Sentiment Analysis of Movie Reviews

Rohan Roy and Sarthak Gangwal

Department of Mathematics,Indian Institute Of Technology,Delhi

February 9, 2025

Abstract

Sentiment analysis of movie reviews provides valuable insights into public opinion on films. In this paper, we present a comprehensive sentiment analysis of movie reviews using a range of models, from traditional approaches like Logistic Regression to state-of-the-art Transformer models. After thorough data analysis and preprocessing, we implement and compare multiple models to evaluate their effectiveness. Our results show that the weighted averaging ensemble of Transformer models outperforms all other approaches, achieving the highest accuracy.

Keywords: Review, sentiment, Machine Learning, Transformer

1 Introduction

Sentiment Analysis aims to extract subjective information from text, allowing us to determine the underlying sentiment of a given statement. Analyzing movie reviews helps to estimate the general reception of a film, providing insights into audience opinions. In this report, we perform sentiment analysis on movie reviews using a variety of Machine Learning models, ranging from traditional methods to state-of-the-art Transformer-based approaches.

Processing textual data presents several challenges, including variations in length and structure, as well as the presence of nuanced expressions such as sarcasm. Addressing these complexities is crucial for improving model performance.

This paper is organized as follows: Section 2 describes the dataset used in this study. Section 3 outlines the preprocessing steps applied to the data. Section 4 presents an analysis of the dataset. Section 5 details the approaches taken for the non-competitive part of the study, while Section 6 covers the competitive segment. In Section 7, we compare the performance of different models, followed by conclusions in Section 8.

2 Dataset Description

The dataset consisted of 417,057 entries, with each entry containing a review and its corresponding class label. Reviews were categorized into four classes: A, B, C, and D, where A represented the most positive reviews and D represented the most negative ones. For the non-competitive part, this four-class classification was used as is. However, for the competitive part, the dataset was modified by merging classes A and B into a single "AB" class (representing positive reviews) and classes C and D into a "CD" class (representing negative reviews), reducing the problem to binary classification. The test dataset contained 48,000 reviews for evaluation.

3 Preprocessing Steps

The dataset contains raw text taken from movie reviews, which had to be processed to improve sentiment analysis results. The following preprocessing steps were applied:

3.1 Text Normalization

Lowercasing: All text was converted to lowercase since case distinction is generally unnecessary in sentiment analysis. For example, "Good" and "good" carry the same meaning. However, uppercase text can sometimes convey emphasis, such as in “This was good” vs. “This was GOOD,” where the latter indicates stronger positive sentiment.

3.2 Cleaning and Standardization

Removal of HTML and URLs: These elements do not contribute to sentiment and were removed.

Replacing chat words with standard meanings: Social media slang and abbreviations were expanded (e.g., “gg” to “good game” and “rizz” to “charisma”) to ensure proper embeddings and accurate sentiment interpretation.

Handling numbers: Numbers were often used in rating contexts, such as “I rate it 4/5.” These fractions were evaluated, and based on their value, replaced with corresponding sentiment words:

- $x \geq 0.8 \rightarrow$ "awesome"
- $0.6 \leq x < 0.8 \rightarrow$ "good"
- $0.5 \leq x < 0.6 \rightarrow$ "neutral"
- $0.2 \leq x < 0.5 \rightarrow$ "bad"
- $x < 0.2 \rightarrow$ "worst"

An exception was made for the fraction ’9/11’ as there are a lot of references to the event of 9/11 like ‘This is the best movie on 9/11’ .In such cases it will not be wise to replace something like ‘9/11’ with ‘awesome’

Converting numbers to words: After transformation, numeric values were converted into words, e.g., “I rate it 5 stars” became “I rate it five stars,” as NLP models often struggle with digit-based numbers.

3.3 Emoji and Special Character Processing

Handling emojis and emoticons: Emojis and emoticons contribute to sentiment but are difficult for models to process in raw form, so they were replaced with descriptive text (e.g., “:)” to “smiling face”).

Removing contractions: Common contractions (e.g., "don’t" to "do not") were expanded.

Removing punctuation: Punctuation was removed in most cases, except when it contributed to sentiment (e.g., "This is great" vs. "This is great!").

Removing special characters: Non-alphanumeric characters (except underscores) were removed.

3.4 Stopword Removal and Lemmatization

Removing stopwords: Words that do not contribute to sentiment (e.g., "the," "is," "and") were removed.

Lemmatization: Words were reduced to their base form (e.g., "running" to "run") to improve consistency.

These preprocessing steps ensured that the text was clean, normalized, and more interpretable for sentiment analysis models.

Figure 1: Wordcloud before preprocessing

4 Exploratory data analysis

4.1 Value counts for classes

In this dataset, classes A (137,662) and B (168,936) form the majority, while classes C (90,116) and especially D (20,344) are in the minority. This imbalance can create significant challenges in a classification problem. Since the model is exposed to a larger number of samples from A and B, it may become biased toward these classes, leading to poor generalization for C and, more critically, for D. As a result, the model might struggle to correctly classify instances from the minority classes, significantly reducing its effectiveness. Additionally, standard evaluation metrics like accuracy can be misleading, as the model may achieve high accuracy by simply favoring the majority classes while ignoring the minority ones. To address this issue, different techniques such as undersampling majority classes, creating more instances of minority class by duplication and stratified splitting of data will be used.

4.2 Wordcloud for text visualization

The word cloud analysis revealed that the most frequently occurring words are stop words, which should be removed as they do not contribute to sentiment analysis. Additionally, terms associated with the movie industry appear frequently, along with words that describe sentiment, highlighting their relevance to the task.

4.3 Sentence length analysis

Plotted histogram for visualization. Average sentence length was 12.34

4.4 Word length analysis

Plotted histogram for visualization. Average word length was 4.84

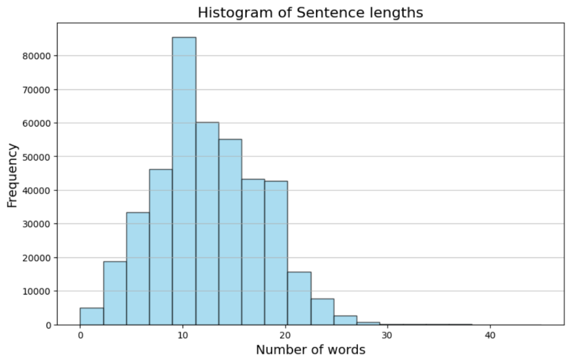

Figure 2: Histogram of Sentence length

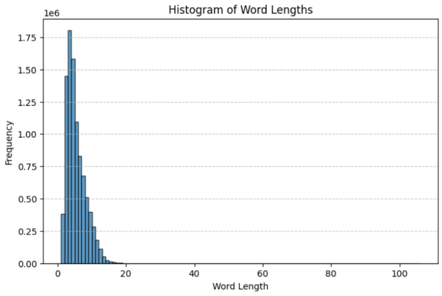

Figure 3: Histogram of word length

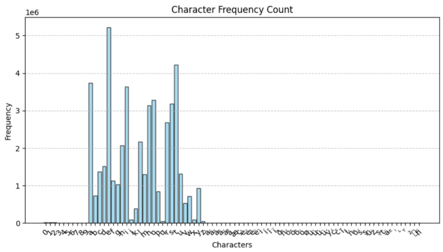

Figure 4: Bar graph for character frequencies

Figure 5: Wordcloud after preprocessing

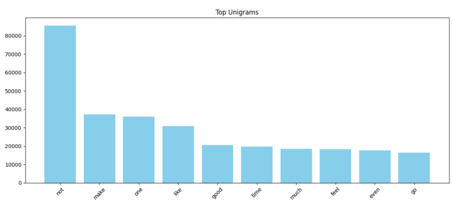

Figure 6: Unigram

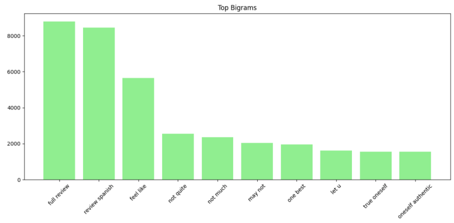

Figure 7: Bigram

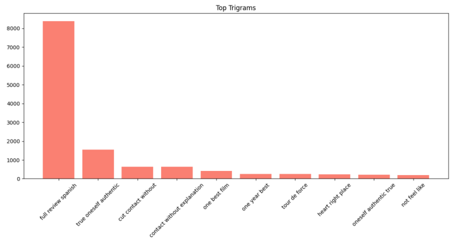

Figure 8: Trigram

5 Methodologies for Non-Competetitve

5.1 Word embedding used

We used the Google News Word2Vec 300-dimensional word embeddings for our movie review sentiment analysis. This pretrained model, trained on a large news corpus, captures word meanings and relationships well. Despite being based on news, its broad vocabulary makes it useful for movie reviews too. The high-dimensional embeddings help distinguish positive and negative sentiments. Using a pretrained model also saved computation time since we didn't have to train embeddings from scratch on our limited dataset. For computational efficiency, we limited vocabulary of Word2vec model to top 100000 words.

Total words in dataset: 5148198

Words covered in Word2Vec: 4465205

Vocabulary Coverage: 86.73 %

Examples of out of vocabulary words are hollywood, writerdirector, scifi, oscar, etc.

5.2 Norm used to convert sentence to vectors

Word Embedding converted each word into a 300-dimensional vector. To represent a sentence as a vector, we needed to choose a method to combine individual word vectors. We experimented with mean, sum, and max norms, comparing their performance using logistic regression. All three performed similarly, with the mean norm showing a slight edge. Therefore,

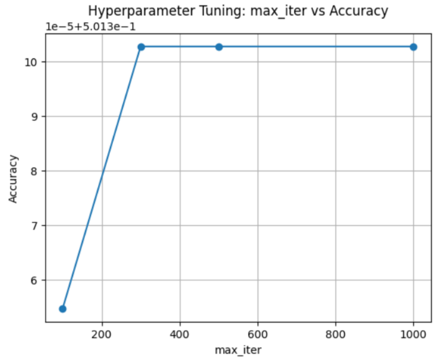

Figure 9: Hyperparameter tuning for batch gradient descent

we used the mean norm for the rest of the non-competitive part.

5.3 Splitting into train and test

We split the data 80-20 for training and testing. Since the dataset was imbalanced, we used stratified splitting to maintain the class distribution.

5.4 Logistic Regression with Batch gradient descent

From the `sklearn.linear_model` module, we used the `LogisticRegression` function to create our model with `multiclass='ovr'` and `solver='lbfgs'`. The `multiclass='ovr'` (one-vs-rest) setting trains a separate binary classifier for each class, treating each class independently. The `solver='lbfgs'` (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) is an optimization algorithm that uses batch gradient descent for training. It efficiently handles large datasets and supports L2 regularization, making it well-suited for logistic regression.

Our logistic regression model achieved **50.14%** accuracy on the test data and **50.41%** accuracy on the training data, showing no major overfitting. Class B had the highest recall (64%), while Class D performed the worst, with very low recall (2%). Class A and Class B had decent f1-scores (0.55), while Class C and Class D struggled, especially Class D, which was rarely predicted correctly. The overall results show that the model favors majority classes, making it less effective for minority classes.

Hyperparameter tuning on maximum iterations revealed that convergence was achieved at about 300 iterations.

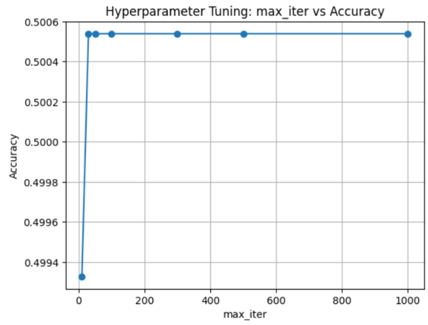

Figure 10: Hyperparameter tuning for stochastic gradient descent

5.5 Logistic Regression with Stochastic gradient descent

For the stochastic gradient descent (SGD) logistic regression model, we used the `SGDClassifier` from `sklearn.linear_model` with `loss='log_loss'`, which applies logistic regression. A fixed `random_state=42` ensured reproducibility. We applied L2 regularization (`penalty='l2'`) with a regularization strength of `alpha=0.0001` to prevent overfitting.

Our SGD logistic regression model achieved **50.05%** accuracy on the test data and **50.30%** on the training data, indicating no overfitting. Precision, recall, and f1-scores were similar to the batch gradient descent model. Hyperparameter tuning for maximum iterations showed that the stochastic logistic regression model converged in about 20 epochs. Unlike batch gradient descent, which processes the entire dataset at once, stochastic gradient descent updates model parameters after each training example, allowing it to reach an optimal solution faster. This resulted in a more efficient training process while maintaining similar accuracy and classification performance.

5.6 Logistic Regression with Mini-batch gradient descent

For the mini-batch gradient descent logistic regression model, we used `SGDClassifier` with `loss='log_loss'`. We set the mini-batch size to 100 and trained the model for 10 epochs using `partial_fit`. In each epoch, the training data was shuffled, and the model was updated iteratively on mini-batches. This approach balanced efficiency and convergence speed, achieving similar accuracy to the batch and stochastic gradient descent models.

5.7 Random Forest classifier

For the Random Forest model, we used the `RandomForestClassifier` from the `sklearn.ensemble` module. The model consisted of 100 decision trees (`n_estimators=100`) with a maximum depth of 10 (`max_depth=10`) to prevent overfitting. We required a minimum of 5 samples to split a node (`min_samples_split=5`) and at least 2 samples per leaf node (`min_samples_leaf=2`) to improve generalization. The `criterion='gini'` was used to measure node purity, and `random_state=42` ensured reproducibility.

Our Random Forest model achieved an accuracy of **46.51%** on the test data and **50.94%** on the training data, showing slight overfitting. Class B had the highest recall, while Classes C and D performed poorly, with very low recall. The overall precision, recall, and f1-scores were lower than the logistic regression models, indicating that the Random Forest struggled with classifying minority classes. This suggested the need for effecting hyperparameter tuning for all different parameters.

We performed hyperparameter tuning for the Random Forest model using `RandomizedSearchCV` from the `sklearn.model_selection` module. A randomized search was conducted over a predefined parameter grid, varying the number of estimators (`n_estimators`), maximum tree depth (`max_depth`), minimum samples required for node splitting (`min_samples_split`), and minimum samples per leaf node (`min_samples_leaf`). We used `StratifiedShuffleSplit` for cross-validation, ensuring class distribution was preserved.

The best hyperparameters were selected based on validation accuracy, and the optimized model achieved **60 %** accuracy on test data. Thus after hyperparameter tuning RandomForest model performed better than Logistic regression model but it still had bias towards majority classes.

6 Methodologies for Competetitve

We used multiple models starting from the foundational Machine Learning models like Logistic Regression to state of the art models like BERT .

6.1 Random Forest

Similar to as in non-comp part, we used we used the `RandomForestClassifier` from the `sklearn.ensemble` module and tuned it using `RandomizedSearchCV` with `StratifiedShuffleSplit` for cross-validation. The best random forest classifier thus found resulted in accuracy of **65 %** on test data.

6.2 VADER Sentiment Analysis

We utilized the VADER (Valence Aware Dictionary and sEntiment Reasoner) model to generate positive, negative, and compound sentiment scores for each review. A threshold of 0.5 was set for the compound score:

- If the compound score < 0.5, the review was classified as **negative**.
- Otherwise, it was classified as **positive**.

6.3 VADER with Fuzzy Logic

To enhance the VADER-based sentiment classification, we applied fuzzy logic to refine decision-making. The steps involved are as follows:

6.3.1 Computing Sentiment Scores

We used the VADER sentiment analyzer to compute the compound sentiment score for each review, which ranges from -1 (most negative) to 1 (most positive).

6.3.2 Defining Fuzzy Logic Components

We modeled sentiment classification as a fuzzy inference system with:

- **Input (Antecedent):** The compound sentiment score.
- **Output (Consequent):** The classification label (negative or positive).

Membership Functions:

- The compound score was categorized into three fuzzy sets:
  - **Negative:** Scores in the range $[-1, -1, 0]$
  - **Neutral:** Scores in the range $[-0.5, 0, 0.5]$
  - **Positive:** Scores in the range $[0, 1, 1]$
- The classification output was mapped to:
  - **Negative:** Values in the range $[0, 0, 0.5]$
  - **Positive:** Values in the range $[0.5, 1, 1]$

6.3.3 Fuzzy Rules

We formulated fuzzy rules to classify reviews based on the computed sentiment score:

- If the score is **negative**, classify as **negative**.
- If the score is **neutral**, classify as **positive** (adjustable based on dataset characteristics).
- If the score is **positive**, classify as **positive**.

6.3.4 Defuzzification and Prediction

The fuzzy inference system generated a classification score between 0 and 1. We used a threshold-based approach:

- If the defuzzified output ≥ 0.5, classify as **positive**.
- Otherwise, classify as **negative**.

6.3.5 Implementation and Evaluation

The fuzzy classification was applied to the test dataset:

- First, compound sentiment scores were computed.
- Then, fuzzy rules were applied to determine the sentiment class.

This approach improved upon the standard VADER model by introducing more nuanced decision-making, reducing misclassification in ambiguous cases.

6.4 Naïve Bayes

We implemented a Bernoulli Naïve Bayes classifier for sentiment classification.

6.5 Convolutional Neural Network (CNN)

We implemented a Convolutional Neural Network (CNN) . The model consists of the following layers:

- **Embedding Layer:** Pre-trained GloVe word embeddings were used, initialized with a 100-dimensional embedding matrix. The embeddings were frozen during training.
- **Convolutional Layers:**
  - A `Conv1D` layer with 128 filters, a kernel size of 5, and ReLU activation.
  - A dropout layer (rate = 0.2) to prevent overfitting.
  - Another `Conv1D` layer with 256 filters and a kernel size of 5.
- **Global Max Pooling:** Extracts the most significant features from the convolutional outputs.
- **Dense Layer:** A fully connected layer with a sigmoid activation function to output a binary classification.

The model was compiled using the Adam optimizer and binary cross-entropy loss. It was trained for 50 epochs with a batch size of 128 and a validation split of 20%.

6.6 LSTM-Based Sentiment Analysis

To incorporate contextual word representations, we implemented an LSTM-based model using GloVe embeddings.

The LSTM model was structured as follows:

- **Input Layer:** Accepts sequences of length 100.
- **LSTM Layer:** Contains 128 units with sequential memory retention.
- **Dropout Layer:** A dropout rate of 0.5 was applied to reduce overfitting.
- **Fully Connected Layers:**
  - A dense layer with 64 neurons and ReLU activation.
  - A final dense layer with a sigmoid activation for binary classification.

6.7 LSTM-GRU Based Model

To improve the performance of sequential text modeling, we implemented a hybrid Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) model.

We designed a hybrid recurrent neural network with the following layers:

- **Embedding Layer:** Converts input tokens into dense vector representations using an embedding dimension of 256.
- **Bidirectional LSTM Layer:** A 128-unit bidirectional LSTM to capture contextual dependencies from both past and future words.
- **GRU Layer:** A 128-unit GRU to further refine sequential representations and reduce vanishing gradient issues.

- **Dropout Layer:** A dropout rate of 0.5 was applied to prevent overfitting.
- **Fully Connected Layers:**
  - A dense layer with 64 neurons and ReLU activation.
  - A final dense layer with a sigmoid activation for binary classification.

6.8 In-Context Learning

We used the `google/flan-t5-large` model, a pre-trained sequence-to-sequence language model designed for instruction-following tasks.

6.8.1 Prompt Engineering

We adopted a two-shot learning approach, where the model was provided with a few labeled examples before predicting the sentiment of a new review. The prompt was structured as follows:

- A brief task description: *"Classify these movie reviews as positive (1) or negative (0):"*
- Two positive and two negative example reviews, randomly sampled from the training dataset.
- The target review to be classified.
- A placeholder for the sentiment prediction.

Following this, for each review we combined it with example pairs to form the prompt.

6.9 Zero-Shot Classification

We used the `facebook/bart-large-mnli` model, which is optimized for natural language inference (NLI) tasks and effective for zero-shot classification.The Hugging Face `zero-shot-classification` pipeline was utilized for inference.

6.9.1 Classification Process

- The model was provided with a set of predefined candidate labels: **positive** and **negative**.
- Each review was classified by computing its similarity to these labels in the NLI framework.
- The label with the highest confidence score was assigned as the predicted sentiment.
- The predictions were converted into binary values: 1 for **positive**, 0 for **negative**.

6.10 Transformer-Based Models

We experimented with various transformer models for sentiment classification.

- We tested several pre-trained transformer models:
  - `deepseek-ai/DeepSeek-V3`
  - `textattack/roberta-base-imdb`
  - `lvwerra/gpt2-imdb`
  - `valhalla/distilbart-mnli-12-3`
  - `siebert/sentiment-roberta-large-english` (best-performing)
- The Hugging Face `text-classification` pipeline was used for inference .

Siebert's Sentiment RoBERTa outperformed all other models in terms of accuracy and consistency. This model was selected as the final transformer-based classifier for sentiment analysis.

6.11 Ensemble of Transformer Models

We experimented with multiple ensemble techniques, combining predictions from different transformer-based classifiers.

6.11.1 Majority Voting Ensemble

In this approach, we combined predictions from three models:

- `siebert/sentiment-roberta -large-english`
- `textattack/roberta-base-imdb`
- `distilbert-base-uncased -finetuned-sst-2-english`

Each review was classified by all models, and the final label was determined using majority voting (i.e., if at least two models predicted positive, the final output was positive).

6.11.2 Soft-Voting Ensemble

We implemented a soft-voting ensemble method, which considers the probability scores from multiple models instead of hard classifications. This method works as follows:

- Three transformer-based sentiment classification models were used:
  - `siebert/sentiment-roberta-large-english`
  - `textattack/roberta-base-imdb`

- `distilbert-base-uncased -finetuned-sst-2-english`

- Each model was configured to return probability distributions instead of discrete labels.

- For each review, the probability of the positive sentiment class was extracted from each model.

- The final sentiment classification was determined by averaging the probabilities:

  - If the average probability > 0.5, the review was classified as **positive**.
  - Otherwise, it was classified as **negative**.

6.11.3 Weighted Averaging Ensemble (Best Performing)

This approach assigned different importance to models based on validation performance:

- The same three models were used.

- Each model's sentiment score was weighted:

  - `siebert/sentiment-roberta-large-english`: 0.4
  - `textattack/roberta-base-imdb`: 0.4
  - `nlptown/bert-base -multilingual-uncased-sentiment`: 0.2

- The final prediction was based on the weighted average of sentiment scores.

This method provided the highest accuracy and consistency.

6.11.4 Meta-Classifier (Logistic Regression)

We also trained a logistic regression model as a meta-classifier:

- Feature vectors were created using the confidence scores from each base model.

- A logistic regression model was trained on these features using the training data.

- The final predictions were made using this meta-classifier.

While effective, it did not outperform the weighted averaging ensemble.

Among all approaches, the **weighted averaging ensemble** produced the best results, balancing performance across multiple transformer-based classifiers.

7 Results

We experimented with traditional machine learning models, deep learning approaches, transformer-based models, and ensemble methods. The primary evaluation metric used was accuracy.

Table 1 summarizes the accuracies of different models. We observe that traditional models like Random Forest and VADER perform relatively poorly, whereas transformer-based approaches yield significantly higher accuracy.

Model	Accuracy
Random Forest	0.652
VADER	0.491
VADER with Fuzzy Logic	0.678
Naïve Bayes	0.831
CNN	0.349
LSTM	0.499
LSTM with GRU	0.512
In-Context Learning	0.412
Zero-Shot Learning	0.864
Individual Transformer -Based Method	0.941
Ensemble of Transformer-Based Models	0.956

Table 1: Accuracy Comparison of Different Models

Table 2 summarizes the accuracies of the ensembling methods of the various transformer models.

Model	Accuracy
Majority Voting Ensemble	0.946
Soft Voting Ensemble	0.948
Weighted Averaging Ensemble	0.956
Meta Classifier	0.947

Table 2: Comparison of the Different Ensembling techniques

Conclusion

Among the traditional Machine Learning models, Bernoulli Naive Bayes significantly performed better.It benefited from strong feature independence assumptions, making it well-suited for sentiment classification.The VADER model alone had an accuracy of 0.49, but applying fuzzy logic improved it to 0.67. This suggests that incorporating uncertainty into sentiment classification enhances performance.CNN performed poorly with an accuracy of 0.34, likely due to its inability to capture

sequential dependencies. LSTM and LSTM with GRU performed better (0.49 and 0.51, respectively), highlighting the importance of modeling long-term dependencies in sentiment classification.Still we would say models like CNN and LSTM performed lower than our expectations. We could test the In-Context learning method only with the FLAN-T5,which is a lightweight model and achieved poor accuracy in that one.Further testing with other large language models like Mistral 7B or fine tuning the FLAN-T5 on the training data can be done to achieve better results. Zero-shot classification using BART achieved 0.86, demonstrating the strength of large pre-trained models in generalizing without explicit training. The best-performing individual model was Siebert's Sentiment RoBERTa with an accuracy of 0.94.
The ensemble of transformers performed best, as combining multiple models reduced individual weaknesses and improved robustness.The weighted averaging ensemble of the models `siebert/sentiment-Roberta-large-english` ,`textattack/roberta-base-imdb` and distilbert-base-uncased finetuned-sst-2-english outperformed all the other models,achieving an accuracy of 0.956 making it most suitable for the task for sentiment classification. The weighted averaging ensemble outperforming other ensemble methods can be due to several key factors:

- **Incorporation of Model Reliability:** By assigning higher weights to models with better validation performance, the ensemble leveraged the strengths of the most accurate models. For instance, `siebert/sentiment-roberta-large-english` was given greater influence because of its consistently high accuracy.
- **Mitigation of Noise from Weaker Models:** Lower-performing models contributed less to the final decision, reducing the overall noise and minimizing the impact of their errors.
- **Balanced Decision Making:** Unlike majority voting—which makes hard decisions based solely on counts—the weighted approach takes into account the confidence of each prediction, yielding a smoother and more reliable decision boundary.
- **Leveraging Complementary Strengths:** Different models capture various aspects of sentiment. The ensemble benefits from this diversity by combining general sentiment detection with domain-specific nuances, resulting in improved robustness.

References

[1] Sentiment Analysis – Wikipedia. https://en.wikipedia.org/wiki/Sentiment_analysis

[2] NLTK Stopwords Corpus. http://www.nltk.org/book/ch02.html

[3] A Comprehensive Guide to Text Preprocessing for Twitter Data: Getting Ready for Sentiment Analysis. https://shorturl.at/KKHcf

[4] J. Devlin, M. W. Chang, K. Lee, K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, June 2-7, 2019, Minneapolis, Minnesota.

[5] Tong Xiao and Jingbo Zhu. Foundations of Large Language Models, January 17, 2025. https://arxiv.org/pdf/2501.09223

[6] Niladri Chatterjee, Tanya Aggarwal, and Rishabh Maheshwari. Sarcasm Detection Using Deep Learning-Based Technique. In *Deep Learning-Based Approaches for Sentiment Analysis*, edited by Basant Agarwal, Richi Nayak, Namita Mittal, and Srikanta Patnaik, Springer, pp. 237-258, 2020.

[7] Danilo Dessí, Mauro Dragoni, Gianni Fenu, Mirko Marras, and Diego Reforgiato Recupero. Deep Learning Adaptation with Word Embeddings for Sentiment Analysis on Online Course Reviews. In *Deep Learning-Based Approaches for Sentiment Analysis*, edited by Basant Agarwal, Richi Nayak, Namita Mittal, and Srikanta Patnaik, Springer, 2020.

[8] Taha533. *Sentiment Analysis of IMDB Movie Reviews*. GitHub Repository. Available at: https://github.com/Taha533/Sentiment-Analysis-of-IMDB-Movie-Reviews