Bidirectional LSTM Classifier for Political-Party Text Attribution

Abstract

This report documents a neural text classification pipeline for attributing short political texts to one of two party classes. The approach combines subword tokenization, a trainable embedding layer, a bidirectional LSTM encoder, masked mean pooling, and a linear classification head. The model is trained end-to-end using cross-entropy loss and AdamW optimization with mixed-precision GPU acceleration. On a held-out test set, the system reaches an accuracy of 0.8635 at a loss of 0.3307, with stronger performance on the majority class. We provide a formal mathematical specification, implementation details, and an empirical assessment including class-wise metrics and error analysis.

1 Problem setting and data

Let $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$ denote a corpus of texts $x^{(n)}$ and binary labels $y^{(n)} \in \{0, 1\}$ indicating party membership. Raw party strings are normalized into two categories and all instances outside this mapping are removed. Missing text or label entries are discarded. The dataset is divided into training, validation, and test partitions using stratified sampling to preserve class proportions. The test set contains 12,770 instances of class 0 and 20,691 of class 1, yielding prevalences of 0.3816 and 0.6184, respectively. This imbalance motivates careful evaluation beyond overall accuracy.

2 Model specification

2.1 Tokenization and inputs

Each text is mapped to a fixed-length sequence of subword token identifiers using a WordPiece-style tokenizer with vocabulary size V. After padding or truncation to length $T = 128$, each sample is represented by:

$$\mathbf{t} = (t_1, \dots, t_T), \quad t_i \in \{1, \dots, V\},$$

and a binary mask

$$\mathbf{m} = (m_1, \dots, m_T), \quad m_i \in \{0, 1\},$$

where $m_i = 1$ indicates a true token and $m_i = 0$ indicates padding. The tokenizer is used only to obtain indices; semantic representations are learned by the network.

2.2 Trainable embedding layer

Let $E \in \mathbb{R}^{V \times d}$ be the embedding matrix with $d = 200$. Tokens are embedded as

$$\mathbf{x}_i = E[t_i] \in \mathbb{R}^d, \quad \mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_T) \in \mathbb{R}^{T \times d}.$$

The padding index is set so that padded positions do not update embeddings.

2.3 Bidirectional LSTM encoder

A forward LSTM and a backward LSTM encode $\mathbf{X}$. For a single LSTM direction, gates and state updates at timestep i are:

$$\begin{aligned}
\mathbf{i}_i &= \sigma(W_i\mathbf{x}_i + U_i\mathbf{h}_{i-1} + \mathbf{b}_i), && (1)\\
\mathbf{f}_i &= \sigma(W_f\mathbf{x}_i + U_f\mathbf{h}_{i-1} + \mathbf{b}_f), && (2)\\
\mathbf{o}_i &= \sigma(W_o\mathbf{x}_i + U_o\mathbf{h}_{i-1} + \mathbf{b}_o), && (3)\\
\tilde{\mathbf{c}}_i &= \tanh(W_c\mathbf{x}_i + U_c\mathbf{h}_{i-1} + \mathbf{b}_c), && (4)\\
\mathbf{c}_i &= \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \tilde{\mathbf{c}}_i, && (5)\\
\mathbf{h}_i &= \mathbf{o}_i \odot \tanh(\mathbf{c}_i), && (6)
\end{aligned}$$

with sigmoid $\sigma(\cdot)$ and elementwise product $\odot$. The hidden size is $H = 128$ per direction. The bidirectional representation at position i is

$$\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i] \in \mathbb{R}^{2H},$$

yielding a sequence output

$$\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_T) \in \mathbb{R}^{T\times 2H}.$$

2.4 Masked mean pooling

To obtain a fixed-dimensional vector while excluding padding, the model computes:

$$\mathbf{z} = \frac{\sum_{i=1}^{T} m_i\mathbf{h}_i}{\sum_{i=1}^{T} m_i + \varepsilon} \in \mathbb{R}^{2H},$$

where ε is a small constant to avoid division by zero.

2.5 Classification head

Dropout is applied to $\mathbf{z}$ with probability $p = 0.2$:

$$\tilde{\mathbf{z}} = \mathrm{Dropout}(\mathbf{z}).$$

A linear layer produces logits:

$$\mathbf{o} = W\tilde{\mathbf{z}} + \mathbf{b} \in \mathbb{R}^2.$$

The predictive distribution is

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}), \qquad \hat{y} = \arg\max_{k\in\{0,1\}} \hat{\mathbf{y}}_k.$$

2.6 Training objective

Parameters θ are learned by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log \hat{\mathbf{y}}^{(n)}_{y^{(n)}}.$$

Optimization uses AdamW with learning rate 10^{-3}. Mixed-precision training (automatic casting to FP16 with dynamic gradient scaling) is employed to improve throughput and numerical stability on modern GPUs.

2.7 Computational considerations

For each batch, the dominant cost arises from the BiLSTM. For sequence length T and hidden size H, time complexity scales as $\mathcal{O}(T \cdot H^2)$ per direction, and memory scales as $\mathcal{O}(T \cdot H)$ for storing activations. Fixing $T = 128$ stabilizes runtime and enables cuDNN kernel autotuning.

3 Implementation details

3.1 Environment and reproducibility

Training is performed on CUDA hardware with cuDNN benchmarking activated, selecting optimized kernels for the fixed tensor shapes. Random seeds are set for Python, NumPy, and PyTorch (CPU and GPU) to reduce run-to-run variation.

3.2 Data pipeline

The pipeline proceeds as:

1. Read CSV and keep relevant columns (text and party label).
2. Remove missing rows and normalize party strings.
3. Map party strings into binary labels and filter all others.
4. Stratified splitting into train/validation/test (80/10/10).
5. Pre-tokenize each split to input IDs and masks.

Pre-tokenization on CPU in large minibatches amortizes tokenizer overhead and prevents GPU under-utilization during training.

3.3 Training loop

The training routine iterates for 10 epochs. Each epoch consists of:

1. Forward pass with autocasting.
2. Cross-entropy loss computation.
3. Backward pass on scaled loss.
4. AdamW parameter update.

Model selection is based on validation accuracy; the best validation checkpoint is stored and reloaded before final test evaluation. DataLoaders use batch size 256 with pinned memory and persistent workers for efficient host-device transfer.

4 Empirical assessment

4.1 Headline results

On the test partition, the model achieves:

$$\text{Test loss} = 0.3307, \qquad \text{Test accuracy} = 0.8635.$$

The closeness of validation-selected performance and test performance indicates stable generalization without severe overfitting.

Table 1: Test-set classification report.

Class	Precision	Recall	F_1	Support
0	0.8347	0.8008	0.8174	12770
1	0.8801	0.9021	0.8910	20691
Accuracy		0.8635		33461
Macro avg	0.8574	0.8515	0.8542	33461
Weighted avg	0.8627	0.8635	0.8629	33461

4.2 Class-wise metrics

Table 1 reports precision, recall, and F_1 by class.

Performance is stronger for class 1, consistent with its higher prevalence. The macro-averaged F_1 is slightly lower than the weighted average, reflecting imbalance-driven asymmetry.

4.3 Confusion matrix and derived rates

The confusion matrix (rows: true, columns: predicted) is:

$$\mathbf{C} = \begin{pmatrix} 10226 & 2544 \\ 2025 & 18666 \end{pmatrix}.$$

Key derived quantities (treating class 1 as positive) are:

$$\begin{aligned}
\text{TPR (recall}_1) &= \frac{18666}{18666 + 2025} = 0.9021, \\
\text{TNR (specificity)} &= \frac{10226}{10226 + 2544} = 0.8008, \\
\text{FPR} &= \frac{2544}{10226 + 2544} = 0.1992, \\
\text{FNR} &= \frac{2025}{18666 + 2025} = 0.0979, \\
\text{Balanced accuracy} &= \frac{0.9021 + 0.8008}{2} = 0.8515, \\
\text{MCC} &= 0.7088.
\end{aligned}$$

The false-positive rate is about twice the false-negative rate, showing that the classifier is more prone to labeling class-0 items as class 1 than vice versa. In applications where mislabeling class 0 has higher cost, this skew may require threshold adjustment or reweighting.

4.4 Error profile

Inspection of the confusion matrix suggests two dominant error types:

1. **Class-0 $\rightarrow$ Class-1 errors (2544 cases):** likely driven by lexical overlap or stylistic convergence in the two parties' rhetoric, aggravated by class imbalance.

2. **Class-1 $\rightarrow$ Class-0 errors (2025 cases):** fewer and may correspond to moderate or cross-partisan language patterns.

Given that embeddings are learned from scratch, the model relies heavily on dataset-specific word usage rather than external semantic priors, which can increase confusion in stylistically similar texts.

4.5 Comparative perspective

A BiLSTM with masked pooling is a strong sequential baseline. However, architectures initialized with pretrained contextual encoders typically outperform purely task-trained embeddings on political text tasks because they embed broader semantic and syntactic regularities. The present performance indicates that the dataset is large enough to train effective representations, but also hints at a ceiling imposed by the lack of pretrained knowledge.

5 Limitations and possible extensions

5.1 Limitations

1. **No pretrained representations:** learning E from scratch can underutilize semantic information that pretrained models capture.
2. **Fixed truncation length:** texts longer than 128 tokens lose tail content, potentially discarding discriminative cues.
3. **Moderate imbalance:** the decision boundary shifts toward class 1, as seen in higher FPR.
4. **Pooling simplicity:** mean pooling assumes equal importance of token positions and may dilute localized signals.

5.2 Extensions

Several modifications could plausibly improve accuracy or reduce skew:

1. **Class-weighted loss:** set weights w_0, w_1 in cross-entropy to penalize minority errors more.
2. **Pretrained encoder swap:** replace the embedding+BiLSTM block with a frozen or fine-tuned transformer encoder while keeping masked pooling and the head.
3. **Attention pooling:** learn token weights α_i to compute $\mathbf{z} = \sum_i \alpha_i \mathbf{h}_i$ with masking.
4. **Longer or adaptive T:** increase T or bucket by length to preserve more context without undue padding cost.
5. **Calibration and thresholding:** tune the probability cutoff using validation curves to trade off FPR vs. FNR for downstream objectives.

6 Conclusion

The implemented pipeline offers a computationally efficient neural baseline for binary party attribution in political texts. By combining subword token IDs with trainable embeddings and a BiLSTM encoder, the system captures sequential patterns and achieves solid test performance (accuracy 0.8635, MCC 0.7088). Evaluation reveals a mild bias toward predicting the majority class, visible in an elevated false-positive rate for class 0. The model's simplicity, GPU-friendly batching, and stable selection via validation accuracy make it a robust starting point. Future work should focus on incorporating pretrained contextual representations and imbalance-aware objectives to further improve minority-class recall and overall calibration.