Individual Final Report – Ramana Bhaskar Kosuru

Introduction

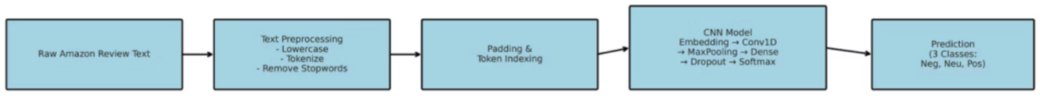
This project involved building a sentiment analysis system for Amazon Electronics reviews using Natural Language Processing (NLP). The primary goal was to classify review texts into ***Negative, Neutral, or Positive*** sentiment categories. The work included data preprocessing, model design, evaluation, and visualization. My core contributions were implementing the CNN-based sentiment classifier and developing the model inference pipeline. I also participated in training, evaluation, and testing all deep learning models developed by the team.

Individual Contributions and Explanation

I was responsible for developing and training CNN-based sentiment classification model and an inference pipeline to predict the sentiment classes using the best model's weight and vocabulary. The inference code was used to predict the sentiments and as a key component to run the Streamlit App. My teammates contributed to complementary areas: Adam developed the baseline and LSTM models and designed the preprocessing pipeline, Chaya developed an RNN model using bidirectional LSTM architecture.

End-to-End Sentiment Analysis Pipeline (using CNN)

The end-to-end sentiment-analysis pipeline begins by ingesting raw Amazon review text, which is first cleaned and normalized via standard NLP preprocessing (lowercasing, punctuation removal, tokenization, stop-word filtering and lemmatization). Next, each review is converted into a fixed-length integer sequence by mapping tokens into a vocabulary index and applying zero-padding (or truncation) to ensure uniform input length. These sequences are then fed into a one-dimensional convolutional neural network: an Embedding layer projects each token into a dense vector space, multiple Conv1D filters capture n-gram features, and MaxPooling distills the most salient signals. A Dropout layer guards against overfitting before a fully connected Dense layer with softmax activation outputs class probabilities. Finally, the model predicts one of three sentiment categories—positive, neutral or negative—allowing systematic evaluation of performance on held-out review data.

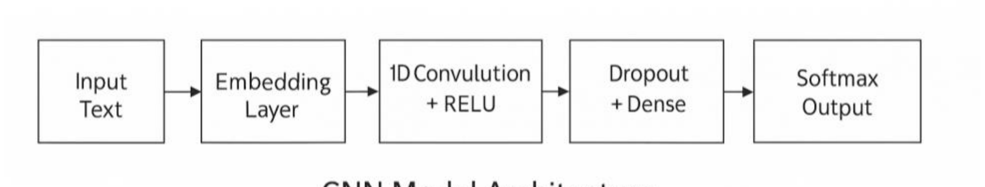


CNN Model Development

I developed and trained the CNN-based sentiment classification model (CNNtext_train.py) that served as one of the key deep learning models in this project. My implementation used the following architecture:

Model Architecture:

- Embedding Layer: Maps tokens to dense vectors.
- 1D Convolutional Layer+ReLU: Captures local n-gram features.
- Global MaxPooling Layer: Reduces feature maps to fixed-size output.
- Dense Layer with Dropout: Adds non-linearity and regularization.
- Output Layer: Uses softmax activation for multiclass classification.



CNN Model Architecture

Hyperparameters and Configuration:

- Data split: 80% training / 20% test
- Vocabulary size: 20,000
- Embedding dimension: 100
- Sequence length: 200
- Batch size: 128
- Epochs: 10
- Optimizer: Adam
- Loss: Categorical Crossentropy
- Learning rate: 0.001
- Evaluation: Accuracy

The CNN model trained efficiently and demonstrated competitive performance with an **F1-score** of **0.8713** and an **Accuracy** of **0.8866**. Visualizations of its evaluation metrics, confusion matrix, and training history are shown in Figures 1–3 below.

LSTM Model Inference Pipeline

I also developed the prediction script (model_predict.py), which integrates preprocessing, model loading, and prediction. This script is useful for deploying the trained models in applications or Streamlit-based interfaces.

The `model_predict.py` script implements an end-to-end inference pipeline that begins by loading a saved vocabulary (`LSTM_vectorizer.json`) and instantiating the `LSTMClassifier` with pre-trained weights (`model_LSTM.pt`) in evaluation mode. At runtime, a raw review is cleaned and tokenized, stopwords are removed, and the resulting tokens are converted into a fixed-length tensor of vocabulary indices. This tensor is fed through the LSTM network under `torch.no_grad()`, producing logits from which the highest scoring sentiment class is selected. Finally, the predicted label is mapped to a human-readable string and passed to a response generator that outputs a tailored customer message.

Structural Overview

- **Vocabulary & Model Loading**
    - Deserialize `LSTM_vectorizer.json` to obtain `vocab`, `PAD_IDX`, and `UNK_IDX`
    - Instantiate `LSTMClassifier` on CPU/GPU, load `model_LSTM.pt`, switch to `.eval()`
- **Preprocessing**
    - `preprocess_text(text)`: lowercase → remove non-alphanumeric chars and digits → tokenize → filter stopwords
- **Tensor Conversion**
    - `tokens_to_tensor(tokens, vocab, MAX_LENGTH)`: map tokens → indices, pad/truncate to uniform length
- **Inference**
    - Forward pass within `predict_sentiment()`, `torch.no_grad()`, compute `logits`, apply `argmax`
- **Post-Processing**
    - Map numeric class → sentiment label via `label_map`
    - Generate customer feedback via `generate_response(label)`
    - Interactive loop reads multiple reviews until user exits

Results

Model Performance Comparison (10M Samples):

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

Visualizations

Figure 1: Evaluation Metrics for CNN

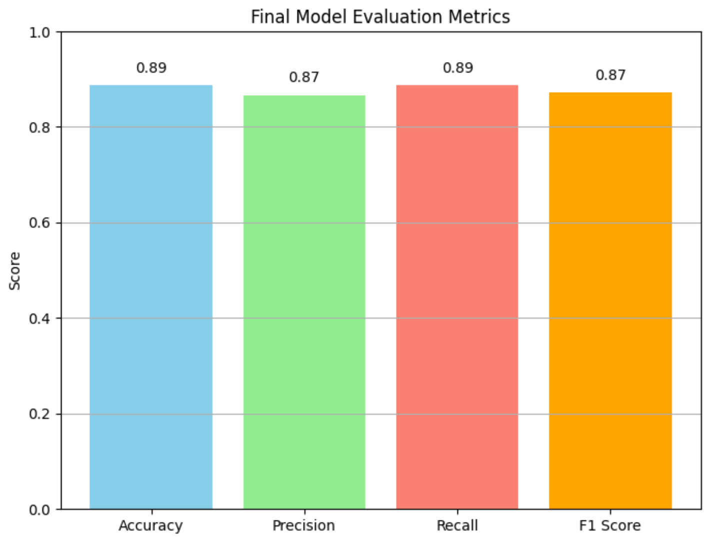
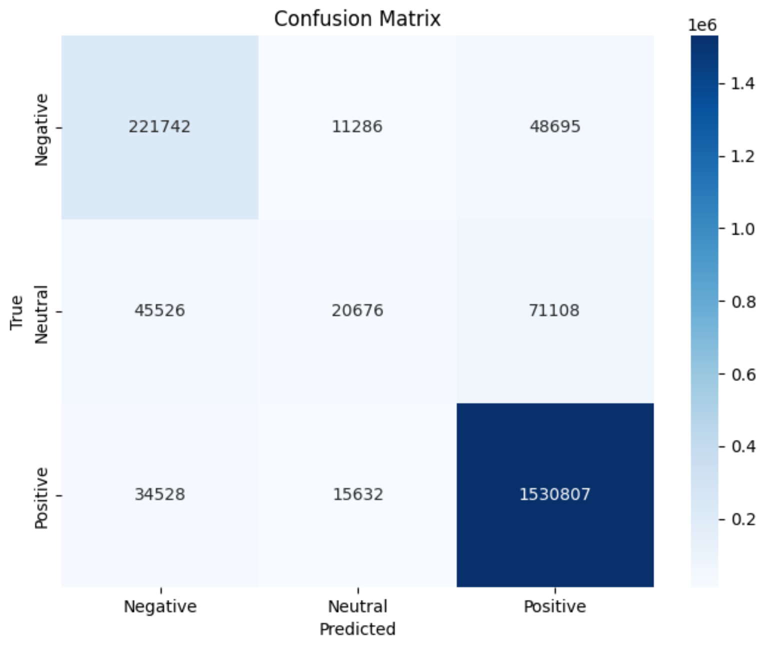
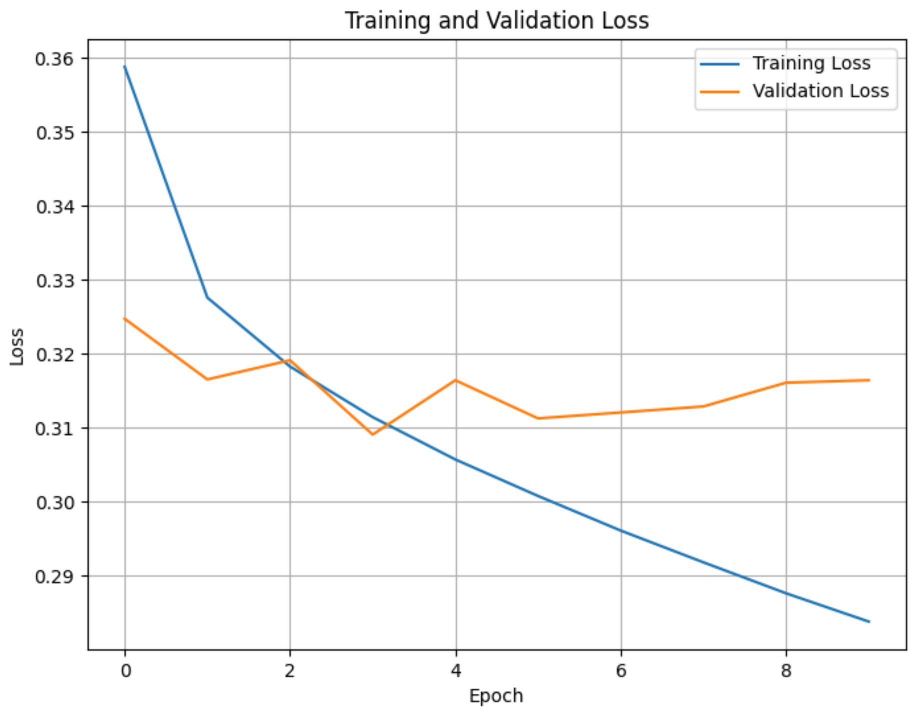


Figure 2: Confusion Matrix



The confusion matrix for the CNN-based sentiment classifier reveals that the model excels at identifying positive reviews—correctly classifying 1,530,807 out of 1,580,967 true positives (≈97% recall) and mislabeling only 34,528 as negative and 15,632 as neutral. Its performance on negative sentiment is respectable but lower, with 221,742 true negatives out of 281,723 (≈79% recall), while 48,695 and 11,286 negative reviews are mistakenly predicted as positive or neutral, respectively. In contrast, neutral reviews pose a significant challenge: of 137,310 true neutral examples, only 20,676 are correctly identified (≈15% recall), whereas 71,108 are misclassified as positive and 45,526 as negative. This pattern underscores strong discrimination between positive and negative language but highlights the model's difficulty in capturing the more subtle, balanced tone of neutral sentiment—likely exacerbated by class imbalance—and suggests that targeted strategies (e.g., reweighting, data augmentation, or specialized neutral–class features) may be needed to bolster neutral-class performance.

Figure 3: Training vs Validation Loss



The training loss for the CNN model steadily declines from roughly 0.36 at epoch 0 to about 0.29 by epoch 9, showing that the network continues to fit the training data throughout all ten epochs. In contrast, the validation loss falls from around 0.32 to its minimum of approximately 0.308 by epoch 3 but then plateaus and even creeps back up to roughly 0.313 by epoch 9. This divergence—where training loss keeps improving while validation loss stalls and slightly increases—suggests the model begins to overfit after the third or fourth epoch. To preserve generalization, one might employ early stopping around epoch 3–4 or introduce stronger regularization (for example, higher dropout, weight decay, or data augmentation) to curb this mild overfitting.

Summary and Conclusions

The CNN-based sentiment classifier I developed proved to be a highly effective model, achieving an F1-score of **0.8713**, accuracy of **88.66%**, precision of **0.8664**, and recall of **0.8866** on a test set of 2 million samples. Training the model on 10 million Amazon Electronics reviews took approximately **3 hours**, with an average of **17 minutes per epoch** over 10 epochs. While the LSTM model slightly outperformed the CNN with a marginal gain in accuracy and F1-score, the CNN offered a robust and computationally efficient alternative, especially suitable for deployment scenarios. This project provided hands-on experience in deep learning model development, optimization, and evaluation at scale. Looking ahead, incorporating transformer-based models like BERT or experimenting with hybrid CNN-RNN architectures could further improve classification performance and contextual understanding.

Code Attribution

All my code in this project is my own work, aside from auxiliary routines provided by teammates. While I leveraged generative AI for scaffolding assistance, no code was directly copied from any external source.