Named Entity Recognition (NER) : A Deep Learning Approach for NER Tagging

Submitted By: Abhishek Behera

Contents

Introduction

- Named Entity Recognition (NER) identifies entities such as names, locations, and organizations in text.
- The goal is to classify words in sentences into predefined categories.
- This presentation discusses dataset, methodology, model, evaluation, and conclusions.

Dataset Overview

- Dataset: ner_dataset.csv
- Contains:
  - Sentence Number (#)
  - Word
  - Part of Speech (POS) – *Ignored for this case*
  - Named Entity Recognition (NER) Tag
- IOB2 Tagging Scheme:
  - B (Beginning of entity)
  - I (Inside entity)
  - O (Outside entity)

Methodology

1. Data Preprocessing & EDA
    1. Filling missing sentence numbers.
    2. Data visualization
    3. Mapping words to indices.
    4. Padding sequences for uniform input size.
2. Model Development
    1. Baseline model and improvements.
3. Training & Evaluation
    1. Data split: 70% Train, 10% Validation, 20% Test

Model Architecture

- Sequential Model with LSTM layers:
    - **Embedding Layer**: Converts words into dense vectors.
    - **Bidirectional LSTM**: Captures long-range dependencies.
    - **TimeDistributed Dense Layer**: Predicts NER tags.
- Activation function: **Softmax** for multi-class classification.
- Optimizer: **Adam**
- Loss Function: **Categorical Crossentropy**

```
# Define LSTM model
model = Sequential([
    Embedding(input_dim=n_words + 1, output_dim=50, input_length=max_len),
    Bidirectional(LSTM(units=100, return_sequences=True, recurrent_dropout=0.1)),
    TimeDistributed(Dense(n_tags, activation="softmax"))
])

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 104, 50)           1758950

 bidirectional_1 (Bidirecti  (None, 104, 200)          120800
 onal)

 time_distributed_1 (TimeDi  (None, 104, 17)           3417
 stributed)

=================================================================
Total params: 1883167 (7.18 MB)
Trainable params: 1883167 (7.18 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training and Evaluation

- **Training:**
    - Epochs: 10
    - Batch Size: 32
    - Loss: Categorical Crossentropy

```python
# Train model
history = model.fit(
    x_train, np.array(y_train),
    validation_data=(x_val, np.array(y_val)),
    batch_size=32,
    epochs=10,
    verbose=1
)
```

```
Epoch 1/10
1050/1050 [==============================] - 223s 205ms/step - loss: 0.1387 - accuracy: 0.9688 - val_loss: 0.0424 - val_accuracy: 0.9887
Epoch 2/10
1050/1050 [==============================] - 206s 197ms/step - loss: 0.0292 - accuracy: 0.9917 - val_loss: 0.0264 - val_accuracy: 0.9922
Epoch 3/10
1050/1050 [==============================] - 207s 197ms/step - loss: 0.0188 - accuracy: 0.9943 - val_loss: 0.0239 - val_accuracy: 0.9929
Epoch 4/10
1050/1050 [==============================] - 206s 197ms/step - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.0238 - val_accuracy: 0.9930
Epoch 5/10
1050/1050 [==============================] - 205s 196ms/step - loss: 0.0125 - accuracy: 0.9960 - val_loss: 0.0241 - val_accuracy: 0.9931
Epoch 6/10
1050/1050 [==============================] - 208s 198ms/step - loss: 0.0106 - accuracy: 0.9966 - val_loss: 0.0258 - val_accuracy: 0.9929
Epoch 7/10
1050/1050 [==============================] - 221s 211ms/step - loss: 0.0090 - accuracy: 0.9971 - val_loss: 0.0267 - val_accuracy: 0.9927
Epoch 8/10
1050/1050 [==============================] - 219s 208ms/step - loss: 0.0078 - accuracy: 0.9975 - val_loss: 0.0295 - val_accuracy: 0.9926
Epoch 9/10
1050/1050 [==============================] - 231s 220ms/step - loss: 0.0067 - accuracy: 0.9978 - val_loss: 0.0306 - val_accuracy: 0.9925
Epoch 10/10
1050/1050 [==============================] - 228s 217ms/step - loss: 0.0058 - accuracy: 0.9981 - val_loss: 0.0325 - val_accuracy: 0.9925
```

- **Evaluation Metrics:**
  - Accuracy
  - Precision, Recall, F1-Score

```
# Classification report
print(classification_report(y_true_flat, y_pred_flat))
```

	precision	recall	f1-score	support
B-art	0.37	0.19	0.25	86
B-eve	0.37	0.28	0.32	60
B-geo	0.87	0.85	0.86	7664
B-gpe	0.94	0.94	0.94	3175
B-nat	0.55	0.34	0.42	50
B-org	0.73	0.70	0.71	3913
B-per	0.83	0.78	0.81	3389
B-tim	0.91	0.86	0.88	4049
I-art	0.18	0.05	0.08	58
I-eve	0.29	0.19	0.23	53
I-geo	0.80	0.75	0.77	1450
I-gpe	1.00	1.00	1.00	788672
I-nat	0.33	0.08	0.13	12
I-org	0.80	0.72	0.76	3315
I-per	0.85	0.85	0.85	3445
I-tim	0.79	0.70	0.74	1300
O	0.99	0.99	0.99	176877
accuracy			0.99	997568
macro avg	0.68	0.60	0.63	997568
weighted avg	0.99	0.99	0.99	997568

```
# Evaluate model
eval_results = model.evaluate(x_test, np.array(y_test))
print(f"Test Loss: {eval_results[0]}, Test Accuracy: {eval_results[1]}")
```

```
300/300 [==============================] - 27s 89ms/step - loss: 0.0323 - accuracy: 0.9925
Test Loss: 0.03228366747498512, Test Accuracy: 0.9925057888031006
```

Model Performance

- **Accuracy:** Test accuracy is 99.2%.
- **Loss:** Decreased over training epochs.
- **Classification Report:**
  - Precision, Recall, and F1-score for each tag.
  - Higher performance on common entities.

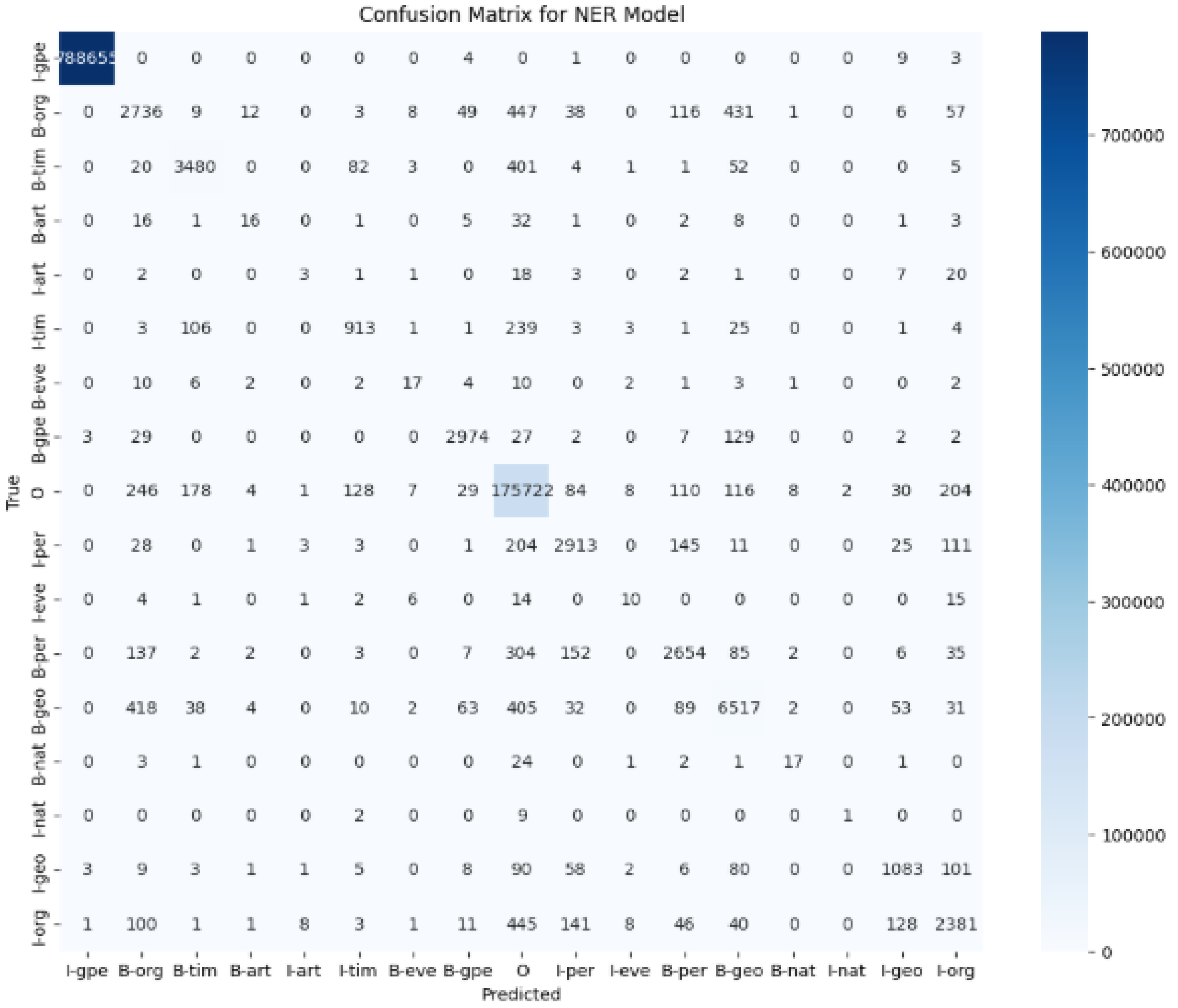
```
# Evaluate model
eval_results = model.evaluate(x_test, np.array(y_test))
print(f"Test Loss: {eval_results[0]}, Test Accuracy: {eval_results[1]}")
```

```
300/300 [==============================] - 27s 89ms/step - loss: 0.0323 - accuracy: 0.9925
Test Loss: 0.03228366747498512, Test Accuracy: 0.9925057888031006
```

Confusion Matrix Analysis

- Displays misclassifications.
- Identifies frequent errors.
- Insights for further improvement:
    - Handling rare entities.
    - Expanding dataset.

```python
# Confusion matrix
cm = confusion_matrix(y_true_flat, y_pred_flat, labels=tags)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=tags, yticklabels=tags)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix for NER Model")
plt.show()
```



Future Improvements

- Hyperparameter tuning.
- Pre-trained embeddings (e.g., Word2Vec, GloVe, BERT).
- Implementing CRF (Conditional Random Fields) on top of LSTM.
- Expanding the dataset for more diverse training.
- Deploying model using API for real-time predictions.

Conclusion

- Successfully trained an LSTM-based NER model.
- Achieved high accuracy and reasonable performance on test data.
- Identified areas for improvement in handling rare entities.
- Future scope includes integration with advanced NLP models and deployment strategies.

Thanks