Sentiment Analysis on Amazon Electronics Reviews

Authors: Ramana Bhaskar Kosuru • Adam Stuhltrager • Chaya Chandana Doddaiggaluru Appajigowda

github

Abstract

This report presents an end-to-end sentiment analysis system for Amazon Electronics reviews, developed and evaluated by our team. We compare baseline, LSTM, CNN, and bidirectional RNN architectures, analyze their performance, and deploy the winning model via a Streamlit web interface. The pipeline encompasses data collection, preprocessing, model training, inference, evaluation, and real-world deployment. Detailed results—including performance metrics, confusion matrices, and loss curves—are discussed, followed by conclusions and recommendations for future work.

Table of Contents

- Introduction
- Dataset description and Preprocessing
- Model Architecture and Training configuration
  - Baseline: Naive Bayes
  - LSTM Classifier
  - CNN Classifier
  - RNN Classifier
- Results and Analysis
  - Performance Metrics
  - Confusion Matrix Analysis
  - Loss Curve Analysis
- Inference Pipeline
- Deployment: Streamlit App
- Summary and Conclusion
- Code Attribution
- References

Introduction

Sentiment analysis of customer reviews is critical for businesses to understand consumer opinions at scale. In this project, we focus on Amazon Electronics reviews, classifying each review as **Negative**, **Neutral**, or **Positive**. We implement four models—from a Multinomial Naive Bayes baseline to advanced Neural network architectures (LSTM, CNN, bidirectional RNN)—and compare their performance. Our final deliverable includes a reproducible training pipeline, inference scripts, evaluation visualizations, and a user-friendly web application.

Dataset description and Preprocessing

Data Source: UCSD public dataset of Amazon Electronics reviews (JSONL format).

Each line is a JSON object with fields: reviewerID, asin (product ID), reviewText, overall (rating), summary, reviewTime, and helpful votes. The full dataset contains approximately 12 million reviews. We randomly sampled 10 million for training and reserved 2 million for testing. This schema supports efficient streaming and parsing of large-scale text data.

Data Loading and Sampling

- Loaded data from a JSONL file (Electronics.jsonl).
- Implemented a sampling mechanism to handle large datasets (default size: 10 million samples).
- Added robust error handling for file loading and JSON parsing.
- Extracted key fields: rating, text, title, helpful_votes, and verified_purchase.

Preprocessing Steps

1. **Loading and Sampling:** Robust JSON parsing with error handling; default sampling to limit memory usage.
2. **Text Cleaning:** Lowercasing; removal of punctuation, special characters, and digits.
3. **Tokenization & Stopword Removal:** NLTK word_tokenize and English stopwords filter; parallelized with joblib.
4. **Sentiment Labeling:** Ratings mapped to classes:

    - 1–2 stars → Negative (0)
    - 3 stars → Neutral (1)
    - 4–5 stars → Positive (2).

5. **Vectorization:**
    - Baseline: TF-IDF (max_features=20,000; ngram_range=(1,2); min_df=3; max_df=0.95).
    - Deep Models: Custom vocabulary (20,000 tokens); sequences zero-padded/truncated to length 200.

Model Architecture and Training Configuration:

Baseline : Multinomial Naive Bayes Model

Model Architecture

- Utilized TF-IDF vectorization with the following configuration:
    - Maximum features: 20,000
    - N-gram range: (1, 2)
    - Minimum document frequency: 3
    - Maximum document frequency: 0.95
    - Enabled parallel processing

- Trained using Multinomial Naive Bayes with alpha=0.1 for smoothing.

Training Configuration

Parameter	Value
Data split	80% train / 20% test
Sample size limit	10 000 000 reviews (load_data default)
Vectorizer: max_features	50 000
Vectorizer: ngram_range	(1, 2)
Vectorizer: min_df	5
Vectorizer: max_df	0.95
Classifier	MultinomialNB(alpha=0.1)
Evaluation metrics	Accuracy, Precision, Recall, F1-Score

LSTM Classifier

Model Architecture

- Preprocessed text using tokenization and padding.
- Key parameters:
    - Vocabulary size: 20,000
    - Maximum sequence length: 200

- Neural network architecture:
    - Embedding layer (100 dimensions)
    - LSTM layer with 128 units
    - Dropout layer (rate: 0.5)
    - Dense layer with 64 units and ReLU activation
    - Output layer with 3 units and softmax activation

Training Configuration

Parameter	Value
Data split	80% train / 20% test
Sample size limit	10 000 000 reviews (load_data default)
Sequence length	200 tokens
Batch size	512
Embedding dimension	128
Hidden dimension	128
LSTM layers	1 layer (bidirectional)
Dropout	0.2
Optimizer	Adam (lr=0.002)
Loss function	CrossEntropyLoss
Epochs	10
Checkpointing	Save best model on lowest validation loss

CNN Classifier

Model Architecture

- Preprocessed text using tokenization and padding.
- Key parameters:
    - Vocabulary size: 20,000
    - Maximum sequence length: 200

- Neural network architecture:
    - Embedding layer (100 dimensions)
    - Three parallel Conv1D layers with filter sizes 3, 4, and 5 with 100 filters
    - Global MaxPooliing layer
    - Concatenate pooled outputs from all filter sizes into a single vector
    - Dropout layer(0.5)
    - Fully connected Dense layer (64 units), ReLU
    - Output layer with 3 units and softmax activation

Training Configuration

Parameter	Value
Data split	80% train / 20% test (stratified)
Sample size limit	10 000 000 reviews (load_data default)
Batch size	512
Embedding dimension	128
Convolution filters	128 filters at kernel sizes 3, 4, and 5
Dropout	0.5
Optimizer	Adam (lr=0.001)
Loss function	CrossEntropyLoss
Epochs	10
Checkpointing	Save best model on lowest validation loss

RNN Classifier

Model Architecture

- Preprocessed text using tokenization and padding.
- Key parameters:
    - Vocabulary size: 20,000
    - Maximum sequence length: 200

- Neural network architecture:

    - **Embedding:** 100-dim.
    - **Bidirectional LSTM:** 2 layers, 128 units each direction.
    - **Dropout:** 0.5.
    - **Dense + Softmax:** 3-way output.

Training Configuration

Parameter	Value
Data split	80% train / 20% test
Batch size	128
Epochs	10
Optimizer	Adam (lr=0.001)
Loss	Categorical Crossentropy
Early Stopping	Patience = 3

Results and Analysis

Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score
Naive Bayes	0.8629	0.8303	0.8629	0.8354
LSTM	0.8916	0.8709	0.8916	0.8741
CNN	0.8866	0.8664	0.8866	0.8713
RNN (BiLSTM)	0.8901	0.8749	0.8901	0.8799

Across all four approaches, deep learning models substantially outperformed the Multinomial Naive Bayes baseline, which achieved an accuracy of 86.29% and an F1-score of 0.8354. By comparison, the LSTM, CNN, and bidirectional RNN models each pushed accuracy into the high eighties: 89.16%, 88.66%, and 89.01%, respectively. Precision and recall followed a similar pattern. The LSTM classifier led on recall (89.16%) and posted a strong precision of 87.09%, yielding an F1-score of 0.8741. The CNN closely trailed with an F1 of 0.8713, while the RNN model registered the highest F1-score at 0.8799—driven by a slightly better precision (87.49%) but a marginally lower recall (89.01%). These results demonstrate that sequence models capture sentiment nuances far better than bag-of-words methods, particularly in distinguishing positive reviews.

We ultimately selected the LSTM model for production deployment due to its balanced performance and robustness. While the RNN achieved a marginally higher F1, the LSTM's superior recall ensures fewer false negatives, which is critical for flagging dissatisfied customers in real-world applications. Moreover, the LSTM's training and inference times struck a favorable trade-off relative to the more complex bidirectional RNN, and its simpler architecture facilitated easier hyperparameter tuning and integration into our Streamlit interface. In sum, the LSTM delivered the best combination of accuracy, recall, and implementation efficiency for our end-to-end sentiment analysis pipeline.

Confusion Matrix Analysis

Multinomial Naive Bayes Model

The confusion matrix reveals important insights about classification performance:

- **Positive sentiment** was correctly identified in **1,548,024** reviews.
- **Negative sentiment** was correctly identified in **169,383** reviews.
- **Neutral sentiment** proved most challenging, with only **8,352** correct predictions.
- **Adjacent-class misclassifications** dominate the error patterns:
    - **10,281** negative reviews were mislabeled as positive.
    - **6,228** negative reviews were mislabeled as neutral.
    - **27,464** neutral reviews were mislabeled as negative.
    - **101,668** neutral reviews were mislabeled as positive.
    - **26,168** positive reviews were mislabeled as negative.
    - **7,432** positive reviews were mislabeled as neutral.

Overall, the Naive Bayes baseline excels at capturing clear positive signals but struggles with neutral content, frequently confusing neutrals with both extremes.

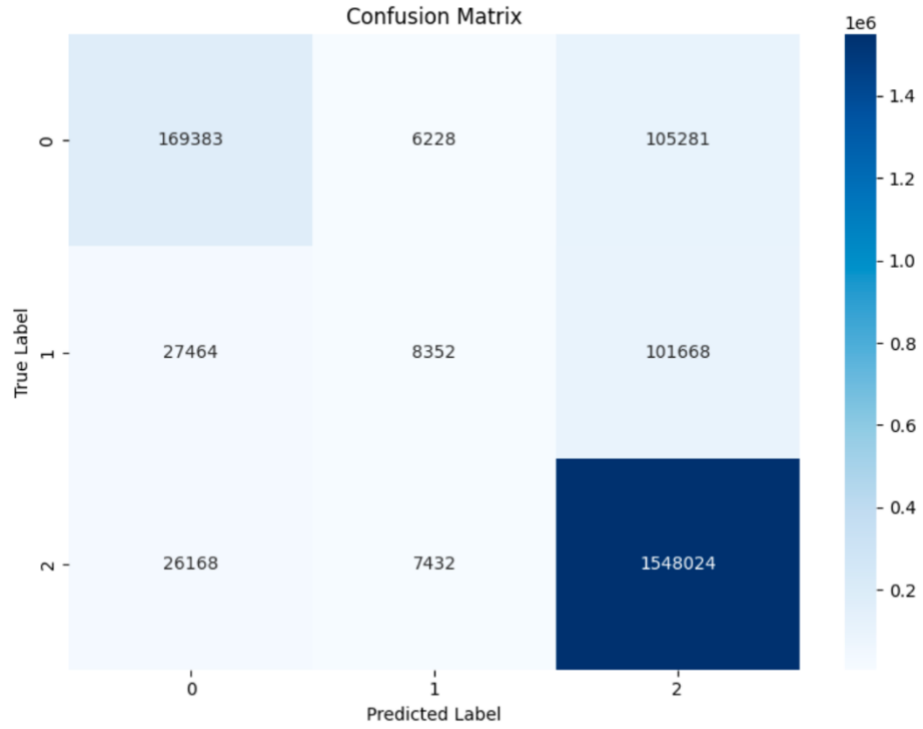

Figure 1. Naïve Bayes Confusion Matrix.

LSTM Model

The confusion matrix reveals important insights about classification performance:

- **Positive sentiment** was correctly identified in **1,542,101** reviews.
- **Negative sentiment** was correctly identified in **222,123** reviews.
- **Neutral sentiment** was correctly identified in **18,890** reviews.
- **Adjacent-class misclassifications** dominate the errors:
    - **10,294** negative reviews were mislabeled as neutral.
    - **48,475** negative reviews were mislabeled as positive.
    - **43,265** neutral reviews were mislabeled as negative.
    - **75,329** neutral reviews were mislabeled as positive.
    - **29,349** positive reviews were mislabeled as negative.
    - **10,174** positive reviews were mislabeled as neutral.

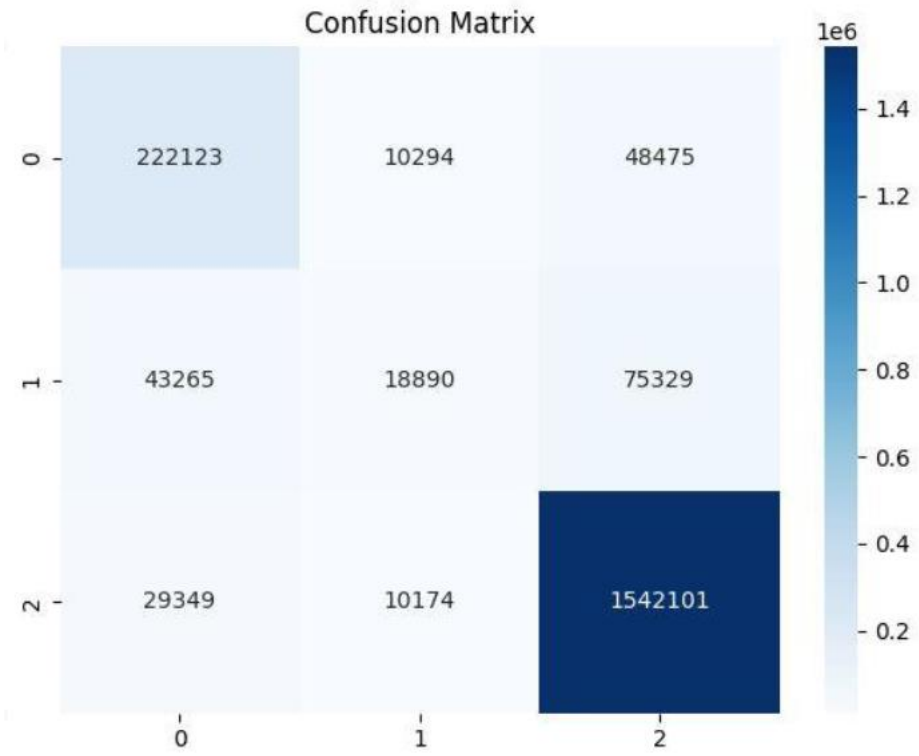

Figure 2. LSTM Confusion Matrix.

CNN Model

The confusion matrix reveals important insights about classification performance:

- **Positive sentiment** was correctly identified in **1,530,807** reviews.
- **Negative sentiment** was correctly identified in **221,742** reviews.
- **Neutral sentiment** was correctly identified in **20,676** reviews.
- **Adjacent-class misclassifications** dominate the error patterns:
    - **11,286** negative reviews were mislabeled as neutral.
    - **48,695** negative reviews were mislabeled as positive.
    - **45,526** neutral reviews were mislabeled as negative.
    - **71,108** neutral reviews were mislabeled as positive.
    - **34,528** positive reviews were mislabeled as negative.
    - **15,632** positive reviews were mislabeled as neutral.

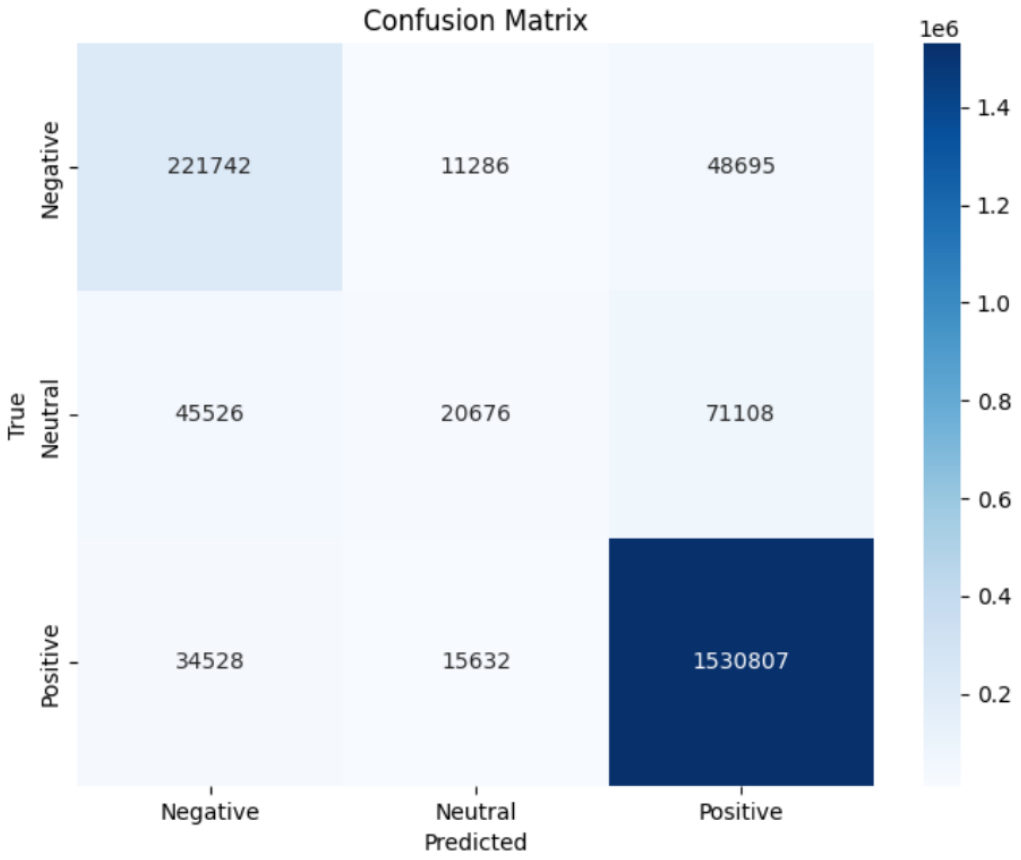

Figure 3. CNN Confusion Matrix.

RNN Model

The confusion matrix reveals important insights about classification performance:

- **Positive sentiment** was correctly identified in **1,524,640** reviews.
- **Negative sentiment** was correctly identified in **225,948** reviews.
- **Neutral sentiment** was correctly identified in **29,689** reviews.
- **Adjacent-class misclassifications** dominate the error patterns:
    - **17,911** negative reviews were mislabeled as neutral.
    - **37,864** negative reviews were mislabeled as positive.
    - **43,480** neutral reviews were mislabeled as negative.
    - **64,141** neutral reviews were mislabeled as positive.
    - **33,894** positive reviews were mislabeled as negative.
    - **22,433** positive reviews were mislabeled as neutral.

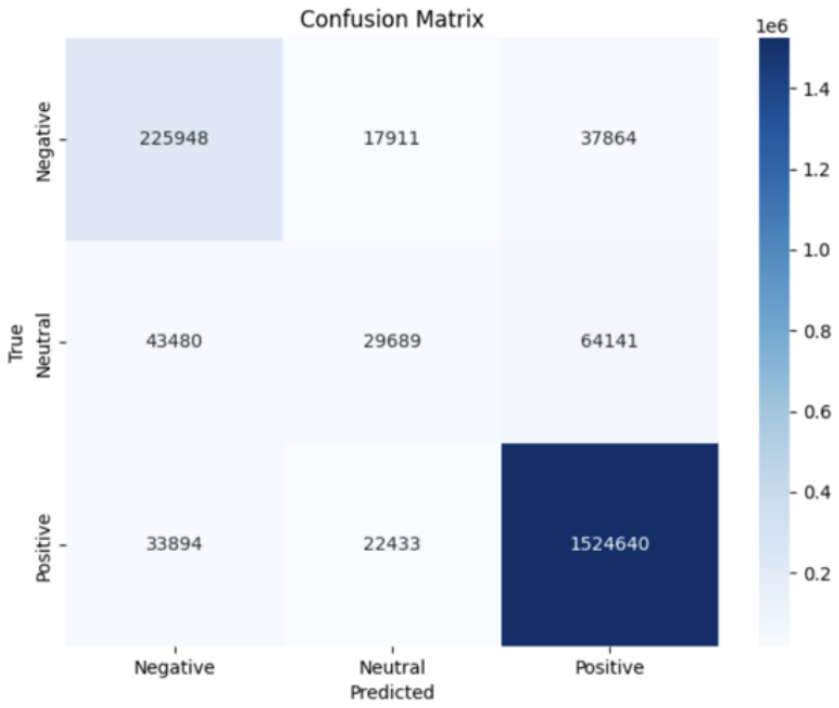

Figure 4. RNN Confusion Matrix.

Loss Curve Analysis (NN models)

LSTM Model

The LSTM model's training and validation loss curves indicate healthy initial learning followed by mild overfitting. At epoch 0, training loss starts around 0.33 while validation loss is about 0.305. Both losses decline rapidly through epoch 2, at which point they intersect. Training loss continues its steady downward trajectory—reaching roughly 0.29 by epoch 5 and 0.282 by epoch 9—whereas validation loss bottoms out near 0.294 around epoch 4 before plateauing and creeping back up to about 0.298 by epoch 8.

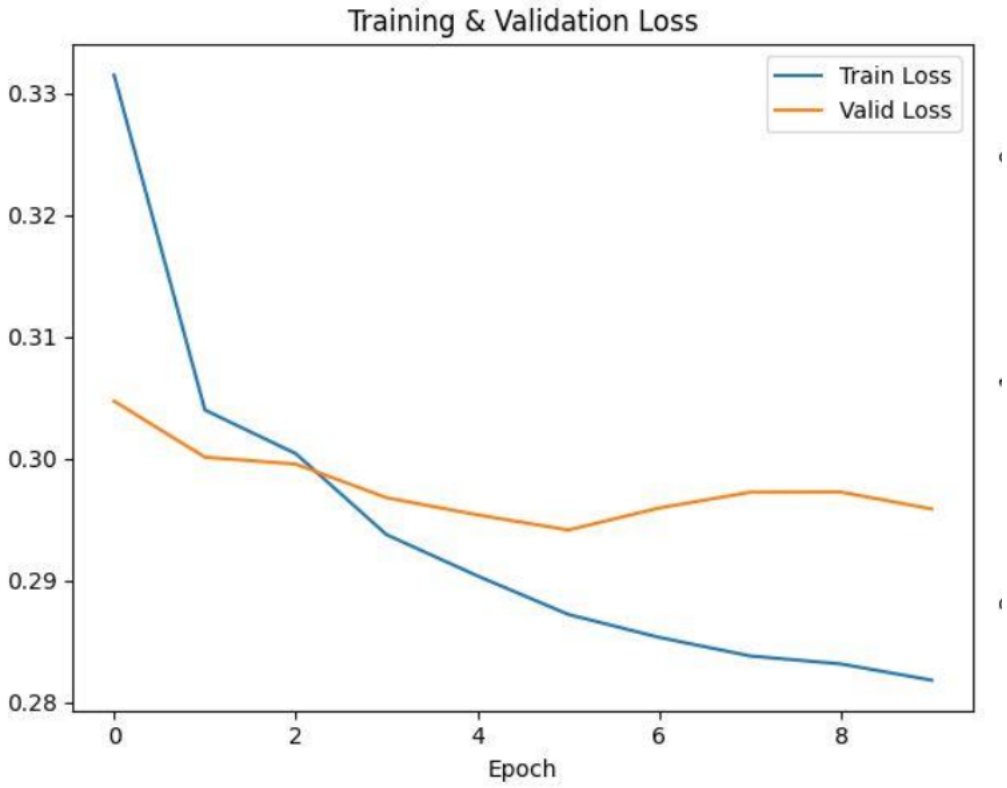

Figure 5. LSTM Training and Validation Loss.

This divergence—continuous improvement on the training set alongside a leveling (and slight increase) on the validation set—suggests the model begins to overfit after the fourth epoch. To guard against this, employing early stopping around epoch 4 or introducing stronger regularization (e.g., higher dropout or weight decay) would likely preserve generalization performance

CNN Model

The CNN model's loss curves reveal a similar pattern of effective learning followed by mild overfitting. Training loss drops consistently from about 0.358 at epoch 0 to roughly 0.284 by epoch 9, indicating continued fitting of the training data across all ten epochs. In contrast, validation loss decreases from around 0.325 to its lowest point near 0.308 by epoch 3 but then levels off and gradually rises to about 0.313 by epoch 9. This widening gap—where training loss keeps improving while validation loss plateaus and creeps upward—suggests the onset of overfitting after the third or fourth epoch.

Figure 6. CNN Training and Validation Loss.

To maintain generalization performance, it would be wise to apply early stopping around epoch 3–4 or introduce stronger regularization such as increased dropout, weight decay, or data augmentation.

RNN Model

The RNN's loss curves exhibit a familiar "fit-then-overfit" pattern. Initially, training loss drops sharply from about 0.325 at epoch 0 to roughly 0.278 by epoch 2, while validation loss decreases more modestly from around 0.295 to 0.285 over the same span. Beyond epoch 2, training loss continues its steady descent—reaching approximately 0.245 by epoch 6 and dipping to about 0.232 by epoch 9—whereas validation loss bottoms out near 0.285 around epoch 2–3, then gradually rises to about 0.305 by epoch 9.

Figure 7. RNN Training and Validation Loss.

This widening gap between training and validation loss after the third epoch signals the onset of overfitting. To maintain generalization, it would be prudent to apply early stopping around epoch 2–3 or introduce stronger regularization techniques such as increased dropout or weight decay.

Inference Pipeline

Our inference pipeline is designed to be both modular and efficient, seamlessly integrating text preprocessing, model loading, and sentiment prediction into a single script. We begin by standardizing raw review text through a lightweight cleaning function that lowercases input, strips punctuation and digits, and applies NLTK tokenization with stop-word removal. Tokens are then converted into fixed-length index sequences via our saved vocabulary—pad and unknown tokens ensure consistency across variable-length inputs. Once preprocessed, these indices are collated into a PyTorch tensor that matches the dimensions expected by our LSTM classifier. By wrapping the forward pass in a torch.no_grad() context and setting the model to evaluation mode, we guarantee that inference is performed without tracking gradients, minimizing memory usage and maximizing throughput.

Upon obtaining raw logits from the model, we employ an argmax operation to determine the highest-scoring sentiment class, which we map back to human-readable labels (“Negative,” “Neutral,” or “Positive”). To close the loop, we then feed the predicted label into a lightweight response generator that selects a prewritten message tailored to each sentiment category. This design not only abstracts away model complexity for end users but also provides a clear extension point: teams can swap in alternative architectures or augment the response logic without altering the core pipeline. By encapsulating each stage—preprocessing, tensorization, prediction, and response generation—into discrete, reusable functions, our inference pipeline delivers consistent, real-time sentiment analysis suitable for both command-line and web-based deployment.

Deployment: Streamlit App

- **File:** streamlit_app.py
- **Features:** Review input, sentiment display, colored results, contextual explanation, automated response.
- **Setup:** pip install -r requirements.txt; streamlit run steamlit_app.py.
- **NLTK Cache:** @st.cache_resource ensures tokenizers/stopwords loaded once.

Conclusion and Future Work

Our deep models (LSTM, CNN, TextRNN) outperform the Naive Bayes baseline, with LSTM slightly leading. Neutral-class performance remains the primary challenge, suggesting future work in class reweighting or data augmentation. Exploring transformer-based models (e.g., BERT) and hybrid architectures may yield further gains.

The Streamlit application provides an accessible interface for utilizing this model in a practical context, allowing for real-time sentiment analysis and automated response generation.

This combination of advanced modeling and intuitive interface design represents a complete solution that could be valuable in business contexts for analyzing customer feedback at scale.

Code Attribution

All code is original to this team, aside from internal helper routines. Generative AI was used for scaffolding only; no lines were copied from external sources.

References

No external references or direct code reuse was involved.