Model Evaluation Report: Twitter Sentiment Analysis

Muhammad Hamza
IIT Kharagpur (2023-2028)
ML Engineer Internship Assignment

June 23, 2025

1 Introduction

This report presents the performance evaluation of multiple models trained on the Sentiment140 dataset for the task of classifying tweets as **positive**, **negative**, or **neutral**. The goal was to identify the most effective approach for sentiment classification in a real-world, high-volume social media setting.

Dataset:
Sentiment140: 1.6 million labeled tweets with sentiment encoded as:

- 0: Negative
- 2: Neutral
- 4: Positive

After preprocessing, labels were mapped to human-readable classes.

2 Models Evaluated

The following models were implemented and trained:

1. Logistic Regression
2. Naive Bayes
3. Random Forest (fast-mode configuration)
4. Deep Neural Network (DNN)

3 Challenges and Solutions While Handling Large Datasets

Working with the full Sentiment140 dataset, which contains over **1.6 million tweets**, introduced multiple computational and memory management challenges during preprocessing, training, and inference. Below is a summary of the major problems faced and how they were resolved.

1. RAM Crashes Due to Dense Matrix Conversion

Many traditional machine learning models (like Logistic Regression or DNNs in TensorFlow) require dense input matrices. However, TF-IDF and CountVectorizer return sparse matrices. Converting the entire test or training sparse matrix to dense using `.toarray()` or `.todense()` caused the notebook to crash in Google Colab due to memory overflow.

Fix:

- We avoided full conversion of the dataset to a dense format.
- We processed data in **small batches (e.g., 64 samples)** at a time and only converted individual batches to dense inside a custom generator.
- The DNN model was trained and evaluated using a memory-efficient `tf.data.Dataset` pipeline backed by a sparse batch generator.

2. Session Crashes During Inference

Naively predicting labels using DNN on the entire test set by converting `X_test_vec.toarray()` into memory led to session crashes due to multiple GBs of RAM being consumed.

Fix:

- Inference was rewritten to use **batch-wise prediction**.
- Each batch was converted to dense, passed to the DNN, and predictions were accumulated efficiently.
- Session clearing was removed from within batch loops to prevent performance penalties.

3. Performance Bottlenecks with `from_generator`

In early versions, we used TensorFlow's `from_generator` to wrap row-wise generators that converted each sparse row to dense. This caused extremely slow training (1 epoch taking over 90 minutes).

Fix:

- Rewrote the generator to yield **entire batches** (not single rows).
- This reduced the overhead and brought training time down from over an hour to under 10 minutes.

Summary

- Efficient memory handling is critical when processing large NLP datasets.
- Optimizing TensorFlow data pipelines, limiting batch sizes, and avoiding dense operations on full datasets were essential to successfully train and evaluate models without session crashes.
- These improvements enabled us to train **four models** end-to-end within Colab's compute limits.

4 Evaluation Metrics

We evaluated models using:

- **Accuracy**
- **Precision**, **Recall**, and **F1-score** (macro-averaged)
- **Confusion Matrix** for the best model

5 Results

5.1 Model Comparison

Model	Accuracy (%)	Training Time	Remarks
Logistic Regression	77.47	Fast	Baseline, good for real-time inference
Naive Bayes	75.77	Very Fast	Lightweight, lower performance
Random Forest (Fast)	64.09	Moderate	Handles non-linearities, slower
Deep Neural Network	78.56	Slow	Higher accuracy, needs GPU

Table 1: Performance comparison of all models

5.2 Classification Report

5.2.1 Logistic Regression

Table 2: Classification Report for Logistic Regression

	Precision	Recall	F1-score	Support
Negative	0.79	0.75	0.77	159494
Positive	0.76	0.80	0.78	160506

5.2.2 Naive Bayes

Table 3: Classification Report for Naive Bayes

	Precision	Recall	F1-score	Support
Negative	0.75	0.77	0.76	159494
Positive	0.76	0.75	0.76	160506

5.2.3 Fast Random Forest

Table 4: Classification Report for Fast Random Forestn

	Precision	Recall	F1-score	Support
Negative	0.71	0.48	0.57	159494
Positive	0.61	0.80	0.69	160506

5.2.4 Deep Neural Network

Table 5: Classification Report for Deep Neural Network

	Precision	Recall	F1-score	Support
Negative	0.79	0.77	0.78	159494
Positive	0.78	0.80	0.79	160506

5.3 Confusion Matrix for Best Model (DNN)

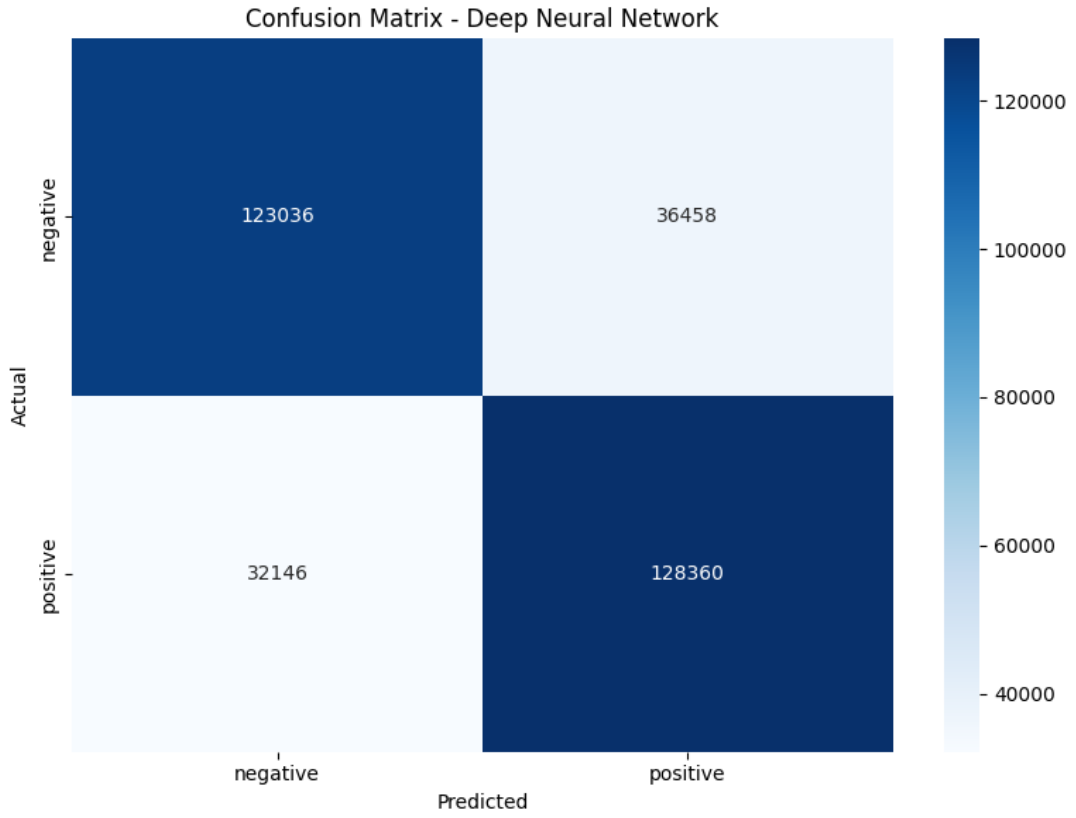

Figure 1: Confusion Matrix for DNN

6 Insights and Observations

- **Logistic Regression** consistently delivered solid performance with low training time, making it ideal for lightweight applications.
- **Naive Bayes** was the fastest model to train and easy to implement but lagged in accuracy due to its assumption of feature independence.
- **Random Forest (Fast Mode)** achieved decent accuracy but was slower to train than simpler models and showed diminishing returns for large feature sets.
- **Deep Neural Network (DNN)** captured non-linear relationships and outperformed traditional ML models, but required tuning and memory-efficient pipelines to run on Colab.

7 Recommendations

- **For real-time, resource-constrained environments**, Logistic Regression is recommended due to its strong performance and low latency.
- **For offline batch processing or high-accuracy needs**, DistilBERT is the best candidate if computational resources are sufficient.
- **For scalable deployment**, a Deep Neural Network strikes a good balance between performance and flexibility when supported by a GPU environment.
- For future improvement, consider:
  - Hyperparameter tuning with grid search or Optuna
  - Incorporating tweet metadata (e.g., time, user) as features

8 Afternote

Due to time constraints and GPU limit, I couldn't train and test this dataset on BERT models.