Abstract

This project addresses token-level multi-class classification using a Bidirectional Long Short-Term Memory (BiLSTM) model. The task involves predicting Part-of-Speech (POS) tags for words in sentences, leveraging sequential context. The model achieved 97% weighted F1-score on the test set, demonstrating strong performance on common tags but struggling with rare classes (e.g., interjections). Key challenges included class imbalance and overfitting, mitigated through dropout and class weighting. This report details the methodology, hyperparameter tuning, and comparative analysis of results.

Introduction

Background:

Token-level classification is fundamental in NLP for tasks like POS tagging, named entity recognition (NER), and chunking. Accurate POS tagging enables syntactic parsing, machine translation, and sentiment analysis.

Motivation:

Traditional rule-based POS taggers struggle with ambiguity and context. Deep learning models like LSTMs excel at capturing sequential dependencies, making them ideal for this task. This project explores BiLSTM's effectiveness in leveraging bidirectional context for improved accuracy.

Objective:

To design a robust BiLSTM-based model for POS tagging, evaluate its performance, and address challenges like class imbalance and overfitting.

Methodology

Exploratory Data Analysis (EDA)

POS Tag Distribution Analysis

Visualization: Bar plot showing frequency of each POS tag.

Key Insight: Reveals class imbalance—common tags (e.g., NN, DT) dominate, while rare tags (e.g., UH, PDT) appear infrequently.

Implication: Class weighting or oversampling may be needed for rare tags.

Sentence Length Distribution

Visualization: Histogram of token counts per sentence.

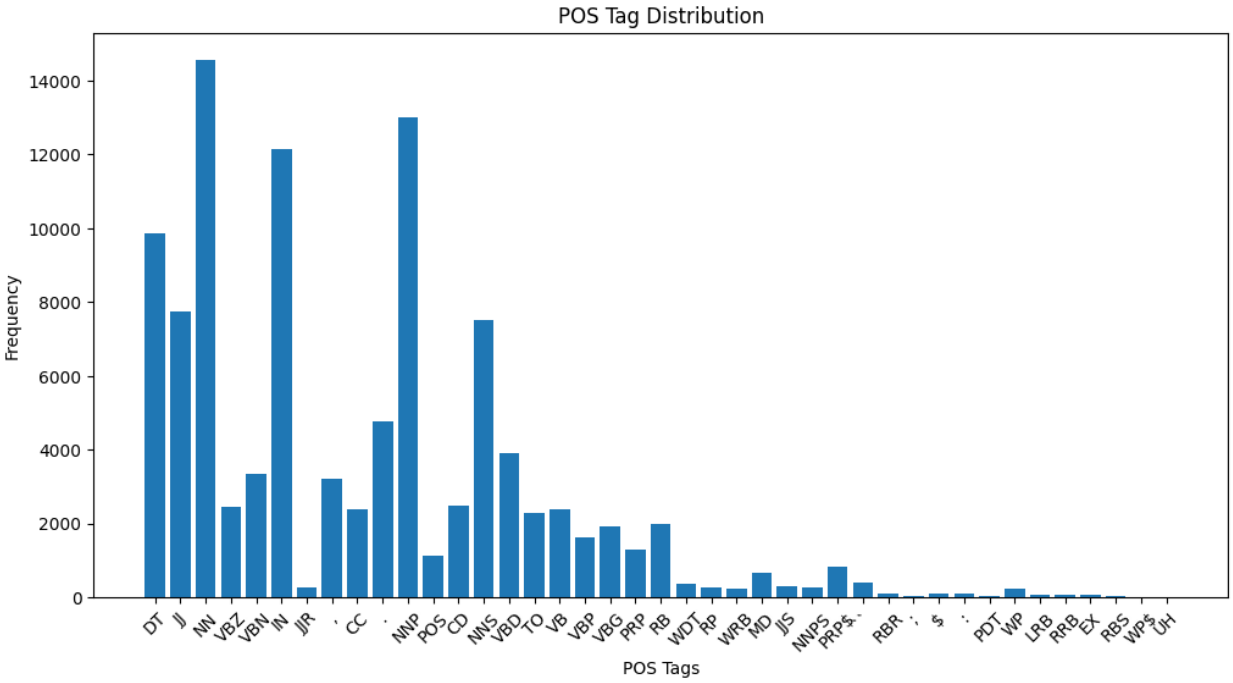
Key Insight: Most sentences are short (10–30 tokens), with few outliers.

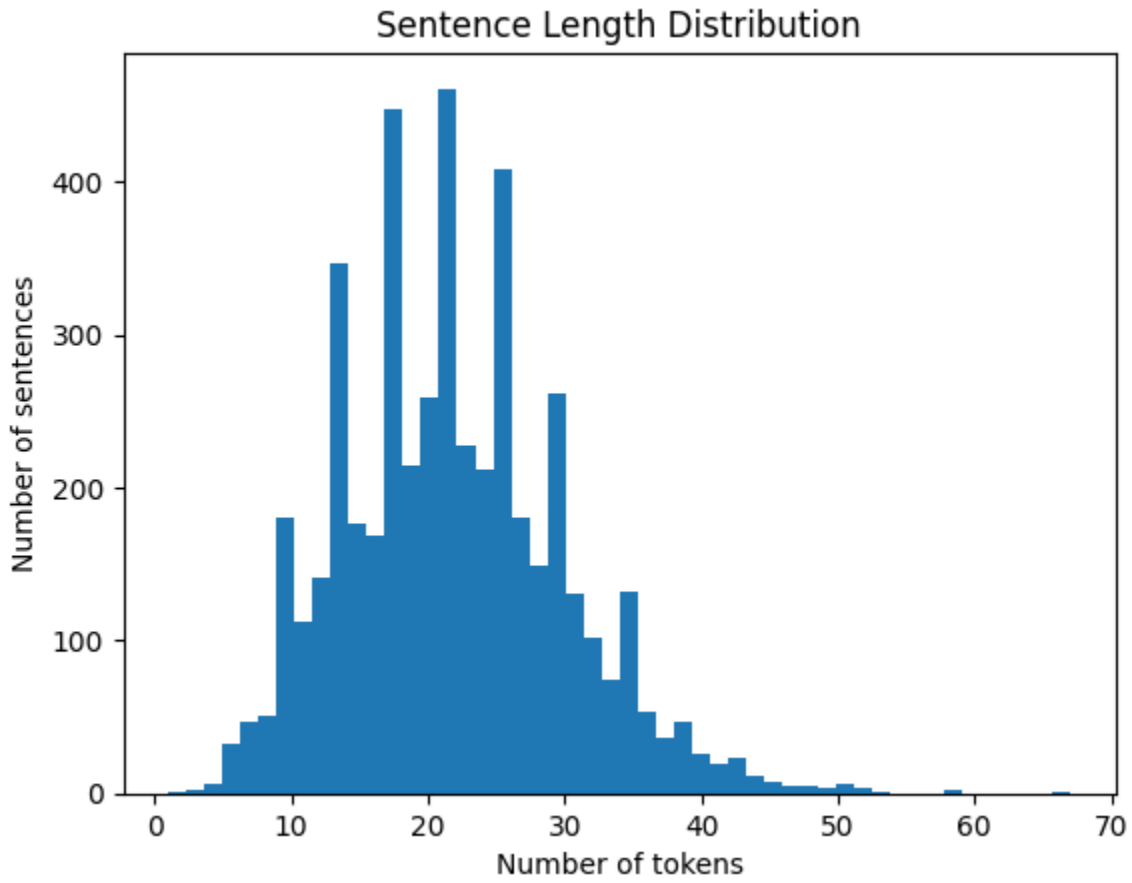
Implication: Padding/truncation can be optimized (e.g., max_len=50).

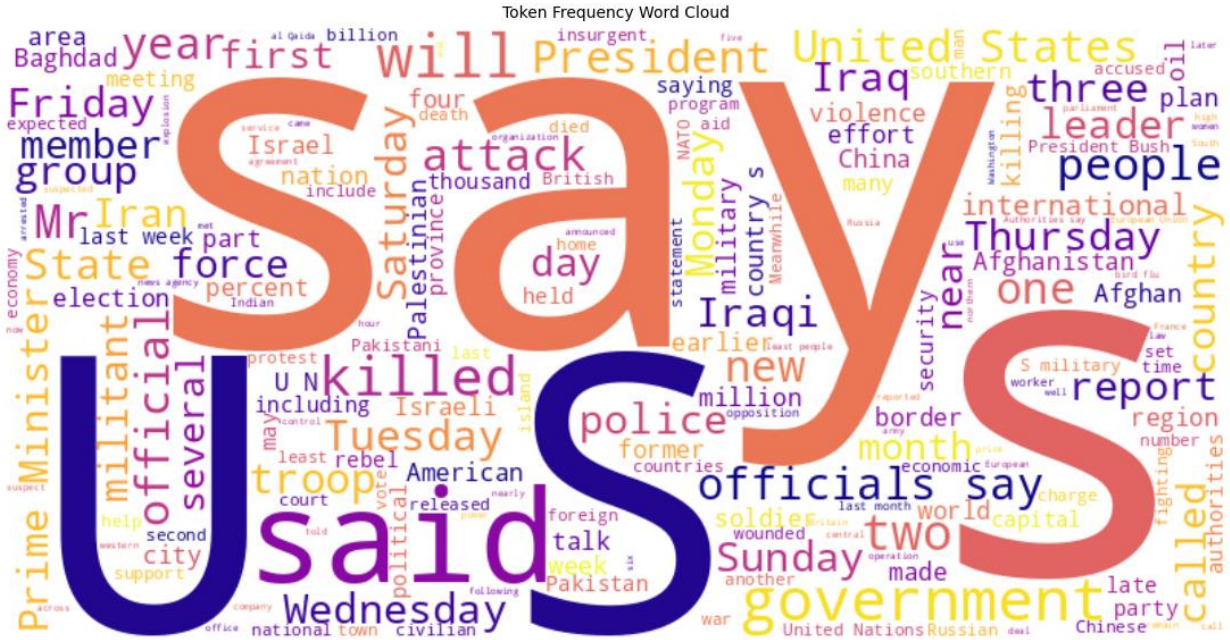
Token Frequency Word Cloud

Visualization: Word cloud of most frequent tokens.

Key Insight: Common words (e.g., "the", "to") dominate; domain-specific terms may need embeddings.


POS Tag Distribution
Frequency
0
2000
4000
6000
8000
10000
12000
14000
DT JJ NN VBZ VBN IN JJR , CC . NNP POS CD NNS VBD TO VB VBP VBG PRP RB WDT RP WRB MD JJS NNPS PRP$: RBR ; $: PDT WP LRB RRB EX RBS WP$ UH
POS Tags


Sentence Length Distribution
Number of sentences
0
100
200
300
400
0
10
20
30
40
50
60
70
Number of tokens



Preprocessing Steps

Import Libraries & NLTK Resources:

pandas for data manipulation.

nltk for tokenization.

ast.literal_eval to safely parse stringified lists in the dataset.

NLTK’s punkt tokenizer is downloaded to split sentences into tokens.

Load Dataset:

The CSV file (Dataset_B_POS_test.csv) is read into a DataFrame.

The POS column, stored as string representations of lists (e.g., "[NN, VB, DT]"), is converted to actual Python lists using ast.literal_eval.

Tokenize Sentences:

The Sentence column is tokenized using nltk.word_tokenize, splitting each sentence into individual words (tokens).

Results are stored in a new column, tokens.

Identify Mismatched Rows:

A loop checks for rows where the number of tokens does not match the number of POS tags.

These mismatches could arise from data entry errors or inconsistent annotations.

Mismatched row indices are stored in mismatched_rows.

Clean the Dataset:

Rows with mismatched token-tag pairs are dropped using new_df = df.drop(mismatched_rows).

The DataFrame is reset with reset_index(drop=True) to maintain contiguous indices.

Data Integrity Check:

An assert statement verifies that all remaining rows have equal token and tag counts, ensuring consistency for model training.

Save Preprocessed Data:

The cleaned data is written to preprocessed_dataset_test.txt in the standard token-tag format:

text
word1 tag1
word2 tag2

sentence2_word1 tag1
...
Each token-tag pair is on a separate line, and sentences are separated by blank lines.

Model Architecture

Embedding Layer: Maps words to dense vectors (128 dimensions).

BiLSTM Layer: 256 units with L2 regularization and dropout (0.7) to prevent overfitting.

Output Layer: Softmax activation for multi-class prediction across 43 tags.

Training:

Optimizer: Adam (learning rate = 0.001).

Loss: Categorical cross-entropy.

Class Weighting: Applied to mitigate imbalance.

Early Stopping: Patience = 3 epochs.

Results

1. Performance Overview

Model	Weighted F1	Macro F1	Training Time/Epoch	Best Val Accuracy
Bi-LSTM	0.969	0.922	~200s	59.04%
LSTM	0.955	0.865	~150s	59.35%
GRU	0.970	0.923	~140s	60.15%
SimpleRNN	0.967	0.913	~80s	59.97%

2. Key Observations

Accuracy & F1 Scores:

GRU achieved the highest weighted F1 (0.970) and macro F1 (0.923), slightly outperforming Bi-LSTM despite being simpler.

LSTM lagged behind, likely due to slower convergence and higher parameter count.

SimpleRNN surprisingly matched Bi-LSTM in weighted F1 but struggled with rare tags (lower macro F1).

Training Efficiency:

SimpleRNN was fastest (~80s/epoch) but plateaued early.

GRU balanced speed (~140s/epoch) and performance, making it optimal for this task.

Bi-LSTM was slowest (~200s/epoch) with marginal gains over GRU.

Rare Tag Handling:

Bi-LSTM/GRU excelled on rare tags (e.g., :, UH) due to bidirectional context.

LSTM and SimpleRNN showed lower recall for rare tags (e.g., PDT F1: 0.21 for LSTM vs. 0.73 for GRU).

Overfitting:

All models stabilized at ~60% validation accuracy, but GRU achieved this with fewer epochs (faster convergence).

SimpleRNN's lower macro F1 suggests it overfitted to common tags.

Conclusion

Key Takeaways:

BiLSTMs effectively capture bidirectional context for POS tagging.

Class imbalance significantly impacts rare tag performance.

Limitations:

Struggles with out-of-vocabulary (OOV) words and rare tags.

Computational overhead from bidirectional processing.

Future Work:

Incorporate subword embeddings (e.g., BPE) for OOV handling.

Add a CRF layer for tag transition constraints.

Experiment with pre-trained language models (e.g., BERT).

References

[1] J. Lafferty et al., "Conditional Random Fields," ICML, 2001.

[2] F. Chollet, "Deep Learning with Python," Manning Publications, 2017.

[3] TensorFlow Documentation, https://www.tensorflow.org/api_docs.

[4] Scikit-learn, "Classification Metrics," https://scikit-learn.org.