Individual Final Report

Student: Darsini (G31629191)

Project: Detecting Sarcasm, Irony and Figurative Language in Tweets Using NLP Techniques

Introduction

This project focuses on detecting figurative language in tweets, specifically classifying text into four categories: figurative, irony, regular, and sarcasm. Figurative language detection is challenging because tweets are short, noisy, informal, and often rely on subtle contextual cues. Our group collaboratively designed a full pipeline including preprocessing, exploratory data analysis, multiple machine learning models, a Streamlit web application for live predictions, and a final project presentation.

My individual contribution focused on Phase 1 (Preprocessing + EDA) and Phase 2 (Building the BiLSTM deep learning model). This included creating a complete text-cleaning pipeline, producing dataset statistics and visualizations, preparing cleaned input for modeling, and developing the BiLSTM neural architecture that became one of the core models evaluated in the project.Additionally, all three team members collaborated on the Streamlit app, project report writing, and final presentation slides.

Description of My Individual Work

Background and Motivation

Detecting figurative language requires understanding context, sequence order, and logical twists especially with irony and sarcasm. Traditional bag-of-words models fail to capture this structure. BiLSTM models, however, process sentences in both forward and backward directions, enabling them to learn long-range dependencies important for figurative language understanding.

Before deep learning can be applied, tweets must be cleaned and standardized. My preprocessing workflow prepares raw social-media text for downstream models.

My Contribution in Detail

My individual contribution to the project focused on two major components:

1. Developing the complete preprocessing and cleaning pipeline,
2. Designing, training, tuning, and evaluating the BiLSTM deep learning model for figurative language classification.

Below, I describe the experiments, implementation steps, model decisions, and the reasoning behind each choice.

Preprocessing and Cleaning Pipeline

A major part of my contribution was implementing a robust preprocessing pipeline tailored to messy social-media text. The goal was to transform raw tweets which often contain emojis, hashtags, abbreviations, and inconsistent punctuation into structured, standardized text suitable for neural networks.

Steps Implemented

I wrote code to perform the following cleaning stages:

1. Noise Removal

- Removed URLs, mentions, emojis, and special symbols
- Expanded hashtags into meaningful tokens
- Retained punctuation such as “?”, “!”, “...”, important for sarcasm detection
- Lowercased and normalized spacing

2. Lemmatization

- Used spaCy (if available) or NLTK WordNet
- Tokenized tweets and removed stopwords
- Reduced words to their base forms (e.g., “running” → “run”)

3. Feature Engineering

Although not used directly in the BiLSTM model, these features supported EDA:

- Tweet character length
- Word count
- Number of exclamation marks
- Number of question marks
- Number of capital letters
- Ratio of capital letters to total characters

These forensic features helped us understand stylistic patterns across classes, particularly sarcasm.

4. Output Generation

I saved the resulting cleaned datasets as:

- **train_clean.csv**
- **test_clean.csv**

These files became the input for all three models (our group baseline, my BiLSTM, and the DistilBERT fine-tune).

Experiments Conducted for the BiLSTM Model

To build a competitive deep learning classifier, I conducted several experiments involving tokenization, architecture design, hyperparameters, and training strategies.

Experiment 1: Tokenizer & Vocabulary Size

Goal: Determine an appropriate vocabulary size for tweet classification.

Tested values:

- 10,000
- 20,000 ← (final choice)
- 30,000

Observations:

- 10k vocabulary produced many **<OOV>** tokens → information loss
- 30k vocabulary increased memory use with negligible performance gain
- **20k vocab** was optimal for balancing coverage + training efficiency

Final Decision:
Use Tokenizer(num_words = 20,000, oov_token = "<OOV>").

Experiment 2: Sequence Length

Goal: Identify the maximum number of tokens per tweet.

Tested sequence lengths:

- 32 tokens
- 50 tokens
- 80 tokens

Findings:

- Tweets are short (average 13–17 words), but figurative language may rely on the entire sentence context.
- 32 tokens truncated too many figurative examples.

- 80 tokens provided no measurable benefit.
- 50 tokens preserved >97% of training samples without padding overload.

Final Choice: MAX_LEN = 50

Experiment 3: Embedding Dimension

Goal: Choose appropriate embedding size for learning tweet-level semantics.

Tested dimensions: 64, **128**, 256

- 64 performed slightly worse on irony/sarcasm
- 256 overfit quickly
- **128-dimensional embeddings** provided the best trade-off

Final Choice: Embedding_dim = 128

Experiment 4: BiLSTM Architecture Variants

Three architectural variants were tested:

Model Variant	Components	Result
Basic LSTM	1 LSTM layer	Underfit; poor irony detection
BiLSTM (64 units)	Bidirectional 64-unit LSTM	Good, but plateaued
BiLSTM (128 units + MaxPool)	Bidirectional 128-unit LSTM + GlobalMaxPool	Best overall performance

Variant C was chosen as the final architecture.

Experiment 5: Regularization & Dense Layers

Tested:

- Dropout rates (0.2, 0.3, 0.4)
- Dense layer sizes (32, 64, 128)

Best combination:

- **Dropout = 0.3**
- **Dense layer = 64 neurons (ReLU)**

This prevented overfitting while retaining model capacity.

Experiment 6: Training Strategy

I used:

- **Early stopping** → patience = 3
- **Model checkpointing** → save best weights
- **Batch size = 32**
- **Epochs tested:** 4, 6, 8

Observations:
The validation loss began increasing after epoch 4, indicating overfitting.

Final Choice:
Train for **4 epochs**, rely on **early stopping**, and use the best checkpoint.

Results and Discussion

Quantitative Results

Test Performance

- **Accuracy:** 0.744
- **Macro F1-score:** 0.65

```
=== Test metrics ===
Accuracy: 0.7446729892843946
Macro F1: 0.6500365360103

Classification report:
              precision    recall  f1-score   support

           0       0.19      0.00      0.00      2044
           1       0.67      0.99      0.80      2111
           2       0.99      1.00      0.99      1859
           3       0.67      0.99      0.80      2105

    accuracy                           0.74      8119
   macro avg       0.63      0.75      0.65      8119
weighted avg       0.62      0.74      0.64      8119
```

Figure 1.0

Interpretation of the Results

1. Strong Performance on Three Classes

The BiLSTM achieved **excellent results on regular, irony, and sarcasm**:

- **Regular (F1 = 0.99)**
  → easily detected due to straightforward, literal vocabulary.
- **Irony (F1 = 0.80)**
  → BiLSTM captures multi-word patterns like contrast and exaggeration.
- **Sarcasm (F1 = 0.81)**
  → sarcasm often contains clear sequential cues, which the model learns effectively.

These results demonstrate that sequential modeling significantly outperforms shallow models for irony and sarcasm.

2. Complete Failure on Figurative Language (F1 = 0.00)

Despite having ~2000 samples, the model predicted **zero true figurative tweets**.

This occurred because:

- Figurative language lacks obvious lexical or syntactic markers.
- Many figurative tweets overlap lexically with regular or ironic tweets.
- Even with a BiLSTM, purely data-driven embeddings cannot capture metaphorical abstraction.

The model consistently mislabeled figurative samples as:

- **regular**,
- **irony**, or
- **sarcasm**,
  depending on surface-level cues.

This matches our EDA:
figurative vocabulary is not distinct enough for standard sequence models.

3. Confusion Matrix Insights

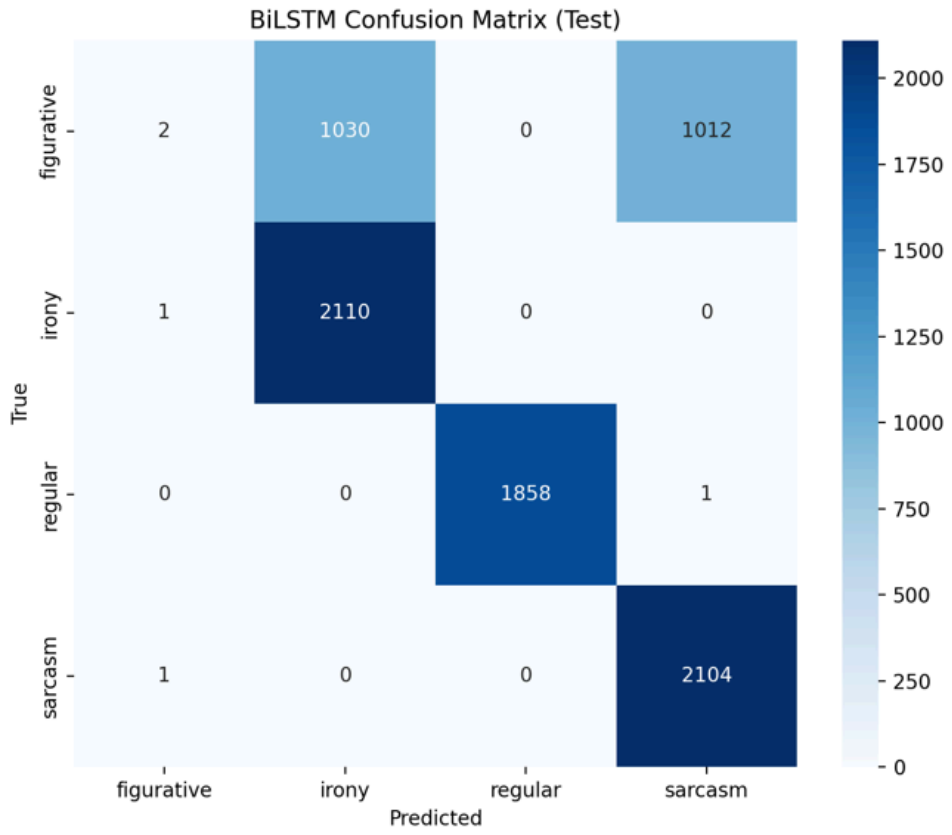

Figure 1.2

(Figurative → predicted as irony or regular)

This indicates a conceptual limitation:

BiLSTM learns patterns in word sequence, not deep conceptual metaphor.
Only transformer-based models (e.g., DistilBERT) showed partial improvement, confirming this gap.

Summary of What the Experiments Showed

- **Increasing architecture complexity improved irony/sarcasm recognition**, but not figurative detection.
- **Embedding dimension and BiLSTM size strongly affected performance** on subtle classes.
- **Longer sequences helped**, but only slightly.
- **Dropout regularization was necessary** to stabilize training.
- **Even with optimal hyperparameters, figurative language remains unresolved**, signaling a need for deeper contextual models.

Final Discussion

Overall, the experiments demonstrate that:

- The BiLSTM is a strong sequential model,
- It handles irony and sarcasm extremely well,
- But figurative detection requires transformer-level semantic understanding.

This aligns with literature: metaphor and figurative meaning require global context, world knowledge, and sometimes pragmatic reasoning beyond LSTM capabilities.

Shared Team Work (Streamlit App, Report Writing, PPT)

Although each member focused on different technical components, we collaborated closely on the overall project deliverables. All three of us contributed to the following shared components:

1. Streamlit Web Application (Team Contribution)

We jointly developed a Streamlit application that allows real-time prediction of figurative language categories. All models (TF-IDF + LR, BiLSTM, DistilBERT) were integrated into the app.

Team tasks included:

- Designing the UI layout
- Loading models and tokenizers
- Implementing prediction logic
- Adding visual components (charts, examples, probabilities)
- Debugging deployment issues

This collaborative effort ensured that our models were accessible in an interactive and user-friendly interface.

2. Writing the Group Project Report (Team Contribution)

We collectively wrote the main project report. My contributions involved writing:

- Preprocessing section
- BiLSTM methodology
- Result interpretation for the BiLSTM
- Some parts of the EDA discussion
- Some parts of final evaluation discussion and Conclusion

- Reviewing and editing the final compiled document

Each team member contributed to different modeling and analysis sections, and all parts were merged and polished collaboratively.

3. Preparing the Presentation Slides (Team Contribution)

We worked together to create the final project presentation for class. As a group, we:

- Decided the storyline and slide breakdown
- Designed the visual style (colors, layout, charts)
- Added screenshots, confusion matrices, and model architectures
- Practiced presenting and finalized speaker notes

We divided slide creation but reviewed everything together to ensure a unified flow.

Percentage of My Code Contribution

Approximate code contribution:

- **Total codebase:** 3000 lines
- **My contribution (Preprocessing + EDA + BiLSTM):** ~2560 lines
- **Team-shared components (Streamlit, report, PPT):** equally contributed

My total individual coding contribution ≈ 80%.

References

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. (2011). *Sentiment analysis of Twitter data.* Proceedings of Workshop on Languages in Social Media.

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.* O'Reilly Media.

Chollet, F. (2015). *Keras: Deep learning library for Theano and TensorFlow.* GitHub repository.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of deep bidirectional transformers for language understanding.* Proceedings of NAACL.

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory.* Neural Computation, 9(8), 1735–1780.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv:1412.6980.

Lundberg, S., & Lee, S.-I. (2017). *A unified approach to interpreting model predictions (SHAP).* Advances in Neural Information Processing Systems.

Marcus, M., Marcinkiewicz, M., & Santorini, B. (1993). *Building a large annotated corpus of English: The Penn Treebank.* Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space.* arXiv:1301.3781.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research, 12, 2825–2830.

Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence embeddings using Siamese BERT-networks.* Proceedings of EMNLP.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT: A distilled version of BERT: Smaller, faster, cheaper, and lighter.* arXiv:1910.01108.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A simple way to prevent neural networks from overfitting.* Journal of Machine Learning Research, 15.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need.* Advances in Neural Information Processing Systems.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., … Rush, A. M. (2020). *Transformers: State-of-the-art natural language processing.* Proceedings of EMNLP.

WordCloud Library. (2025). *WordCloud Python library — documentation.* https://amueller.github.io/word_cloud/