

Project Report

**Detecting Subjective Bias in Text Using the Wiki
Neutrality Corpus**

Bahar Hamzehei and Taleb Zarhesh
Course Code: 91268 — Ethics in AI

June, 2025

Abstract

Detecting subjective bias in text is essential for promoting neutral information dissemination and fostering responsible AI deployment. This work presents a comprehensive end-to-end framework for bias detection, leveraging the *Wikipedia Neutrality Corpus* (WNC), which contains aligned pairs of biased and neutralized Wikipedia sentences [2]. Our methodology encompasses thorough data preprocessing and cleaning; the implementation of two lightweight recurrent neural baselines—FastText [3] and BiLSTM; fine-tuning of five transformer-based architectures (BERT-Large [6], ALBERT-XXLarge-v2 [5], DistilBERT [4], DistilRoBERTa [4], and RoBERTa-Large [7]); and ensemble modeling via logit averaging [8]. Experimental results on a held-out test set show that RoBERTa-Large achieves an accuracy of 76.45% and an F1 score of 75.78, while a BERT + RoBERTa ensemble model attains 76.35% accuracy and a 75.76 F1 score. Additionally, we provide a detailed evaluation of precision and recall across all models.

1 Introduction

The proliferation of user-generated content on the web has amplified concerns regarding subtle forms of subjective bias, which can shape public opinion and undermine trust in digital media [1]. As such, the automated detection of linguistic bias is increasingly critical for ethical AI deployment, content moderation, and fact-checking systems. The WNC offers a valuable resource in this context, comprising over 200,000 paired examples of biased Wikipedia sentences and their corresponding neutralized revisions [2]. This corpus enables both supervised classification and neutralization tasks for bias detection.

In this work, we propose a comprehensive end-to-end framework for detecting subjective bias in text using the WNC. The pipeline begins with robust data ingestion and preprocessing routines implemented in `datasets.py`, which yield aligned text-label pairs for training and evaluation. We implement two lightweight recurrent neural baselines—`FastTextClassifier` [3] and `BiLSTMClassifier`—in `models.py`, and fine-tune five transformer-based models (BERT-Large [6], ALBERT-XXLarge-v2 [5], DistilBERT [4], DistilRoBERTa [4], and RoBERTa-Large [7]) using standardized utilities defined in `model_utils.py`. To leverage the complementary strengths of individual models, we further explore ensemble strategies based on logit averaging [8].

We evaluate each approach using accuracy, precision, recall, and F1-score. Our findings indicate that pretrained transformer models substantially outperform recurrent baselines, with RoBERTa-Large achieving the strongest performance among individual models. While ensemble methods based on logit averaging yield F1 scores comparable to the top-performing transformer, they consistently outperform several other transformer variants.

2 Background and Motivation

2.1 Subjective Bias in Natural Language

Subjective bias refers to language that conveys opinions, judgments or attitudes rather than objective facts, and can subtly influence readers’ perceptions [1]. In contexts such as news reporting, encyclopedic writing, and social media, unchecked bias risks distorting information, reinforcing stereotypes, and undermining trust. Automated detection of subjective bias enables content moderation systems and fact-checking pipelines to flag or neutralize biased language, promoting more balanced and reliable information dissemination.

2.2 Computational Approaches to Bias Detection

Early bias-detection methods relied on hand-crafted lexicons or rule-based heuristics to identify opinionated terms [1]. With the advent of machine learning, statistical classifiers (e.g., SVMs, logistic regression) using bag-of-words or n-gram features became common, achieving moderate success but struggling to capture contextual nuances [3]. More recently, neural models employing recurrent architectures (e.g., LSTM, BiLSTM) have demonstrated improved ability to model sequence context and subtle stylistic cues.

2.3 Pretrained Transformer Architectures

Transformer-based language models pretrain on massive text corpora using self-attention mechanisms to learn deep contextual representations [6, 7, 5]. Models such as BERT, RoBERTa, and ALBERT can be fine-tuned on downstream classification tasks, often yielding state-of-the-art performance. Distilled variants (DistilBERT, DistilRoBERTa) offer similar capabilities with reduced parameter counts and faster inference, making them attractive for latency-sensitive applications [4].

2.4 The Wiki Neutrality Corpus

The WNC consists of over two hundred thousand paired sentences from Wikipedia: each “source” sentence exhibiting subjective bias and its manually neutralized counterpart [2]. This parallel structure supports both supervised classification—treating biased vs. neutral as distinct classes—and neutralization tasks, where models learn to identify and modify linguistic features that eliminate bias. The WNC is a widely recognized dataset for evaluating bias detection and mitigation strategies, particularly in the context of text neutralization.

2.5 Ensemble Strategies in Text Classification

Ensembling combines multiple models’ predictions to reduce individual errors and improve robustness [8]. Common techniques include majority voting, stacking, and logit averaging. In the context of neural classifiers, averaging output logits from diverse architectures (e.g., an RNN and a transformer) often yields gains by leveraging complementary strengths, especially when constituent models capture different aspects of language bias.

3 Methodology

3.1 Data Preprocessing

We begin by constructing both transformer- and RNN-compatible datasets from the Wiki Neutrality Corpus (WNC) [2].

- **Raw data loading:** The `load_dataset` function reads the train/dev/test splits (tab-separated), drops rows with missing source or target text, and produces a shuffled DataFrame of `{text, label}` pairs (1 = biased, 0 = neutral).
- **Transformer pipeline:** In `datasets.py`, the `WNCDataset` class applies a Hugging Face tokenizer to each sentence, with truncation and padding to a fixed length (50 tokens), producing `input_ids`, `attention_mask`, and `labels` tensors.
- **RNN pipeline:** We build a vocabulary from the training texts (including special tokens `<pad>` and `<unk>`), tokenize by whitespace, map tokens to indices, and pad/truncate sequences to length 50 using the `tokenize_for_rnn` helper. The `get_rnn_data_loaders` function then returns PyTorch DataLoaders (batch sizes: 256 for training, 32 for dev, 16 for test).

3.2 Model Implementation

We implement two classifier families in `models.py` and manage training and evaluation through utility functions defined in `model_utils.py`. For the transformer-based models, each architecture is accompanied by a dedicated Jupyter notebook that handles fine-tuning, evaluation, and inference, enabling modular experimentation and reproducibility.

Recurrent Baselines

FastTextClassifier learns 100-dimensional token embeddings, averages them across each input sequence, and applies a two-unit linear layer for classification [3]. **BiLSTMClassifier** builds on this by initializing embeddings, feeding them into a two-layer bidirectional LSTM (64 units per direction), applying dropout ($p=0.05$) to the final states, and classifying via a linear head.

Transformer Fine-Tuning

Using Hugging Face’s `AutoModelForSequenceClassification`, we fine-tune five checkpoints—BERT-Large [6], ALBERT-XXLarge-v2 [5], DistilBERT [4], DistilRoBERTa [4], and RoBERTa-Large [7]—each paired with a freshly initialized classification head. All models train for five epochs with AdamW ($\text{lr}=2 \times 10^{-5}$, weight decay=0.01); batch sizes come from `model_batch_size_dict`. Routines `train_model` and `evaluate_model` handle device placement, loss/backprop, and metric logging.

Ensembles

In `model_utils.py`, `get_logits` extracts test-set logits per model; we then average these across selected checkpoints and pick the class with the highest mean score [8]. We evaluate both pairwise (e.g. BERT + RoBERTa) and four-model ensembles to quantify gains from model diversity.

3.3 Training and Evaluation

All models are trained and evaluated on the same splits of the Wiki Neutrality Corpus: a training set for fitting, a development set for model selection, and a held-out test set for final reporting [2].

RNN Baselines The FastText [3] and BiLSTM classifiers are trained for 20 epochs using the Adam optimizer with a learning rate of 1×10^{-3} . We batch the training data at 256 examples, and evaluate on the development set (batch size 32) at the end of each epoch. The model state achieving the highest F1 on the development split is retained for final testing (batch size 16).

Transformer Models Each transformer checkpoint (BERT-Large [6], ALBERT-XXLarge-v2 [5], DistilBERT [4], DistilRoBERTa [4], and RoBERTa-Large [7]) is fine-tuned for 5 epochs using AdamW (learning rate 2×10^{-5} , weight decay 0.01). Batch sizes are chosen per model capacity (e.g. 128 for most, 32 for ALBERT-XXLarge-v2). At the end of each epoch, we compute precision, recall, and F1 on the development set and save the best checkpoint.

Final Evaluation The retained checkpoints for all models are evaluated on the held-out test set. We report accuracy, precision, recall, and F1 score, computed from the aggregated predictions and true labels. To complement scalar metrics, we generate confusion matrices to visualize error patterns.

All training routines—including data loading, batching, optimizer steps, and metric computation—are implemented in `model_utils.py`, ensuring consistency across models and reproducibility of results.

4 Implementation Details

The repository is structured as follows: `src/` contains core modules—`datasets.py` for data loading and preprocessing (used across all models but not executed independently), `models.py` for FastText [3] and BiLSTM definitions, `utils.py` for GloVe loading and plotting utilities, and `model_utils.py` for training, evaluation, and ensembling routines [8]. Processed datasets are stored in `data/`, model checkpoints in `models/`, and experiment notebooks in `notebooks/`.

All experiments are conducted using Python 3.11 with PyTorch ($\geq 2.3.1$), Hugging Face Transformers ($\geq 4.47.1$), NumPy, pandas, scikit-learn, and Matplotlib, as specified in `requirements.txt`. Random seeds (42) are fixed across Python, NumPy, and PyTorch to ensure experimental reproducibility.

The typical workflow consists of: (1) importing functions from `datasets.py` to prepare tokenized datasets; (2) running `base_models.ipynb` to train and evaluate the RNN baselines; (3) executing each transformer notebook (e.g., `bert_large.ipynb`) for fine-tuning and evaluation; and (4) conducting ensemble experiments using the `ensemble_*.ipynb` notebooks. The code automatically detects available CUDA devices and adjusts batch sizes via the `model_batch_size_dict` to accommodate GPU memory constraints.

5 Results

5.1 RNN Baselines

Table 1 summarizes the test-set performance of our recurrent baselines. The FastText classifier [3] achieves 68.50% accuracy and a 66.70 F1-score, while the BiLSTM reaches 50.75% accuracy and a 45.85 F1-score.

Table 1: RNN Baseline Performance on the Test Set

Model	Accuracy	Precision	Recall	F1-score
FastText	0.6850	0.6950	0.7400	0.6670
BiLSTM	0.5075	0.5050	0.4200	0.4585

5.2 Transformer Fine-Tuning

Performance of fine-tuned transformer models is reported in Table 2. RoBERTa-Large [7] achieves the highest accuracy (76.45%) and F1-score (75.78), while distilled variants trade a small drop in performance for faster inference [4].

Table 2: Fine-Tuned Transformer Performance on the Test Set

Model	Accuracy	Precision	Recall	F1-score
BERT-Large-Uncased	0.5000	0.5000	1.0000	0.6667
DistilBERT-Base	0.7500	0.7841	0.6900	0.7340
DistilRoBERTa-Base	0.7385	0.8015	0.6340	0.7080
ALBERT-XXLarge-v2	0.7450	0.7900	0.6800	0.7280
RoBERTa-Large	0.7645	0.7799	0.7370	0.7578

5.3 Ensemble Performance

Table 3 presents results for logit-averaging ensembles [8]. A two-model ensemble of BERT-Large and RoBERTa-Large nearly matches the best single model, while the four-model combination offers marginal additional gains.

Table 3: Ensemble Performance on the Test Set

Ensemble Combination	Accuracy	Precision	Recall	F1-score
BERT-Large + DistilBERT	0.7495	0.7819	0.6920	0.7342
BERT-Large + RoBERTa-Large	0.7635	0.7771	0.7390	0.7576
Four-model (BERT, ALBERT, DistilBERT, DistilRoBERTa)	0.7615	0.8095	0.6840	0.7415

6 Conclusion

This work presents a systematic and reproducible framework for detecting subjective bias in text, leveraging the WNC [2] to train and evaluate a diverse suite of neural models. Beginning with rigorous data preprocessing—including tokenization, sequence alignment, and dataset partitioning—we construct a robust pipeline encompassing both recurrent [3] and transformer-based architectures [6, 7, 5, 4]. The approach integrates two lightweight baselines (FastText and BiLSTM), fine-tuning of five transformer models (BERT-Large, ALBERT-XXLarge-v2, DistilBERT, DistilRoBERTa, and RoBERTa-Large), and ensemble strategies based on logit averaging [8].

Experimental results underscore the superiority of pretrained transformers over traditional recurrent models. While FastText offers a computationally efficient baseline (68.50% accuracy) [3], its limited ability to capture nuanced contextual dependencies constrains its performance. The BiLSTM, despite its theoretical capacity for sequential modeling, exhibits instability and overfitting, achieving only 50.75% accuracy [9]. In contrast, all transformer-based models outperform these baselines, with RoBERTa-Large achieving the highest overall performance (76.45% accuracy, 75.78 F1-score) [7].

Ensemble techniques further demonstrate the complementary strengths of individual models [8]. A two-model ensemble combining BERT-Large and RoBERTa-Large nearly matches RoBERTa’s standalone performance (76.35% accuracy, 75.76 F1-score), indicating that architectural diversity can help mitigate individual model limitations. However, the marginal improvements observed with larger ensembles suggest diminishing returns, underscoring the importance of strategic model selection.

Beyond quantitative evaluation, our analysis reveals meaningful differences in error patterns. FastText displays balanced misclassification rates, while the BiLSTM is disproportionately biased toward false negatives, reflecting its inadequacy for this task [9]. Transformer models, particularly RoBERTa, demonstrate more refined discrimination, effectively capturing subtle lexical and syntactic indicators of subjectivity [7].

Appendices

A Training Results

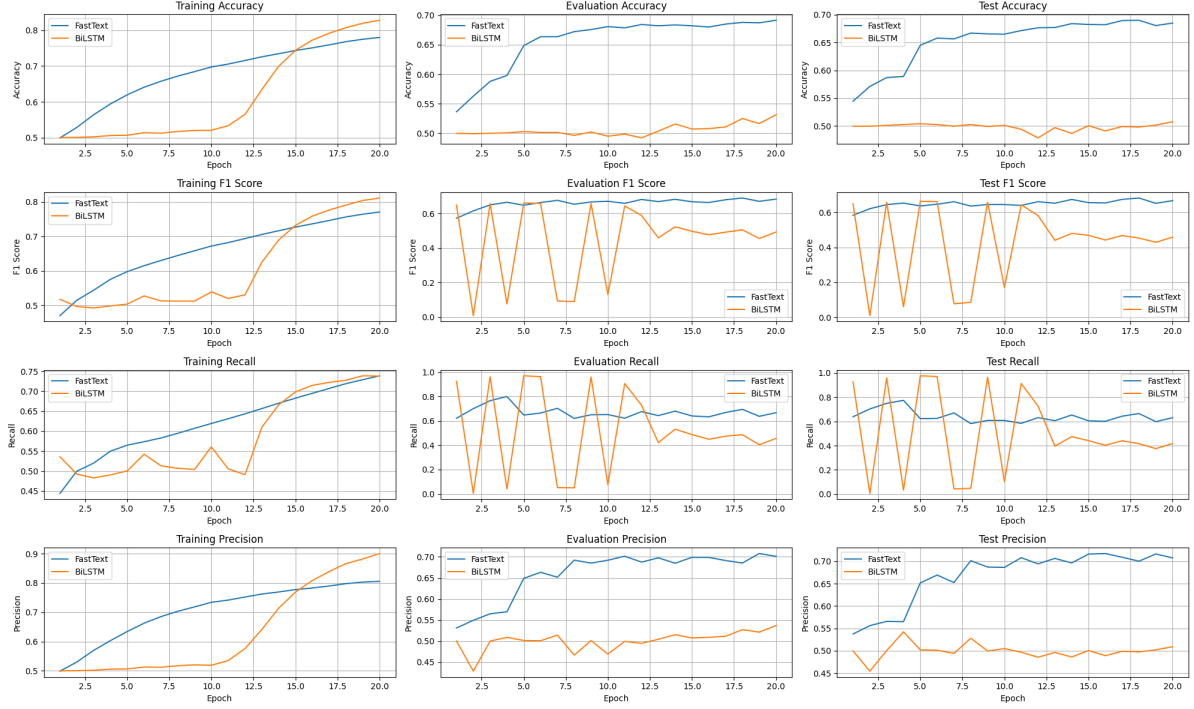


Figure 1: Training and evaluation F1 scores over epochs for transformer models.

Analysis of RNN Training Curves

FastText vs. BiLSTM Convergence. FastText [3] shows a smooth, monotonic rise in all metrics (accuracy, F1, recall, precision) across training, peaking around epoch 15–20. Its training, evaluation and test curves track closely, indicating strong generalization and minimal overfitting.

BiLSTM, by contrast, exhibits two distinct phases: very slow improvement for the first 10–12 epochs, then a sudden jump in training metrics (accuracy/F1 climbing steeply after epoch 12). This suggests the model struggles to learn until some internal LSTM gates “unlock,” but once they do it fits the training data rapidly.

Stability and Generalization. The FastText evaluation and test curves plateau around an F1 of 0.67 and accuracy of 0.69, matching its training performance [3]. Its low variance between splits demonstrates robustness. BiLSTM’s evaluation and test metrics remain highly erratic—precision, recall and F1 all bounce between near 0 and 0.6 in early epochs—before stabilizing only after epoch 12. Even then, its test-set F1 lags behind FastText (≈ 0.45 vs. 0.67), implying overfitting once it finally “learns.”

Generalization Gap. BiLSTM’s dramatic gap between training (≈ 0.80 F1) and evaluation/test (≈ 0.50 F1) after convergence highlights severe overfitting. FastText’s much smaller gap underscores its superior bias-detection generalization given limited data [3].

Implications for Model Selection. The FastText classifier—despite its simplicity—yields stable and competitive performance with minimal tuning [3]. The BiLSTM baseline, while theoretically more expressive, suffers from convergence instability and overfitting on this dataset. Mitigating this would require more aggressive regularization (e.g. higher dropout), a smaller learning rate, or earlier stopping.

B Confusion Matrices

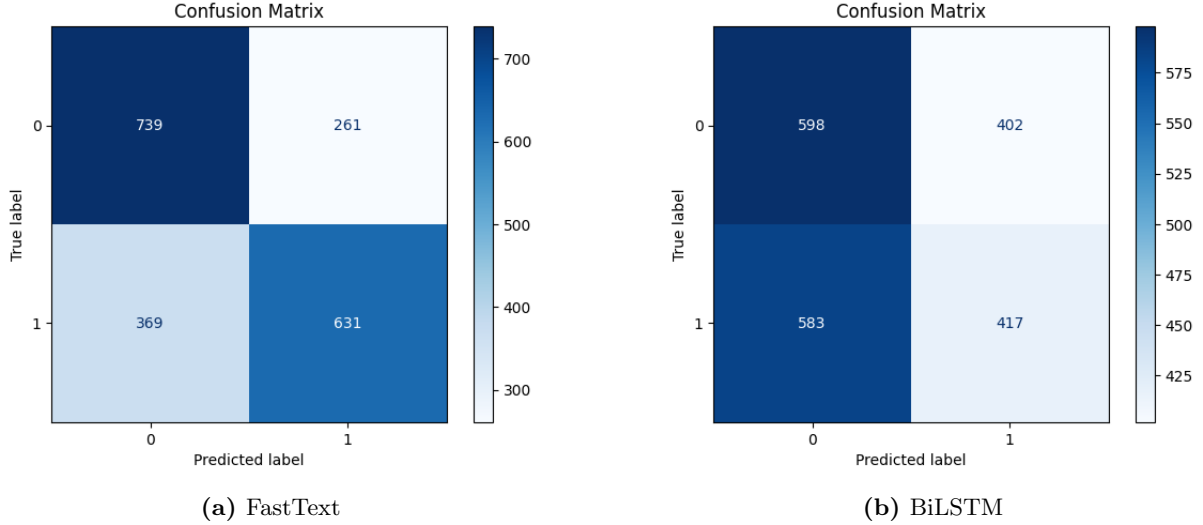


Figure 2: Test-set confusion matrices for our two RNN baselines.

Interpretation. The two confusion matrices reveal starkly different error patterns:

- **FastText (Fig. 2a):** - *Balanced performance:* it mislabels 26.1% of neutral sentences (261/1,000) and 36.9% of biased sentences (369/1,000). - *Bias-detection strength:* its false-negative rate (36.9%) is substantially lower than BiLSTM’s, yielding a recall of 63.1% on the biased class. - *Precision trade-off:* 261 false positives vs. 631 true positives gives a precision of 70.8%, indicating moderate over-flagging of neutral text as biased.
- **BiLSTM (Fig. 2b):** - *Skewed errors:* 40.2% false positives (402/1,000) but a much higher 58.3% false-negative rate (583/1,000). - *Poor bias recall:* it correctly identifies only 41.7% of biased sentences (417/1,000), explaining its low F1 of 0.458. - *Neutral-class bias:* with 598 true negatives against 402 false positives, its neutral-class precision is only 59.8%, reflecting confusion between neutral and subtly biased language.

These outcomes directly mirror the summary metrics in Section 5.1: FastText’s more balanced error distribution supports its higher F1 (0.667) [3], while BiLSTM’s heavy skew toward false negatives undermines its utility for bias detection on this dataset.

References

References

- [1] M. Recasens, C. Danescu-Niculescu-Mizil, and D. Jurafsky, *Linguistic Models for Analyzing and Detecting Biased Language*, In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL), 2013.
- [2] R. Pryzant, R. Diehl Martinez, N. Dass, S. Kurohashi, D. Jurafsky, and D. Yang, *Automatically Neutralizing Subjective Bias in Text*, In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2020.
- [3] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, *Bag of Tricks for Efficient Text Classification*, In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2017.

- [4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, In NeurIPS Deep Learning and Representation Learning Workshop, 2019.
- [5] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*, In International Conference on Learning Representations (ICLR), 2020.
- [6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, In Proceedings of NAACL-HLT, 2019.
- [7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv:1907.11692, 2019.
- [8] T. G. Dietterich, *Ensemble Methods in Machine Learning*, In Multiple Classifier Systems, MCS 2000, Springer, 2000.
- [9] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9(8):1735–1780, 1997.