NLP Experiment 1

Aim: Study various applications of NLP and NLP Tools

Theory:

Natural Language Processing (NLP) has a wide range of applications. Some of them are:

- **Sentiment Analysis**: Sentiment analysis involves determining the emotional tone or sentiment expressed in a text, often used in social media monitoring and customer feedback analysis.
- **Chatbots**: Chatbots use NLP to engage in natural conversations with users. They can be found in customer support, virtual assistants, and more.
- **Machine Translation**: NLP is used for translating text from one language to another, such as Google Translate.
- **Named Entity Recognition (NER)**: NER identifies entities like names of people, organisations, locations, and more in text, which is useful in information retrieval and data categorisation.
- **Text Summarization**: NLP is used to automatically generate concise summaries of longer texts, which can be handy for news articles or academic papers.
- **Speech Recognition**: Converting spoken language into text is crucial in voice assistants (e.g., Siri, Alexa) and transcription services.
- **Text Classification**: Assigning predefined categories or labels to text, like spam detection in emails or classifying articles into topics.
- **Question Answering**: This is used in search engines to provide direct answers to user questions, like the featured snippets in Google search results.

We aim to solve the problem of Text Classification from our given problem statement. Text classification, in a more technical sense, is a natural language processing (NLP) task where machine learning algorithms are trained to assign predefined labels or categories to text data. It involves extracting relevant features from the text, such as words or phrases, and using these features to train a model to recognize patterns and associations between the text and the categories. Once the model is trained, it can automatically classify new, unseen text into the appropriate categories based on the patterns it has learned. Text classification finds applications in various fields, from sentiment analysis and spam detection to topic categorization and language identification, aiding in automating the organization and management of textual data.

NLP Experiment 2

Aim: Study Various Applications of NLP and Formulate the Problem Statement for Mini Project

Theory:

1. Introduction to Natural Language Processing (NLP)
Natural Language Processing (NLP) is a field of Artificial Intelligence that focuses on the interaction between computers and human language. It enables machines to understand, interpret, and generate human language in a meaningful way. NLP plays a crucial role in many real-world applications by processing large amounts of natural language data and extracting valuable information from it.

2. Applications of NLP
NLP has a wide range of applications, many of which are embedded into everyday technologies:

- **Machine Translation**: Automatic translation of text from one language to another, such as Google Translate. It involves complex techniques to maintain the context, grammar, and nuances of the original text.

- **Text Categorization**: Automatically categorizing or classifying text into predefined categories (e.g., email spam filtering, news classification). It uses methods such as supervised learning to label documents.

- **Text Summarization**: Creating concise summaries of long pieces of text while preserving the core message. Summarization can be extractive (selecting important parts) or abstractive (generating a new summary).

- **Chatbot Development**: Virtual assistants that simulate human conversation to provide customer service, technical support, or personal assistance. Modern chatbots leverage deep learning models to understand user input and deliver relevant responses.

- **Plagiarism Detection**: Detecting similarities in text content to ensure originality, widely used in academic and professional environments. NLP-based models can identify reworded text and paraphrased content.

- **Spelling & Grammar Checkers**: Systems that detect and correct spelling mistakes, grammatical errors, and syntactical issues in text. These systems use both rule-based and machine-learning approaches for improving text quality.

- **Sentiment/Opinion Analysis**: Extracting and analyzing opinions or sentiments expressed in text, widely used in market analysis, customer reviews, and social media monitoring.

- **Question Answering Systems**: Answering user queries by retrieving relevant information from databases or documents. Such systems range from simple FAQ retrieval to complex models that provide specific answers to complex questions.

- **Personal Assistants**: NLP-based digital assistants like Siri, Alexa, and Google Assistant help users perform tasks through voice commands, such as setting reminders, searching the web, or controlling smart home devices.

- **Tutoring Systems**: AI-driven tutoring platforms that provide personalized learning experiences, assess student progress, and offer tailored educational content.

3. Literature Review of Recent Advancements in NLP

In recent years, NLP has witnessed significant advancements due to the rise of deep learning models like transformers. Key innovations include:

- **Transformers and Attention Mechanisms**: The introduction of the transformer architecture (e.g., BERT, GPT) revolutionized NLP by improving model performance in tasks like translation, summarization, and question answering. Attention mechanisms allow the model to focus on different parts of the input data, improving contextual understanding.

- **Pre-trained Language Models**: Large pre-trained models, such as GPT-3 and BERT, have become the foundation of many NLP applications. These models are trained on vast datasets and can be fine-tuned for specific tasks, drastically reducing the need for task-specific training data.

- **Transfer Learning in NLP**: Transfer learning, where a model trained on one task is fine-tuned for another, has been highly effective in NLP. This allows smaller datasets to benefit from pre-trained knowledge, leading to more robust solutions.

- **Multilingual Models**: New models capable of handling multiple languages simultaneously, such as mBERT, enable multilingual machine translation, cross-lingual tasks, and more without requiring language-specific data.

4. Problem Statement for Mini Project

Based on the study of various applications, the mini-project will focus on **developing a text summarization tool** that can generate concise summaries of legal documents. Legal texts tend to be lengthy, and summarization tools can help extract key information efficiently for legal professionals. The project will aim to explore both extractive and abstractive summarization techniques using pre-trained models like BERT or GPT-2.

Objective: To create an efficient and accurate text summarization tool for legal documents using state-of-the-art NLP techniques, with a focus on improving the quality of the generated summaries.

Our problem statement for the NLP project is a Text classification problem. We have a dataset which consists of:

- Synopsis of a movie
- Genre of the movie

The synopsis for each movie will labelled with its genre. We aim to develop a model that can correctly predict the Genre of a movie when given a synopsis.

File Name: train.csv

Details:

Number of rows: 54,000

Columns:
Id, movie, name, synopsis

Column Description:
id: ID of the movie
movie_name: Name of the movie
genre: Genre of the movie

Primarily the "synopsis" column will be used for all NLP experiments while the "genre" column will be used for labelling the data in the NLP Mini project.

Sample data:

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

5. Conclusion
NLP has become a cornerstone of modern AI applications, with use cases in various domains such as language translation, customer service, and legal assistance. This project will contribute to the growing field by tackling the challenge of text summarization for legal texts, enabling users to process information faster and more effectively.

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",
```

```
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)
```

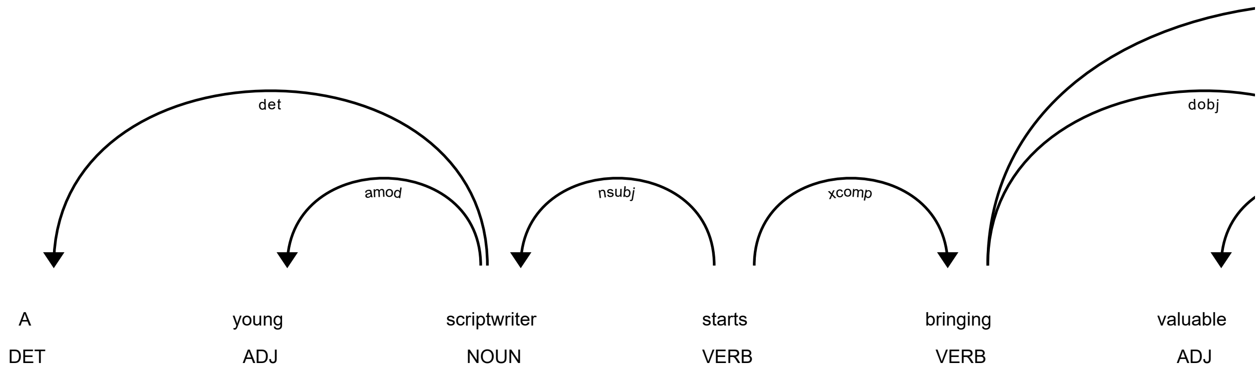
```
deep_parse_results[0]
```

```
 {'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 Day - 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': 0.0208333333

Emission Probabilities:
{'UNKNOWN': {'A': 0.020833333333333332, 'young': 0.020833333333333332, 'scriptwriter': 0.020833333333333332, 'starts': 0.0208333333
```

```
# 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 wildl
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	

0	0	0	44978	Super Me	starts bringing valuable ...	fantasy	starts bringing valuable ob...	y br
1	1	1	50185	Entity Project	a director and her friends renting a haunted h...	horror	director friends renting haunted house capture...	ha
				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'
```

```
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')
```

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

```python
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 [==============================] - 9s 8ms/step - loss: 0.3233 - accuracy
Epoch 3/100
1053/1053 [==============================] - 9s 9ms/step - loss: 0.3074 - accuracy
Epoch 4/100
1053/1053 [==============================] - 9s 9ms/step - loss: 0.2953 - accuracy
Epoch 5/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.2870 - accuracy
Epoch 6/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2797 - accuracy
Epoch 7/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.2724 - accuracy
Epoch 8/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2653 - accuracy
Epoch 9/100
1053/1053 [==============================] - 9s 8ms/step - loss: 0.2581 - accuracy
Epoch 10/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2509 - accuracy
Epoch 11/100
1053/1053 [==============================] - 9s 8ms/step - loss: 0.2439 - accuracy
Epoch 12/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2371 - accuracy
Epoch 13/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2302 - accuracy
Epoch 14/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2236 - accuracy
Epoch 15/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2168 - accuracy
Epoch 16/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2109 - accuracy
Epoch 17/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2044 - accuracy
Epoch 18/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1988 - accuracy
Epoch 19/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1937 - accuracy
Epoch 20/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.1884 - accuracy
Epoch 21/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1831 - accuracy
Epoch 22/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.1785 - accuracy
Epoch 23/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1740 - accuracy
Epoch 24/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.1696 - accuracy
Epoch 25/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1652 - accuracy
Epoch 26/100
 053/1053 [==============================] - 8s 8ms/step - loss: 0.1622 - accurac
```

```
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: GENRE PREDICTION

Problem Statement:

Text classification uses the movie's synopsis to predict the genre of the movie.

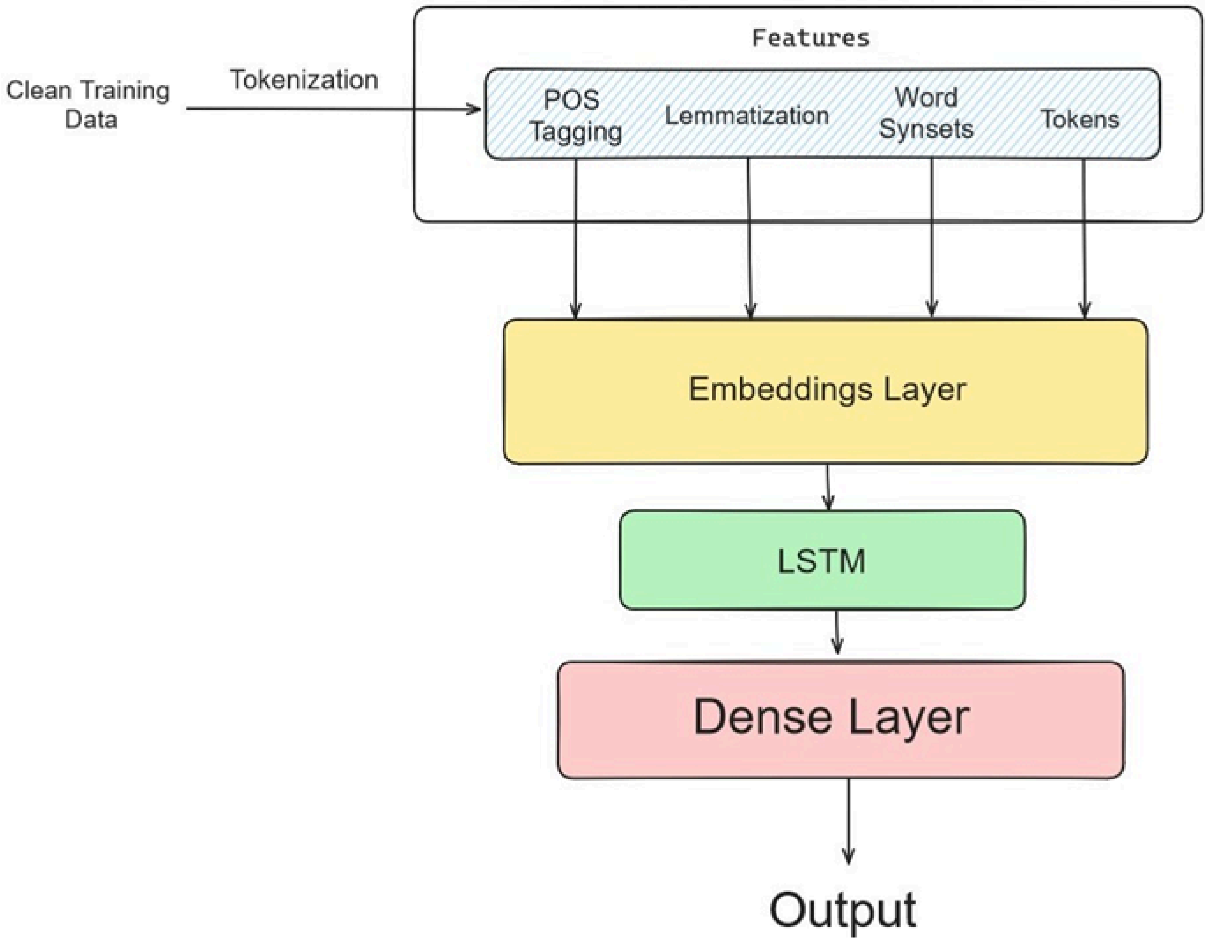
Sample Input:

A young scriptwriter returns valuable objects from his short nightmares of being chased by a demon. Selling them makes him rich.

Sample Output:

Fantasy

Flowchart:



Libraries Used:

- Pandas (for loading data)

- NLTK (Lemmatizing, tokenizing, Wordnet analysis)

- Spacy (POS tagging)

- Tensorflow and Keras (for developing neural networks to develop a text classification model)

Project Description:

The project aims to tackle a text classification problem. It aims to train on the synopsis of a movie and classify its genre. The dataset is webscrapped data from IMDB with 54000 rows of movies and a total of 10 unique genres to classify from.

Workflow:

- Loading pre-processed data:
    - Stop words are removed
    - Words are lemmatized
    - Words outside the character set and noise in the data are removed
    - Words are already lowercased

- Extracting features from the data to add multiple data points for a neural network to learn from.
- We use multilabel binarized which encodes labels in the following manner:
  eg: labels = [apple, banana, mango]
  Encode: [[
  1,0,0
  0,1,0
  0,0,1
  ]]

The hyperparameters we have used are:

max_words: The maximum number of words to be tokenized. This hyperparameter limits the vocabulary size to the most frequently occurring words in your text data.

max_sequence_length: The maximum sequence length for input data sequences. This hyperparameter determines the length to which input sequences will be padded or truncated.

output_dim: The output dimension of the word embedding layer (Embedding). In this case, it's set to 300, meaning each word will be represented as a 300- dimensional vector in the embedding space.

LSTM: The number of LSTM units in the LSTM layer. In our case, there are 128 LSTM units.

activation: The activation function used for the output layer. sigmoid is used because it's suitable for multi-label classification problems, where each label is binary.

loss: The loss function for model training. It's set to 'binary_crossentropy', which is appropriate for multi-label classification tasks.

optimizer: The optimization algorithm for training the model. Adam is a popular optimizer, and the learning rate is set to 0.001.

- The data points used are:
    - Tokens of each synopsis
    - POS tags for each token
    - Lemmatizing each word (although training data is lemmatized beforehand)
    - Word Synset Analysis
- We use these features as training data for the neural network.
- We develop a simple neural network consisting of:
    - Embedding layer
    - LSTM layer
    - Dense layer
- We train the neural network for 100 epochs.

Results and Evaluation:

After training the neural network for 100 epochs, we achieved an accuracy of around 76.22% and a loss of around 0.0841

```
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')
```

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

```python
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 [==============================] - 9s 8ms/step - loss: 0.3233 - accuracy
Epoch 3/100
1053/1053 [==============================] - 9s 9ms/step - loss: 0.3074 - accuracy
Epoch 4/100
1053/1053 [==============================] - 9s 9ms/step - loss: 0.2953 - accuracy
Epoch 5/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.2870 - accuracy
Epoch 6/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2797 - accuracy
Epoch 7/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.2724 - accuracy
Epoch 8/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2653 - accuracy
Epoch 9/100
1053/1053 [==============================] - 9s 8ms/step - loss: 0.2581 - accuracy
Epoch 10/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2509 - accuracy
Epoch 11/100
1053/1053 [==============================] - 9s 8ms/step - loss: 0.2439 - accuracy
Epoch 12/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2371 - accuracy
Epoch 13/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2302 - accuracy
Epoch 14/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2236 - accuracy
Epoch 15/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2168 - accuracy
Epoch 16/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.2109 - accuracy
Epoch 17/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.2044 - accuracy
Epoch 18/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1988 - accuracy
Epoch 19/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1937 - accuracy
Epoch 20/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.1884 - accuracy
Epoch 21/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1831 - accuracy
Epoch 22/100
1053/1053 [==============================] - 7s 7ms/step - loss: 0.1785 - accuracy
Epoch 23/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1740 - accuracy
Epoch 24/100
1053/1053 [==============================] - 8s 7ms/step - loss: 0.1696 - accuracy
Epoch 25/100
1053/1053 [==============================] - 8s 8ms/step - loss: 0.1652 - accuracy
Epoch 26/100
 053/1053 [==============================] - 8s 8ms/step - loss: 0.1622 - accurac
```

```
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(
```