.

23MSD7018 - SUMEDHA

Named Entity Recognition (NER) using Bi-LSTM

In [16]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

In [17]:
```
data = pd.read_csv(r'C:\Users\USER\Downloads\archive (12)\ner_dataset.csv', encoding= 'unicode_escape')
data.head()
```

Out[17]:

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O

In [18]:

```python
# Display basic information about the dataset
print(data.info())

# Check for missing values
print(data.isnull().sum())

# Distribution of IOB tags
print(data['Tag'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 4 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   Sentence #  47959 non-null    object
 1   Word        1048575 non-null  object
 2   POS         1048575 non-null  object
 3   Tag         1048575 non-null  object
dtypes: object(4)
memory usage: 32.0+ MB
None
Sentence #    1000616
Word                0
POS                 0
Tag                 0
dtype: int64
O        887908
B-geo     37644
B-tim     20333
B-org     20143
I-per     17251
B-per     16990
I-org     16784
B-gpe     15870
I-geo      7414
I-tim      6528
B-art       402
B-eve       308
I-art       297
I-eve       253
B-nat       201
I-gpe       198
I-nat        51
Name: Tag, dtype: int64
```

In [19]:

```
from sklearn.preprocessing import LabelEncoder

# Fill missing values in 'Sentence #' column
data['Sentence #'] = data['Sentence #'].ffill()

# Initialize label encoders for Word, POS, and Tag
word_encoder = LabelEncoder()
pos_encoder = LabelEncoder()
tag_encoder = LabelEncoder()

# Encode columns
data['Word'] = word_encoder.fit_transform(data['Word'])
data['POS'] = pos_encoder.fit_transform(data['POS'])
data['Tag'] = tag_encoder.fit_transform(data['Tag'])
```

In [20]:

```
# Group words, POS tags, and tags by sentence
sentences = data.groupby('Sentence #')['Word'].apply(list).values
pos_tags = data.groupby('Sentence #')['POS'].apply(list).values
tags = data.groupby('Sentence #')['Tag'].apply(list).values

# Pad sequences to handle variable-length sentences
MAX_LEN = 100  # Set a maximum length based on the distribution of sentence lengths
X_words = pad_sequences(sentences, maxlen=MAX_LEN, padding='post')
X_pos = pad_sequences(pos_tags, maxlen=MAX_LEN, padding='post')
y = pad_sequences(tags, maxlen=MAX_LEN, padding='post')
y = [to_categorical(i, num_classes=len(tag_encoder.classes_)) for i in y]
```

Feature Engineering

In [21]:

```python
from tensorflow.keras.layers import Input, Embedding, Concatenate, Bidirectional, LSTM, Dense, TimeDistributed
from tensorflow.keras.models import Model

# Define input layers for words and POS tags
input_words = Input(shape=(MAX_LEN,), dtype='int32')
input_pos = Input(shape=(MAX_LEN,), dtype='int32')

# Embedding layers for words and POS tags
embedding_dim = 50  # Set based on your chosen embedding
vocab_size_words = len(word_encoder.classes_)
vocab_size_pos = len(pos_encoder.classes_)

word_embedding = Embedding(input_dim=vocab_size_words, output_dim=embedding_dim, input_length=MAX_LEN)(input_words
pos_embedding = Embedding(input_dim=vocab_size_pos, output_dim=embedding_dim, input_length=MAX_LEN)(input_pos)

# Concatenate word and POS embeddings
merged_embeddings = Concatenate()([word_embedding, pos_embedding])
```

```
C:\Users\USER\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_len
gth` is deprecated. Just remove it.
  warnings.warn(
```

Model Design(Bi-LSTM Model)

In [22]:
```
# Bi-LSTM layer to capture context
bi_lstm = Bidirectional(LSTM(units=64, return_sequences=True, recurrent_dropout=0.1))(merged_embeddings)

# Output layer with softmax activation for multi-class classification
output = TimeDistributed(Dense(len(tag_encoder.classes_), activation='softmax'))(bi_lstm)

# Define the model
model = Model(inputs=[input_words, input_pos], outputs=output)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100)	0	-
input_layer_1 (InputLayer)	(None, 100)	0	-
embedding_2 (Embedding)	(None, 100, 50)	1,758,900	input_layer[0][0]
embedding_3 (Embedding)	(None, 100, 50)	2,100	input_layer_1[0][0]
concatenate_2 (Concatenate)	(None, 100, 100)	0	embedding_2[0][0], embedding_3[0][0]
bidirectional_1 (Bidirectional)	(None, 100, 128)	84,480	concatenate_2[0][0]
time_distributed_1 (TimeDistributed)	(None, 100, 17)	2,193	bidirectional_1[0][0]

Total params: 1,847,673 (7.05 MB)

Trainable params: 1,847,673 (7.05 MB)

Non-trainable params: 0 (0.00 B)

Training the Model

In [23]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets
X_words_train, X_words_val, X_pos_train, X_pos_val, y_train, y_val = train_test_split(
    X_words, X_pos, y, test_size=0.2, random_state=42
)

# Train the model
history = model.fit(
    [X_words_train, X_pos_train], np.array(y_train),
    validation_data=([X_words_val, X_pos_val], np.array(y_val)),
    batch_size=32,
    epochs=5
)
```

```
Epoch 1/5

C:\Users\USER\anaconda3\Lib\site-packages\keras\src\models\functional.py:225: UserWarning: The structure of `inpu
ts` doesn't match the expected structure: ['keras_tensor_13', 'keras_tensor_14']. Received: the structure of inpu
ts=('*', '*')
  warnings.warn(
```

```
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 170s 117ms/step - accuracy: 0.9613 - loss: 0.1700 - val_accuracy: 0.9925 - val_los
s: 0.0268
Epoch 2/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 142s 118ms/step - accuracy: 0.9936 - loss: 0.0223 - val_accuracy: 0.9937 - val_los
s: 0.0213
Epoch 3/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 140s 117ms/step - accuracy: 0.9950 - loss: 0.0166 - val_accuracy: 0.9939 - val_los
s: 0.0201
Epoch 4/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 144s 120ms/step - accuracy: 0.9957 - loss: 0.0133 - val_accuracy: 0.9937 - val_los
s: 0.0205
Epoch 5/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 145s 121ms/step - accuracy: 0.9962 - loss: 0.0116 - val_accuracy: 0.9940 - val_los
s: 0.0207
```

In [29]:

```
import warnings
warnings.filterwarnings('ignore')
```

Evaluation

In [30]:

```
from sklearn.metrics import classification_report, confusion_matrix

import numpy as np

# Predict on the validation set
y_pred = model.predict([X_words_val, X_pos_val])

# Convert predictions and true labels back to their original tag format

y_pred_classes = np.argmax(y_pred, axis=-1)  # Get the index of the highest probability
y_val_classes = np.argmax(y_val, axis=-1)

# Flatten the lists for sklearn metrics
y_pred_flat = y_pred_classes.flatten()
y_val_flat = y_val_classes.flatten()

# Generate the classification report
print("Classification Report:")
print(classification_report(y_val_flat, y_pred_flat, target_names=tag_encoder.classes_))

# Confusion Matrix Visualization
conf_matrix = confusion_matrix(y_val_flat, y_pred_flat)
```

```
300/300 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step
Classification Report:
              precision    recall  f1-score   support

       B-art       1.00      1.00      1.00    749464
       B-eve       0.63      0.24      0.35        70
       B-geo       0.86      0.92      0.89      7558
       B-gpe       0.98      0.94      0.96      3142
       B-nat       0.33      0.20      0.25        40
       B-org       0.80      0.75      0.77      4151
       B-per       0.85      0.84      0.84      3400
       B-tim       0.93      0.89      0.91      4077
       I-art       0.33      0.01      0.02        84
       I-eve       0.50      0.15      0.24        65
       I-geo       0.83      0.79      0.81      1462
       I-gpe       0.94      0.48      0.64        33
       I-nat       0.00      0.00      0.00        13
       I-org       0.83      0.79      0.81      3394
       I-per       0.88      0.88      0.88      3406
       I-tim       0.87      0.75      0.80      1251
           O       0.99      1.00      0.99    177590

    accuracy                           0.99    959200
   macro avg       0.74      0.62      0.66    959200
weighted avg       0.99      0.99      0.99    959200
```

Model without POS Tags:

To evaluate the model without POS tags, we can modify the input data to include only the word sequences (X_words)

In [35]:

```
# Use only words as input features (no POS tags)
X_words_no_pos = X_words  # Same word sequences without POS tags
```

In [36]:

```python
# Split the data into training and validation sets (without POS tags)
X_words_train_no_pos, X_words_val_no_pos, y_train_no_pos, y_val_no_pos = train_test_split(
    X_words_no_pos, y, test_size=0.2, random_state=42
)
```

Train Model Without POS Tags

In [37]:

```
# Define input layers for words only (no POS input)
input_words_no_pos = Input(shape=(MAX_LEN,), dtype='int32')

# Word embedding layer
word_embedding_no_pos = Embedding(input_dim=vocab_size_words, output_dim=embedding_dim, input_length=MAX_LEN)(inpu

# Bi-LSTM layer to capture context
bi_lstm_no_pos = Bidirectional(LSTM(units=64, return_sequences=True, recurrent_dropout=0.1))(word_embedding_no_pos

# Output layer with softmax activation for multi-class classification
output_no_pos = TimeDistributed(Dense(len(tag_encoder.classes_), activation='softmax'))(bi_lstm_no_pos)

# Define the model (without POS tags)
model_no_pos = Model(inputs=input_words_no_pos, outputs=output_no_pos)
model_no_pos.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model_no_pos.summary()

# Train the model (without POS tags)
history_no_pos = model_no_pos.fit(
    X_words_train_no_pos, np.array(y_train_no_pos),
    validation_data=(X_words_val_no_pos, np.array(y_val_no_pos)),
    batch_size=32,
    epochs=5
)
```

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 100)	0
embedding_5 (Embedding)	(None, 100, 50)	1,758,900
bidirectional_3 (Bidirectional)	(None, 100, 128)	58,880
time_distributed_3 (TimeDistributed)	(None, 100, 17)	2,193

Total params: 1,819,973 (6.94 MB)

Trainable params: 1,819,973 (6.94 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 216s 109ms/step - accuracy: 0.9508 - loss: 0.2214 - val_accuracy: 0.9908 - val_los
s: 0.0326
Epoch 2/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 130s 108ms/step - accuracy: 0.9922 - loss: 0.0267 - val_accuracy: 0.9927 - val_los
s: 0.0248
Epoch 3/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 132s 110ms/step - accuracy: 0.9945 - loss: 0.0183 - val_accuracy: 0.9932 - val_los
s: 0.0229
Epoch 4/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 159s 132ms/step - accuracy: 0.9954 - loss: 0.0145 - val_accuracy: 0.9932 - val_los
s: 0.0229
Epoch 5/5
1199/1199 ━━━━━━━━━━━━━━━━━━━━ 133s 111ms/step - accuracy: 0.9961 - loss: 0.0122 - val_accuracy: 0.9931 - val_los
s: 0.0240
```

Predict and Evaluate Without POS Tags

In [38]:

```
# Predict on the validation set (without POS tags)
y_pred_no_pos = model_no_pos.predict(X_words_val_no_pos)

# Convert predictions and true labels back to their original tag format
y_pred_classes_no_pos = np.argmax(y_pred_no_pos, axis=-1)
y_val_classes_no_pos = np.argmax(y_val_no_pos, axis=-1)

# Flatten the lists for sklearn metrics
y_pred_flat_no_pos = y_pred_classes_no_pos.flatten()
y_val_flat_no_pos = y_val_classes_no_pos.flatten()

# Generate the classification report for the model without POS tags
print("Classification Report (Without POS tags):")
print(classification_report(y_val_flat_no_pos, y_pred_flat_no_pos, target_names=tag_encoder.classes_))
```

```
300/300 ━━━━━━━━━━━━━━━━━━━━ 11s 30ms/step
Classification Report (Without POS tags):
              precision    recall  f1-score   support

       B-art       1.00      1.00      1.00    749464
       B-eve       0.65      0.24      0.35        70
       B-geo       0.86      0.89      0.88      7558
       B-gpe       0.96      0.94      0.95      3142
       B-nat       0.80      0.10      0.18        40
       B-org       0.82      0.69      0.75      4151
       B-per       0.84      0.81      0.83      3400
       B-tim       0.92      0.88      0.90      4077
       I-art       0.00      0.00      0.00        84
       I-eve       0.25      0.05      0.08        65
       I-geo       0.83      0.75      0.79      1462
       I-gpe       0.94      0.52      0.67        33
       I-nat       0.00      0.00      0.00        13
       I-org       0.81      0.77      0.79      3394
       I-per       0.88      0.83      0.86      3406
       I-tim       0.81      0.78      0.80      1251
           O       0.99      0.99      0.99    177590

    accuracy                           0.99    959200
   macro avg       0.73      0.60      0.64    959200
weighted avg       0.99      0.99      0.99    959200
```

Compare Performance with and without POS Tags

Overall Accuracy remains unaffected by the inclusion of POS tags. Precision slightly improves for some tags (like B-gpe), but decreases for others (like B-nat). Recall is generally improved for some tags, such as B-org and I-per, but worsens for others like I-gpe. F1-score sees some improvements with POS tags, particularly in cases where both precision and recall benefit from POS tags (like B-gpe, B-per, and B-tim). Macro and Weighted averages show slight improvements in macro metrics, while the weighted metrics remain stable.

REPORT

1. Data Preprocessing Steps: The dataset used for this Named Entity Recognition (NER) task is the Kaggle Annotated Corpus for NER, containing the columns Sentence #, Word, POS (Part-of-Speech), and Tag (the entity label).

Handling Missing Data: The dataset consists of 1,048,575 records, some of which have missing values in the Sentence # column. These missing values were imputed using the ffill() method to ensure that words are correctly grouped under their respective sentences.

Encoding: The Word, POS, and Tag columns are categorical. These were encoded using Scikit-learn's LabelEncoder. The Word and POS columns were converted into integer sequences to be used as input features, while the Tag column was transformed into integer labels corresponding to the named entities.

Grouping by Sentences: The data is organized by sentence, where each sentence contains multiple words. We grouped the data by the Sentence # and converted the words, POS tags, and entity labels into lists for each sentence.

Padding Sequences: Since sentences vary in length, we padded them to a consistent length of 100 words (MAX_LEN) using Keras' pad_sequences. This ensures that all input sequences have the same length for batch processing in neural networks.

One-Hot Encoding of Tags: After encoding the Tag column, it was converted into a one-hot encoded format using Keras' to_categorical, suitable for multi-class classification.

Model Architecture and Hyperparameters: The model is built using a Bi-directional Long Short-Term Memory (Bi-LSTM) network, a type of Recurrent Neural Network (RNN) commonly used for sequence labeling tasks like NER. Below are the details of the model architecture:

Input Layers: Two input layers were defined—one for the word sequences (input_words) and another for the POS tag sequences (input_pos). Both inputs have the shape (MAX_LEN,) and are integer-encoded.

Embedding Layers: Both the word and POS inputs were passed through Embedding layers to map each word and POS tag into dense vector representations. The embedding dimension was set to 50, and the vocabulary size was based on the number of unique words and POS tags.

Concatenation Layer: The word and POS embeddings were concatenated into a single tensor, combining both feature types.

Bi-LSTM Layer: A Bidirectional LSTM layer was added to capture contextual information from both the past and the future words in the sentence, enhancing the model's ability to understand word meaning in context. The LSTM layer has 64 units, and a recurrent dropout of 0.1 was applied to prevent overfitting.

Output Layer: A TimeDistributed Dense layer was used as the output, predicting the entity label for each word in the sentence. The softmax activation function was used to handle multi-class classification.

Hyperparameters:

Batch size: 32 Epochs: 5 Optimizer: Adam Loss function: Categorical Cross-Entropy Performance Metrics and Error Analysis: The model was trained on the preprocessed data for 5 epochs, and its performance was evaluated using standard classification metrics such as accuracy, precision, recall, and F1-score.

Training Results: Training accuracy: 99.62% Validation accuracy: 99.40% Training loss: 0.0116 Validation loss: 0.0207 The model demonstrates strong performance on the validation set, achieving high accuracy and low loss after 5 epochs. However, performance varies across different entity types.

Classification Report: The model performs well for entity types like B-per, I-per, B-org, and O, with high precision, recall, and F1-scores close to 1.0. However, some entity types like B-eve, I-art, and I-nat show lower recall and F1-scores. This suggests that the model struggles more with rarer or less frequent entity classes, possibly due to dataset imbalance.

Error Analysis: The model's recall for certain entities, such as I-art (F1-score of 0.02), is low, indicating that it does not effectively capture less frequent entities. This is a common challenge in imbalanced datasets. The confusion matrix also supports this, showing that certain labels (like I-art) are predicted less accurately compared to others like O (non-entity), which has high accuracy. Techniques like class weighting, data augmentation for rare entities, or further model fine-tuning could help improve the model's performance for these underrepresented classes.

Evaluation of Model without POS Tags: To assess the impact of POS tags, we trained a variant of the model without POS tags, using only word sequences as input. The model's accuracy and performance were slightly lower without the POS tags, indicating that POS information contributes

In [ ]:

```

```