Natural Language Programming Project

Sentiment Analysis

Code Explanation

Imported Libraries :

```python
# Importing Libraries Will Be Needed
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

#For embedding
from sentence_transformers import SentenceTransformer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# For Model Building
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import LSTM, Bidirectional, Dropout, Dense, Embedding


from sklearn.model_selection import train_test_split

# Import metrics for detailed evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

- **Data Handling**: `pandas`, `numpy` for cleaning and processing data.
- **Text Preprocessing**: `nltk`, `re` for cleaning and tokenizing text.
- **Feature Extraction**: `CountVectorizer`, `TfidfVectorizer` for text to numeric conversion.
- **Embeddings**: `SentenceTransformer`, `Tokenizer`, `pad_sequences` for embedding creation.
- **Model Building**: `Sequential`, `LSTM`, `Dense`, `Adam` for neural networks.

- **Data Splitting**: `train_test_split` to divide training/testing sets.
- **Evaluation**: `accuracy`, `precision`, `recall`, `f1` for performance metrics.

```python
# Download NLTK resources
nltk.download('stopwords')
nltk.download('punkt')  # For tokenization
nltk.download('wordnet')  # For lemmatization
nltk.download('omw-1.4')  # For WordNet Lemmatizer's language support
```

[2]

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\elora\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\elora\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\elora\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\elora\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!

True
```

Download NLTK Resources:

- `stopwords`: For removing common irrelevant words.
- `punkt`: For tokenizing text into words or sentences.
- `wordnet`: For lemmatization tasks.
- `omw-1.4`: Provides language support for `WordNetLemmatizer`.

```python
# Load the dataset
data = pd.read_csv('Book1.csv')

# Display the first few rows of the dataset
print(data.head())
```

```
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive
```

- **Load Dataset**: Reads a CSV file (`Book1.csv`) containing text reviews and sentiment labels.
- **Preview Data**: Displays the first few rows to confirm the structure:
  - **Columns**:
    - `review`: Text of the review.
    - `sentiment`: Sentiment label (positive/negative).

```python
data['review'].duplicated().sum()
```

```
np.int64(0)
```

- **Check for Duplicate Reviews**: Counts the number of duplicate entries in the `review` column.
- **Result**: There are no duplicate reviews (`0`).

```python
# Find duplicates in review column
duplicates = data[data['review'].duplicated()]

duplicates
```

review	sentiment

- **Find Duplicate Reviews**: Identifies rows in the `review` column that are duplicates.
- **Result**: No duplicates are found in the dataset (empty output).

```
data.isnull().sum()
```

[6]

```
review       0
sentiment    0
dtype: int64
```

- **Check for Missing Values**: Verifies if there are any null or missing entries in the dataset.

```
data.describe()
```

[7]

	review	sentiment
count	1999	1999
unique	1999	2
top	I loved this movie! It was all I could do not ...	positive
freq	1	1005

- **Dataset Summary**: Provides statistical information about the data:
  - **Count**: 1,999 rows in both `review` and `sentiment` columns.
  - **Unique**: 1,999 unique reviews, indicating no duplicates.
  - **Top**: Most frequent review is "I loved this movie! It was all I could do not ..." and most frequent sentiment is "positive."
  - **Frequency**: Sentiment "positive" appears 1,005 times.

```
data['sentiment'].value_counts()
```

[8]

```
sentiment
positive    1005
negative     994
Name: count, dtype: int64
```

- **Sentiment Distribution**:
  - **Positive**: 1,005 reviews.

- **Negative**: 994 reviews.
- The dataset is nearly balanced between the two sentiment classes.

Data Cleaning and Preprocessing

```
# Removing duplicates in review column
data = data.drop_duplicates(subset=['review'])
```
[9]

```
data['review'].duplicated().sum()
```
[10]

```
np.int64(0)
```

- **Remove Duplicates**: Eliminates duplicate entries in the `review` column using `drop_duplicates`.
- **Verify Duplicates**: Confirms no duplicate reviews remain in the dataset (`0 duplicates`).
- Ensures data integrity for analysis.

```python
def clean_text(text):
    # Lowercase
    text = text.lower()

    # Remove HTML tags
    text = re.sub(r'<[^>]+>', '', text)

    # Remove URLs
    text = re.sub(r'https?://\S+', '', text)

    # Remove special characters (punctuations) and numbers
    text = re.sub(r"[^a-zA-Z\s]", ' ', text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word not in stop_words)

    # Tokenization (word-level)
    word_tokens = word_tokenize(text)  # Split into words

    # Stemming
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in word_tokens]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = [lemmatizer.lemmatize(word) for word in stemmed_words]

    # Combine the final processed words into a single string
    final_text = ' '.join(lemmatized_words)
    return final_text

# Apply the cleaning function to the review column
data['cleaned_review'] = data['review'].apply(clean_text)
```

[11]

- **Text Cleaning Function**:
  - Converts text to lowercase for consistency.
  - Removes HTML tags, URLs, special characters, and numbers for clarity.
  - Removes stopwords (e.g., "the," "and") to focus on meaningful words.
  - Tokenizes text into individual words.
  - Applies stemming (reduces words to root form) and lemmatization (converts words to base form).
  - Combines processed words back into a single string.
- **Application**: Cleans all reviews in the `review` column and stores the results in a new column called `cleaned_review`.

```
[12] print(data.head())

...                                               review sentiment  \
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive

                                      cleaned_review
0  one review mention watch oz episod hook right ...
1  wonder littl product film techniqu unassum old...
2  thought wonder way spend time hot summer weeke...
3  basic famili littl boy jake think zombi closet...
4  petter mattei love time money visual stun film...
```

- **Data Preview**:
  - Displays the first few rows of the dataset after cleaning.
  - `review`: Original text reviews with raw formatting (HTML tags, stopwords, etc.).
  - `cleaned_review`: Preprocessed reviews, cleaned and ready for analysis.
  - The `cleaned_review` column shows simplified and processed text for better model input.

Text Representation

```python
reviews = data['cleaned_review']
sentiments = data['sentiment']  # Target labels
```

```python
# 1. Bag of Words (BoW)
bow_vectorizer = CountVectorizer()
bow_features = bow_vectorizer.fit_transform(reviews)

print("BoW Shape:", bow_features.shape)
print("Sample BoW Vector:", bow_features[0].toarray())
```

```
... BoW Shape: (1999, 17155)
    Sample BoW Vector: [[0 0 0 ... 0 0 0]]
```


- **Text Representation**:
  - Converts cleaned reviews into numerical features for model training.
- **Bag of Words (BoW)**:
  - Uses `CountVectorizer` to create a sparse matrix of word frequencies.
  - **Shape**: `(1999, 17155)` - 1999 reviews and 17,155 unique words.
  - **Sample Vector**: Represents the word frequency of a single review as an array.
- **Purpose**: Transforms text data into a format suitable for machine learning models.

```
# 2. TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_features = tfidf_vectorizer.fit_transform(reviews)

print("TF-IDF Shape:", tfidf_features.shape)
print("Sample TF-IDF Vector:", tfidf_features[0].toarray())
```

```
TF-IDF Shape: (1999, 17155)
Sample TF-IDF Vector: [[0. 0. 0. ... 0. 0. 0.]]
```

- **TF-IDF Representation**:
  - Converts cleaned reviews into numerical features using the **TF-IDF (Term Frequency-Inverse Document Frequency)** technique.
- **Details**:
  - **Shape**: `(1999, 17155)` - 1999 reviews and 17,155 unique words.
  - **Sample TF-IDF Vector**: Represents the importance of each word in a review as a weighted value.
- **Purpose**: Captures the importance of words in reviews relative to the dataset, making it more informative than simple word counts.

```
# 3. N-Grams (bi-grams or tri-grams)
ngram_vectorizer = CountVectorizer(ngram_range=(2, 2))  # Bi-grams
ngram_features = ngram_vectorizer.fit_transform(reviews)

print("N-Gram Shape:", ngram_features.shape)
print("Sample N-Gram Vector:", ngram_features[0].toarray())
```

```
N-Gram Shape: (1999, 177377)
Sample N-Gram Vector: [[0 0 0 ... 0 0 0]]
```

- **N-Grams Representation**:
  - Uses `CountVectorizer` to create features based on bi-grams (pairs of consecutive words).
- **Details**:
  - **Shape**: `(1999, 177377)` - 1999 reviews and 177,377 unique bi-grams.
  - **Sample N-Gram Vector**: Represents the frequency of bi-grams in a single review.
- **Purpose**: Captures contextual word pairs, providing more insight into word relationships within reviews.

```
# Load pre-trained BERT-based SentenceTransformer model
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

# Generate embeddings for all reviews
data['embeddings'] = data['cleaned_review'].apply(lambda x: model.encode(x))

# Convert to NumPy array for use in models
review_embeddings = np.array(data['embeddings'].tolist())
print("Embeddings Shape:", review_embeddings.shape)
```
[17]

```
Embeddings Shape: (1999, 384)
```

- **Sentence Embeddings**:
  - Uses the pre-trained `SentenceTransformer` model (`paraphrase-MiniLM-L6-v2`) to generate dense semantic embeddings for each cleaned review.
- **Details**:
  - **Embedding Shape**: `(1999, 384)` - 1999 reviews represented as 384-dimensional vectors.
  - Converts textual data into meaningful numerical representations capturing semantic context.
- **Purpose**: Provides a powerful and context-aware representation of reviews, suitable for advanced machine learning models.

Padding

```
# Step 1: Initialize the tokenizer
tokenizer = Tokenizer(num_words=5000)

# Step 2: Fit the tokenizer on the cleaned review data
tokenizer.fit_on_texts(data['cleaned_review'])

# Step 3: Convert text to sequences of integers
sequences = tokenizer.texts_to_sequences(data['cleaned_review'])

# Step 4: Define the maximum sequence length
max_length = 500

# Step 5: Apply padding to standardize sentence lengths
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post', truncating='post')
```
[18]

- **Padding Process**:
  1. **Initialize Tokenizer**: Limits the vocabulary to the top 5,000 most frequent words.
  2. **Fit Tokenizer**: Maps words in the cleaned reviews to integer indices.
  3. **Convert to Sequences**: Transforms text reviews into sequences of integers.
  4. **Define Maximum Length**: Sets a fixed sequence length (`500`).
  5. **Apply Padding**: Adds zeros to shorter sequences to standardize all to the same length.
- **Purpose**: Ensures all input sequences have uniform length for compatibility with machine learning models.

Model Building

```python
# Step 6: Define the model architecture
vocab_size = len(tokenizer.word_index) + 1  # Vocabulary size from tokenizer
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=max_length),  # Embedding layer
    Bidirectional(LSTM(64, return_sequences=False)),  # Bidirectional LSTM
    Dropout(0.5),  # Dropout for regularization
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

[19]

- **Model Architecture**:
  - **Embedding Layer**: Converts words to dense vectors of size `128` based on the vocabulary size and input sequence length.
  - **Bidirectional LSTM**: Processes text in both forward and backward directions with 64 units, capturing context from both ends.
  - **Dropout**: Adds a 50% dropout rate to prevent overfitting.
  - **Dense Layer**: A single neuron with a sigmoid activation function for binary classification (outputting probabilities for positive or negative sentiment).
- **Purpose**: Builds a robust and context-aware model suitable for sentiment analysis.

Train the model

```python
# Define X and y for training and testing
# X: Padded sequences (input features)
X = padded_sequences

# y: Sentiment labels (target)
# Convert sentiment into binary format: 1 for 'positive', 0 for 'negative'
y = (data['sentiment'] == 'positive').astype(int)  # Ensure 'sentiment' column exists in the dataset

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Display shapes to confirm
print(f"Training Data Shape: X_train={X_train.shape}, y_train={y_train.shape}")
print(f"Testing Data Shape: X_test={X_test.shape}, y_test={y_test.shape}")
```

[27]

```
Training Data Shape: X_train=(1199, 500), y_train=(1199,)
Testing Data Shape: X_test=(800, 500), y_test=(800,)
```


- **Train-Test Split**:
  - **Input Features (X)**: Uses `padded_sequences` as input data.
  - **Target Labels (y)**: Converts `sentiment` to binary format (`1` for positive, `0` for negative).
  - Splits the data into:
    - **Training Set**: 60% of the data for training the model (`X_train`, `y_train`).
    - **Testing Set**: 40% of the data for evaluating the model (`X_test`, `y_test`).
- **Shapes**:
  - Training Data: 1,199 reviews with 500 features each.
  - Testing Data: 800 reviews with 500 features each.
- **Purpose**: Prepares the dataset for training and ensures separate evaluation data for unbiased model testing.

```python
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),  # Optimizer
              loss='binary_crossentropy',            # Loss function for binary classification
              metrics=['accuracy'])                  # Accuracy metric for evaluation
```

- **Compile the Model**:
  - **Optimizer**: `Adam` with a learning rate of `0.001` for efficient gradient-based optimization.
  - **Loss Function**: `binary_crossentropy`, used for binary classification tasks.
  - **Metrics**: Tracks accuracy during training and evaluation.
- **Purpose**: Prepares the model for training by defining the optimization and evaluation strategies.

```python
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Train the model with early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, verbose=1)

# Train the model with validation split, early stopping, and learning rate scheduler
history = model.fit(X_train, y_train,               # Training data
                    epochs=15,                      # Number of epochs
                    batch_size=32,                  # Batch size
                    validation_split=0.4,           # Validation split (40% of training data)
                    callbacks=[early_stopping, lr_scheduler],
                    verbose=1)                      # Verbosity mode

# Display the model summary
print(model.summary())
```

```
Epoch 1/15
23/23 ━━━━━━━━━━━━━━━━━━━━ 14s 439ms/step - accuracy: 0.8906 - loss: 0.3097 - val_accuracy: 0.7729 - val_loss: 0.4833 - learning_rate: 0.0010
Epoch 2/15
23/23 ━━━━━━━━━━━━━━━━━━━━ 10s 448ms/step - accuracy: 0.9618 - loss: 0.1581 - val_accuracy: 0.8021 - val_loss: 0.4788 - learning_rate: 0.0010
Epoch 3/15
23/23 ━━━━━━━━━━━━━━━━━━━━ 10s 448ms/step - accuracy: 0.9868 - loss: 0.0922 - val_accuracy: 0.8062 - val_loss: 0.5250 - learning_rate: 0.0010
Epoch 4/15
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 414ms/step - accuracy: 0.9967 - loss: 0.0386
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
23/23 ━━━━━━━━━━━━━━━━━━━━ 11s 476ms/step - accuracy: 0.9966 - loss: 0.0386 - val_accuracy: 0.8083 - val_loss: 0.5945 - learning_rate: 0.0010
Epoch 5/15
23/23 ━━━━━━━━━━━━━━━━━━━━ 10s 452ms/step - accuracy: 0.9985 - loss: 0.0221 - val_accuracy: 0.8146 - val_loss: 0.5929 - learning_rate: 2.0000e-04

Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 500, 128)	2,198,784
bidirectional (Bidirectional)	(None, 128)	98,816
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129

```
Total params: 6,893,189 (26.30 MB)

Trainable params: 2,297,729 (8.77 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 4,595,460 (17.53 MB)

None
```

- **Training**:
  - `EarlyStopping`: Stops training if validation loss does not improve for 3 epochs and restores the best weights.
  - `ReduceLROnPlateau`: Reduces learning rate after 2 epochs of no improvement in validation loss.
  - Model trains for up to 15 epochs with a batch size of 32, but stops early due to callbacks.
- **Training Metrics**:
  - Accuracy improves steadily, reaching `99.85%` on training data.
  - Validation accuracy reaches `81.46%` after early stopping.
  - Learning rate is reduced during training to fine-tune performance.
- **Model Summary**:
  - **Embedding Layer**: Maps input into a dense vector space with `2,198,784` parameters.
  - **Bidirectional LSTM**: Processes sequences in both forward and backward directions with `98,816` parameters.
  - **Dropout Layer**: Adds no parameters; used for regularization.
  - **Dense Output Layer**: Final layer with 1 neuron for binary classification.
- **Model Details**:
  - **Total Parameters**: `6,893,189`.
  - **Trainable Parameters**: `2,297,729`.
  - **Optimizer Parameters**: `4,595,460`.
- **Purpose**: Achieves high accuracy while preventing overfitting through callbacks and regularization.

Evaluate the model

```python
# Evaluate on test data
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)

# Predict on test data
y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)  # Convert probabilities to binary labels

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)

# Display metrics
print(f"Test Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

[30]

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 83ms/step
Test Accuracy: 0.84
Precision: 0.84
Recall: 0.85
F1-Score: 0.85
```

- **Model Evaluation**:
  - Evaluates the trained model on test data to calculate **loss** and **accuracy**.
  - Test accuracy: `84%`.
- **Predictions**:
  - Converts predicted probabilities to binary labels (`0` or `1`) using a threshold of `0.5`.
- **Performance Metrics**:
  - **Accuracy**: `84%` - Overall correctness of predictions.
  - **Precision**: `84%` - Proportion of true positives among predicted positives.
  - **Recall**: `85%` - Proportion of true positives identified correctly.
  - **F1-Score**: `85%` - Balance between precision and recall.
- **Purpose**: Confirms the model's effectiveness in classifying sentiments with good accuracy and balanced precision/recall.