Sentiment Classification with Neural Text Representation on Amazon Reviews

Loris Giunta
l.giunta1@studenti.unipi.it

Andrea Lepori
a.lepori1@studenti.unipi.it

Mihnea Molnar
m.molnar@studenti.unipi.it

Gemma Ragadini
g.ragadini@studenti.unipi.it

Pan Zhang
p.zhang@studenti.unipi.it

Abstract

This project, developed for the Human Language Technologies course at the University of Pisa during the 2024/2025 academic year, aims to explore the task of sentiment analysis on Amazon product reviews through two distinct approaches. The first approach leverages static embeddings generated by models such as Word2Vec, GloVe, and FastText, combined with traditional Machine Learning algorithms or Deep Learning models (based on RNNs). The second approach relies on contextual embeddings derived from transformer-based models such as BERT and RoBERTa. Several experiments were conducted to compare and better understand the strengths and limitations of the different models, also evaluating the impact of various text preprocessing strategies on overall performance.

1 Introduction

Sentiment analysis is the task of analyzing digital text to determine the emotional tone conveyed, typically categorized as positive, negative, or neutral. The objective of this project is to develop a sentiment classifier for product reviews using neural representations. Specifically, sentiment classification was applied to an Amazon review dataset, following two distinct methodological approaches.

The first approach employs static word embeddings, which represent the tokens in a review as dense, semantically-informed vectors that remain the same regardless of the context in which a token appears. The second approach, by contrast, leverages contextual word embeddings produced by transformer-based encoders. These models are capable of incorporating contextual information about a word's surrounding tokens, allowing them to capture different meanings or nuances of the same word depending on the context in which it occurs.

The approach based on static embeddings is further divided into two conceptually distinct pipelines. In the first pipeline, referred to as "Static Embedding + Traditional ML", a single embedding is generated for each review by aggregating all the individual token embeddings into one single vector. Sentiment classification is then performed using traditional machine learning algorithms that operate on these flattened feature vectors. In the second pipeline, "Static Embedding + RNN", the entire sequence of embeddings that constitutes a review is used as input to sequence-based deep learning models, such as Recurrent Neural Networks (RNNs) and their gated variants.

HLT final project report (2025).

The second, more recent approach, leverages contextual embeddings generated by transformer-based encoders such as BERT and RoBERTa. These pretrained models—either fine-tuned or used with frozen weights—are combined with classification heads to perform the sentiment classification task.

The objective of this project is to evaluate the effectiveness of the proposed approaches for automatic sentiment classification, using appropriate evaluation metrics. The results obtained are discussed and compared across the various models, highlighting their performance, strengths, weaknesses, and potential limitations. Furthermore, the development of an effective and sustainable sentiment prediction model in the domain of product reviews could prove highly beneficial. For companies, it offers a fast and automated tool to gauge customer perception of their products. For consumers, it can support more informed purchasing decisions.

2 Background

Sentiment classification, a core task in sentiment analysis, has evolved substantially over the past decade, transitioning from lexicon-based methods to sophisticated deep learning architectures. Wankhade et al. [2022] provide a comprehensive survey of the current landscape, categorizing sentiment analysis methods into three broad paradigms: lexicon-based, machine learning-based, and deep learning-based approaches.

Lexicon-based methods rely on predefined sentiment dictionaries and rules to infer polarity. Though interpretable and resource-efficient (unsupervised), these approaches often struggle with handling sarcasm, context dependency, and domain specificity. Their limitations have led to increasing reliance on machine learning-based techniques, such as Naive Bayes, Support Vector Machines (SVM), and logistic regression, which learn sentiment patterns from labeled datasets. These approaches offer greater adaptability but still face constraints when dealing with complex linguistic structures. As highlighted by Wankhade et al., hybrid approaches are an emerging trend in literature, combining the strengths of both Machine Learning techniques and Lexicon-Based methods.

Recent advances in deep learning have opened the door to a new era of sentiment classification, addressing many challenges faced by earlier techniques. Neural network models, particularly those leveraging Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNNs), have shown superior performance in capturing contextual and sequential information in text.

Tang et al. [2023] further emphasize the importance of combining semantic-rich representations with context-aware models. In their comparative study, the integration of Word2Vec embeddings with LSTM networks outperformed traditional and shallow machine learning models across multiple sentiment classification benchmarks. Their findings reinforce the view that word embeddings enable models to better capture the underlying sentiment-bearing semantics of words, while LSTM's memory capabilities support contextual sequence modeling.

The introduction of the attention mechanism marked a pivotal shift in sentiment analysis. Unlike sequential models, attention allows models to dynamically focus on the most relevant parts of a text. Transformers, such as BERT and GPT, build upon this by employing multi-head self-attention to capture global contextual relationships in parallel, resulting in state-of-the-art performance in sentiment analysis. These models are pretrained on massive corpora and can be fine-tuned on downstream tasks such as sentiment classification, offering strong performance even with limited labeled data.

This paradigm, known as transfer learning, enables pretrained language models to reuse previously acquired linguistic knowledge and adapt it to new domains or tasks with minimal additional supervision. Models like BERT and RoBERTa provide rich contextual embeddings that surpass static representations in capturing semantic nuances. More recently, parameter-efficient fine-tuning methods (PEFT), such as adapters and LoRA have emerged, allowing effective adaptation of large models without updating all parameters, thus reducing computational cost while maintaining performance.

In summary, modern sentiment classification favors deep learning models with contextual embeddings and attention mechanisms, notably transformer-based architectures like BERT and GPT. Transfer learning and fine-tuning of pretrained models enable strong results even with limited data. Yet, challenges like domain adaptation, multilingual and code-mixed texts, explainability, sarcasm detection and high computational costs remain open research areas.

3 Dataset

The first step in this work, following an in-depth review of the relevant literature and research papers related to our task, was the selection and thorough analysis of a suitable dataset. The dataset employed in this project is the *Amazon Review Full Score Dataset (Version 3, updated September 9, 2015)*[1] . It was created by Xiang Zhang based on the extensive collection of Amazon product reviews originally compiled by McAuley and Leskovec (2013). This dataset contains product reviews along with their corresponding star ratings — i.e., numerical scores from 1 to 5 reflecting user satisfaction.

The dataset is structured as follows:

- *Training Set*: 3,000,000 samples, with 600,000 reviews for each star rating from 1 to 5.
- *Test Set*: 650,000 samples, with 130,000 reviews per rating category.

For this work, *only the training set* was used. Each entry in the dataset includes three fields:

- *Rating*: An integer from 1 to 5 indicating the number of stars assigned by the user.
- *Title*: A short textual field representing the title of the review.
- *Review*: A longer textual field containing the full content of the review.

The data is provided in CSV format, specifically in the files `train.csv` and `test.csv`. An important characteristic of this dataset is its *perfect class balance*, with an equal number of reviews for each rating level. This property makes it a robust benchmark for evaluating models on fine-grained sentiment analysis and review score prediction tasks.

3.1 Dataset Preprocessing

The initial preprocessing step involved cleaning the dataset by detecting and removing null values and duplicate entries. Furthermore, since it was decided to focus on a binary classification task, a new column 'sentiment' was created, this column contains the value 1 (positive sentiment) if the corresponding rating is greater than 3, the value 0 (negative sentiment) if the corresponding rating is less than 3. Rows with rating equal to 3 were discarded because they were considered neutral reviews. Next, we combined the 'Title' and 'Review' fields into a single field named 'Text', which was used as the input for subsequent analyses and modeling. A statistical analysis of the 'Text' column revealed that the average review length is approximately 80 words, but with high variability. Reviews range from as few as 20 words to well over 100 words, with a standard deviation of 43.26, indicating considerable dispersion in review length.

From the full dataset, a subset of 500,000 samples was extracted for experiments involving static embedding approaches. Within this subset, a further reduced sample of 100,000 entries was used specifically for traditional machine learning models that also rely on static embeddings. For contextual embedding approaches, based on transformer models such as BERT and RoBERTa, we worked with multiple subsets of 30,000, 60,000, and 200,000 samples, which were drawn directly from the full dataset.

4 Method

4.1 Static Embeddings Approaches

4.1.1 Text Preprocessing

Starting from the 500,000-sample subset, we designed a preprocessing pipeline specifically oriented toward sentiment analysis. To guide this process, we referred to several relevant papers, including *Agarwal et al. [2011]– Sentiment Analysis of Twitter Data, Columbia University* and *Krouska et al. [2016]- The Effect of Preprocessing Techniques on Twitter Sentiment Analysis, University of Piraeus*.

The main objective of our text preprocessing approach was to investigate whether traditional machine learning models (such as SVM, Logistic Regression, XGBoost, KNN, etc.) and recurrent neural

[1] View the original Dataset

network models (such as LSTM, BiLSTM, GRU etc.) perform better or worse when the embeddings are built on top of preprocessed text as opposed to minimally preprocessed.

The minimal preprocessed text consist in lowercasing and line-break removal.
As for the complex preprocessing pipeline, the following steps were applied:

First, custom tokens were created for certain elements such as emojis (both textual and Unicode-based) and punctuation marks. For example, if an emoji like ' ☺' appears in the dataset, it is automatically replaced with the token 'HAPPY' using regular expressions (Regex). Similarly, other emotional tokens such as 'ANGRY' and 'SAD' were introduced based on the emoji detected.

The same principle was applied to punctuation marks. For instance, when encountering elements like '!' or '?!', they were replaced with tokens such as 'EXCLAMATION_MARK' and 'SURPRISE', respectively. Additional punctuation tokens were also introduced based on their relevance to sentiment, as summarized in the table in appendix (Table 1).

Moreover, when a word or a group of words appears in uppercase, we append the token 'ALL_CAPS' at the end of the word or phrase to retain the emphasis information that could contribute to sentiment understanding.

The second part of the preprocessing pipeline included the following steps:

1. Expansion of contractions
2. Lowercasing the entire review
3. Removal of URLs and mentions
4. Removal of special characters and digits
5. Stopword removal

These steps were implemented using *Regular Expressions (Regex)* and the `NLTK` library in Python. Regarding stopword removal, a custom stopword list was created to retain words that are generally considered stopwords but may carry important sentiment information. Examples include: 'not', 'no', 'never', 'nobody', and similar terms.

The final step of the preprocessing phase involved applying lemmatization using the `spaCy` library. We opted for lemmatization over stemming, even though it is computationally more expensive, because it provides better grammatical accuracy and preserves the syntactic integrity of the text — a crucial aspect in sentiment analysis tasks.

4.1.2 Static Embedding + Traditional ML Approach

In the first methodological approach, we used static word embeddings combined with traditional machine learning (ML) classifiers to perform sentiment classification. The pipeline (shown in figure 1) is structured as follows:

1. *Input Text and Preprocessing*:
the input consists of review texts extracted from the raw dataset. Two levels of preprocessing were explored:
   - *Minimal Preprocessing*: basic text normalization including lowercasing and line breaks removal.
   - *Heavier preprocessing*: the one discussed in subsection 4.1.1.

2. *Tokenization*:
the preprocessed text is tokenized into individual words.

3. *Static Word Embeddings*:
tokens are mapped to pretrained GloVe embeddings (Global Vectors for Word Representation). Each word is represented as a fixed dimensional dense vector that captures semantic relationships between words. These embeddings are static, meaning each word has a single vector regardless of its context in a sentence. This allows the model to leverage general semantic knowledge learned from large corpora, even though it does not account for word sense disambiguation in different contexts.

4. *Embedding Pooling*:
   to generate a single fixed-length vector for each review, two pooling strategies were tested:
   - *Average pooling*: computes the mean of all word vectors in a review.
   - *Element-wise maximum pooling*: selects the maximum value per dimension across all word vectors in a review.

   These strategies condense variable-length sequences into fixed-size input suitable for classical ML models.

5. *Classification*:
   The resulting review embeddings are fed into a variety of traditional ML classifiers implemented with scikit-learn. The classifiers explored include:
   - *Support Vector Machine (SVM)*
   - *Logistic Regression (LR)*
   - *Multi-Layer Perceptron (MLP)*
   - *Random Forest (RF)*
   - *k-Nearest Neighbors (KNN)*
   - *XGBoost*

These baseline techniques are consistent with those discussed in Wankhade et al. [2022] which highlights their relevance and frequent application in Machine Learning approach to sentiment classification tasks.

This approach allowed us to assess the performance of different ML model architectures and preprocessing strategies on the task of binary sentiment prediction.

This static embedding pipeline serves as a strong baseline and a historical example that is representative of a well-established NLP methodology that combines unsupervised word representation with classical supervised learning techniques.

4.1.3 Static Embedding + RNN approach

To explore the effectiveness of recurrent architectures for sentiment classification, we implemented a series of models following a consistent pipeline (figure 1 in Appendix), while varying both the recurrent layer type and the classifier head.

Raw text reviews were preprocessed according to the two different types of preprocessing exposed before. The cleaned text was tokenized using the `Tokenizer` utility from Keras. The tokenized sequences were padded or truncated to a fixed maximum length, selected based on the token-per-review distribution observed in the dataset.

Embedding Layer

Each review was represented as a *sequence of static embedding vectors*. Two main strategies were employed:

- **Random initialization:** Using the Keras `Embedding` layer initialized randomly and trained from scratch.
- **Pretrained embeddings:** Leveraging pretrained vectors such as GloVe, Word2Vec and fastText. These embeddings were optionally fine-tuned during training by setting `trainable=True`.

Recurrent Layer

The embedded sequences were processed by different types of recurrent layers:

- *Simple Vanilla RNN*
- *Gated Recurrent Unit (GRU)*
- *Long Short-Term Memory (LSTM)*

- *Bidirectional LSTM (BiLSTM)*
- *Stacked BiLSTM*

These layers were configured to output either the final hidden state or a pooled representation.

Classification Head

Each recurrent layer was followed by a classification head, which was a multi-layer perceptron (MLP) or convolutional layers followed by dense layers (CNN+Dense). The final output was a sigmoid-activated neuron producing a binary sentiment classification.

The use of these architectures and combination of layers for this task is shown by Tang et al. [2023].

Implementation Tools. All models were implemented using `Keras`, which enabled fast prototyping and seamless GPU utilization on Colab.

4.2 Contextual Embeddings (Transformer based) Approaches

4.2.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) was introduced by researchers at Google in October 2018. It is a bidirectional Transformer model pretrained on large-scale unlabeled text using two self-supervised objectives: predicting masked tokens within a sentence (Masked Language Modeling) and determining whether one sentence follows another (Next Sentence Prediction). BERT is highly versatile, as its learned language representations can be fine-tuned for various downstream NLP tasks, such as sentiment analysis, by adding a task-specific layer or head, as shown by Almeida et al. [2022]. Architecturally, BERT is built entirely on the Transformer encoder and consists of 12 stacked Transformer blocks. In this work, we use the BERT-base-uncased version, which includes 110 million parameters, 12 attention heads, 12 hidden layers, a hidden size of 768, vocabulary tokens 30,522, an intermediate size of 3,072, and uses the GELU activation function.

Introduction to Model Architecture In the contextual embedding approach, we employ BERT combined with a fully connected neural network (**FCNN**) to address the binary sentiment classification task. The overall model architecture is as figure 2 in the appendix.

Introduction to classifier We then add a classification head on top of the BERT model, consisting of three fully connected layers. The first layer projects the BERT hidden representation to 512 units, followed by BatchNorm1d (512), a GELU activation function, and Dropout with a rate of 0.5. The second layer reduces the dimensionality from 512 to 256 units, and is followed by BatchNorm1d (256), a SiLU activation, and Dropout with a rate of 0.3. The final layer maps the 256-dimensional vector to the number of output classes, which is 2 in our binary classification setting.

Introduction to Data and Data Preprocessing We also utilize the Amazon Review dataset, which contains approximately 3 million reviews, for our sentiment classification task. Prior to training, several preprocessing steps were applied. First, we performed ratings mapping: reviews with ratings less than 3 were labeled as 0 (negative), while those with ratings greater than 3 were labeled as 1 (positive). We then sampled 1% of the entire dataset as a working subset, from which 80% was used for training and 20% for validation. To evaluate the model's generalization capability, we sampled another 1% from the remaining 99% of the original dataset to serve as the test set. Finally, we applied text normalization techniques, such as replacing missing titles (NaNs) with the string 'NULL', converting all text to lowercase, and removing extra whitespace.

Model Training The model was trained using binary cross-entropy loss and the AdamW optimizer with a batch size of 16. We applied early stopping with a patience of 3 during fine-tuning to prevent overfitting. To optimize different parts of the model effectively, we used a lower learning rate (1e-5) for the BERT encoder and a higher learning rate (1e-3) for the classifier head. The best model was selected and saved based on the highest validation accuracy achieved during training.

4.2.2 RoBERTa

RoBERTa (A Robustly Optimized BERT Pretraining Approach), developed by Meta AI, is a transformer-based model that improves upon BERT through a series of optimizations. It retains the 12-layer transformer architecture but introduces the following key modifications:

- **No Next Sentence Prediction (NSP)**: Unlike BERT, RoBERTa removes the NSP objective, as it was found to provide minimal benefit while complicating training. This allows the model to fully dedicate its training budget to masked language modeling.
- **Dynamic Masking**: RoBERTa dynamically selects the 15% of masked tokens at each training epoch, instead of using a fixed mask like BERT. This exposes the model to a more diverse set of contexts, improving its ability to generalize.
- **Larger Training Corpus**: RoBERTa is trained on 160GB of ten times the volume used for BERT, providing broader linguistic coverage.
- **Tokenizer**: It uses byte-level Byte-Pair Encoding (BPE), which operates on raw UTF-8 bytes.

Dataset Preparation. We started with the full raw dataset. Ratings 1–2 were mapped to class 0 (negative), and ratings 4–5 to class 1 (positive), excluding neutral reviews (rating = 3). We used stratified sampling to maintain class balance and generated two subsets:

- 30,000 reviews for a pilot run.
- 200,000 reviews for main experiments.

Each subset was split into 50% training, 20% validation, and 30% internal testing.

Text Preprocessing and Tokenization As for BERT, preprocessing was intentionally kept lightweight to reduce distortion of the original input semantics. The steps included dropping NaN values, merging title and text, and replacing "\n" with spaces.

A sample of 10,000 reviews was tokenized to determine sequence length statistics. As we can see in Figure 5 in the appendix, the 95th percentile was found to be approximately 200 tokens. Based on this, we set MAX_LEN = 205, which trims only the longest 5% of reviews and reduces padding for the rest. Tokenization was performed using RoBERTa's byte-level BPE tokenizer. To enhance efficiency and reproducibility, we cached the input_ids, attention_mask, and label arrays to .npz files. These are reused in subsequent training runs, eliminating the need to re-tokenize and ensuring input consistency across experiments.

Data Pipeline The training pipeline was implemented using TensorFlow's *tf.data* API and structured as follows:

- **From generator**: a Python generator lazily loads pre-tokenized .npz files and yields (input_ids, attention_mask, label) tuples to Tensorflow API which dynamically builds a *tf.data.Dataset*.
- **Dynamic padding (padded_batch)**: each batch is padded only to the maximum sequence length within that batch, saving GPU memory compared to global padding.
- **Prefatching**: *tf.data.AUTOTUNE* dynamically adjusts buffer sizes to maximize GPU utilization without unnecessary memory consumption.

Model Architecture The core of the model is the *facebook/roberta-base* encoder, which serves as the backbone of the architecture. It includes 12 transformer layers with hidden states of 768 dimensions (embeddings size). Each layer contains a feed-forward sublayer with 3,072 units and uses the GELU activation function. The entire encoder consists of approximately 125 million parameters.

To adapt RoBERTa to our binary sentiment classification task, we added a custom classification head. This head begins with a dense layer that projects the 768-dimensional pooled output down to 256 units, followed by a dropout layer with a probability of 0.3. This is then further reduced to 64 units through another dense layer and finally to a single output neuron for binary classification. Overall,

this head adds only around 213,000 trainable parameters, keeping the model compact, especially when the encoder is frozen and only the head is trained.

Training was carried out using the binary cross-entropy loss function and the AdamW optimizer, with a batch size of 32. To avoid overfitting and accelerate convergence, we applied early stopping. We also enabled *restore_best_weights*, which ensures that, at the end of training, the model reverts to the version that achieved the best validation loss.

5 Experimental analysis

5.1 Experimental settings

5.1.1 Experimental Settings for Traditional ML

To evaluate the impact of different text preprocessing strategies on traditional machine learning models, we conducted experiments using a 100,000-instance subsample of the full dataset. The data was split into training and test sets using an 80/20 ratio, ensuring a consistent and fair basis for evaluation across all experiments.

This binary classification task was carried out under two preprocessing configurations:

- **Experiment A:** Employed *minimal and coarse-grained preprocessing*
- **Experiment B:** Used *more sophisticated and accurate preprocessing techniques*, as detailed in the earlier sections of this paper.

Both experiments were designed to be comparable under identical modeling conditions. Specifically, we used the same pretrained word embeddings, fixed embedding dimensionality, and identical train/test partitions across the two setups.

To assess model performance, we report standard evaluation metrics: **Accuracy**, **Precision**, **Recall**, and **F1 score**. Additionally, we consider **processing time** to evaluate the computational cost associated with each model.

5.1.2 Experimental Settings for RNN

For experiments involving Recurrent Neural Networks (RNNs), we used a 500,000-instance subsample of the dataset, which was again balanced across the target classes. The data was partitioned into training and test sets with an 80/20 split.

Model performance was evaluated on the internal test set using the standard classification metrics: **Accuracy**, **Precision**, **Recall**, and **F1 score**.

Training was conducted using the Adam optimizer to minimize binary cross-entropy loss. To prevent overfitting, we employed **early stopping** by monitoring the validation loss with a predefined patience threshold.

Unless explicitly stated otherwise, in the table with results, minimal text preprocessing was applied to the input data.

In experiments labeled with **'+PreProc'**, the more extensive text preprocessing pipeline, previously described, was applied.

5.1.3 Experimental Settings for BERT

Due to limited computational resources, we conducted only a single experiment for the BERT part using an NVIDIA A100 GPU. The model was trained on 60,000 samples for 10 epochs, with the BERT fully trainable.

5.1.4 Experimental Settings for RoBERTa

We designed four controlled experiments to isolate the effects of data size, encoder fine-tuning, and training duration:

1. **Experiment A** (pilot): 30k samples, encoder trainable, 3 epochs.
2. **Experiment B**: 200k samples, encoder trainable, 3 epochs.
3. **Experiment C**: 200k samples, encoder frozen, 5 epochs
4. **Experiment D**: 200k samples, encoder frozen, 20 epochs.

Each experiment modifies a single variable: A→B tests data scaling; B→C isolates the effect of freezing the encoder; C→D measures the impact of extended training with a frozen encoder.

As with the BERT experiment, these experiments were run on Colab Pro with an NVIDIA A100 GPU.

5.2 Results

5.2.1 Results of Traditional ML

As shown in Table 2, applying heavier preprocessing in Experiment (B) led to only marginal improvements or, in some cases, no gain compared to Experiment (A). For instance, SVM, Logistic Regression, KNN and XGBoost showed slight increases in accuracy, the last only when combined with average pooling, while other models like MLP and Random Forest remained largely unchanged. Overall, average pooling consistently outperformed max pooling across nearly all models, suggesting it captures more representative features for this task. The results highlight that while preprocessing can help slightly, the choice of pooling strategy has a more noticeable impact on performance.

Among the models evaluated, MLP achieved the highest accuracy (0.80) with average pooling in Experiment (A), as well as SVM in Experiment (B) with the same pooling strategy. In contrast, KNN consistently produced the lowest accuracy, particularly with max pooling, likely due to its sensitivity to high-dimensional representations. Regarding training time, models such as Logistic Regression and XGBoost were notably fast, completing in a few seconds, whereas SVM and MLP required significantly more time. This suggests a trade-off between performance and computational efficiency when choosing traditional models for text classification.

5.2.2 Results of RNN

As shown in Table 3, RNN-based models achieved excellent performance, with several configurations reaching F1 scores above 0.93. The best-performing model was the BiLSTM with minimal preprocessing, achieving an F1 score of 0.935, followed closely by BiLSTM + CNN (0.933), GRU (0.932), and Stacked BiLSTM (0.931). Notably, these high scores were attained using only minimal text preprocessing. Models leveraging pretrained embeddings such as GloVe, FastText, and Word2Vec also performed strongly, though they did not significantly outperform simpler configurations, indicating that architectural choices played a more critical role than embedding source.

Interestingly, adding heavier preprocessing did not improve and in many cases slightly degraded performance. For example, +PreProc + BiLSTM dropped to an F1 of 0.906, and other models showed similar or greater declines. This suggests that the RNN architectures, particularly with sufficient capacity and training data, are robust enough to learn effective representations from raw or lightly processed text.
Training time varied across models, with Vanilla RNN and GRU offering faster runtimes (under 1.5 minutes), while more complex architectures like BiLSTM and Stacked BiLSTM required in some cases over 4 minutes. However, the performance gains achieved by deeper models generally justify the added computational cost.

5.2.3 Results of BERT

Table 4 in Appendix presents the overall performance of the BERT plus a fully connected neural network (**FCNN**) model trained on 60,000 samples. The model achieved a training accuracy of 98.57%, validation accuracy of 93.73%, and test accuracy of 93.98%, indicating good generalization ability. Figure 3 and Figure 4 further illustrate the model's behavior over training epochs. The accuracy graph shows a steady improvement and convergence, while the loss graph demonstrates a consistent decrease in training loss and a clear stabilization of validation loss, confirming the model's convergence and effectiveness in learning from data.

5.2.4 Results of RoBERTa

Table 5 in Appendix summarizes the four experiments conducted with RoBERTa + FCNN, designed to isolate the effects of data size, encoder fine-tuning, and training duration.

A→B: Increasing the dataset from 30k to 200k improved test accuracy (94.5% → 95.7%), probably due to reduced overfitting and better generalization, at the cost of longer training time (7.5 → 41 min).

B→C: Freezing the encoder reduced training time (41 → 35 min) but led to a drop in terms of accuracy (95.7% → 86.3%), highlighting the benefit of fine-tuning.

C→D: Extending training from 5 to 20 epochs with a frozen encoder slightly improved accuracy (86.3% → 89.4%), but required ~4× more time (35 → 135 min).

Among all configurations, **Experiment B** (200k, fine-tuned) showed the best balance between accuracy and efficiency

6 Discussion

6.1 Discussion of Static Embedding Approach

The deep learning approach based on RNN architectures consistently achieved superior performance, significantly outperforming traditional machine learning models. Notably, both approaches performed well with minimal text preprocessing, and heavier preprocessing did not necessarily lead to consistent improvements, occasionally resulting in slight performance degradation. This suggests that both model families are capable of extracting meaningful features from relatively raw text. Furthermore, the combination of static word embeddings with traditional ML models and RNNs remains a simple yet effective solution, demonstrating that even well-established methods can yield competitive results in modern NLP tasks.

6.2 Overall Discussion

The following summary provide a high-level comparison of the various approaches explored, focusing on both accuracy and training efficiency.

Traditional machine learning models based on static embeddings, trained on 100,000 examples and executed on Colab's CPU, were simple to implement and train, with training times ranging from 2 to 14 minutes depending on the algorithm. However, they achieved limited performance, approximately 80% accuracy, due to the lack of contextual and sequential information.

RNN-based models, combined with static embeddings, performed significantly better. Trained on a larger dataset of 500,000 examples, they reached about 93.5% accuracy. Their training times remained impressively low (from 40 seconds to 4.5 minutes) when executed on a standard Colab GPU.

Moving to contextual embedding models, BERT was trained on a dataset of 60,000 examples (30,000 for training) and achieved 94% accuracy with a training time of 14 minutes. This marked a clear improvement over RNNs, despite using a smaller dataset.

Finally, RoBERTa was evaluated on two configurations: one with 30,000 examples and another with 200,000 (using a 50/20/30 split). These correspond to 15,000 and 100,000 training samples respectively. The model reached 94.5% and 95.7% test accuracy, slightly outperforming BERT. Training times ranged from 7.5 to 41 minutes, depending on data size. Although these models required access to a Colab Pro A100 GPU, the observed performance improvements often justify the additional computational cost.

We also tested RoBERTa with a frozen encoder. Although training became faster per epoch, performance peaked at 89.4%, even after 20 epochs. This confirms that fine-tuning is essential to fully leverage transformer-based architectures.

In summary, while fine-tuned contextual models deliver the best trade-off between accuracy and efficiency, RNN-based models remain a strong alternative when computational resources are limited.

7 Conclusions

This work presented a comprehensive study on the task of sentiment classification using neural text representations on Amazon product reviews. The project unfolded through several key phases, beginning with an in-depth review of the relevant literature and state-of-the-art approaches. This analysis guided the selection of suitable methodologies and helped define the application domain for our sentiment classifier.

Following this, we designed and implemented two distinct methodological pipelines. The first relied on traditional machine learning models and RNN-based architectures combined with static word embeddings, an approach rooted in earlier NLP practices. The second adopted a more modern strategy, leveraging pretrained Transformer-based encoders to generate contextual embeddings, aligning with current trends in deep learning for NLP.

Careful attention was devoted to dataset selection, understanding, and text preprocessing, which was tailored for sentiment classification. We conducted extensive experiments to evaluate and compare the performance of these models under different preprocessing regimes, following best practices in machine learning evaluation and employing standard metrics.

Despite computational constraints, all planned models were successfully trained and tested. The experimental results revealed insightful trends, including the robustness of nearly all models with minimal preprocessing, the relatively limited impact of heavier text preprocessing on model performance and the very excellent performance reached by transfomer-based models. The findings also highlighted performance trade-offs between traditional and deep learning models under varying resource conditions, offering guidance on model selection depending on practical constraints.

Many of the implemented models achieved strong performance, demonstrating their potential for real-world deployment in sentiment analysis applications. Future work could explore model generalization across domains and multilingual or code-mixed text, or fine-tuning Transformer models with Parameter-Efficient techniques.

References

Apoorv Agarwal, Boyi Xie, Ilya Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38. Association for Computational Linguistics, 2011.

Tales Almeida, José Souza, Antônio Braga, and Teresa Ludermir. A bert framework to sentiment analysis of tweets. *Procedia Computer Science*, 199:395–402, 2022.

Anastasia Krouska, Christos Troussas, and Maria Virvou. The effect of preprocessing techniques on twitter sentiment analysis. *University of Piraeus, Department of Informatics*, 1(1):1–8, 2016. Technical Report.

Haodong Tang, Nan Zhang, Xinyi Yu, Tengze Mao, and Lidong Wang. Enhancing sentiment analysis with word2vec and lstm: A comparative study. *Journal of Basic and Applied Research International*, 29(3):8342, 2023. doi: 10.56557/JOBARI/2023/v29i38342.

Mayur Wankhade, Annavarapu Chandra Sekhara Rao, and Chaitanya Kulkarni. A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 2022. doi: 10.1007/s10462-022-10144-1.

A Appendix

Table 1: Special tokens used in place of punctuation during preprocessing

Punctuation	Replacement Token
`...`	`ELIPSIS`
`!!!` (or more)	`STRONG_EXCLAMATION`
`???` (or more)	`STRONG_QUESTION`
`?!` or `!?`	`SURPRISE`
`?`	`QUESTION_MARK`
`!`	`EXCLAMATION_MARK`

Table 2: Accuracy and training time of traditional ML models in Experiment (A) and (B), using average and max pooling strategies

Experiment	Model	Accuracy (Avg)	Time (Avg)	Accuracy (Max)	Time (Max)
A	SVM	0.79	9 min	0.71	13 min
	Logistic Regression	0.78	5 s	0.68	5 s
	MLP	0.80	5 min	0.70	10 min
	Random Forest	0.76	2 min	0.72	1 min
	XGBoost	0.77	10 s	0.78	8 s
	KNN	0.70	10 s (test)	0.61	12 s (test)
B	SVM	0.80	9 min	0.72	14 min
	Logistic Regression	0.79	2 s	0.69	2 s
	MLP	0.79	2 min	0.69	8 min
	Random Forest	0.76	2 min	0.73	1 min
	XGBoost	0.79	10 s	0.78	8 s
	KNN	0.71	12 s (test)	0.62	12 s (test)

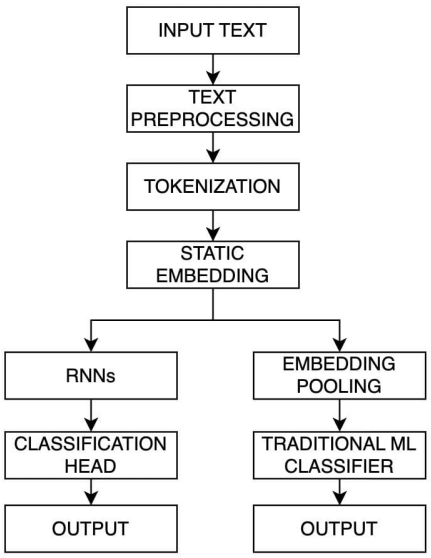

Figure 1: Pipeline of the static embedding approach using traditional ML (right) and RNN (left). Shared preprocessing steps are shown at the top.

Table 3: Performance of RNN-based models under various preprocessing and embedding settings

Model	Accuracy	Precision	Recall	F1 Score	Training Time
Minimal Preprocessing					
BiLSTM	0.935	0.939	0.931	0.935	4 min 30 s
BiLSTM + CNN	0.932	0.923	0.942	0.933	3 min
GRU	0.932	0.931	0.932	0.932	1 min 15 s
LSTM	0.930	0.937	0.923	0.930	1 min
Vanilla RNN	0.905	0.903	0.907	0.905	1 min 30 s
Stacked BiLSTM	0.931	0.928	0.934	0.931	4 min 30 s
+ Pretrained Embeddings					
GloVe + BiLSTM	0.931	0.941	0.920	0.930	4 min 15 s
FastText + BiLSTM	0.930	0.927	0.935	0.931	3 min 45 s
W2V + LSTM	0.930	0.930	0.930	0.930	2 min 30 s
+Advanced Preprocessing (+PreProc)					
+PreProc + BiLSTM	0.905	0.895	0.918	0.906	2 min
+PreProc + BiLSTM + CNN	0.902	0.910	0.893	0.901	1 min
+PreProc + GRU	0.901	0.892	0.911	0.902	40 s
+PreProc + LSTM	0.901	0.900	0.902	0.901	38 s
+PreProc + Vanilla RNN	0.884	0.891	0.875	0.883	33 s
+PreProc + Stacked BiLSTM	0.903	0.908	0.896	0.902	1 min
+PreProc + GloVe + BiLSTM	0.903	0.895	0.912	0.903	2 min
+PreProc + FastText + BiLSTM	0.902	0.887	0.921	0.903	2 min
+PreProc + W2V + LSTM	0.910	0.910	0.910	0.910	2 min 30 s

Table 4: Performance of BERT + FCNN on a 60k-sample dataset

Model	Total Samples	Train Acc.	Val. Acc.	Test Acc.	Time (min)
BERT + FCNN	60,000	98.57%	93.73%	93.98%	14

Table 5: Summary of RoBERTa experiments

Experiment	Total Samples	Encoder	Epochs	Val Acc.	Test Acc.	Time (min)
A	30k	Trainable	3	94.1%	94.5%	7.5
B	200k	Trainable	3	95.5%	95.7%	41
C	200k	Frozen	5	83.2%	86.3%	35
D	200k	Frozen	20	89.0%	89.4%	135

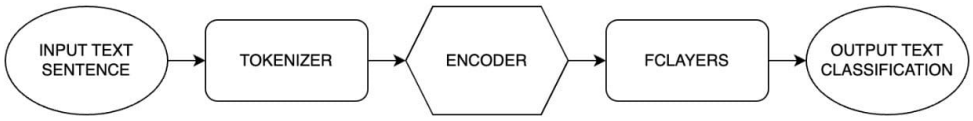

Figure 2: Architecture of the BERT + FCNN pipeline

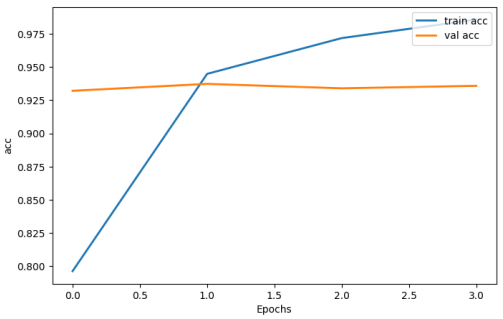

Figure 3: Validation accuracy across epochs for BERT

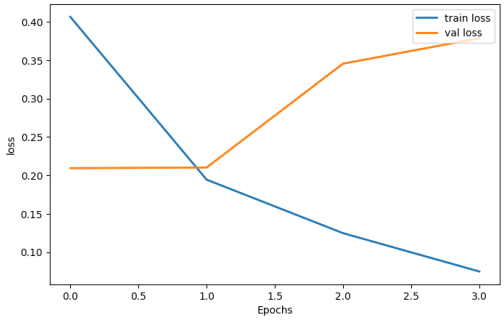

Figure 4: Validation loss across epochs for BERT

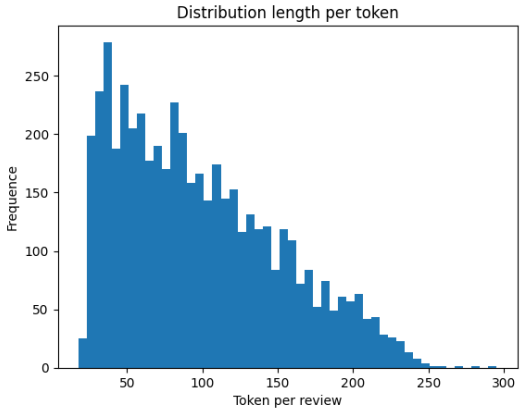

Figure 5: Distribution of review lengths (in tokens)