sheet

Removing inconsistence in synopsis

```
string_to_remove = "... See full synopsis¬†¬M"
df['synopsis'] = df['synopsis'].str.replace(string_to_remove, '').str.strip()
```

Removing noisy data

```
english_alphabet_pattern = re.compile(r'^[a-zA-Z\s]*$')
df['movie_name'] = df['movie_name'].apply( \
    lambda x: x if re.match(english_alphabet_pattern, x) else '')
df = df[df['movie_name'] != '']
```

```
df.head()
```

	id	movie_name	synopsis	genre
0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy
1	50185	Entity Project	A director and her friends renting a haunted h...	horror
2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family
3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi
4	2206	Apat na anino	Buy Day - Four Men Widely - Apart in Life - By...	action

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.to_csv('/content/clean_train.csv')
```

Tokenizing

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
```

```
tokenizer.fit_on_texts(df['synopsis'])
```

```
word_index = tokenizer.word_index
```

```
print(word_index)
```

```
{'<OOV>': 1, 'a': 2, 'the': 3, 'to': 4, 'of': 5, 'and': 6, 'in': 7, 'his': 8, 'is': 9, 'an': 10, 'her': 11, 'with': 12, 'on': 13, 't
```

```
sequences = tokenizer.texts_to_sequences(df['synopsis'])
```

```
sequences
```

```
   122,
   292,
   10,
   672,
   12,
   2,
   174,
   1,
   1540,
   8,
   87,
   4,
   728,
   13,
   2,
   5084,
   243,
   3,
   75,
   588,
   231],
  [7,
   10,
   1393,
   1168,
   1,
   662,
   1466,
   541,
   3362,
   4943,
   1,
   332,
   23,
   3,
   1473,
   4329,
   1,
   445,
   5,
   3,
   4476,
   1,
   1549,
   5,
   1,
   1798,
   3,
   1124,
   4,
   2279,
   60,
   5,
   1473,
   3,
   97,
   160],
  ...]
```

NLP Experiment 4

Aim: To implement Advanced Text Pre-processing Techniques.

Theory:

We first remove stopwords. Stopwords are frequently occurring words that carry little to no additional information for analysing the meaning of the sentence.

Steps for removal of stopwords:

- We download a list of stopwords using nltk.download("stopwords")
- NLTK provides a list of stop words in multiple languages (we download that in the step above). We use NLTK to remove these stopwords from text data, allowing one to focus on the more meaningful words and phrases when performing text analysis, such as text classification or sentiment analysis.
- Split every sentence into words (splitting by space).
- Form a new sentence, if a word is present in the list of stopwords formed earlier we do not add that word back in the sentence.
- We form a new sentence by eliminating all stopwords using this.
-

Lemmatization is the method of reducing a word into its dictionary form (these reduced words are known as lemma) that is normalizing words to their root words. This practice makes it easier to analyse sentences.

- We perform lemmatization by using "wordnet" in nltk
- WordNet is a lexical database and resource for natural language processing and linguistic research. It's an extensive lexical database of English, developed at Princeton University. WordNet is organized around the concept of a "lexicon," which is essentially a comprehensive dictionary of English words and their relationships.
- In NLTK (Natural Language Toolkit), WordNetLemmatizer is a class that provides lemmatization functionality based on WordNet. We use this class to lemmatize words in a sentence. (As used in the function lemmatize_words(text:str))

Libraries and Tools Used:

- Pandas
- nltk

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk
import pandas as pd
import os
```

```
os.listdir('/content/Dataset')
```

```
['train.csv', 'clean_train.csv', 'test.csv']
```

```
df = pd.read_csv('./Dataset/clean_train.csv')
df.head(5)
```

	Unnamed: 0	id	movie_name	synopsis	genre
0	0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy
1	1	50185	Entity Project	A director and her friends renting a haunted h...	horror
2	2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family
3	3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi
4	4	2206	Apat na anino	Buy Day - Four Men Widely - Apart in Life - By...	action

Next steps: Generate code with df View recommended plots New interactive sheet

```
nltk.download('stopwords')
```

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

```
stop_words = set(nltk.corpus.stopwords.words('english'))
stop_words
```

```
        's',
        'same',
        'shan',
        "shan't",
        'she',
        "she's",
        'should',
        "should've",
        'shouldn',
        "shouldn't",
        'so'
```

```
def remove_stopwords(text:str):
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)
```

```
df['filtered_synopsis'] = df['synopsis'].apply(remove_stopwords)
df['filtered_synopsis'][:5]
```

	filtered_synopsis
0	young scriptwriter starts bringing valuable ob...
1	director friends renting haunted house capture...
2	educational video families family therapists d...
3	Scientists working Austrian Alps discover glac...
4	Buy Day - Four Men Widely - Apart Life - Night...

dtype: object

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
lemmatizer = WordNetLemmatizer()
```

```
def lemmatize_words(text):
    words = text.split()
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(lemmatized_words)
```

```
df['lemmatized_synopsis'] = df['filtered_synopsis'].apply(lemmatize_words)
df['lemmatized_synopsis'][:5]
```

	lemmatized_synopsis
0	young scriptwriter start bringing valuable obj...
1	director friend renting haunted house capture ...
2	educational video family family therapist desc...
3	Scientists working Austrian Alps discover glac...
4	Buy Day - Four Men Widely - Apart Life - Night...

```
df.to_csv('./Dataset/lemmatized_data.csv')
```

NLP Experiment 5

Aim: To implement shallow and deep Parsing

Theory:

Shallow Parsing (also known as chunking) focuses on grouping words or phrases based on their syntactic structures.

- We first tokenize each word.
- We perform pos tagging on the text.
- We perform chunking using RegExParser.
  "GP: {<JJ.*|VBG><NN.*>+}" is our RegexPattern, where:
  GP stands for genre phrase which gives data points that help get information regarding the genre of the movie

- JJ stands for adjective, JJ.* could parse superlative or comparative adjective used in the synopsis
- and NN.* stands for the nouns used in the movie synopsis
- VBG stands for gerund where any verbs ending with "ing" would be parsed
  It will either parse and adjective-noun combination or a gerund noun combination to gain information on the genre of the movie by its synopsis

- We can make a chunker object entering the above RegEx pattern.
- The RegEx pattern enables the chunker to parse objects in the pos-tagged entities.
- The parsed entities are then used to visualize a parsing tree.
- The libraries used in this:
  The NLTK (Natural Language Toolkit) downloads you've listed include various resources and models used for natural language processing (NLP) tasks, including shallow parsing and other related tasks. Let's briefly describe each of them and their relevance to shallow parsing:

  Maxent_treebank_pos_tagger:
  - This is a part-of-speech tagger model trained on the Treebank corpus. It assigns part-of-speech tags to words in a text, which is a fundamental step in shallow parsing. Part-of-speech tagging helps identify the grammatical roles of words in a sentence, which is essential for chunking and other syntactic analysis.

  Treebank:
  - The Treebank corpus is a large collection of parsed and annotated English sentences. It serves as a valuable resource for training and evaluating syntactic parsers and taggers. Shallow parsers and chunkers can benefit from using this corpus for training and testing.

Punkt:

- The Punkt tokenizer is a pre-trained sentence tokenizer that can segment text into sentences. While not directly related to shallow parsing, sentence segmentation is often a preliminary step before any parsing or tagging operation.

Words:

- The 'words' resource contains a list of words in English. It can be useful for various linguistic operations, including vocabulary analysis and text processing. Shallow parsing may involve working with words, so having access to a comprehensive list can be beneficial.

Maxent_ne_chunker:

- Named Entity Recognition (NER) is a task often associated with shallow parsing. While the 'maxent_ne_chunker' resource is primarily for NER, it shares some components with POS tagging and syntactic analysis, which are relevant to shallow parsing.

Averaged_perceptron_tagger:

- Similar to 'maxent_treebank_pos_tagger', this is another part-of-speech tagger model. It's trained using the averaged perceptron algorithm, and it can be used for assigning part-of-speech tags to words. Accurate POS tagging is crucial for shallow parsing tasks.

Deep parsing aims to provide a more comprehensive and detailed analysis of a sentence's grammatical structure.

- We import spacy for deep parsing.
- We load the model named “spacy_en_core_sm” to deep-parse sentences.
- “spacy_en_core_sm” is a model, where the sm indicates small, where the smaller lightweight models are downloaded.
- The spacy_en_core_sm model is designed for various NLP tasks, including tokenization, part-of-speech tagging, named entity recognition, and dependency parsing.
- Put through each sentence through the nlp() function.
- We parse the following from the words:
  - word: The original word.
  - lemma: The root of the original word.
  - pos: The part of speech tag of the word.
  - dep: Dependency, refers to the syntactic dependency relationship between the token and its parent in the parse tree or dependency tree.
  - head: represents the head of the token to which the current token is syntactically related in the parse tree.

Libraries and tools used:

1. nltk (for shallow parsing)
2. spacy (for deep parsing)
3. Pandas (for loading CSV files)

```
import pandas as pd
import nltk
from nltk import RegexpParser
from nltk.parse.stanford import StanfordParser
import spacy
```

```
nltk.download('maxent_treebank_pos_tagger')
nltk.download('treebank')
nltk.download('punkt')
nltk.download('words')
nltk.download('maxent_ne_chunker')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package maxent_treebank_pos_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/maxent_treebank_pos_tagger.zip.
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
df = pd.read_csv('/content/lemmatized_data.csv')
df.head(5)
```

	Unnamed: 0.1	Unnamed: 0	id	movie_name	synopsis	genre	filtered_synopsis	lemmatized_synopsis
0	0	0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy	young scriptwriter starts bringing valuable ob...	young scriptwriter start bringing valuable obj...
1	1	1	50185	Entity Project	A director and her friends renting a haunted h...	horror	director friends renting haunted house capture...	director friend renting haunted house capture ...
2	2	2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family	educational video families family therapists d...	educational video family family therapist desc...
3	3	3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi	Scientists working Austrian Alps discover glac...	Scientists working Austrian Alps discover glac...
4	4	4	2206	Apat na anino	Buy Day - Four Men Widely - Apart in	action	Buy Day - Four Men Widely	Buy Day - Four Men Widely -

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['tokenized'] = df['synopsis'].apply(nltk.word_tokenize)
df['tokenized']
```

	tokenized
0	[A, young, scriptwriter, starts, bringing, val...
1	[A, director, and, her, friends, renting, a, h...
2	[This, is, an, educational, video, for, famili...
3	[Scientists, working, in, the, Austrian, Alps,...
4	[Buy, Day, -, Four, Men, Widely, -, Apart, in,...
...	...
42102	[A, ragtag, gang, of, international, talking-d...
42103	[A, seductive, woman, gets, involved, in, rela...
42104	[Duyen, ,, a, wedding, dress, staff, ,, who, d...
42105	[The, people, of, a, crowded, colony, in, Coim...
42106	[Margo, is, a, little, mouse, that, lives, qui...

42107 rows × 1 columns

```
df['entities'] = df['tokenized'].apply(nltk.pos_tag)
df['entities']
```

	entities
0	[(A, DT), (young, JJ), (scriptwriter, NN), (st...
1	[(A, DT), (director, NN), (and, CC), (her, PRP...
2	[(This, DT), (is, VBZ), (an, DT), (educational...
3	[(Scientists, NNS), (working, VBG), (in, IN), ...
4	[(Buy, NNP), (Day, NNP), (-, :), (Four, CD), (...
...	...
42102	[(A, DT), (ragtag, NN), (gang, NN), (of, IN), ...
42103	[(A, DT), (seductive, JJ), (woman, NN), (gets,...
42104	[(Duyen, NNP), (,, ,), (a, DT), (wedding, NN),...
42105	[(The, DT), (people, NNS), (of, IN), (a, DT), ...
42106	[(Margo, NNP), (is, VBZ), (a, DT), (little, JJ...

42107 rows × 1 columns

```
grammar_pattern = """
    GP: {<JJ.*|VBG><NN.*>+}
 """

"""
 GP stands for genre phrase which gives datapoints that help get information
 regarding the genre of the movie- JJ stands for adjective, JJ.* could parse superlative or comparitive
 adjective used in the synopsis- and NN.* stands for the nouns used in the movie synopsis- VBG stands for gerund where any verbs ending
 It will either parse and adjective - noun combination or a gerund noun
 combination to gain inofrmation on the genre of the movie by its synopsis
 """
```

```
'\n GP stands for genre phrase which gives datapoints that help get information\n regarding the genre of the movie- JJ stands for a
djective, JJ.* could parse superlative or comparitive\n adjective used in the synopsis- and NN.* stands for the nouns used in the m
ovie synopsis- VBG stands for gerund where any verbs ending with "ing" would be parsed\n It will either parse and adjective - noun
```

```
chunker = RegexpParser(grammar_pattern)
```

```
df['chunks'] = df['entities'].apply(chunker.parse)
df['chunks']
```

	chunks
0	[(A, DT), [(young, JJ), (scriptwriter, NN)], (...
1	[(A, DT), (director, NN), (and, CC), (her, PRP...
2	[(This, DT), (is, VBZ), (an, DT), [(educationa...
3	[(Scientists, NNS), (working, VBG), (in, IN), ...
4	[(Buy, NNP), (Day, NNP), (-, :), (Four, CD), (...
...	...
42102	[(A, DT), (ragtag, NN), (gang, NN), (of, IN), ...
42103	[(A, DT), [(seductive, JJ), (woman, NN)], (get...
42104	[(Duyen, NNP), (,, ,), (a, DT), (wedding, NN),...
42105	[(The, DT), (people, NNS), (of, IN), (a, DT), ...
42106	[(Margo, NNP), (is, VBZ), (a, DT), [(little, J...

42107 rows × 1 columns

```
nltk.Tree.fromstring(str(df['chunks'][42105])).pretty_print()
```

```
                                                                                                                                                  S
    ______________________________________________________________________________________________________________________________________________|____
   |        |         |     |     |          |            |       |         |        |      |     |      |          |            |             |
The/DT people/NNS of/IN a/DT in/IN Coimbatore/NNP city/NN go/VBP through/IN a/DT as/IN a/DT few/JJ heavily/RB armed/VBN criminals/NN
```

Deep Parsing

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

```
deep_parse_results = []
for sentence in df['synopsis']:
    doc = nlp(sentence)
```

```
dependencies = []

for token in doc:
    dependencies.append({
        "word":token.text,
        "lemma":token.lemma_,
        "pos":token.pos_,
        "dep":token.dep_,
        "head":token.head.text
    })
    deep_parse_results.append(dependencies)
```

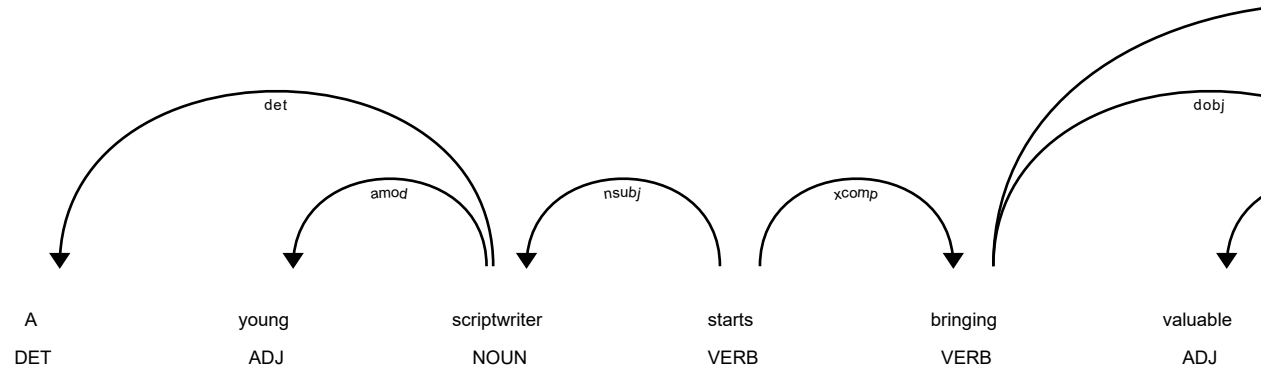
```
deep_parse_results[0]
```

```
{'word': 'who', 'lemma': 'who', 'pos': 'PRON', 'dep': 'nsubj', 'head': 'are'},
{'word': 'are',
 'lemma': 'be',
 'pos': 'AUX',
 'dep': 'relcl',
 'head': 'people'},
{'word': 'not', 'lemma': 'not', 'pos': 'PART', 'dep': 'neg', 'head': 'are'},
{'word': 'willing',
 'lemma': 'willing',
 'pos': 'ADJ',
 'dep': 'acomp',
 'head': 'are'},
{'word': 'to', 'lemma': 'to', 'pos': 'PART', 'dep': 'aux', 'head': 'make'},
{'word': 'make',
 'lemma': 'make',
 'pos': 'VERB',
 'dep': 'xcomp',
 'head': 'willing'},
{'word': 'life',
 'lemma': 'life',
 'pos': 'NOUN',
 'dep': 'nsubj',
 'head': 'easy'},
{'word': 'easy',
 'lemma': 'easy',
 'pos': 'ADJ',
 'dep': 'ccomp',
 'head': 'make'},
{'word': 'for', 'lemma': 'for', 'pos': 'ADP', 'dep': 'prep', 'head': 'make'},
{'word': 'Margo',
 'lemma': 'Margo',
 'pos': 'PROPN',
 'dep': 'pobj',
 'head': 'for'},
{'word': '.', 'lemma': '.', 'pos': 'PUNCT', 'dep': 'punct', 'head': 'are'}]
```

```
from spacy import displacy
text = nlp(df['synopsis'][0])
```

```
displacy.serve(text, style="dep")
```

```
/usr/local/lib/python3.10/dist-packages/spacy/displacy/__init__.py:106: UserWarning: [W011] It looks like you're calling displacy.se
  warnings.warn(Warnings.W011)
```



NLP Experiment 6

Aim: To implement Text Processing Models

Theory:

In this assignment, we delve into the practical implementation of two foundational text processing techniques: the N-gram model (including 2-gram and 3-gram) and the Term Frequency- Inverse Document Frequency (TF-IDF) model. These methodologies are integral to natural
language processing (NLP), providing crucial insights into text patterns and enabling tasks such as text prediction and document similarity analysis.

The N-gram model involves calculating probabilities of word sequences, where 2-gram and 3-gram models capture the likelihood of a word given its preceding words. Tokenization and N-gram generation are facilitated using the Natural Language Toolkit (NLTK). The 2-gram model utilizes a defaultdict structure to efficiently capture relationships between prefixes and suffixes.

Enhancements include factoring in word frequencies for more nuanced predictions.

On the other hand, the TF-IDF model focuses on term frequency and inverse document frequency. Scikit-learn is employed for TF-IDF vectorization, providing a robust toolkit for numerical operations. The TF-IDF vectorizer is configured with English stop words for effective preprocessing, ensuring a cleaner and more representative analysis of textual data. Cosine similarity calculation is integrated for a comprehensive measure of document similarity analysis.

Example for N-Gram Model:
Consider the input text "A young scriptwriter." For 2-grams, predictions for 'scriptwriter' include terms like screenwriter, working, and novelist. For 3-grams, predictions include phrases like 'who had just,' 'who is,' and 'who dreams.'

Example for TF-IDF Model:
For the TF-IDF model, take the input text "Three best friends spy on their families, sneak into each other's house, and organize elaborate pranks." This yields top 5 similar documents with corresponding cosine similarity values.

Formula

Bigram: $$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

N-gram: $$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

Libraries and Tools Used:

- NLTK (Natural Language Toolkit).
- Collections.Counter
- Math
- Pandas

```
import string
import random
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('reuters')
from nltk.corpus import reuters
from nltk import FreqDist
import pandas as pd
import re
from nltk import ngrams , defaultdict, Counter
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from nltk.lm.preprocessing import padded_everygram_pipeline
```

```
[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 reuters to /root/nltk_data...
```

NGRARM

```
data = pd.read_csv('/content/Dataset/clean_train.csv')
data.head(5)
sents = data['synopsis']
```

```
def preprocess_text(text):
    # Implement text cleaning and tokenization here (if needed)
    tokens = word_tokenize(text)
    return tokens
```

```
data_list = list(data['synopsis'].apply(word_tokenize))
```

```
n = 2
train_data, padded_sents = padded_everygram_pipeline(n, data_list)
```

```
from nltk.lm import MLE
model = MLE(n)
```

```
len(model.vocab)
```

```
0
```

```
model.fit(train_data, padded_sents)
print(model.vocab)
```

```
<Vocabulary with cutoff=1 unk_label='<UNK>' and 50706 items>
```

```
len(model.vocab)
```

```
50706
```

```
print(model.vocab.lookup(data_list[0]))
```

```
('A', 'young', 'scriptwriter', 'starts', 'bringing', 'valuable', 'objects', 'back', 'from', 'his', 'short', 'nightmares', 'of', 'bei
```

```
print(model.vocab.lookup('language is never random lah .'.split()))
```

```
('language', 'is', 'never', 'random', '<UNK>', '.')
```

```
model.counts['nightmares']
```

```
115
```

```
model.counts[['nightmares']]['are']
```

```
5
```

```
model.score('are', 'nightmares'.split())
```

```
0.043478260869565216
```

TF-IDF

```
df_train = pd.read_csv('/content/Dataset/clean_train.csv', index_col=0)
# df_test  = pd.read_csv('./dataset/test.csv', index_col=0)
```

```
#using only 5 sentences for training
df_train = df_train.head(5)
```

```
data_list_train = list(df_train['synopsis'].apply(word_tokenize))
# data_list_test = list(df_test['synopsis'].apply(word_tokenize))
```

```
data_list_train
```

```
        'the',
        'Fire',
        'of',
        'their',
        'Fury',
        'Against',
        'the',
        'Hated',
        'Oppressors',
        '.']]
```

```
for i in data_list_train:
    print(i)
```

```
['A', 'young', 'scriptwriter', 'starts', 'bringing', 'valuable', 'objects', 'back', 'from', 'his', 'short', 'nightmares', 'of', 'bei
['A', 'director', 'and', 'her', 'friends', 'renting', 'a', 'haunted', 'house', 'to', 'capture', 'paranormal', 'events', 'in', 'order
['This', 'is', 'an', 'educational', 'video', 'for', 'families', 'and', 'family', 'therapists', 'that', 'describes', 'the', 'Behavior
['Scientists', 'working', 'in', 'the', 'Austrian', 'Alps', 'discover', 'that', 'a', 'glacier', 'is', 'leaking', 'a', 'liquid', 'that
['Buy', 'Day', '-', 'Four', 'Men', 'Widely', '-', 'Apart', 'in', 'Life', '-', 'By', 'Night', 'Shadows', 'United', 'in', 'One', 'Figh
```

```
from nltk.stem.porter import PorterStemmer


def tokenize(text):
    tokens = nltk.word_tokenize(text)
    stems = []
    for item in tokens:
        stems.append(PorterStemmer().stem(item))
    return stems


# create object
tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words='english')

# get tf-df values
result = tfidf.fit_transform(df_train['synopsis'])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:521: UserWarning: The parameter 'token_pattern' will not
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:406: UserWarning: Your stop_words may be inconsistent wit
  warnings.warn(
```

```
feature_names = tfidf.get_feature_names_out()

# Create a DataFrame to display the TF-IDF values for the first movie synopsis
tfidf_df = pd.DataFrame(result[0].T.todense(), index=feature_names, columns=['TF-IDF'])
tfidf_df = tfidf_df.sort_values(by=['TF-IDF'], ascending=False)
```

```
tfidf_df.head(30)
```

	TF-IDF
young	0.258992
bring	0.258992
scriptwrit	0.258992
sell	0.258992
make	0.258992
demon	0.258992
short	0.258992
start	0.258992
chase	0.258992
rich	0.258992
hi	0.258992
valuabl	0.258992
object	0.258992
nightmar	0.258992
.	0.246823
order	0.000000
rent	0.000000
popular	0.000000
oppressor	0.000000
prove	0.000000
night	0.000000
psychiatr	0.000000
paranorm	0.000000
seriou	0.000000
scientist	0.000000
shadow	0.000000
therapi	0.000000
therapist	0.000000
thi	0.000000
unit	0.000000

Next steps: Generate code with tfidf_df View recommended plots New interactive sheet

```
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names_out(), tfidf.idf_):
    print(ele1, ':', ele2)
```

```
behavior : 2.09861228866811
bring : 2.09861228866811
buy : 2.09861228866811
captur : 2.09861228866811
chase : 2.09861228866811
day : 2.09861228866811
deal : 2.09861228866811
demon : 2.09861228866811
describ : 2.09861228866811
director : 2.09861228866811
discov : 2.09861228866811
educ : 2.09861228866811
event : 2.09861228866811
famili : 2.09861228866811
```

```
liquid : 2.09861228866811
local : 2.09861228866811
make : 2.09861228866811
men : 2.09861228866811
night : 2.09861228866811
nightmar : 2.09861228866811
object : 2.09861228866811
oppressor : 2.09861228866811
order : 2.09861228866811
paranorm : 2.09861228866811
popular : 2.09861228866811
prove : 2.09861228866811
psychiatr : 2.09861228866811
rent : 2.09861228866811
rich : 2.09861228866811
scientist : 2.09861228866811
scriptwrit : 2.09861228866811
sell : 2.09861228866811
seriou : 2.09861228866811
shadow : 2.09861228866811
short : 2.09861228866811
start : 2.09861228866811
therapi : 2.09861228866811
therapist : 2.09861228866811
thi : 2.09861228866811
unit : 2.09861228866811
valuabl : 2.09861228866811
vent : 2.09861228866811
video : 2.09861228866811
wide : 2.09861228866811
wildlif : 2.09861228866811
work : 2.09861228866811
young : 2.09861228866811
```

```
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf value:')
# print(result)
```

```
Word indexes:
{'young': 66, 'scriptwrit': 50, 'start': 55, 'bring': 10, 'valuabl': 60, 'object': 40, 'hi': 29, 'short': 54, 'nightmar': 39, 'chase

tf-idf value:
```

```
# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

```
  0.         0.25899237 0.         0.         0.25899237 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.25899237
  0.         0.         0.         0.         0.         0.
  0.25899237 0.         0.         0.25899237 0.25899237 0.
  0.         0.         0.         0.         0.         0.
  0.25899237 0.         0.25899237 0.25899237 0.         0.
  0.25899237 0.25899237 0.         0.         0.         0.
  0.25899237 0.         0.         0.         0.         0.
  0.25899237]
 [0.         0.13627206 0.         0.         0.         0.
  0.         0.         0.28598221 0.         0.         0.
  0.28598221 0.         0.         0.         0.         0.
  0.28598221 0.         0.         0.28598221 0.         0.
  0.28598221 0.         0.         0.         0.28598221 0.
  0.28598221 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.28598221 0.28598221 0.28598221 0.28598221 0.         0.28598221
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.10342437 0.         0.         0.         0.
  0.21704766 0.         0.         0.21704766 0.         0.
  0.         0.         0.         0.21704766 0.         0.21704766
  0.         0.         0.21704766 0.         0.65114297 0.
  0.         0.         0.         0.         0.         0.
  0.         0.21704766 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.21704766 0.
```

```
[0.         0.13627206 0.28598221 0.28598221 0.         0.28598221
 0.         0.28598221 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.28598221 0.         0.         0.         0.
 0.         0.         0.28598221 0.         0.         0.
 0.         0.         0.28598221 0.         0.28598221 0.28598221
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.28598221 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.28598221 0.28598221
 0.        ]
[0.62247822 0.0988714  0.         0.         0.20749274 0.
```

NLP Experiment 7

Aim: To study and implement Morphological Analysis of English sentences.

Theory:

Morphological analysis in Natural Language Processing (NLP) involves the study and decomposition of words into their morphemes, which are the smallest units of meaning within a language. Morphemes can be prefixes, suffixes, roots, and other linguistic units that contribute to the meaning and structure of a word.

To perform morphological analysis we do the following steps:

- Tokenize the sentences (for which we download the “punkt” module).
- Remove stopwords (for which we download the “stopwords” module).
- We perform Stemming to find the suffix or prefix of a word.

To perform stemming:

- We make a list of affixes (suffixes as well as prefixes)
- first 18 are prefixes, the next 15 are suffixes.
- If it starts with a prefix or ends with a suffix we extract that from the word and show the extracted suffix or prefix for every word.

Libraries and Tools Used:

- Pandas (for loading data)
- nltk (for tokenizing and removing stopwords)

```
import spacy
import pandas as pd

# import en_core_web_sm
import spacy.cli
spacy.cli.download("en_core_web_sm")

# nlp = en_core_web_sm.load()
```

```
✓ 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.
```

```
# Load the English language model
# nlp = en_core_web_sm.load()
nlp = spacy.load('en_core_web_sm')


data = pd.read_csv('clean_train.csv')
data = data.head(10)


def perform_morphological_analysis(text):
    doc = nlp(text)
    analyzed_tokens = []

    for token in doc:
        analyzed_tokens.append({
            'Token': token.text,
            'Lemma': token.lemma_,
            'POS': token.pos_
        })

    return analyzed_tokens

# Iterate through the dataset and perform morphological analysis for each synopsis
for index, row in data.iterrows():
    synopsis = row['synopsis']
    analyzed_tokens = perform_morphological_analysis(synopsis)

    # Print the results for the first few tokens in the synopsis
    print(f"Synopsis {index + 1}:")
    for token_info in analyzed_tokens[:5]:  # Display the first 5 tokens
        print(f"Token: {token_info['Token']}, Lemma: {token_info['Lemma']}, POS: {token_info['POS']}")
    print("\n")
```

```
Synopsis 1:
Token: A, Lemma: a, POS: DET
Token: young, Lemma: young, POS: ADJ
Token: scriptwriter, Lemma: scriptwriter, POS: NOUN
Token: starts, Lemma: start, POS: VERB
Token: bringing, Lemma: bring, POS: VERB


Synopsis 2:
Token: A, Lemma: a, POS: DET
Token: director, Lemma: director, POS: NOUN
Token: and, Lemma: and, POS: CCONJ
Token: her, Lemma: her, POS: PRON
Token: friends, Lemma: friend, POS: NOUN


Synopsis 3:
Token: This, Lemma: this, POS: PRON
Token: is, Lemma: be, POS: AUX
Token: an, Lemma: an, POS: DET
Token: educational, Lemma: educational, POS: ADJ
Token: video, Lemma: video, POS: NOUN


Synopsis 4:
Token: Scientists, Lemma: scientist, POS: NOUN
Token: working, Lemma: work, POS: VERB
Token: in, Lemma: in, POS: ADP
Token: the, Lemma: the, POS: DET
Token: Austrian, Lemma: Austrian, POS: PROPN
```

```
Synopsis 5:
Token: Buy, Lemma: buy, POS: VERB
Token: Day, Lemma: Day, POS: PROPN
Token: -, Lemma: -, POS: PUNCT
Token: Four, Lemma: four, POS: NUM
Token: Men, Lemma: Men, POS: PROPN


Synopsis 6:
Token: A, Lemma: a, POS: DET
Token: video, Lemma: video, POS: NOUN
Token: voyeur, Lemma: voyeur, POS: NOUN
Token: stalks, Lemma: stalk, POS: VERB
Token: women, Lemma: woman, POS: NOUN


Synopsis 7:
Token: Twin, Lemma: Twin, POS: PROPN
Token: brothers, Lemma: brother, POS: NOUN
Token: separated, Lemma: separate, POS: VERB
Token: at, Lemma: at, POS: ADP
Token: birth, Lemma: birth, POS: NOUN


Synopsis 8:
Token: A, Lemma: a, POS: DET
```

NLP Experiment 8

Aim: To implement Part of Speech- tagging using HMM.

Theory:

Part of Speech (POS) tagging is a fundamental task in natural language processing (NLP) that involves assigning parts of speech to each word in a sentence, such as noun, verb, adjective, etc. It's a crucial step in linguistic analysis for various NLP tasks, including text summarization, sentiment analysis, and machine translation.

One common approach to POS tagging is through **Hidden Markov Models (HMMs)**, a probabilistic model that is well-suited for sequential data like text. HMM is often employed because of its efficiency in modelling time series and sequences where an underlying sequence (hidden states) governs the observed data.

Components of HMM:

1. **States (POS Tags)**: The hidden states in the HMM correspond to the POS tags we want to assign. Common tags include Nouns (NN), Verbs (VB), Adjectives (JJ), etc.
2. **Observations (Words)**: The words in the sentence are the observations. These are known data points in the sequence, but their corresponding POS tags are unknown (hidden states).
3. **Transition Probabilities**: These are the probabilities of moving from one hidden state (POS tag) to another. For example, the probability of a noun being followed by a verb.
4. **Emission Probabilities**: These represent the probability of a word (observation) being generated from a particular state (POS tag). For instance, the probability of the word "dog" being a noun.
5. **Initial Probabilities**: These probabilities define the likelihood of starting in each state (POS tag) at the beginning of the sentence.

Steps in POS Tagging Using HMM:

1. **Data Preprocessing**: Prepare a tagged corpus of sentences (like the Penn Treebank), where each word in the sentences has a corresponding POS tag.
2. **Training**:
    - Compute the **initial probabilities** by counting how often each POS tag appears at the start of a sentence.
    - Calculate the **transition probabilities** by counting how often one POS tag follows another.
    - Calculate the **emission probabilities** by counting how often a word is associated with a specific POS tag.
3. **Decoding (Viterbi Algorithm)**: After training, the HMM can be used to predict the sequence of POS tags for a new, unseen sentence. This prediction is done using the

Viterbi algorithm, which finds the most probable sequence of hidden states (POS tags) given the observed sequence of words.

- The Viterbi algorithm is a dynamic programming algorithm that efficiently computes the most likely sequence of hidden states by combining both transition and emission probabilities at each step.

Why Use HMM for POS Tagging?

- **Efficiency**: HMM efficiently handles sequences of words and considers the likelihood of transitions between POS tags, which helps capture linguistic structures like noun-verb agreements.
- **Data Sparsity**: Despite limited training data, HMM performs well due to the probabilistic approach, allowing for generalizations from sparse data.
- **Markov Assumption**: The HMM assumes that the current state (POS tag) depends only on the previous state, simplifying the computation of sequences while still providing accurate predictions.

Limitations:

- **Independence Assumptions**: HMM assumes that the probability of a word depends only on its corresponding POS tag and that the current POS tag depends only on the previous tag, which might oversimplify language dependencies.
- **Limited Context**: Since HMM only looks at one preceding word, it may struggle with more complex dependencies that span across multiple words or phrases.

```
import pandas as pd
import os
```

```
os.listdir('datasets')
```

```
df = pd.read_csv('/content/clean_train.csv')
df.head(5)
```

	Unnamed: 0	id	movie_name	synopsis	genre
0	0	44978	Super Me	A young scriptwriter starts bringing valuable ...	fantasy
1	1	50185	Entity Project	A director and her friends renting a haunted h...	horror
2	2	34131	Behavioral Family Therapy for Serious Psychiat...	This is an educational video for families and ...	family
3	3	78522	Blood Glacier	Scientists working in the Austrian Alps discov...	scifi
4	4	2206	Apat na anino	Buy Dav - Four Men Widely - Apart in Life - By...	action

Next steps: Generate code with df View recommended plots New interactive sheet

```
len(df)//2
```

```
21053
```

```
import nltk
from nltk import word_tokenize, pos_tag
```

```
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
```

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

```
print(pos_tag(word_tokenize(df['synopsis'][0])))
```

```
[('A', 'DT'), ('young', 'JJ'), ('scriptwriter', 'NN'), ('starts', 'VBZ'), ('bringing', 'VBG'), ('valuable', 'JJ'), ('objects', 'NNS
```

```
training_data = []
for index, sentence in enumerate(df['synopsis'][:10]):
    training_data.append(pos_tag(word_tokenize(df['synopsis'][index])))

print(training_data[0])
print(training_data[1])
print(training_data[2])
```

```
[('A', 'DT'), ('young', 'JJ'), ('scriptwriter', 'NN'), ('starts', 'VBZ'), ('bringing', 'VBG'), ('valuable', 'JJ'), ('objects', 'NNS
[('A', 'DT'), ('director', 'NN'), ('and', 'CC'), ('her', 'PRP$'), ('friends', 'NNS'), ('renting', 'VBG'), ('a', 'DT'), ('haunted',
[('This', 'DT'), ('is', 'VBZ'), ('an', 'DT'), ('educational', 'JJ'), ('video', 'NN'), ('for', 'IN'), ('families', 'NNS'), ('and', '(
```

```
states = set()
state_list = []
```

```
for x in training_data:
    for y in x:
        state_list.append(y[1])

states = set(state_list)

print(states)
```

```
{'CD', 'DT', 'TO', 'VBZ', 'NNPS', ',', 'VBG', 'WRB', 'VBD', 'VBP', 'JJ', 'POS', 'PRP', 'NNS', 'WDT', 'VBN', '.', 'RB', 'NNP', 'VB',
```

```
 # Dictionary mapping POS tags to shorter descriptions (values) and comments (provided as Python comm
pos_tag_mapping = {
    'PRP$': 'Possessive',
    # CD: Cardinal Number (e.g., "one", "3")
    'CD': 'Cardinal',
```

```
    # VBN: Past Participle Verb (e.g., "gone", "written")
    'VBN': 'Past Participle',
    'TO': 'to', # TO: "to" (e.g., "to go")
    'PRP': 'Personal',
    # WDT: Wh-determiner (e.g., "which", "whose")
    'WDT': 'Wh-determiner',
    # DT: Determiner (e.g., "the", "this")
    'DT': 'Determiner',
    # VBG: Present Participle Verb (Gerund) (e.g., "running", "swimming")
    'VBG': 'Gerund',
    # RP: Particle (e.g., "up", "out")
     'RP': 'Particle',
    # VB: Base Form Verb (e.g., "run", "eat")
    'VB': 'Base Verb',
    # JJ: Adjective (e.g., "happy", "red")
    'JJ': 'Adjective',
    # CC: Coordinating Conjunction (e.g., "and", "but")
    'CC': 'Conjunction',
    # VBZ: Third Person Singular Present Verb (e.g., "he runs")
    'VBZ': '3rd Person Singular Verb',
    # WP: Wh-pronoun (e.g., "who", "what")
    'WP': 'Wh-pronoun',
    # NN: Noun, Singular or Mass (e.g., "cat", "money")
    'NN': 'Noun',
    # RB: Adverb (e.g., "quickly", "very")
    'RB': 'Adverb',
    # IN: Preposition (e.g., "in", "on")
    'IN': 'Preposition',
    # VBP: Non-3rd Person Singular Present Verb (e.g., "I run")
    'VBP': 'Non-3rd Person Singular Verb',
    # WRB: Wh-adverb (e.g., "why", "where")
    'WRB': 'Wh-adverb',
    # VBD: Past Tense Verb (e.g., "he ran")
    'VBD': 'Past Tense Verb',
    # NNS: Noun, Plural (e.g., "cats", "dogs")
    'NNS': 'Plural Noun'
    }
```

```
for x, y in enumerate(training_data):
    for i, j in enumerate(y):
        training_data[x][i] = (j[0], pos_tag_mapping.get(j[1], 'UNKNOWN'))

print(training_data[0])
```

```
[('A', 'UNKNOWN'), ('young', 'UNKNOWN'), ('scriptwriter', 'UNKNOWN'), ('starts', 'UNKNOWN'), ('bringing', 'UNKNOWN'), ('valuable',
```

```
import collections
```

```
initial_counts = collections.defaultdict(int)
transition_counts = collections.defaultdict(lambda: collections.defaultdict(int))
emission_counts = collections.defaultdict(lambda: collections.defaultdict(int))
```

```
for sentence in training_data:
    initial_counts[sentence[0][1]] += 1
    for i in range(len(sentence) - 1):
        current_tag, next_tag = sentence[i][1], sentence[i + 1][1]
        transition_counts[current_tag][next_tag] += 1
        word = sentence[i][0]
        emission_counts[current_tag][word] += 1
```

```
total_sentences = len(training_data)

initial_probabilities = {tag: count / total_sentences for tag, count in initial_counts.items()}
transition_probabilities = {current_tag: {next_tag: count / sum(transition_counts[current_tag].values())
                                          for next_tag, count in transition_counts[current_tag].items()}
                            for current_tag in transition_counts}
emission_probabilities = {tag: {word: count / sum(emission_counts[tag].values())
                                for word, count in emission_counts[tag].items()}
                          for tag in emission_counts}
```

```
print("Initial Probabilities:")
print(initial_probabilities)
print("\nTransition Probabilities:")
print(transition_probabilities)
print("\nEmission Probabilities:")
print(emission_probabilities)
```

```
Initial Probabilities:
{'UNKNOWN': 0.3, 'Determiner': 0.5, 'Plural Noun': 0.2}

Transition Probabilities:
{'UNKNOWN': {'UNKNOWN': 0.6458333333333334, 'Gerund': 0.041666666666666664, 'Noun': 0.041666666666666664, 'Base Verb': 0.020833333333

Emission Probabilities:
{'UNKNOWN': {'A': 0.020833333333333332, 'young': 0.020833333333333332, 'scriptwriter': 0.020833333333333332, 'starts': 0.020833333333
```

```
# Viterbi decoding function
def viterbi_decode(initial_probabilities, transition_probabilities, emission_probabilities, new_data):
    best_path = [None] * len(new_data)
    best_prob = [0.0] * len(new_data)

    # Initialize for the first word
    for tag, prob in initial_probabilities.items():
        emission_prob = emission_probabilities.get(tag, {}).get(new_data[0], 1e-10)
        best_prob[0] = prob * emission_prob
        best_path[0] = tag

    # Process remaining words
    for t in range(1, len(new_data)):
        max_probs = {}
        for current_tag in emission_probabilities.keys():
            max_prob = 0.0
            max_tag = None
            for previous_tag in initial_probabilities.keys():
                transition_prob = transition_probabilities.get(previous_tag, {}).get(current_tag, 1e-10)
                prob = best_prob[t - 1] * transition_prob
                if prob > max_prob:
                    max_prob = prob
                    max_tag = previous_tag
            emission_prob = emission_probabilities.get(current_tag, {}).get(new_data[t], 1e-10)
            max_probs[current_tag] = max_prob * emission_prob
        best_tag = max(max_probs, key=max_probs.get)
        best_prob[t] = max_probs[best_tag]
        best_path[t] = best_tag

    # Backtrack to find the best path
    pos_tags = [None] * len(new_data)
    pos_tags[-1] = best_path[-1]
    for t in range(len(new_data) - 2, -1, -1):
        pos_tags[t] = best_path[t]

    return pos_tags
```

```
new_data = word_tokenize(df['synopsis'][2])
predicted_tags = viterbi_decode(initial_probabilities, transition_probabilities, emission_probabilities, new_data)

print(f'Predicted:\n{predicted_tags}')
print(f'\nOriginal:\n{[x[1] for x in training_data[2]]}')
```

```
Predicted:
['Plural Noun', '3rd Person Singular Verb', 'Determiner', 'Adjective', 'Noun', 'Preposition', 'Plural Noun', 'Conjunction', 'Noun',

Original:
['Determiner', '3rd Person Singular Verb', 'Determiner', 'Adjective', 'Noun', 'Preposition', 'Plural Noun', 'Conjunction', 'Noun',
```

NLP Experiment 9

Aim: To implement Named Entity Recognition for a given real-world application.

Theory:

Named Entity Recognition (NER) is a crucial task in Natural Language Processing (NLP) that involves identifying and categorizing named entities into predefined classes such as persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.
Implementing NER for a real-world application involves several steps, including data preparation, model selection, training, and evaluation.

Named Entity Recognition is done using Spacy's model. Entities that can be recognized with Spacy's model are:

Person: Names of individuals, including first and last names.

1. Organization: Names of companies, institutions, and organizations.
2. Location: Geographical places, such as cities, countries, states, and landmarks.
3. Date: Specific dates or date ranges, including days, months, and years.
4. Time: Time expressions, including specific times or time intervals.
5. Money: Monetary values, such as currency amounts and prices.
6. Percentage: Percentage values, such as percentages of change.
7. Cardinal Number: Numerical values, both cardinal (e.g., "one," "two") and numeric (e.g., "1," "2").
8. Ordinal Number: Ordinal numbers, indicating order or rank (e.g., "first," "second").
9. Quantity: Measurements and quantities, such as distances, weights, and volumes.
10. Language: Names of languages or language-specific terms.
11. Event: Names of specific events, conferences, or occurrences.
12. Law: Legal references, statutes, or legal terms.
13. Product: Names of products, goods, or branded items.
14. Work of Art: Titles of books, movies, songs, paintings, and other artistic works.
15. Drug: Names of pharmaceutical drugs and medications.
16. NORP (Nationalities or Religious/Political Groups): Names of nationalities, religious, or political groups.
17. Facility: Names of buildings, facilities, or physical structures.
18. Email Address: Email addresses or references to electronic mail.
19. Phone Number: Telephone numbers or references to phone communications.
20. URL: Web URLs or internet addresses.
21. GPE (Geopolitical Entity): Names of geopolitical entities, such as countries, cities, and regions.
22. Honorific (Title): Titles, honorifics, and forms of address (e.g., "Mr.," "Dr.").

23. Money Range: Ranges of monetary values.
24. Quantity Range: Ranges of quantities or measurements.
25. Time Range: Ranges of time expressions or intervals.

Libraries and Tools Used:

- Pandas
- Spacy

```
import spacy
import pandas as pd

# import en_core_web_sm
import spacy.cli
spacy.cli.download("en_core_web_lg")
```

```
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_lg')
⚠ 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.
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_lg')
⚠ 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.
```

```
nlp = spacy.load('en_core_web_lg')
```

```
data = pd.read_csv('/content/clean_train.csv')
data = data.head(10)
```

```
# Define a function for NER
def perform_ner(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities

# Iterate through the dataset and perform NER for each synopsis
for index, row in data.iterrows():
    synopsis = row['synopsis']
    ner_results = perform_ner(synopsis)

    # Print NER results for the current synopsis
    print(f"text {index + 1}: {synopsis}")
    print(f"Synopsis {index + 1} NER Results:")
    for entity, label in ner_results:
        print(f"Entity: {entity}, Label: {label}")
    print("\n")
```

```
text 1: A young scriptwriter starts bringing valuable objects back from his short nightmares of being chased by a demon. Selling the
Synopsis 1 NER Results:


text 2: A director and her friends renting a haunted house to capture paranormal events in order to prove it and become popular.
Synopsis 2 NER Results:


text 3: This is an educational video for families and family therapists that describes the Behavioral Family Therapy approach to dea
Synopsis 3 NER Results:
Entity: Behavioral Family Therapy, Label: ORG


text 4: Scientists working in the Austrian Alps discover that a glacier is leaking a liquid that appears to be affecting local wild
Synopsis 4 NER Results:
Entity: Austrian, Label: NORP
Entity: Alps, Label: LOC


text 5: Buy Day - Four Men Widely - Apart in Life - By Night Shadows United in One Fight Venting the Fire of their Fury Against the
Synopsis 5 NER Results:
Entity: One, Label: CARDINAL


text 6: A video voyeur stalks women in the city with a digital camera until he crosses paths with beautiful model who harbors a dark
Synopsis 6 NER Results:


text 7: Twin brothers separated at birth and worlds apart, oblivious to each other existence they cross each other's paths when one
Synopsis 7 NER Results:


text 8: A traffic police officer teams up with his friend and doctors in order to escape a deadly zombie apocalypse in the town and
Synopsis 8 NER Results:


text 9: A legendary tale unravels.
Synopsis 9 NER Results:
```

```
text 10: Millions in diamonds are stolen from a safe in NYC and later the burglar is killed. Shamus is paid $10,000 by the owner to
Synopsis 10 NER Results:
Entity: Millions, Label: CARDINAL
Entity: NYC, Label: GPE
Entity: Shamus, Label: PERSON
Entity: 10,000, Label: MONEY
```

NLP Experiment 10

Aim: Experimenting with an Advanced NLP Problem of Your Choice using Hugging Face Transformers, spaCy, NLTK (Natural Language Toolkit), AllenNLP, and BERTScore for Fine-Tuning Large Language Models (LLMs) for Standard NLP Problems

Theory:

Text classification is a fundamental natural language processing (NLP) task that involves assigning one or more labels or categories to a piece of text. We attempt to classify text using the BERT model.

We have used BERT (Bidirectional Encoder Representations from Transformers) for text classification.

A version of BERT Used: The specific version of BERT is "bert-base-uncased." This version is a base, uncased BERT model pre-trained on a large corpus of text.

Functions:

forward: This method defines the forward pass of the custom classification model.

- optimizer: It configures the AdamW optimizer with a specified learning rate.
- loss_fn: It defines the loss function as CrossEntropyLoss.
- train_loader: It prepares the training data in mini-batches for efficient training.
- Pooling: The code uses the [CLS] token for pooling. In BERT models, the [CLS] token's final hidden state is often used as a fixed-size representation for the entire input sequence.
- Layers: The BERT model consists of several layers, including the embedding layer (word embeddings, position embeddings, token type embeddings), multiple transformer layers (BertLayer), and a pooler layer (BertPooler).

Hyperparameters: Some hyperparameters include the learning rate (lr=1e-5), the number of training epochs (4), batch size (12), and the maximum sequence length (max_length=20).

The steps are:

- Use Bert Tokenizer on the synopses data.
- Encode Labels for the training data.
- Decide attention mask (the most important words in a sentence which need higher attention).
- Fine Tune the BERT model to develop a model that helps classify text.

Libraries and Tools Used:

- Pandas
- transformer

```
import pandas as pd
import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer


import os


os.listdir('./drive/MyDrive/datasets/')
```

```
['test.csv', 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv', 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv', 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
 'preprocessed_train.csv']
```

```
nltk.download('maxent_treebank_pos_tagger')
nltk.download('punkt') nltk.download('words')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package maxent_treebank_pos_tagger to [nltk_data]
/root/nltk_data...
[nltk_data]       Package maxent_treebank_pos_tagger is already up-to- [nltk_data]
                  date!
[nltk_data] Downloading package punkt to /root/nltk_data... [nltk_data]  Package
punkt 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 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 wordnet to /root/nltk_data...
True
```

```
df = pd.read_csv('./drive/MyDrive/datasets/lemmatized_data.csv')
```

```
df.head(5)
```

	Unnamed: 0.1	Unnamed: 0	id	movie_name	synopsis	genre	filtered_synopsis	le
							a young scriptwriter	young scriptwriter

fantasy starts bringing y valuable ob...
br
Super

director friends

friends renting a haunted horror renting haunted
h... ha house capture...

Behavioral this is an

df = df[:5000]

Tokenizing

df['tokenized'] = df['lemmatized_synopsis'].apply(nltk.word_tokenize)

```
<ipython-input-33-b3364e91ab33>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable df['tokenized'] =
    df['lemmatized_synopsis'].apply(nltk.word_tokenize)
```

df['tokenized'][:5]

```
0    [young, scriptwriter, start, bringing, valuabl...
1    [director, friend, renting, haunted, house, ca...
2    [educational, video, family, family, therapist...
3    [scientist, working, austrian, alp, discover, ...
4    [buy, day, four, men, widely, apart, life, nig...
Name: tokenized, dtype: object
```

df['pos'] = df['tokenized'].apply(nltk.pos_tag)

```
<ipython-input-35-a67f6250f7fb>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable df['pos'] =
    df['tokenized'].apply(nltk.pos_tag)
```

```
print(df['pos'][:5])
```

```
0    [(young, JJ), (scriptwriter, JJR), (start, NN)...
1    [(director, NN), (friend, NN), (renting, VBG),...
2    [(educational, JJ), (video, NN), (family, NN),...
3    [(scientist, NN), (working, VBG), (austrian, J...
4    [(buy, VB), (day, NN), (four, CD), (men, NNS),...
Name: pos, dtype: object
```

```
def wordnet_analysis(tokens):
    synsets = [wordnet.synsets(token) for token in tokens] return synsets
```

```
df['synsets'] = df['tokenized'].apply(wordnet_analysis)
```

```
<ipython-input-38-9d62062fc29d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable df['synsets'] =
  df['tokenized'].apply(wordnet_analysis)
```

```
df['synsets'][:5]
```

```
0    [[Synset('young.n.01'), Synset('young.n.02'), ...
1    [[Synset('director.n.01'), Synset('director.n....
2    [[Synset('educational.a.01'), Synset('educatio...
3    [[Synset('scientist.n.01')], [Synset('working....
4    [[Synset('bargain.n.02'), Synset('buy.v.01'), ...
Name: synsets, dtype: object
```

```
!pip install transformers Collecting
    transformers
      Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
    7.7/7.7 MB 43.8 MB/s eta 0:00:00 Requirement already satisfied: filelock in
    /usr/local/lib/python3.10/dist-packages Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
      Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
    302.0/302.0 kB 28.5 MB/s eta 0:00:00 Requirement already satisfied: numpy>=1.17 in
    /usr/local/lib/python3.10/dist-packa Requirement already satisfied: packaging>=20.0 in
    /usr/local/lib/python3.10/dist-p Requirement already satisfied: pyyaml>=5.1 in
    /usr/local/lib/python3.10/dist-packa Requirement already satisfied: regex!=2019.12.17 in
    /usr/local/lib/python3.10/dist Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
    Collecting tokenizers<0.15,>=0.14 (from transformers)
      Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
    3.8/3.8 MB 90.5 MB/s eta 0:00:00 Collecting safetensors>=0.3.1 (from transformers)
      Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3
    MB 71.6 MB/s eta 0:00:00 Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packag
    Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist- equirement already satisfied:
                    typing-extensions>=3.7.4.3 in /usr/local/lib/python
```

Collecting huggingface-hub<1.0,>=0.16.4 (from transformers) Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)

295.0/295.0 kB 26.6 MB/s eta 0:00:00 Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3. Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dis Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dis Installing collected packages: safetensors, huggingface-hub, tokenizers, transform Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1

```
import torch.nn as nn import
torch.optim as optim from tqdm
import tqdm
from sklearn.preprocessing import LabelEncoder import
pandas as pd
from transformers import BertTokenizer, BertModel import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW from torch.utils.data
import DataLoader, TensorDataset
from sklearn.preprocessing import LabelEncoder train_data =

df


label_encoder = LabelEncoder()

train_data['genre'] = train_data['genre'].apply(lambda genres: ', '.join(genres)) y_train_encoded =

label_encoder.fit_transform(train_data['genre'])

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name) embedding_model =
BertModel.from_pretrained(model_name)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
embedding_model.to(device)

max_length = 20
concatenated_text = train_data['synopsis']
encoded_inputs = tokenizer(list(concatenated_text), padding='max_length', truncation=Tr

train_dataset = TensorDataset(torch.tensor(encoded_inputs['input_ids']), torch.tensor(en train_loader =
DataLoader(train_dataset, batch_size=12, shuffle=True)

class CustomClassifier(nn.Module):
    def init (self, embedding_model, num_classes): super(CustomClassifier,
        self). init () self.embedding_model = embedding_model
        self.fc = nn.Linear(embedding_model.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        embeddings = self.embedding_model(input_ids, attention_mask=attention_mask).last logits =
        self.fc(embeddings)
        return logits
```

```
num_classes = len(label_encoder.classes_)
model = CustomClassifier(embedding_model, num_classes)
model.to(device)

optimizer = optim.AdamW(model.parameters(), lr=1e-5) loss_fn =
nn.CrossEntropyLoss()

model.train()
for epoch in range(4):
    progress_bar = tqdm(train_loader, desc=f"Epoch {epoch + 1}/5", leave=False) total_correct = 0
    total_samples = 0

    for batch in progress_bar:
        optimizer.zero_grad()
        input_ids, attention_mask, labels = [item.to(device) for item in batch] logits = model(input_ids,
        attention_mask)
        loss = loss_fn(logits, labels)
        loss.backward() optimizer.step()

        # accuracy
        _, predicted = torch.max(logits, 1)
        total_correct += (predicted == labels).sum().item() total_samples +=
        labels.size(0)
        accuracy = total_correct / total_samples progress_bar.set_postfix({"loss": loss.item(), "accuracy":

        accuracy})

    # Accuracy * epoche
    print(f'Epoch {epoch + 1} - Accuracy: {accuracy:.4f}')



# Eval_model
model.eval() total_correct
= 0
total_samples = 0 with
torch.no_grad():
    progress_bar = tqdm(train_loader, desc="Evaluating", leave=False) for batch in
    progress_bar:
        input_ids, attention_mask, labels = [item.to(device) for item in batch] logits = model(input_ids,
        attention_mask)
        _, predicted = torch.max(logits, 1)
        total_correct += (predicted == labels).sum().item() total_samples +=
        labels.size(0)
        progress_bar.set_postfix({"accuracy": total_correct / total_samples})

accuracy = total_correct / total_samples print(f'Final
Accuracy: {accuracy:.4f}')
```

```
Epoch 1 - Accuracy: 0.2342
Epoch 2 - Accuracy: 0.3982
```

```
Epoch 3 - Accuracy: 0.5048
Epoch 4 - Accuracy: 0.6072
```

Final

```
import os
```

```
os.listdir()
```

```
['test.csv', 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv', 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv', 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
 'preprocessed_train.csv',
 'exp_nine.pth', 'exp_nine.pkl',
 'exp_nine(1).pkl']
```

```
torch.save(model.state_dict(), 'exp_nine(2).pkl')
```

```
class CustomClassifier(nn.Module):
    def init (self, embedding_model, num_classes): super(CustomClassifier,
        self). init () self.embedding_model = embedding_model
        self.fc = nn.Linear(embedding_model.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        embeddings = self.embedding_model(input_ids, attention_mask=attention_mask).last logits =
        self.fc(embeddings)
        return logits
```

```
num_classes = len(label_encoder.classes_)
model = CustomClassifier(embedding_model, num_classes)
```

```
os.listdir()
```

```
['test.csv', 'clean_train_nlp.csv',
 '.ipynb_checkpoints',
 'train.csv', 'clean_train_3.csv',
 'lemmatized_data.csv',
 'mpr_test.csv', 'mpr_train.csv',
 'NLP MPR',
 'lemmatized_data.gsheet',
 'clean_train_3.gsheet',
```

```
 'preprocessed_train.csv',
 'exp_nine.pth', 'exp_nine.pkl',
 'exp_nine(1).pkl',
 'exp_nine(2).pkl']


model.load_state_dict(torch.load('exp_nine(2).pkl'))

    <All keys matched successfully>


model.eval()

    CustomClassifier( (embedding_model):
      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): BertSelfAttention(
                  (query): Linear(in_features=768, out_features=768, bias=True) (key):
                  Linear(in_features=768, out_features=768, bias=True) (value):
                  Linear(in_features=768, out_features=768, bias=True) (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()
        )
      )
      (fc): Linear(in_features=768, out_features=10, bias=True)
    )
```

```
def preprocess_input(input_text, tokenizer, max_length, label_encoder, device): input_data = tokenizer(input_text,
    padding=True, truncation=True, max_length=max_len

    input_ids = torch.tensor(input_data['input_ids'], dtype=torch.long).unsqueeze(0).to attention_mask =
    torch.tensor(input_data['attention_mask'], dtype=torch.long) unsqu

    return input_ids, attention_mask

def predict_genre(input_text, model, tokenizer, max_length, label_encoder, device): input_ids, attention_mask =
    preprocess_input(input_text, tokenizer, max_length, labe

    with torch.no_grad():
        logits = model(input_ids, attention_mask)

    _, predicted = torch.max(logits, 1)

    return "".join(label_encoder.classes_[predicted.item()].split()).replace(",","")


input_text = df['synopsis'][0]
print(predict_genre(input_text, model, tokenizer, max_length, label_encoder, device)) fantasy


"".join(df['genre'][0].split()).replace(",","")
'fantasy'
```

0	0	0	44978
1	1	1	50185

```
pip install nltk spacy gensim
```

```
import pandas as pd
import nltk
import spacy
from nltk.tokenize import word_tokenize, sent_tokenize from nltk.corpus
import wordnet
from nltk.stem import WordNetLemmatizer from
gensim.models import Word2Vec
from collections import defaultdict
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
nltk.download('punkt') nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

```
# Load dataset
df = pd.read_csv('/content/drive/MyDrive/datasets/mpr_train.csv') synopses =

df['synopsis'].tolist()

# Initialize the lemmatizer lemmatizer =
WordNetLemmatizer()

# Tokenize and lemmatize the synopsis
tokenized_synopses = []
for synopsis in synopses:
    words = word_tokenize(synopsis)
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    tokenized_synopses.append(lemmatized_words)

!python -m spacy download en # Download the English model for spaCy
```

```
2023-10-23 08:00:18.401689: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38
⚠ As of spaCy v3.0, shortcuts like 'en' are deprecated. Please use the full pipeline package
```

```
name 'en_core_web_sm' instead.
Collecting en-core-web-sm==3.6.0
    Downloading https://github.com/explosion/spacy-models/releases/download/en_core_
```

12.8/12.8 MB 31.0 MB/s eta 0:00:00 Requirement already satisfied: spacy<3.7.0,>=3.6.0 in /usr/local/lib/python3.10/di Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/pytho Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/pytho Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3 Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/di Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/ Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.10/di Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/d Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/di Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.1 Requirement already satisfied: typer<0.10.0,>=0.3.0 in /usr/local/lib/python3.10/d Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.10/dist-pac Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3. Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/di Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.10/dist-pac Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.1 Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/ Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packag Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-p Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.1 Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3. Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3. Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dis Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dis Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dis Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3. Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/di Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-p

✔ Download and installation successful

You can now load the package via spacy.load('en_core_web_sm')

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


import pandas as pd
import nltk
import spacy
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

# Step 1: Tokenization def
tokenize_text(text):
    tokens = nltk.word_tokenize(text) return
    tokens

# Step 2: Lemmatizing
def lemmatize_text(tokens): lemmatizer =
    WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens] return
    lemmatized_tokens

# Step 3: Language Modeling (using spaCy) nlp =
spacy.load("en_core_web_sm")
```

```
# Step 4: Syntactic Analysis - POS Tagging (using spaCy)
def pos_tagging(text):
    doc = nlp(text)
    pos_tags = [(token.text, token.pos_) for token in doc] return pos_tags

# Step 5: Semantic Analysis - WordNet Analysis (using NLTK) def
wordnet_analysis(tokens):
    synsets = [wordnet.synsets(token) for token in tokens] return synsets

data = pd.read_csv('/content/drive/MyDrive/datasets/lemmatized_data.csv')


data['Tokens'] = data['synopsis'].apply(lambda x: tokenize_text(x)) data['Lemmatized'] =
data['Tokens'].apply(lambda x: lemmatize_text(x)) data['POS Tags'] = data['synopsis'].apply(lambda x:
pos_tagging(x)) data['WordNet Synsets'] = data['Tokens'].apply(lambda x: wordnet_analysis(x))


data.head(5)
```

LSTM

```
from sklearn.model_selection import train_test_split from
sklearn.preprocessing import MultiLabelBinarizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences from
tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense from
tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, classification_report

# Ensure the 'genre' column is in string format data['genre'] =
data['genre'].astype(str)

# Preprocess genre labels
data['genre'] = data['genre'].str.split(',') # Split the genre labels into lists mlb = MultiLabelBinarizer()
y = mlb.fit_transform(data['genre'])

# Combine multiple features (you can add more as needed)
X = data[['Tokens', 'Lemmatized', 'POS Tags', 'WordNet Synsets']]

# Convert list elements to strings and then combine them X['Combined_Features'] = X.apply(lambda
row: ' '.join(map(str, row)), axis=1)

# Tokenize and pad sequences
max_words = 10000 # Maximum number of words to tokenize
max_sequence_length = 100 # Maximum sequence length tokenizer =
Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X['Combined_Features'])
X_seq = tokenizer.texts_to_sequences(X['Combined_Features']) X_padded =
pad_sequences(X_seq, maxlen=max_sequence_length)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.2, random_s

# Build a deep learning model model =
Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100, input_length=max_sequence_lengt
model.add(LSTM(100))
model.add(Dense(y.shape[1], activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32)

# Evaluate the model
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5) # Convert probabilities to binary predictions accuracy =
accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy:.2f}")

# View classification report for more detailed evaluation
```

, ,

<ipython-input-25-ebbf63445ff7>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame. Try using
.loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable X['Combined_Features'] =
X.apply(lambda row: ' '.join(map(str, row)), axis=1)
Epoch 1/100
1053/1053 [==============================] - 27s 24ms/step - loss: 0.3298 - accura
Epoch 2/100
1053/1053 [==============================]
Epoch 3/100
1053/1053 [==============================]
Epoch 4/100
1053/1053 [==============================]
Epoch 5/100
1053/1053 [==============================]
Epoch 6/100
1053/1053 [==============================]
Epoch 7/100
1053/1053 [==============================]
Epoch 8/100
1053/1053 [==============================]
Epoch 9/100
1053/1053 [==============================]
Epoch 10/100
1053/1053 [==============================]
Epoch 11/100
1053/1053 [==============================]
Epoch 12/100
1053/1053 [==============================]
Epoch 13/100
1053/1053 [==============================]
Epoch 14/100
1053/1053 [==============================]
Epoch 15/100
1053/1053 [==============================]
Epoch 16/100
1053/1053 [==============================]
Epoch 17/100
1053/1053 [==============================]
Epoch 18/100
1053/1053 [==============================]
Epoch 19/100
1053/1053 [==============================]
Epoch 20/100
1053/1053 [==============================]
Epoch 21/100
1053/1053 [==============================]
Epoch 22/100
1053/1053 [==============================]
Epoch 23/100
1053/1053 [==============================]
Epoch 24/100
1053/1053 [==============================]
Epoch 25/100
1053/1053 [==============================]
Epoch 26/100
053/1053 [==============================]

```
import os
```

```
os.listdir()
```

```
['.config', 'drive', 'sample_data']
```

```
model.save('drive/MyDrive/datasets/mpr_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWar saving_api.save_model(
```

MINI PROJECT: Text Summarizer

Problem Statement:

To summarize large paragraph to desired number of sentences and lines.

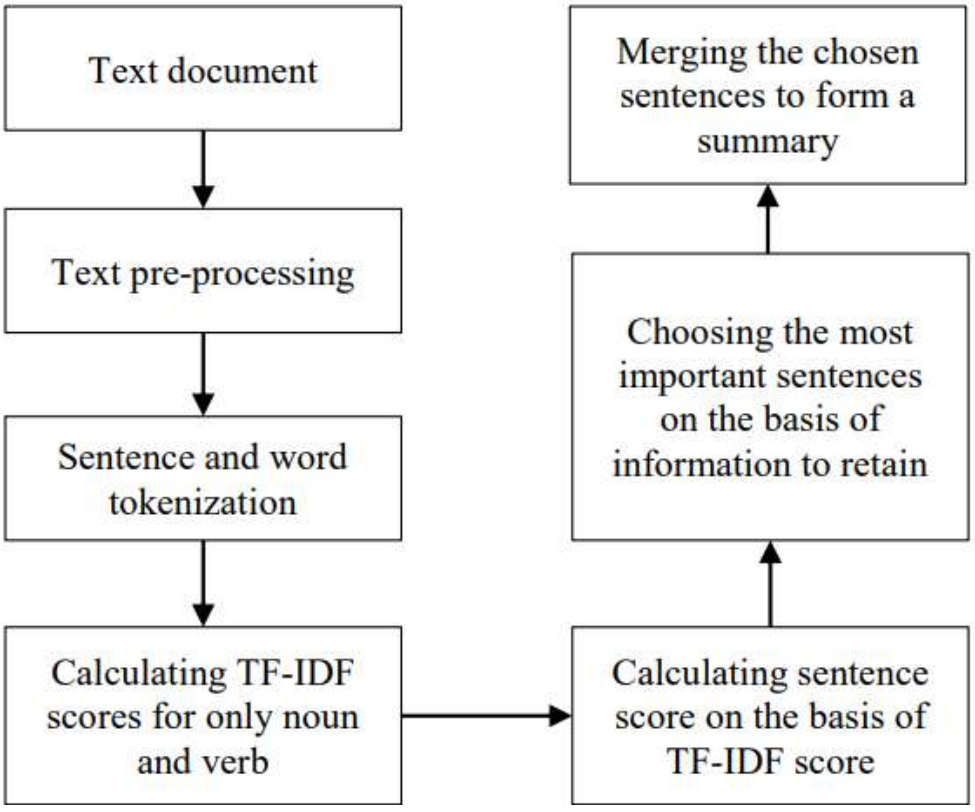
Sample Input:

Text File

Sample Output:

Summarized Text File

Flowchart:



Libraries Used:

- Pandas (for loading data)

- NLTK (Lemmatizing, tokenizing, Wordnet analysis)

Workflow:

1. Import necessary libraries:
- Import NLTK for natural language processing tasks.
- Import the stop words list from NLTK to remove common words during preprocessing.

2. Generate clean sentences:
- Remove special characters, digits, and words from the text.
- Preprocess the text to ensure consistent formatting and remove unnecessary elements.

3. Calculate TF-IDF and generate a matrix:
- Calculate Term Frequency (TF) for each word in a paragraph: the frequency of a word divided by the total number of words in the paragraph.
- Calculate Inverse Document Frequency (IDF) for each word: the logarithm of the total number of documents divided by the number of documents containing the word.
- Multiply the TF and IDF values for each word to generate a weighted matrix representing the importance of each word in the paragraph.

4. Score the sentences:
- Use the TF-IDF word points to assign weights to each sentence in the paragraph.
- Apply a sentence scoring algorithm to determine the overall importance of each sentence based on its word weights.

5. Generate the summary:
- Select the top-scoring sentences based on the calculated weights and the desired retention rate.
- Combine the selected sentences to create a concise summary of the original text.

Codes:

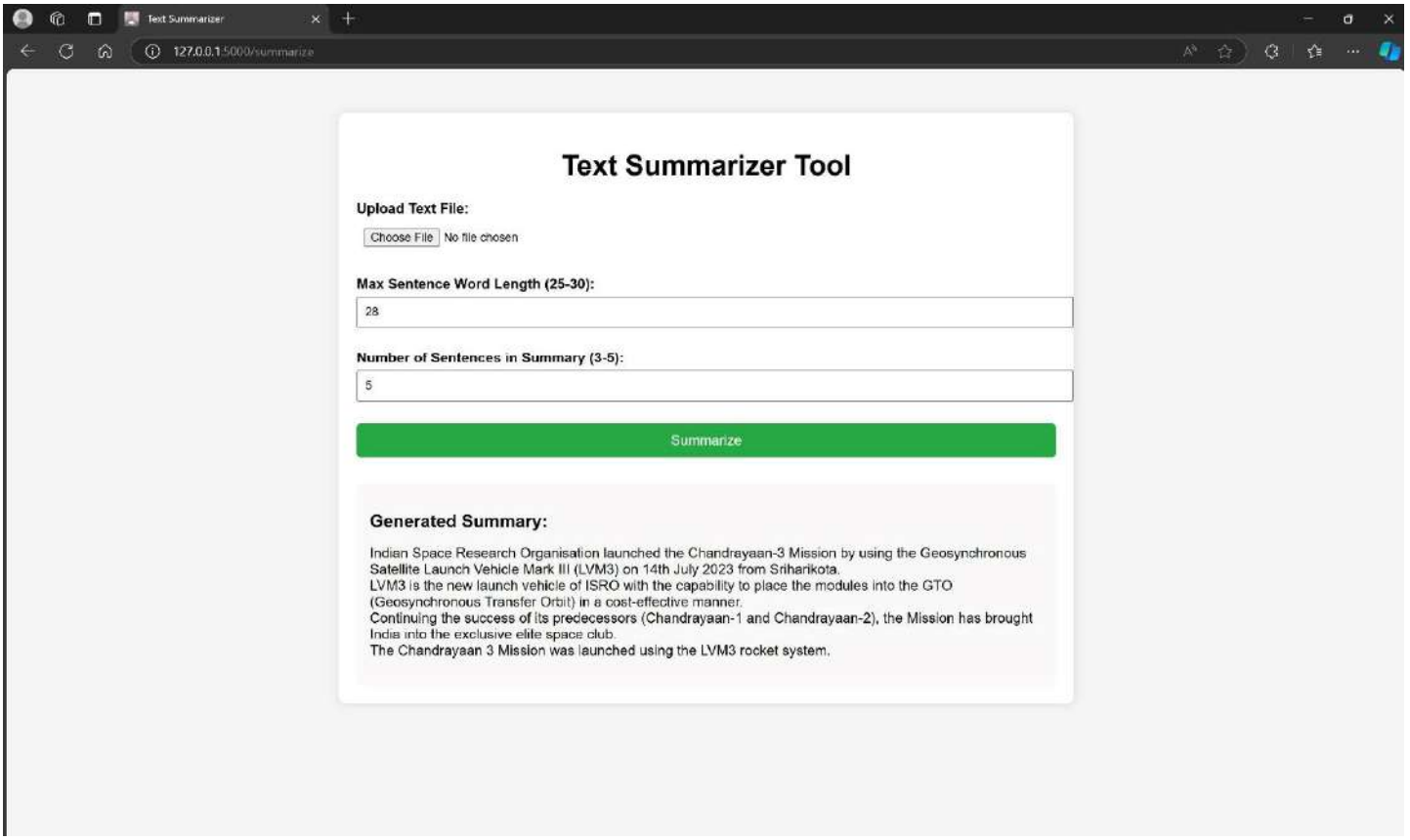
```
import nltk
nltk.download('stopwords')
nltk.download('punkt')

import re
import heapq
import numpy as np
import pandas as pd
import sys


def main():
   file_name = input('1. Enter text file name: ')
```

```
    output_location = input('2. Enter output file name (e.g. summary.txt): ')
    sent_word_length = input('3. Enter max sentence word length (choose between
25 - 30): ')
    top_n = input('4. Enter number of sentences you want in summary (choose
between 3 - 5): ')

    def read_text(file_name):
        """
        Read text from file

        INPUT:
        file_name - Text file containing original text.

        OUTPUT:
        text - str. Text with reference number, i.e. [1], [10] replaced with space, if
any...
        clean_text - str. Lowercase characters with digits & one or more spaces
replaced with single space.
        """
        with open(file_name, 'r') as f:
            file_data = f.read()

        text = file_data
        text = re.sub(r'\[[0-9]*\]',' ',text)
        text = re.sub(r'\s+',' ',text)

        clean_text = text.lower()

        # replace characters other than [a-zA-Z0-9], digits & one or more spaces with
single space
        regex_patterns = [r'\W',r'\d',r'\s+']
        for regex in regex_patterns:
            clean_text = re.sub(regex,' ',clean_text)

        return text, clean_text

    def rank_sentence(text, clean_text, sent_word_length):
        """
        Rank each sentence and return sentence score
```

```
    INPUT:
    text - str. Text with reference numbers, i.e. [1], [10] removed, if any...
    clean_text - str. Clean lowercase characters with digits and additional spaces removed.
    sent_word_length - int. Maximum number of words in a sentence.

    OUTPUT:
    sentence_score - dict. Sentence score
    """
    sentences = nltk.sent_tokenize(text)
    stop_words = nltk.corpus.stopwords.words('english')

    word_count = {}
    for word in nltk.word_tokenize(clean_text):
        if word not in stop_words:
            if word not in word_count.keys():
                word_count[word] = 1
            else:
                word_count[word] += 1

    sentence_score = {}
    for sentence in sentences:
        for word in nltk.word_tokenize(sentence.lower()):
            if word in word_count.keys():
                if len(sentence.split(' ')) < int(sent_word_length):
                    if sentence not in sentence_score.keys():
                        sentence_score[sentence] = word_count[word]
                    else:
                        sentence_score[sentence] += word_count[word]

    return sentence_score

def generate_summary(file_name, sent_word_length, top_n):
    """
    Generate summary

    INPUT:
    file_name - Text file containing original text.
    sent_word_length - int. Maximum number of words in a sentence.
    top_n - int. Top n sentences to display.
```

```
        OUTPUT:
        summarized_text - str. Summarized text with each sentence on each line.
        """
        text, clean_text = read_text(file_name)

        sentence_score = rank_sentence(text, clean_text, sent_word_length)

        best_sentences = heapq.nlargest(int(top_n), sentence_score,
key=sentence_score.get)

        summarized_text = []

        sentences = nltk.sent_tokenize(text)

        for sentence in sentences:
            if sentence in best_sentences:
                summarized_text.append(sentence)

        summarized_text = "\n".join(summarized_text)

        return summarized_text

    # generate summary
    summary = generate_summary(file_name, sent_word_length, top_n)

    # save summary to txt file
    text_file = open(output_location, "w")
    text_file.write(summary)
    text_file.close()
    print('Summarization task completed. Please check your output file.')
if __name__ == '__main__':
    main( *sys.argv[1:] )
```

Results:


Text Summarizer
127.0.0.1:5000/summarize
Text Summarizer Tool
Upload Text File:
Choose File
No file chosen
Max Sentence Word Length (25-30):
28
Number of Sentences in Summary (3-5):
5
Summarize
Generated Summary:
Indian Space Research Organisation launched the Chandrayaan-3 Mission by using the Geosynchronous Satellite Launch Vehicle Mark III (LVM3) on 14th July 2023 from Sriharikota.
LVM3 is the new launch vehicle of ISRO with the capability to place the modules into the GTO (Geosynchronous Transfer Orbit) in a cost-effective manner.
Continuing the success of its predecessors (Chandrayaan-1 and Chandrayaan-2), the Mission has brought India into the exclusive elite space club.
The Chandrayaan 3 Mission was launched using the LVM3 rocket system.

TEXT SUMMARIZER

- Group No-38
- Vedant Devkar
- Kanav Bhatia
- Shirish Shetty
- Parth Dabholkar

PROBLEM STATEMENT

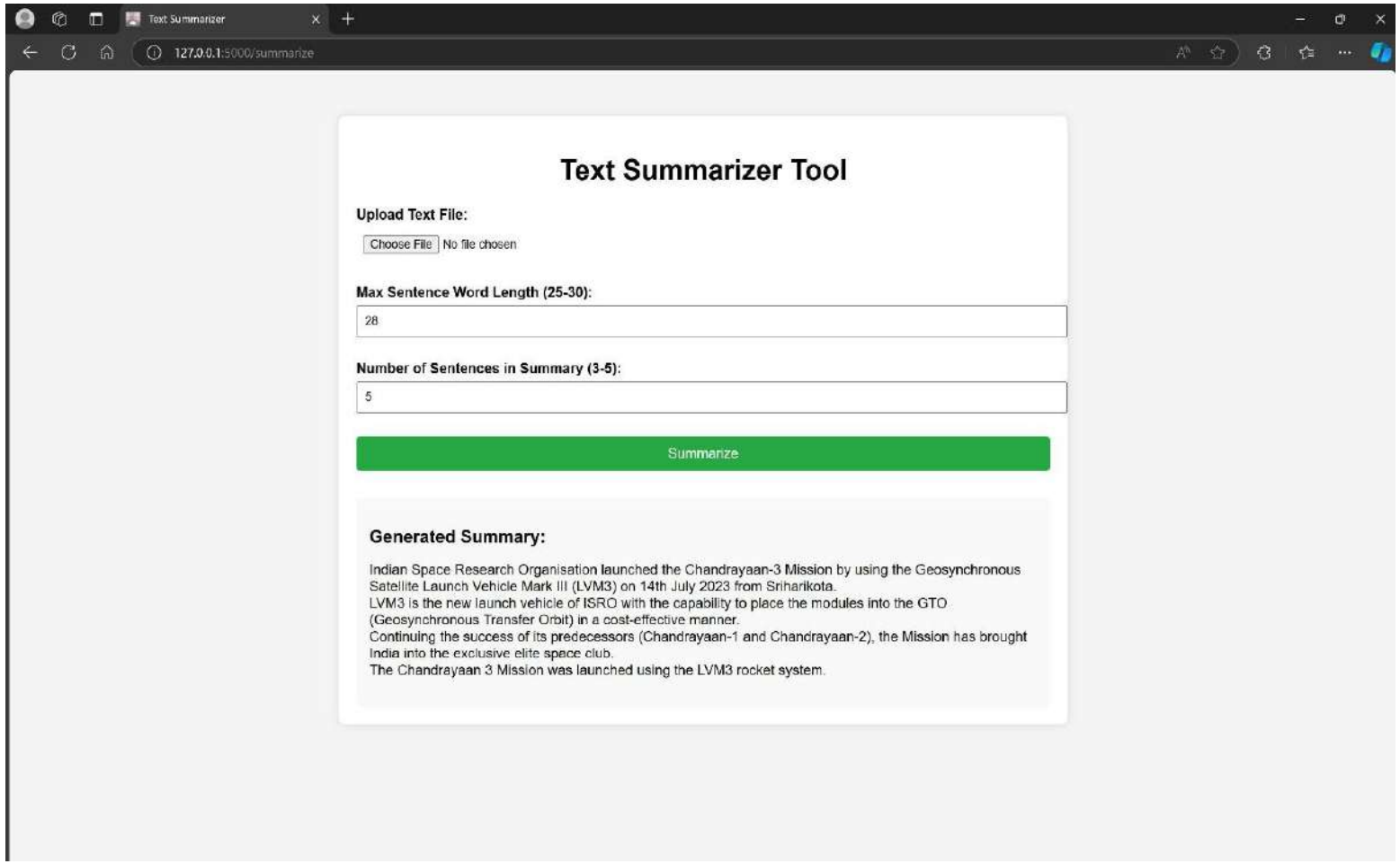
- In an era of information overload, it is challenging for individuals and organizations to quickly digest large volumes of text and extract key insights.
- Manually summarizing text can be time-consuming and inefficient.
- This project aims to develop an automated text summarization tool using Natural Language Processing (NLP) techniques to efficiently generate concise summaries of large documents, helping users save time and focus on the most relevant information.

TECHNOLOGIES USED

- **Natural Language Processing (NLP)**: The core technology for understanding and processing human language.
- **Programming Language (Python)**: Python is commonly used because of its rich ecosystem of NLP libraries such as NLTK, SpaCy, or Scikit-learn.
- **NLTK (Natural Language Toolkit)**: For text preprocessing tasks like tokenization, stop word removal, and stemming/lemmatization.
- **SpaCy**: For more efficient and scalable NLP tasks.
- **Scikit-learn**: For implementing TF-IDF vectorizer and other machine learning models.

ALGORITHM

- **Input Text**: Accept a large block of text that needs to be summarized.
- **Tokenization**: Break the text into individual sentences.
- **Word Tokenization**: Break each sentence into words.
- **Lowercasing**: Convert all text to lowercase to avoid case sensitivity.
- **Stop Word Removal**: Remove common stop words (e.g., "the," "and" "is") that don't contribute much to meaning.
- **Stemming/Lemmatization**: Reduce words to their root or base form (e.g., "running" becomes "run").
- **Term Frequency (TF)**: Calculate how frequently each word appears in a sentence.
- **Inverse Document Frequency (IDF)**: Calculate the inverse frequency of the word across all sentences. Words that appear in many sentences get lower scores.
- **TF-IDF Score**: Multiply the TF and IDF scores for each word. This highlights words that are important in a specific sentence but not common across the text.
- **Sentence Scoring**: For each sentence, calculate a score by summing the **TF-IDF scores** of the words in that sentence. The higher the sentence score, the more relevant it is to the document.
- **Sentence Ranking**: Rank the sentences based on their calculated scores. Sentences with higher scores are considered more important.
- **Summary Generation**: Select the top **N** sentences with the highest scores. The value of **N** can be adjusted depending on the desired summary length. Combine these sentences in the order they appear in the original text to generate the summary.



IMPLEMENTATION

REFERENCES

Jones, Karen Sparck. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation*, vol. 28, no. 1, 1972, pp. 11–21.*This paper introduces the concept of Inverse Document Frequency (IDF), a core component of TF-IDF.*

Ramos, Juan. "Using TF-IDF to Determine Word Relevance in Document Queries." *Proceedings of the First Instructional Conference on Machine Learning*, 2003.*A key resource on understanding how TF-IDF works for document retrieval and text summarization.*

CONCLUSION

The developed text summarizer effectively addresses the challenge of information overload by providing concise summaries of large documents using NLP and TF-IDF techniques. By leveraging term frequency and inverse document frequency, the system highlights the most important sentences, allowing users to quickly grasp the key points without reading the entire text. This approach demonstrates the potential of NLP-based summarization in improving productivity and enhancing information retrieval across various domains. Future improvements could include incorporating advanced techniques like neural network-based models (e.g., transformers) for more context-aware summaries, making the system even more versatile and efficient for different text types.

Parth S. Dabholkar
C31 2103032
Grp no: 38

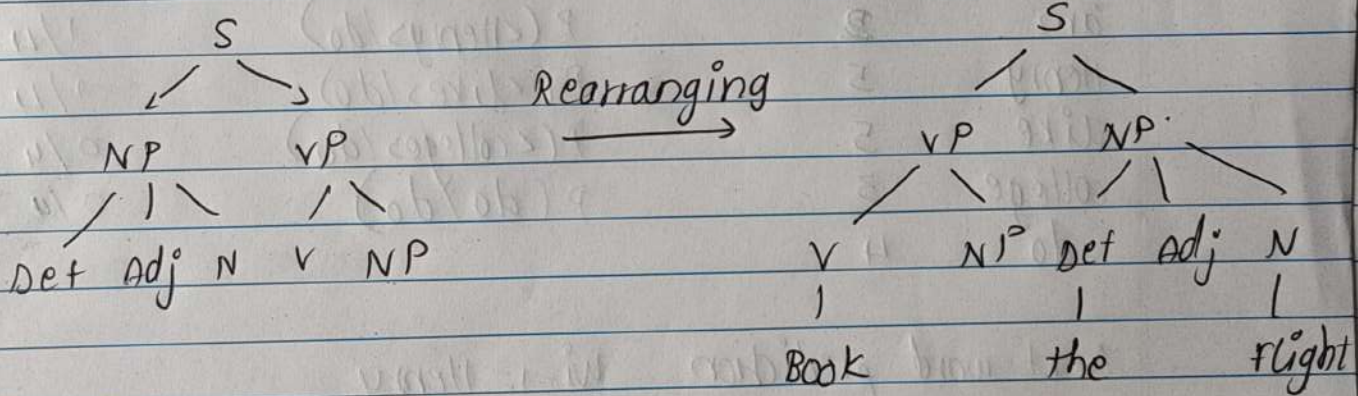
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

NLP ASSIGNMENT

Q1]- Parse the sentence "Book the flight" using the following CFG:

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

⇒ Det → the
VP → Book
N → flight



Q2]- Explain the N-gram model. Given 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>

→ N grams is a fundamental concept in NLP, playing a pivotal role in capturing patterns and relationships within a sequence of words. N grams are continuous sequences of N-items, typically words in the context of NLP. These items can be characters, words or even syllables, depending on the granularity desired. The value of n determines the order of N-gram.

Eg: Unigrams (1-grams): Singe words, eg: cat, dogs
Bigrams (2-grams): Pair of words eg: natural language
Trigrams (3-grams): Triplets of words eg Machine learning model.

Next word prediction w_{i-1} = do

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

Next word	Probability
P(</s>/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 w_{i-1} = Henry

Next word	Probability
P(</s> / 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> / Henry)	0

<s> Do i Like Use trigram P<likes> = 3

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

Next word	Probability next word
P(<s> / like)	0/3
P(<I> / like)	0/3
P(<am> / like)	0/3
P(<Jenny> / like)	1/3
P(<like> / like)	0/3
P(<college> / like)	2/3
P(<do> / like)	0/3

<s> Do I like college? Use four-gram

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

Next word	Probability
P(</s> / I like clg)	2/2
P(<I> / I like clg)	0/2
P(<am> / I like clg)	0/2
P(<Jenny> / I like clg)	0/2
P(<like> / I like clg)	0/2
P(<college> / I like clg)	0/2
P(<do> / I like clg)	0/2

1) <s> I like college </s>

⇒ P(I / <b>) × P(like / I) × P(clg / like) × P(</s> / clg)

⇒ 3/7 × 3/6 × 3/5 × 3/3 × 9/70 = 0.13

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

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

$= 3/7 \times 2/4 \times 3/6 \times 2/5 \times 3/5 = 9/350 = 0.0257$

03) List all bi-grams and compute first five bi gram probabilities for sentence given:

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

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

<s> teachers test student </s>

Bi-grams are:

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

$P(the/student) = P(student/the) / P(student)$

$= 0$

$P(the/<s>) = P(<s>/the) / P(<s>)$

$= 2/3$

$P(student/the) = P(the/student) / P(the)$

$= 2/4$

$P(</s>/test) = P(test/</s>) / P(test)$

$= 1/2$

$P(student/test) = count(test, student) / coun(test)$

$= 1/2$

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.

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

2) Tokenization into N-gram: The text in the corpus is tokenized into N-grams (typically bi-grams or tri-grams).

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

4) Spelling correction: To correct misspelled words, you tokenize the 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. Consider possible correct alternative.

Q4] - <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> teaches/N test/V students/NN </S>

Emmission Probability:

words	Noun	Verb	Model	Preposition	DT
The	0	0	4/4=1	0	
Student	3/8=0.6	0	0	0	
pass	0	1/3=0.3	0	0	

wait	0	0.3	0	0
for	0	0	0	1
teachers	1/5 = 0.2	0	0	0
test	1/5 = 0.2	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>	0.3	0	0.6	0	0
N	0	0.5	0	0	0.5
V	0.3	0	0.3	0.3	0
M	1	0	0	0	0
P	0	0	1	0	0

Q5) In two sentences, different meanings arise from different type of ambiguity.

a) Time flies like an arrow.

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

i) "time flies" [i.e. insects known if flies that exist over time] are similar to an arrow. In this interpretation, "like an

is a simile indicating comparision 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 simile indicating comparision between speech of time and arrow.

b) He crushed the key to my heart"
1) Semantic Ambiguity : The word crushed is the source of semantic ambiguity. It can be interpreted in two ways
- literally as in physically crushing into pieces.
- Figurasively emotional impact : (crushed) it is in emotional.

Q6] - a] Important to Bill

→ Important is an adjective
→ To bill → prepositional phrase
∴ The entire phrase is therefore an adjective phrase.

b] "Looked up the tree"
"Looked is a verb"
"up the bee" is a prepositional phrase.

∴ The entire segment is a prepositional phrase.

Q7] For the given corpus.
N: Noun [Martin, Justin, will, Spot, Pat]
M: Modal [can, will]

V: Verb [watch, spot, put]

statement: Justin will spot Will.

Training Corpus:

<s> Martin Justin can watch will </s>
<s> Spot will watch Martin </s>
<s> will Justin spot Martin </s>
<s> Martin will put spot </s>

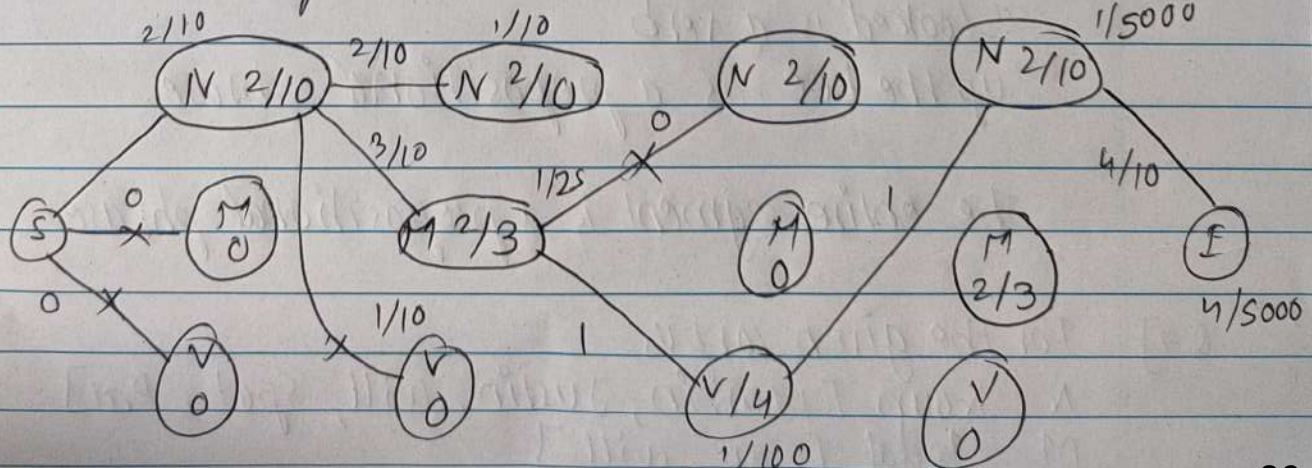
Emission Probability Matrix

	N	M	V
Martin	0.4	0	0
Justin	0.2	0	0
Can	0	0.33	0
watch	0	0	0.5
will	0.2	0.66	0
Spot	0.2	0	0.25
Pdt	0	0	0.25

Transition Probability Matrix

	N	M	V	E
S	1	0	0	0
N	0.2	0.3	0.1	0.4
M	0	0	1	0
V	1	0	0	0

Justin will spot will S → N → M → V → N → E



Q8]- <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>) = 0.5
P(tell|I) = 1
P(to|you) = 1
P(sleep|to) = 0.66
P(and|sleep) = 0.33
P(rest|and) = 1
P(</s>|rest) = 1
P(would|I) = 0.5
P(like|would) = 1
P(to|like) = 1
P(sleep|to) = 0
P(for|sleep) = 0.33
P(an|for) = 1
P(hour|an) = 1
P(</s>|hour) = 1
P(sleep|<s>) = 0.3
P(helps|sleep) = 0.3
P(one|helps) = 1
P(to|one) = 1
P(relax|to) = 0.33
P(</s>|relax) = 1

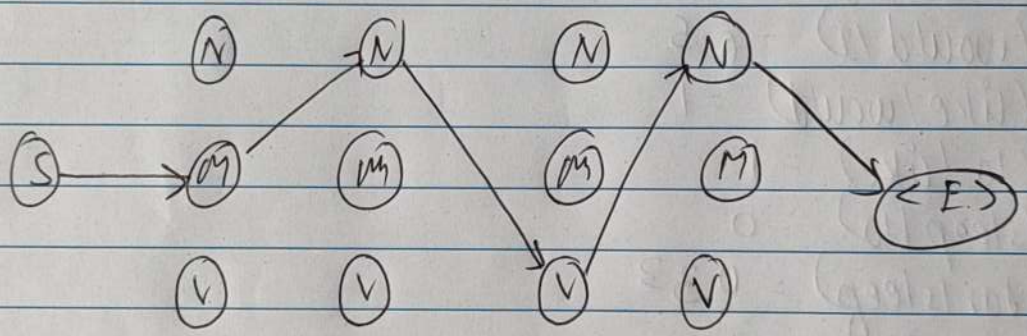
Highest ⇒ P(sleep|to) = 0.66
∴ word is "sleep".

Q9)- Emission Probability

words	Noun	Modal	Verb
Marlin	4/9	0	0
Justin	2/9	0	0
can	0	1/4	0
watch	0	0	2/4
will	1/9	3/4	0
spot	2/9	0	1/4
pat	0	0	1/4

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



Can Justin watch Vill

1) S → N

$V_{11} = P(S \to N) \times P(can/Noun)$
$= 3/4 \times 0 = 0$

$V_{12} = P(S \to N) \times P(can/model)$
$= 0.062$

$V_{13} = P(S \to V) \times P(can/verb)$
$= 0$

2) $V_{21} = V_2 \times P(M \to N) \times (Justin/Noun)$
$= 1/16 \times 1/4 \times 2/9 = 0.034$

$V_{22} = V_{12} \times P(N \to M) \times (Justin/Model)$
$= 1/116 \times 3/9 \times 0 = 0$

$V_{23} = V_{112} \times P(N \to V) \times P(Justin/Verb)$
$= 0$

iii) $V31 = V22 \times P(N \to N) \times P(\text{watch} / \text{Noun})$
$= 0.0035 \times 1/9 \times 0$
$= 0$

$V32 = V22 \times P(N \to M) \times P(\text{watch} / \text{Model})$
$= 0$

$V33 = V22 \times P(N \to V) \times P(\text{watch} / \text{verb})$
$= 0.0035 \times 1/9 \times 2/4$
$= 0.000055$

iv) $V41 = V33 \times P(V \to N) \times (\text{will} / \text{noun})$ $V42 = 0$ $V43 = 0$
$= 0.00056 \times 1 \times 1/9$
$= 9.55 \times 10^{-6}$

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

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

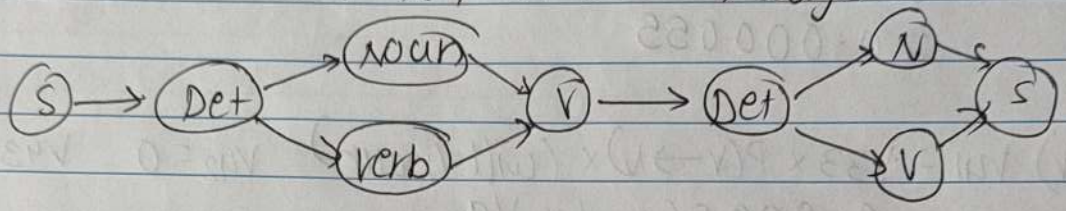
"The park is a book"

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

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

The	park	is	a	book
Det	Noun	verb	Det	Noun
	verb			verb

1 x 2 x 1 x 1 x 2 = 4 ways



P (Det / the, <s>) = 4/6 x 2/4 = 1/3
P (Noun / Park, Det) = 1/6 x 1/3 = 1/18
P (verb / Park, Det) = 1/400 x 1/3 = 1/1200
∴ 1/18 > 1/1200
Park is a Noun.

P(verb / is, Noun) = 1/6 x 1/18 = 1/108
P(Det / a, verb) = 1/6 x 1/108 = 1/648

book is Noun
P(Noun / book, Det) = 1/6 x 1/648 = 1/3888
book as verb
P(verb / book, Det) = 1/4 x 0 = 0

∴ Book is a noun
∴ Result after applying HMM Model

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

Q11)- <s> Mary Jane can see will </s>
<s> Spot will see Mary </s>
<s> Will Jane spot Mary </s>
<s> Mary will put spot </s>

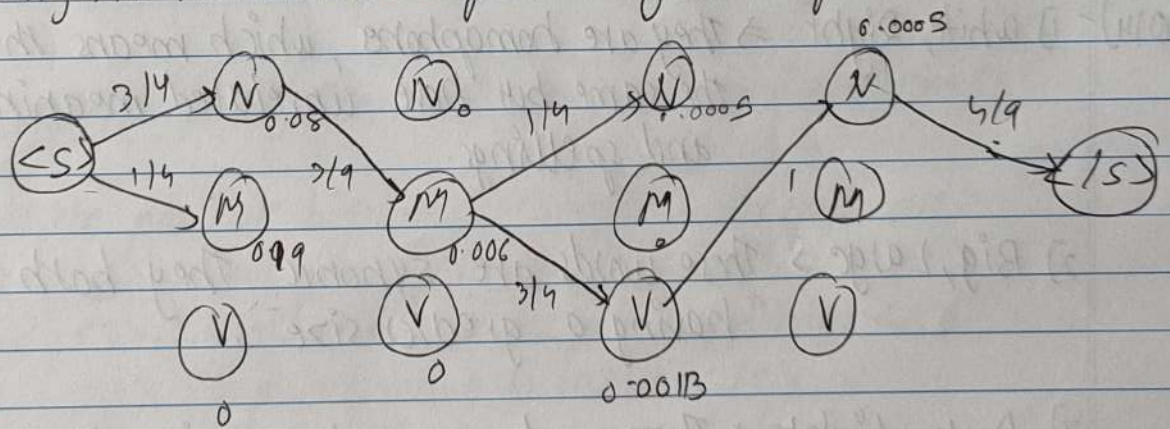
Emmission Prob. Matrix

	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
put	0	0	1/4

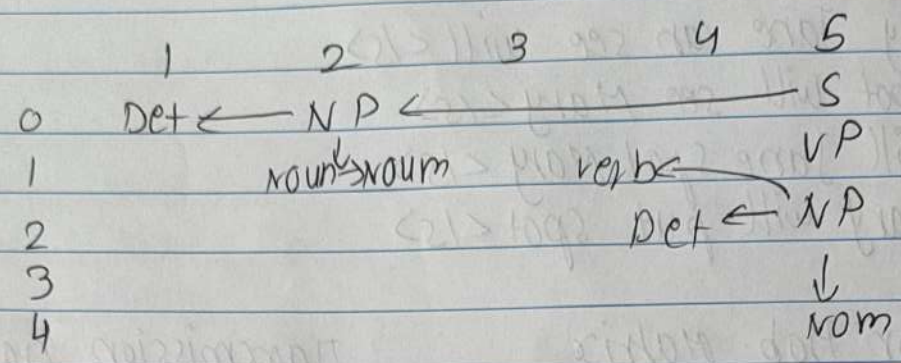
Transmission Prob Matrix

	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

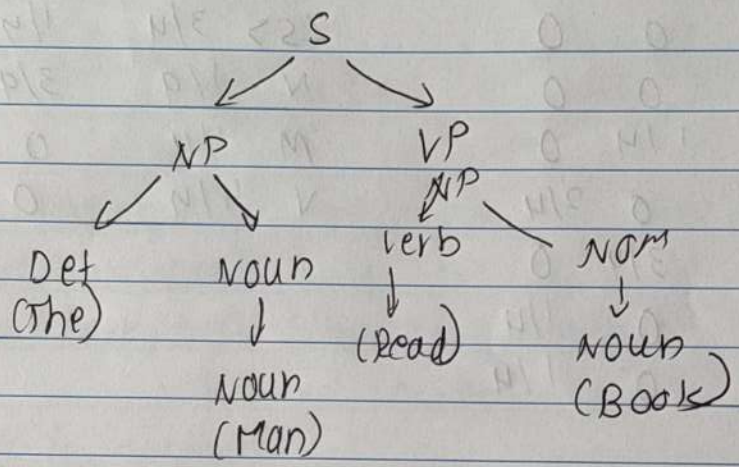
Q12)- Using the transmission probability from previous numerical



Q13) The 1 Man 2 send 3 this 4 book 5



This can be converted into a tree as follows



Q14) 1) White, Right ⇒ They are homophones, which means they sound the same but have unrelated meanings and spellings.

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

3) Dark, Light → These words are antonymic. Dark is opposite to light.

4) Car, vehicle → These words are hypernym, homonym relationship. A car is a type of vehicle.

Q15) Ans is C
(19/20)\d{2} - (0[1-9][0-2]) - ([0-2][1-9]/3[0-1])

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

Q16) i) Homonym → Hyponym

"Invention" [wi] is a Homonym of the "wheel" in the sentence.

2) Hyponym → Hypernym

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

3) Meronym → iii) daffodils → flower
daffodils are part of the flower) → Sentence B

4) ~~Hypernym~~ Holonym → i) wheel → car → Sentence D

Q17) All the possible bigrams in the roll. corpus are:
<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>

Bi-grams:- (<s>, I) (I, tell) (tell, you) (you, to) (to, sleep) (sleep, 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>).

⇒ Context free Grammar (CFG) is a formal grammar used to generate all possible strings in a given formal language. It consists of set of production rules that describe how a strings in the language can be formed from grammar's alphabet.

A CFG consists of:

1. Set of Non-Terminal symbol (N): These are syntactic variables used for denoting sets of strings (eg S, VP, NP).
2. Set of terminal symbols (T): These are actual symbol appear in the string (eg: words)
3. A set of production rules (P): Rules that define how non-terminals can be transformed into terminals or other non-terminals.
4. Start symbol (S): Special Non-terminal → string begins.

Eg: • Non-Terminal (S → sentence, NP → noun phrase, VP → verb phrase)
• Terminals: John, Mary, loves, a, book
• Production Rules: S → NP VP
NP → John | Mary | a book
VP → loves NP
• Start symbol: S

CFG: S → NP - VP → John VP → John loves NP → John loves a book.

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

2) Sub-categorization: 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 and arguments.

3) Movement: NLP allows for movement, like wh-questions, or topicalization. For eg: in question "what did you eat" object moves to front which is challenging for the rule follow fixed heirarchial structure.

Q18) Name the Ambiguity.

⇒ 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 boy saw her.

2) The boy (horse) ran up the hill. It was very steep. It soon got tired
→ Pronoun ambiguity.
The first 'it' refers to the hill.
The second 'it' refers to the horse.

3) The car hit the pole while it was moving
→ Syntactic Ambiguity.
It can refer both car or pole i.e. either 'car was moving when it hit the pole' or 'pole was moving when it was hit by the car'.