Twitter Sentiment Classification Project

Swapnil Prakash Memane

Submission Date:21.11.2025

Dataset: Sentiment140 (Kaggle link)

1. Abstract

This project builds a Twitter sentiment classifier using step-by-step text cleaning and preprocessing. It trains and compares three models—Logistic Regression, LSTM, and DistilBERT—to understand how each handles sentiment prediction. Streamlit apps were created for all three models, allowing users to check the sentiment of one tweet at a time. The project shows how different AI methods perform on real-world tweet analysis.

2. Introduction

- The project aims to classify Twitter posts into **Positive** and **Negative** sentiments using ML and deep learning models.
- Sentiment analysis is important for understanding public opinion and extracting insights from large volumes of social media text.
- The project focuses on handling noisy, informal tweet language to build accurate sentiment classifiers.

3. Dataset Description

- The Sentiment140 dataset, which contains 1.6 million tweets collected through the Twitter API.
- Each tweet is labeled with sentiment categories: 0 = Negative, 2 = Neutral, and 4 = Positive.
- The dataset is designed for training and evaluating sentiment detection models.
- Dataset fields:

  target: Sentiment label of the tweet (0, 2, or 4)

  ids: Unique ID of the tweet

  date: Timestamp of when the tweet was posted

  flag: Query term used (or NO_QUERY if none)

  user: Username of the person who posted the tweet

  text: Actual tweet content

4. Data Preprocessing

- Cleaned tweets by removing URLs, mentions, hashtags, emojis, special characters, and converting text to lowercase.
- Applied TF-IDF encoding for Logistic Regression, sequence tokenization for LSTM, and DistilBERT tokenizer for transformer inputs.
- Checked and addressed class imbalance where needed.
- Performed basic feature preparation such as cleaned text and encoded vectors.
- Split the dataset into train, validation, and test sets for fair model evaluation.

5. Baseline Classical ML Model

Experimentation was done with Logistic Regression, Naive Bayes, SVM, XGBoost models.

Based on the evaluation metrics, best model was chosen i.e Logistic Regression.

Logistic Regression Model

Hyperparameters

- **Vectorizer:** TfidfVectorizer() with default settings
- **Model:** LogisticRegression() (default hyperparameters)
- **Note:** No manual tuning (C, penalty, solver) was applied in this version.

Training Process

- Tweets were vectorized using TF-IDF after converting all inputs to strings and handling missing values.
- The Logistic Regression model was trained on TF-IDF features using the fit() method.
- Model training was fast and stable due to the linear nature of the algorithm.

Evaluation Metrics

- **Accuracy 76.38%**
- **Class-wise Precision, Recall, F1-score**

  Negative (0):

  Precision: 0.7748
  Recall: 0.7389
  F1-score: 0.7564
  Support: 153,888 samples

  Positive (1):

  Precision: 0.7540
  Recall: 0.7884
  F1-score: 0.7708
  Support: 156,195 samples

- **Overall Metrics**

  Macro Avg F1-score: 0.7636
  Weighted Avg F1-score: 0.7637

- **Confusion Matrix (Summary)**

  Model confuses some Positive tweets as Negative and vice-versa, but errors are balanced across both classes.

  True Positive and True Negative counts dominate, showing stable performance.

Observations / Insights

Logistic Regression with TF-IDF performs well as a baseline, reaching ~76% accuracy.

The model shows better precision for Negative tweets and better recall for Positive tweets.

Tweets with mixed or subtle sentiment are the most common misclassifications.

Works fast and efficiently compared to LSTM or DistilBERT, making it suitable for lightweight applications.

---

6. Deep Learning Model (LSTM)

Model Architecture

- **Embedding Layer:** 64-dimensional word embeddings with a vocabulary size of 5000.
- **LSTM Layer:** 32 units with dropout=0.2 and recurrent dropout=0.2 to reduce overfitting.
- **Dense Output Layer:** Single neuron with sigmoid activation for binary (Positive/Negative) classification.

Hyperparameters

- **Embedding size:** 64
- **Sequence length (max_len):** 40
- **Vocab size:** 5000
- **Dropout:** 0.2 (recurrent + standard)
- **Optimizer:** AdamW
- **Loss Function:** Binary Cross-entropy

- **Batch size:** 256
- **Epochs:** 10 (with EarlyStopping, patience=2)
- **Callback:** EarlyStopping (monitors val_loss, restores best weights)

Training Process & Challenges

- Tweets were tokenized using Keras Tokenizer and converted to padded sequences (max_len = 40).
- The model trained smoothly, but LSTM required more computation compared to Logistic Regression.
- EarlyStopping helped prevent overfitting, as validation loss started rising after a few epochs.
- Handling out-of-vocabulary (OOV) words using <OOV> token improved sequence stability.

Evaluation Metrics

(Based on the saved classification_report from your code)

- **Accuracy:** ~0.78 (78.06% validation accuracy during training)
- **Precision / Recall / F1-score:** Stored in lstm_conf_metrics.csv generated by the script.
- **Confusion Matrix:** Shows the distribution of correctly/incorrectly predicted Positive vs Negative tweets.

Observations / Insights

- The LSTM model performs better than Logistic Regression due to its ability to capture sequence patterns.
- Short sequence length (40 tokens) keeps model lightweight while still capturing important tweet context.
- Dropout and EarlyStopping successfully prevented overfitting.
- Although slower to train, LSTM provides stronger generalization on noisy, informal tweet text.

---

7. Transformer Model (BERT / DistilBERT)

- Model architecture & pre-trained model used
- Fine-tuning strategy
- Hyperparameters
- Training process & challenges
- Evaluation metrics:
  - Accuracy
  - Precision / Recall / F1-score Confusion matrix
  -
- Observations / insights

---

8. Model Comparison

Model	Accuracy	Precision	Recall	F1-score	Training Time	Notes
Logistic Regression	0.7638	Class 0: 0.7748 Class 1: 0.7540	Class 0: 0.7389 Class 1: 0.7884	Class 0: 0.7564 Class 1: 0.7708	36.27 seconds	

LSTM/GRU	0.7816	Class 0: 0.7836 Class 1:0.7798	Class 0: 0.7738 Class 1: 0.7894	Class 0: 0.7786 Class 1: 0.7846	33 minutes	On local machine without GPU
Transformer	0.7615	Class 0: 0.7519 Class 1: 0.7718	Class 0: 0.7789 Class 1: 0.7441	Class 0: 0.7651 Class 1: 0.7577	40 minutes	On Colab using T4 GPU

LSTM/GRU performed the best overall

- Highest accuracy (0.7816)
- Best F1-scores for both classes
- Handles sequential patterns better than TF-IDF or transformer on limited training data

Logistic Regression is the fastest

- Trained under 40 seconds
- Strong baseline model with competitive performance for a traditional ML approach

DistilBERT underperformed slightly

- Accuracy ~0.76 despite using GPU
- Likely reasons:
  - Very small training sample (only 8%)
  - Only 2–3 epochs
  - No learning-rate tuning
  - Model requires larger dataset for full potential

Trade-offs

- **Training Speed:** Logistic Regression > LSTM > Transformer
- **Accuracy:** LSTM > Logistic Regression > Transformer
- **Compute Requirement:** Logistic Regression < LSTM < Transformer

---

9. Explainability / Interpretability

- To be implemented

---

10. Deployment

Tools & Libraries (per model)

- **Common: streamlit, pandas, numpy, joblib**
- **Logistic Regression: scikit-learn (TF-IDF vectorizer saved with joblib)**
- **LSTM: tensorflow / keras (Keras Tokenizer saved with joblib)**
- **DistilBERT: transformers (pretrained DistilBertTokenizerFast + transformers model; torch backend or TF variant)**

UI & Functionality

- Simple and lightweight interface where the user can:
  - Enter a **single tweet** in a text box.
  - Click **Predict Sentiment** to get the classification result.
  - View the output as **Positive** or **Negative** with color-coded indicators.
- Separate pages / scripts for each model:
  - **Logistic Regression App**
  - **LSTM App**
  - **DistilBERT App**

11. Challenges & Lessons Learned

Technical Challenges Faced

1. Large dataset size (1.6M tweets) caused slow loading, preprocessing, and training.
2. Slow text cleaning functions, especially regex-heavy operations and repeated NLTK calls.
3. Handling different input formats for three models:
    a. TF-IDF + Logistic Regression
    b. Tokenizer + LSTM
    c. DistilBERT Tokenizer + Transformer
4. Class imbalance slightly affecting LSTM performance.
5. LSTM model underperforming due to poor handling of long sequences and training instability.
6. GPU/CPU limitations, especially while training LSTM and Transformer models.
7. Inconsistent predictions due to improper thresholding (y_pred_probs > 0.5).
8. Deployment issues in Streamlit (file paths, model loading time).

Steps taken to resolve the challenges:

1. Applied different tokenizers appropriately based on the model architecture.
2. Used train-test stratification to maintain balanced class distribution.
3. Tuned LSTM with:
    - Proper sequence length
    - Batch size adjustments
    - Dropout layers
4. For GPU limitation the training of transformer model was done on Google Colab using T4 GPU.

Lessons Learned During the Project

1. One cleaning pipeline cannot fit all models — classical ML, LSTM, and Transformers require different preprocessing.
2. LSTMs perform poorly on long, noisy text compared to modern transformer models.
3. Proper thresholding and label extraction are critical for correct metrics.
4. GPU availability drastically impacts training time for deep learning models.
5. Transformers provide the best balance of accuracy and robustness for text tasks.
6. Streamlit deployment becomes easier when models and tokenizers are cached.
7. Data quality and cleaning make a large difference in final accuracy.

12. Conclusion

The project successfully developed and compared three sentiment classification models—Logistic Regression, LSTM/GRU, and DistilBERT—to classify tweets into positive and negative sentiments. All models demonstrated reliable performance, with LSTM/GRU achieving the highest overall accuracy and F1-score.

Key Takeaways:

- LSTM/GRU performed the best, showing strong ability to capture sequential patterns in text.
- Logistic Regression provided a strong and extremely fast baseline, proving that classical ML techniques can still perform competitively on large text datasets.
- DistilBERT delivered moderate performance, indicating the need for larger training data and hyperparameter tuning to fully leverage transformer-based architectures.

Future Improvements:

1. Train the Transformer model on a larger dataset and with more epochs for better learning.
2. Use GPU-based training for LSTM to significantly reduce training time.
3. Implement additional cleaning for URL, emoticons , repeated words removal in data preprocessing stage
4. Implement LIME/SHAP for model explainability
5. Use logging for recording training time, success, error messages etc.
6. Develop an app to performance batch analysis. (eg, 3 sentences at a time predict 3 different sentiments)
7. Hyperparameter Tuning for all models

13. References

- Dataset source - Sentiment140 (Kaggle link)
- Libraries / frameworks used:

  streamlit

  scikit-learn

pandas
numpy
nltk
pyngrok
tensorflow
tf-keras
transformers

---

14. GitHub Repo Link