```
pip install nltk spacy gensim
```

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

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

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

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

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

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

synopses = df['synopsis'].tolist()
```

```
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

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

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

```
2023-10-23 08:00:18.401689: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38
⚠ As of spaCy v3.0, shortcuts like 'en' are deprecated. Please use the
full pipeline package name 'en_core_web_sm' instead.
Collecting en-core-web-sm==3.6.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_
```

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

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

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

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

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

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

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

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

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


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


data.head(5)
```

LSTM

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

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

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

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

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

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

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

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

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

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

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

# View classification report for more detailed evaluation
```

```
                                ,                ,

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

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
  X['Combined_Features'] = X.apply(lambda row: ' '.join(map(str, row)), axis=1)
Epoch 1/100
1053/1053 [==============================] - 27s 24ms/step - loss: 0.3298 - accura
Epoch 2/100
1053/1053 [==============================] - 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.

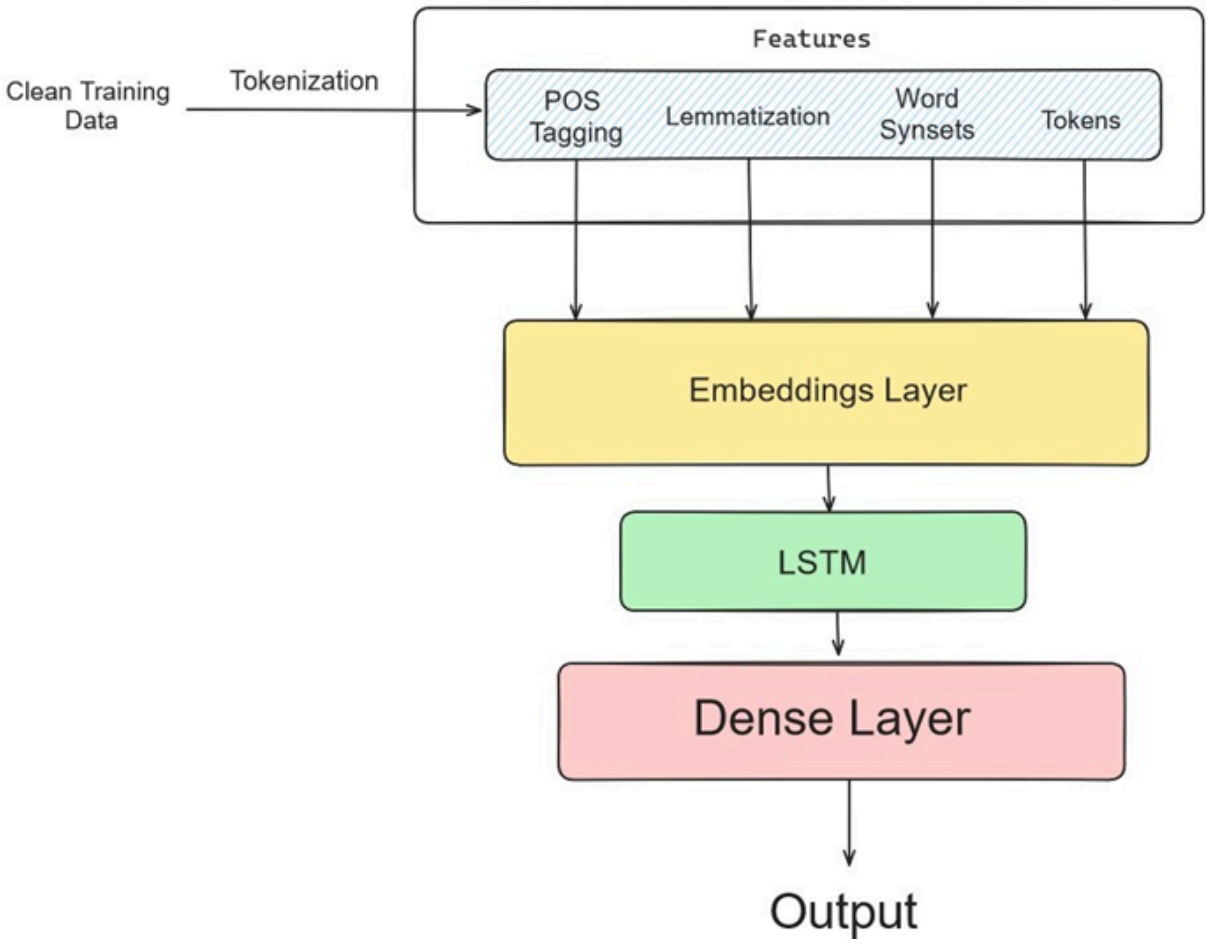
Sample Input:

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