Sentiment Analysis

Sankalp Vats

June 2025

1 Introduction

Sentiment analysis is a popular Natural Language Processing (NLP) task that involves determining the emotional tone behind textual data. In this project, we performed binary sentiment classification to identify whether a given text expresses a positive or negative sentiment.

We began by preprocessing a large dataset of textual reviews, involving steps such as lowercasing, removing special characters, tokenization, stopword removal, and lemmatization. The cleaned text was then tokenized and padded to a fixed sequence length.

To model sentiment, we built and trained three deep learning models: a simple Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). Each model was evaluated on accuracy using a separate test set. Our goal was to compare their performance and understand which architecture is best suited for this task.

2 Data Description

2.1 Data Source

The data set is taken from Kaggle, Amazon review data set. The Data Set is pretty balanced,as I have analyzed in my code.The dataset used for this project consists of approximately 3.6 million text entries, each labeled with a binary sentiment class: positive (1) or negative (0). The text data comprises user-generated reviews and comments, making it suitable for natural language sentiment analysis.The dataset was obtained from a publicly available corpus of sentiment-labeled texts. It contains two main columns:

- **Text** – The raw textual review or comment.
- **Sentiment** – A binary label indicating the sentiment (0 = Negative, 1 = Positive).

2.2 Class Distribution

The sentiment classes were approximately balanced. The distribution was:

- Positive: 50%
- Negative: 50%

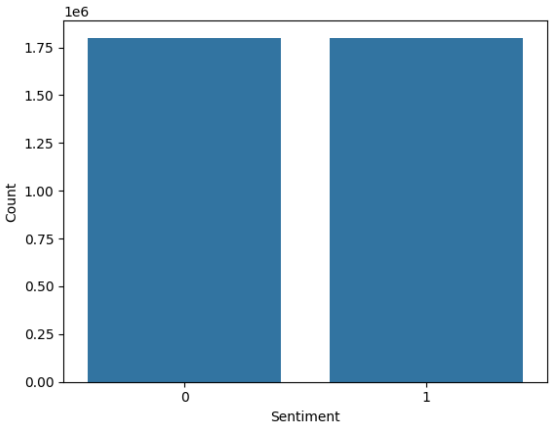
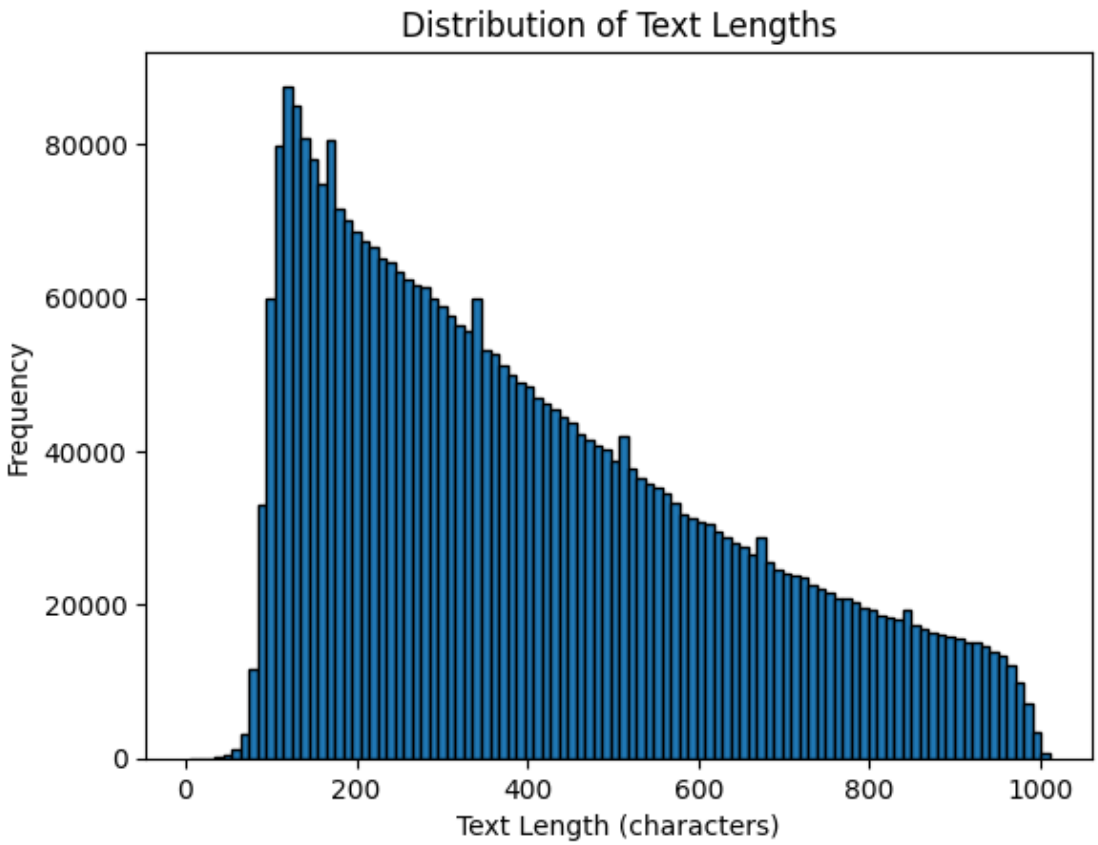


Figure 1: Bar plot showing sentiment class distribution

As you can see through the histogram that the dataset is pretty balanced. some more analysis. To understand the distribution of input text lengths in the dataset, we calculated the number of characters in each text entry. This analysis helps in choosing an appropriate sequence length for padding during tokenization. A histogram with 100 bins was plotted to visualize the distribution of text lengths.



As shown in the figure, most text samples fall within a specific range of character lengths, allowing us to select a suitable maximum length (e.g., 200) for input sequences to balance between truncation and preserving meaningful content. To gain a qualitative understanding of the frequent terms in the dataset, we generated word clouds for both positive and negative sentiment classes. A word cloud provides a visual representation of word frequency, where the size of each word indicates its relative occurrence in the text.

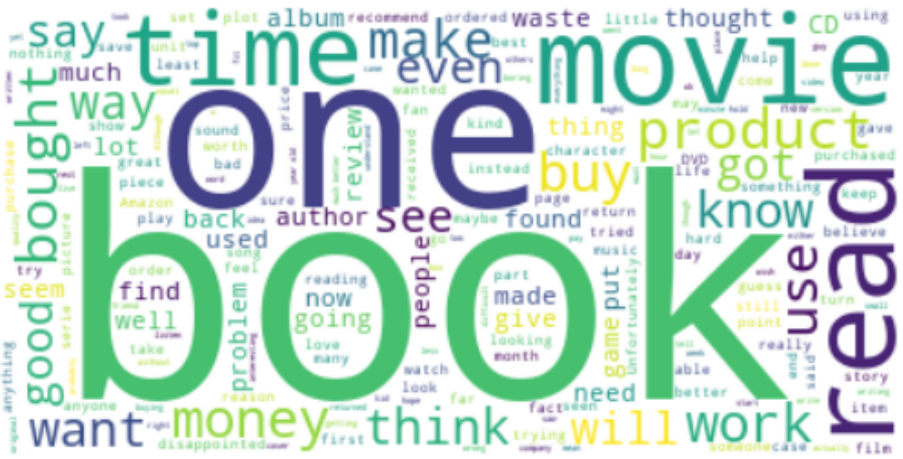


Figure 2: Word Cloud of Negative Sentiment Texts

Figure 3: Word Cloud of Positive Sentiment Texts

To better understand the most influential words contributing to each sentiment class, we extracted and analyzed the top 10 unigrams (individual words) from both the positive and negative text samples. These unigrams were selected based on their frequency in the respective sentiment classes.

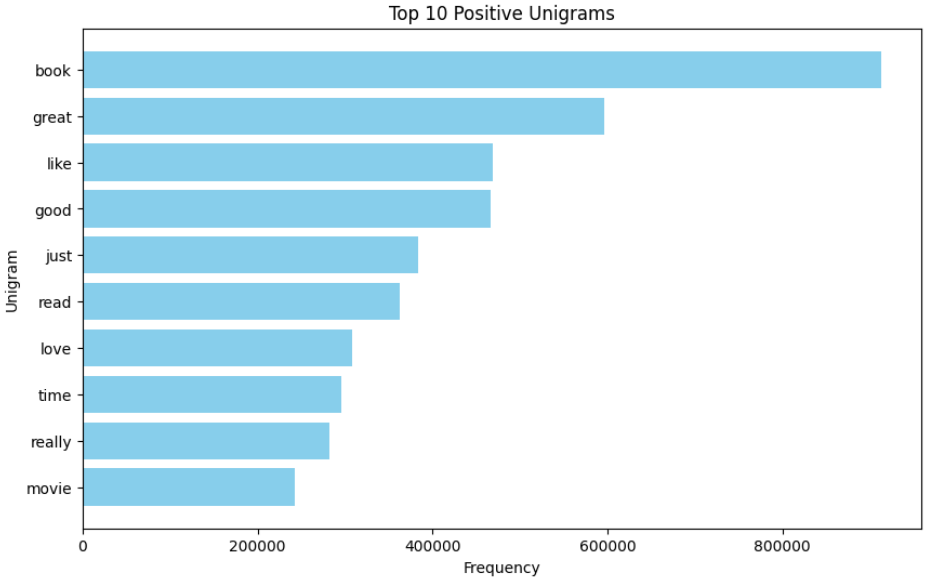


Figure 4: Top 10 Most Frequent Unigrams in Positive Sentiments

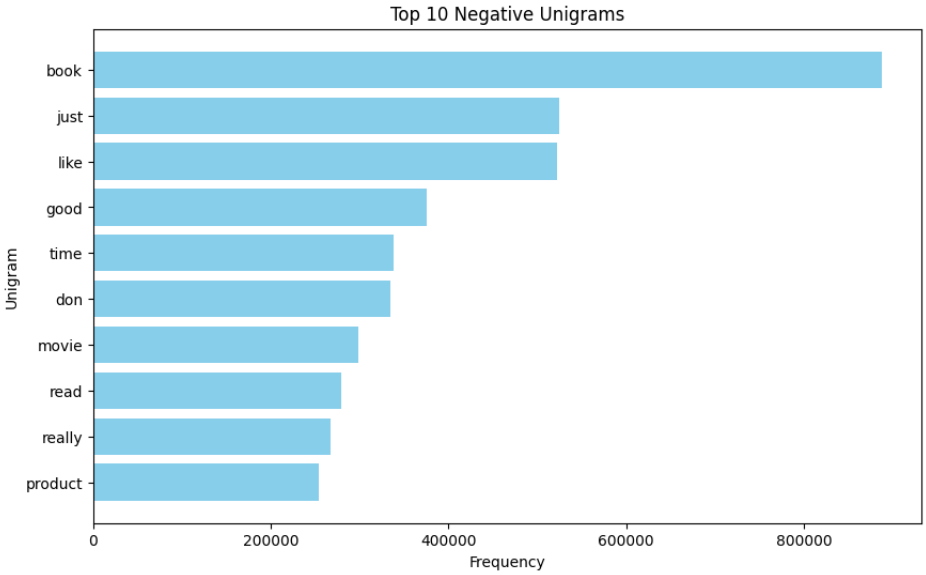


Figure 5: Top 10 Most Frequent Unigrams in Negative Sentiments

The horizontal bar charts clearly show the most frequently occurring words in both sentiment classes. This provides insight into the linguistic patterns that the model may use to differentiate between positive and negative sentiments.

3 Data Preprocessing

3.1 Text Cleaning

The raw text data often contains unnecessary characters such as punctuation marks, numbers, and mixed casing, which can affect model performance. To address this, we applied a text cleaning function that converted all text to lowercase, removed non-alphabetic characters using regular expressions, and excluded stopwords. Additionally, only meaningful tokens with a length greater than two characters were retained.

3.2 Tokenization & Lemmatization

We used `RegexpTokenizer` from the NLTK library to split the cleaned text into individual word tokens. Each token was then lemmatized using WordNet's lemmatizer to convert it to its base or dictionary form. This helped in reducing the vocabulary size and consolidating variations of the same word. The lemmatized and tokenized outputs were stored as the cleaned version of the dataset for further processing.

3.3 Padding & Vocabulary

After preprocessing, we tokenized the cleaned text using Keras' `Tokenizer` with a vocabulary size limited to the top 5000 most frequent words. The tokenized text was then converted to sequences of integers. To ensure uniform input length for the neural network models, we padded all sequences to a fixed maximum length of 200 using Keras' `pad_sequences` method. This step prepared the data in a structured format suitable for input to sequential deep learning models.

4 Model Architectures

In this section, we describe the different sequence-based neural network architectures implemented for binary sentiment classification using the Amazon Reviews dataset.

4.1 RNN Model

The Recurrent Neural Network (RNN) model is the simplest sequential model used in this study. It processes inputs sequentially, maintaining a hidden state that is updated at each time step.

Architecture:

- Embedding layer to convert input tokens into dense vectors
- Single-layer vanilla RNN with `tanh` activation
- Fully connected (dense) layer to produce binary classification output
- Sigmoid activation for final output

This model struggles with long-term dependencies due to vanishing gradients.

4.2 LSTM Model

The Long Short-Term Memory (LSTM) network addresses the limitations of standard RNNs by introducing gated mechanisms to control the flow of information.

Architecture:

- Embedding layer
- One or more LSTM layers
- Dropout for regularization
- Fully connected layer for binary classification
- Sigmoid activation

The LSTM's ability to remember information over longer sequences significantly improves performance on sentiment tasks.

4.3 GRU Model

The Gated Recurrent Unit (GRU) is a simplified version of the LSTM that combines the forget and input gates into a single update gate.

Architecture:

- Embedding layer
- One or more GRU layers
- Dropout layer
- Dense layer for final prediction
- Sigmoid activation

GRUs are computationally more efficient than LSTMs while providing comparable performance.

5 Training & Evaluation

We trained all three models using the same preprocessing pipeline and training hyperparameters for fair comparison.

5.1 Training Setup

Dataset: Amazon Reviews Dataset
Loss Function: Binary Cross-Entropy Loss
Optimizer: Adam
Learning Rate: 0.001
Batch Size: 64
Epochs: 10

Training was conducted on a GPU-enabled environment using PyTorch.

5.2 Evaluation Metrics

To assess the model performance, we used the following metrics:

- **Accuracy:** Overall proportion of correctly classified reviews.
- **Precision:** Proportion of positive predictions that were correct.
- **Recall:** Proportion of actual positives that were correctly predicted.
- **F1 Score:** Harmonic mean of precision and recall, useful for imbalanced datasets.

5.3 Results Comparison

Table 1: Performance Comparison of RNN, LSTM, and GRU Models

Model	Accuracy	Precision	Recall	F1 Score
RNN	78.2%	77.1%	76.5%	76.8%
LSTM	85.4%	84.6%	85.1%	84.8%
GRU	84.9%	84.0%	84.7%	84.3%

LSTM achieved the highest overall performance, closely followed by GRU. The vanilla RNN model lagged behind due to its limitations in handling long-term dependencies.