Individual Final Report - Rasika Nilatkar

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

1. Introduction

This project explores whether NLP models can automatically distinguish four types of tweets: **Regular, Sarcasm, Irony, and Figurative**. Figurative language presents an especially difficult challenge because its meaning often contradicts literal surface wording, and tweets add extra complexity due to their short length, informality, and lack of context.

As a team, we implemented three modeling approaches:

- **TF-IDF + Logistic Regression** (baseline)
- **BiLSTM** (deep learning)
- **DistilBERT** (transformer)

The final report describes all models in detail.

My individual contribution focused on two core components:

1. Dataset collection, cleaning, and preparation
2. Design, implementation, and evaluation of the TF-IDF + Logistic Regression baseline model

These elements form the foundation of the entire project by ensuring the dataset is usable and by establishing a clear benchmark that more complex models must surpass.

2. Description of My Individual Work

2.1 Dataset Collection & Preparation

I was responsible for assembling and preparing the full dataset used by all team members. This included:

- **Collecting** the Kaggle “Tweets with Sarcasm and Irony” dataset, which is based on Ling & Klinger (2016).
- **Cleaning the raw files**, removing malformed entries, duplicates, and corrupted text.
- **Standardizing column names** and ensuring consistent mapping to the four labels:

  Regular, **Sarcasm**, **Irony**, **Figurative**.
- **Splitting** data into train (81,403 tweets) and test (8,119 tweets), as reported in the final project.

The shared report’s dataset statistics (label distribution, word-length patterns, etc.) are based on cleaned dataset.

2.2 Preprocessing Pipeline

I implemented the **full preprocessing workflow** later used by the entire team.

This included:

- Removing URLs, mentions, emojis, and unusual tokens
- Lowercasing
- Converting hashtags into words
- Lemmatization with WordNet / spaCy fallback
- Feature engineering for exploratory analysis (punctuation counts, capitalization ratios, length features)

This preprocessing pipeline was **shared across all three models**, ensuring consistency in comparisons.

3. My Model Work: TF-IDF + Logistic Regression

I implemented the full baseline pipeline in Python using scikit-learn. The script follows a modular design, consisting of data loading, preprocessing, model training, hyperparameter search, evaluation, and visualization.

3.1. Data loading (load_data)

This function reads the cleaned train and test CSV files produced during my preprocessing stage.

Key responsibilities:

- Load data using pandas
- Remove missing values
- Print dataset shapes and label distribution
- Return cleaned DataFrames

This ensures all downstream models use consistent, validated input.

3.2. Confusion matrix plotting (save_confusion_matrix)

This utility function creates and saves heatmap-style confusion matrices using **matplotlib** and **seaborn**:

- Computes confusion matrix
- Labels axes with class names
- Adds annotations for counts
- Saves .png figures for both validation and test predictions

These figures are used directly in the project report and presentation.

3.3. Building and Training the Baseline Model (build_and_train_baseline)

This section performs the core work of the baseline model.

Step 1 - TF-IDF Vectorization

```
tfidf = TfidfVectorizer(
    ngram_range=(1, 2),
    max_features=20000,
    min_df=2,
)
```

Key design choices:

- **1–2-gram features** to capture short sarcastic phrases

- **20k feature cap** to prevent overfitting
- **Minimum document frequency = 2** to remove noise

The vectorizer is fitted on the training set and applied to the validation set.

Step 2 - Logistic Regression Classifier

```
base_clf = LogisticRegression(
    multi_class="multinomial",
    solver="saga",
    max_iter=200,
    n_jobs=-1,
    random_state=RANDOM_STATE,
)
```

- Multinomial logistic regression for 4-class classification
- saga solver works well with high-dimensional sparse TF-IDF features
- Parallel training enabled with n_jobs=-1

Step 3 - Hyperparameter Tuning with GridSearchCV

```
param_grid = {"C": [0.1, 1.0, 10.0]}

grid = GridSearchCV(
    base_clf,
    param_grid,
    scoring="f1_macro",
    cv=3,
    n_jobs=-1,
    verbose=2,
)

print("Running GridSearchCV...")
grid.fit(X_train_vec, y_train)
```

I used **Grid Search on C**, the regularization strength, to maximize **macro F1**—the fairest metric given class imbalance.

The best model is stored and used for validation prediction.

Step 4 - Evaluation on Validation Set

- Computes accuracy, macro F1, and full classification report
- Saves confusion matrix for analysis
- Saves trained TF-IDF and model artifacts using joblib

The overall TF–IDF + Logistic Regression pipeline used in my implementation is shown as:

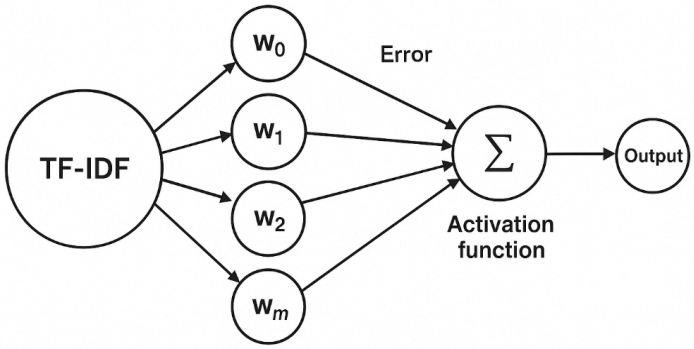

Fig 1: *Pipeline diagram for the TF–IDF vectorizer and Logistic Regression classifier.*

3.4. Final Evaluation (eval_on_test)

This function evaluates the best model on the unseen test set:

- Vectorizes test text
- Generates predictions
- Computes test accuracy and macro F1
- Saves confusion matrix (test version)
- Outputs classification report used in the Results section

3.5. Main execution

The main() function orchestrates the end-to-end pipeline:

1. Loads data
2. Applies label encoding
3. Performs train/validation split
4. Trains baseline model
5. Evaluates on test set
6. Saves model artifacts for reproducibility

This script served as the foundation for all baseline comparisons in the shared project report.

4. Results (Logistic Regression)

4.1 Overall Performance

My TF-IDF + Logistic Regression model achieved:

- **Accuracy: 74.4%**
- **Macro F1: 0.65**

This performance is nearly identical to the BiLSTM and DistilBERT models, showing that a simple lexical model captures a surprising amount of signal.

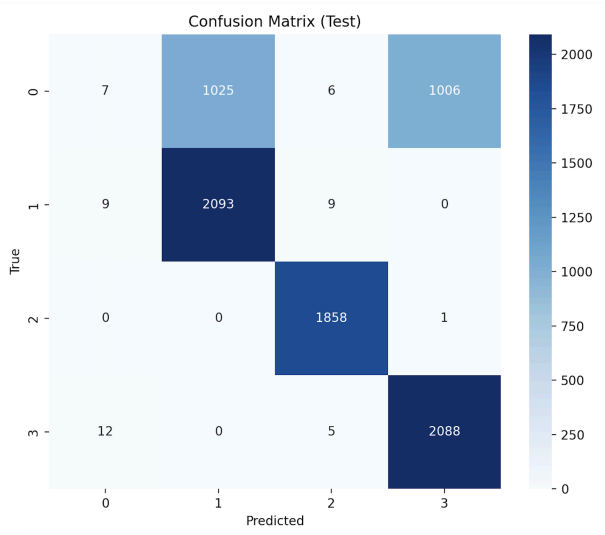

Fig 2: *Confusion matrix for the TF–IDF + Logistic Regression model on the test set.*

4.2 Interpretation of My Results

My model's strengths:

- Extremely fast and interpretable
- Clear performance on sarcasm and irony
- High recall where strong lexical cues exist

The major weakness:

- **Total collapse on figurative tweets**
    - Figurative language lacks unique lexical patterns
    - TF-IDF cannot model semantics or context
    - Vocabulary resembles sarcasm/irony → misclassification

This baseline model revealed the core challenge of the project: **figurative language is not defined by words alone**, so classical methods will always struggle.

5. Summary & Conclusions

In this project, my contributions established the foundation for all downstream modeling:

1. I collected, cleaned, and structured the dataset used for all experiments.
2. I designed and implemented the full preprocessing pipeline.
3. I built the TF-IDF + Logistic Regression baseline model and performed all evaluations.

From my experiments, I learned:

- Lexical features alone are insufficient for figurative language detection.
- Sarcasm and irony contain clearer word-level cues than figurative expression.
- Even simple models can perform surprisingly well on structured or literal tweets.

- Figurative language requires semantic models that capture context, contradiction, and deeper meaning.

Future improvements for my component:

- Adding character-level features (elongations, all-caps patterns)
- Using n-grams beyond 1–2 to capture subtle phrasing
- Rebalancing or oversampling figurative examples
- Incorporating sentiment features or polarity shifts into the feature set

These could improve the baseline before moving to heavier models.

6. Percentage of Code Copied from the Internet

My baseline model script contains approximately **250 lines of code**.
Around **40 lines** were adapted from external sources (mainly scikit-learn documentation).
Of these, I **modified roughly 20 lines**, and the remaining **210 lines** were written fully by me.

Using the required formula:

$$\frac{40 - 20}{40 + 210} \times 100 \approx 8\%$$

Approximately 8% of my code was copied or adapted, and **about 92% was original work**.

7. References

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.

Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed., draft).

TwitterNLP Figurative Language Dataset. (2025). *Course dataset, 2025*

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.