Sentiment Classification of Movie Reviews

Pooria Daneshvar Kakhaki*
Department of Computer Science
Northeastern University
Boston, MA
daneshvarkakhaki.p@northeastern.edu

Neda Ghohabi Esfahani*
Department of Bioengineering
Northeastern University
Boston, MA
ghohabiesfahani.n@northeastern.edu

Abstract

Sentiment analysis is a key task in Natural Language Processing (NLP) that aims to determine the emotional tone of text. In this project, we explore a broad range of machine learning and deep learning models for sentiment classification on the Large Movie Review Dataset (IMDB), which contains 50,000 balanced positive and negative reviews. We benchmark classical models like Logistic Regression and Random Forest trained on Bag-of-Words features (binary count, raw count, and TF-IDF), and develop deep learning approaches including Fully Connected Networks and LSTMs using pretrained, self-trained, and randomly initialized word embeddings. We also train Transformer models with learnable embeddings and fine-tune large pretrained language models such as DistilBERT and RoBERTa, experimenting with strategies like frozen encoders and transfer learning from SST-2. Model performance is primarily evaluated using accuracy, along with F1-score, precision, and recall. Our best result achieves 95.27% accuracy using RoBERTa fine-tuned on IMDB, highlighting the advantage of large-scale pretrained models over traditional and deep learning methods.

1 Introduction

Sentiment analysis is a fundamental task in Natural Language Processing (NLP) that aims to determine the emotional tone, or the sentiment behind textual content [1, 2]. One common application of sentiment analysis is the classification of domain-specific subjective reviews. A common source for such subjective opinions can be found in movie reviews, as they often contain clear sentiment cues, making them a rich source for sentiment classification.

Sentiment classification tasks can be formulated as either a binary or a multiclass classification. In binary classification, the goal is to distinguish between two opposing sentiments, typically positive and negative. On the other hand, multiclass classification extends this by adding additional categories, such as neutral. In some cases, other classes, such as very positive or very negative can also be seen.

Accurate sentiment classification has numerous real-world applications. It can enhance recommendation systems by capturing user preferences, support businesses through customer feedback analysis, and enable automatic summarization of reviews for efficient content consumption. Moreover, sentiment analysis serves as a valuable downstream task for evaluating the performance and generalization ability of language models, making it a useful benchmark in NLP research[3].

In recent years, a variety of machine learning and deep learning techniques have been explored for sentiment classification. Traditional methods such as Naive Bayes, Support Vector Machines (SVM), and Logistic Regression have been widely used with bag-of-words or TF-IDF features [4]. More recent approaches leverage pretrained word embeddings like Word2Vec [5, 6] or GloVe [7] to capture semantic meaning, or use end-to-end deep models such as Convolutional Neural Networks

*These authors contribute equally

CS6120 - Natural Language Processing - (Northeastern University, Spring 2025).

(CNNs), Recurrent Neural Networks (RNNs) [8], and Transformer-based architectures to learn sentiment directly from raw text. These models have shown strong performance on benchmark datasets, particularly when fine-tuned on domain-specific data.

In this project, we investigate the effectiveness of various machine learning and deep learning models for sentiment classification of movie reviews. Our goal is to develop and evaluate approaches that can accurately predict the sentiment expressed in user-generated movie reviews based on their textual content.

2 Related works

Sentiment analysis research has been driven by the development of several benchmark datasets and modeling approaches. The Stanford Sentiment Treebank (SST-2) [9] provides binary sentiment labels for short movie review phrases. The finegrained version of that dataset (SST-5) is annotated with additional neutral, very positive and very negative labels. The Yelp Reviews dataset [10] contains user-generated business reviews with star ratings, while Amazon Reviews [11] offer sentiment annotations across a wide range of product categories. Although these datasets provide diversity in sentiment expression and domain coverage, many consist of short texts or span multiple topics. In contrast, Large Movie Review Dataset (IMDB) [12] 50,000 full-length reviews, offering a more realistic and linguistically complex inputs for sentiment classification.

Early methods for sentiment analysis relied heavily on hand-crafted features such as n-grams, part-of-speech tags, and sentiment lexicons, combined with traditional classifiers like Naive Bayes and Support Vector Machines. The introduction of word embeddings improved the semantic representation of text, enabling simple neural network architectures like fully connected layers over averaged embeddings. Subsequent advances included convolutional neural networks (CNNs) to capture local patterns, recurrent neural networks (RNNs)[8] and LSTMs [13] to model sequence dependencies, and attention mechanisms [14] to focus on sentiment-bearing phrases. Recently, Transformer-based architectures such as BERT citepdevlin2019bert have achieved state-of-the-art results across multiple sentiment benchmarks, though they come with significantly increased computational costs [15].

3 Dataset

For this project, we used the Large Movie Review Dataset (IMDB)[12], which consists of 100,000 movie reviews, with 50,000 labeled examples. The labeled reviews are evenly divided into training and testing sets, each containing 25,000 samples, balanced between 12,500 positive and 12,500 negative reviews. The 50,000 unlabeled samples are typically used for clustering task and zero-shot supervision. We exclude them from all of our experiments.

Since the dataset does not provide a predefined validation set, we further split the training data, reserving 20% for validation. This results in a training set of 20,000 samples (10,000 positive and 10,000 negative) and a validation set of 5,000 samples (2,500 positive and 2,500 negative).

Table 1 summarizes the distribution of samples in our dataset.

Table 1: Distribution of samples in the IMDB dataset.

Split	Positive Samples	Negative Samples	Total Samples
Training	10,000	10,000	20,000
Validation	2,500	2,500	5,000
Testing	12,500	12,500	25,000

We provide several statistics that summarize the characteristics of the dataset in Table 2

Table 2: Summary statistics of the IMDB reviews dataset.

Total Sentences	Total Words	Total Characters	Avg. Sentence Length	Avg. Review Length
528,292	11.68M	51.33M	22.1 words	10.6 sentences (233.7 words)

4 Methodology

4.1 Preprocessing

We applied a series of text normalization techniques to clean and standardize the IMDB movie reviews prior to model training. All text was converted to lowercase to eliminate case-based redundancy. HTML tags and URLs were removed to retain only meaningful content, and common stopwords such as "the," "is," and "and" were filtered out, as they generally carry limited semantic value in sentiment classification.

We also explored several optional enhancements to improve input quality. Punctuation was removed to reduce sparsity in feature-based representations. To reduce words to their base or dictionary form, noth stemming and lemmatization were evaluated, and based on preliminary experiments, we settled on lemmatization as it provided better consistency and performance. Additionally, spell correction was incorporated to address typographical errors and reduce vocabulary noise.

For implementation, we used NLTK for tokenization, stopword removal, stemming, and lemmatization; regular expressions for HTML tag and URL removal; and SymSpell for fast and efficient spell correction. Stopword filtering was performed using the NLTK stopword list, and punctuation removal was implemented using Python's string library.

4.2 Modeling Pipeline

Our modeling pipeline consists of three main stages: text representation, model development, and evaluation. First, raw movie reviews are transformed into structured numerical representations suitable for model input. Then, a range of machine learning deep learning models are trained. Finally we will evaluate our models on our test set for a direct comparison of their performance on the sentiment classification task.

4.2.1 Text Representation

To transform raw text into numerical input suitable for machine learning models, we investigated several word-based representation techniques, each balancing simplicity, expressiveness, and model compatibility.

Bag-of-words (BOW) and TF-IDF: Classical representations included the Bag-of-Words (BoW) model, treating documents as unordered collections of words [16]. Two BoW variants were used: a binary representation that encodes the presence or absence of each word, and a count-based representation that captures the frequency of each word within the document. Building upon BoW, we also applied Term Frequency-Inverse Document Frequency (TF-IDF) weighting, which adjusts raw counts by the relative importance of words across the corpus, reducing the influence of common but uninformative terms [17].

Word vectors: To better capture semantic relationships between words, we employed word embeddings and explored several strategies for constructing an embedding layer as part of our models:

- **Pretrained Word2Vec embeddings:** Vectors trained on other datasets were used to capture semantic information.
- **Domain-trained Word2Vec embeddings:** We trained Word2Vec embeddings directly on the IMDB training set to capture dataset-specific vocabulary and context.
- **Randomly initialized embeddings:** Embeddings were initialized randomly and refined during model training, allowing the network to learn task-specific representations.

This approach allowed us to systematically assess how different embedding initialization methods influenced downstream model performance in sentiment classification.

LLMs native toeknizers: Finally, for transformer-based models such as BERT and DistilBERT, we used their native subword tokenization strategies, including WordPiece and Byte-Pair Encoding. These tokenizers split words into subword units, enabling efficient handling of rare or out-of-vocabulary words while preserving contextual information through positional embeddings.

4.2.2 Modeling Approaches

We developed a range of models spanning classical machine learning, deep learning, and transformer-based architectures to perform sentiment classification on the IMDB dataset.

Classical Machine Learning Models:

- **Logistic Regression:** A linear classifier used as a strong baseline when combined with Bag-of-Words or TF-IDF features [18].
- **Random Forest:** A decision tree ensemble model that captures non-linear patterns and interactions in the data [19].

Deep Learning Models:

- **Fully Connected Networks (FCNs):** Simple feedforward networks trained on dense word embeddings to model sentiment directly from text [20].
- **Recurrent Neural Networks (RNNs):** Sequence models that process text token by token to learn temporal dependencies [8].
- **Long Short-Term Memory Networks (LSTMs):** An RNN variant designed to capture long-range dependencies and mitigate the vanishing gradient problem [13].

Transformer-based Models:

- We implemented a **custom transformer** trained from scratch with learnable embeddings to analyze the impact of self-attention mechanisms [14].
- We also fine-tuned **pretrained large language models (LLMs)**, including **DistilBERT** [21] and **RoBERTa**[22], using multiple strategies such as frozen encoders, full fine-tuning, and sequential transfer from SST-2 to IMDB.

4.2.3 Evaluation

Model performance was primarily evaluated using **accuracy**, given that the IMDB dataset is balanced with an equal number of positive and negative reviews, and both sentiments shave equal value. In addition to accuracy, we reported **F1-score**, **precision**, and **recall** to provide more details on classification performance.

5 Results

5.1 Logistic Regression with Bag-of-Words Representations

As a classical baseline, we implemented Logistic Regression models using Bag-of-Words (BoW) representations to classify sentiment in movie reviews. We experimented with three vectorization schemes: Binary BoW, which encodes word presence or absence; Count BoW, which encodes word frequencies; and TF-IDF, which weights terms by their importance relative to the corpus.

For each vectorization method, we trained Logistic Regression models using different either without (unregularized), or with L1 (Lasso) or L2 (Ridge) penalty. The results of these experiments is provided in Table 3.

For all experiments, the vectorizer settings were fixed across different regularization strategies for a given representation. Specifically, for each vectorizer type (Binary, Count, TF-IDF), the same

Table 3: Logistic Regression Results on Different Vectorizers and Penalties

Vectorizer	Penalty	Accuracy	Precision		Recall		F1	
			P	N	P	N	P	N
Binary	None	0.8446	0.8458	0.8433	0.8427	0.8464	0.8443	0.8448
	L1	0.8620	0.8622	0.8618	0.8618	0.8623	0.8620	0.8621
	L2	0.8603	0.8625	0.8582	0.8574	0.8633	0.8599	0.8607
Count	None	0.8372	0.8460	0.8288	0.8244	0.8499	0.8351	0.8392
	L1	0.8566	0.8609	0.8524	0.8506	0.8626	0.8557	0.8575
	L2	0.8547	0.8608	0.8489	0.8463	0.8631	0.8535	0.8559
TF-IDF	None	0.8466	0.8564	0.8373	0.8328	0.8604	0.8445	0.8487
	L1	0.8746	0.8669	0.8827	0.8852	0.8641	0.8760	0.8733
	L2	0.8816	0.8794	0.8839	0.8846	0.8786	0.8819	0.8812

n-gram range (1,2) and the top 10,000 most informative features were used. Only the model-specific hyperparameters, such as the regularization strength, were tuned individually based on validation set performance to optimize classification accuracy.

An interesting observation is that for both the Count and Binary Count vectorizers, Logistic Regression models with L1 regularization outperform those with L2 regularization. However, for the TF-IDF vectorizer, the opposite can be seen. This difference likely arises because L1 regularization promotes feature selection and sparsity by driving many coefficients to zero, which is advantageous when working with high-dimensional but relatively redundant representations like raw counts. In contrast, TF-IDF weighting already downscales common but uninformative features, producing a more compact and informative feature space.

5.2 Logistic Regression with Word vectors

Beyond Bag-of-Words and TF-IDF representations, we also applied Logistic Regression models directly to word embeddings. Specifically, we explored two approaches: using pretrained Word2Vec word vectors from `GoogleNews-vectors-negative300`, and using Word2Vec word vectors that we trained ourselves on the IMDB training set. For each review, individual word vectors were aggregated by averaging across all words in the review to produce a fixed-size feature representation. No additional embedding layer was employed, as the averaged word vectors served directly as input to the classifier.

The results of these experiments are provided in Table 4.

Table 4: Logistic Regression results using word vector representations

Word Vector Type	Accuracy	Precision		Recall		F1	
		P	N	P	N	P	N
Pretrained Word2Vec	0.8459	0.8484	0.8435	0.8424	0.8494	0.8454	0.8465
Trained Word2Vec 100D	0.8284	0.8318	0.8251	0.8234	0.8335	0.8276	0.8293
Trained Word2Vec 300D	0.8297	0.8315	0.8280	0.8270	0.8324	0.8293	0.8302
Trained Word2Vec 500D	0.8294	0.8313	0.8275	0.8265	0.8323	0.8289	0.8299

The model which uses pretrained Word2Vec vectors consistently outperform those using self-trained Word2Vec word vectors. This result highlights the advantage of leveraging semantic relationships captured by large-scale external datasets. Furthermore, increasing the embedding dimensionality from 100 to 300 led to noticeable improvements in performance, while gains beyond 300 dimensions were marginal.

5.3 Random Forest with Bag-of-Words, TF-IDF, and Word Vector Representations

In addition to Logistic Regression, we implemented Random Forest models to classify sentiment in movie reviews using binary count, count, TF-IDF, and word vector representations.

For each vectorizer type, the vectorizer settings were fixed across experiments, using an n-gram range of (1,2) and selecting the top 10,000 most informative features. Only the model-specific hyperparameters, such as the number of trees and maximum depth and minimum samples at each leaf were tuned based on the performance on the validation set.

The results of these experiments are presented in Table 5.

Table 5: Random Forest results using Bag-of-Words, TF-IDF, and averaged word vector tepresentations

Input Representation	**Accuracy**	**Precision**		**Recall**		**F1**	
		P	**N**	**P**	**N**	**P**	**N**
Binary	0.8463	0.8311	0.8630	0.8693	0.8234	0.8498	0.8427
Count	0.8492	0.8333	0.8666	0.8730	0.8254	0.8527	0.8455
TF-IDF	0.8483	0.8369	0.8604	0.8651	0.8314	0.8508	0.8457
Pretrained Word2Vec	0.8057	0.8005	0.8110	0.8142	0.7971	0.8073	0.8040
Trained Word2Vec (100D)	0.7948	0.7903	0.7996	0.8027	0.7870	0.7964	0.7932
Trained Word2Vec (300D)	0.8030	0.7965	0.8098	0.8140	0.7921	0.8052	0.8009
Trained Word2Vec (500D)	0.8017	0.7963	0.8073	0.8108	0.7926	0.8035	0.7999

Overall, we observed that the Random Forest models achieved slightly lower performance compared to the Logistic Regression models across all representation types. This can be due to our tuning and pruning the model, as Random Forests are highly sensitive to parameters and can be greatly optimized by better pruning.

5.4 Fully Connected Networks with Bag-of-Words and TF-IDF Representations

We then moved into the deep learning domain by implementing Fully Connected Networks (FCNs) trained on Bag-of-Words (BoW) and TF-IDF representations.

Each FCN model consisted of an input layer corresponding to the feature size (10,000 features), followed by two hidden layers with ReLU activation, and a final output layer. Dropout was applied between layers to mitigate overfitting. Like previous experiments, for each representation type, the vectorizer settings (n-gram range (1,2) and top 10,000 features) were fixed. We also made sure to initialize the FCN models weights the say across all experiments.

The results of these experiments are presented in Table 6.

Table 6: Fully Connected Network results using Bag-of-Words and TF-IDF representations

Input Representation	**Accuracy**	**Precision**		**Recall**		**F1**	
		P	**N**	**P**	**N**	**P**	**N**
Binary	0.8692	0.8722	0.8662	0.8651	0.8732	0.8686	0.8697
Count	0.8681	0.8708	0.8655	0.8646	0.8717	0.8676	0.8686
TF-IDF	0.8796	0.8860	0.8733	0.8712	0.8879	0.8785	0.8806

Interestingly, while FCNs were able to capture non-linear combinations of features, their performance was comparable to or slightly below that of Logistic Regression models on BoW and TF-IDF inputs. This suggests that for simple bag-of-words style representations, the linear decision boundaries learned by Logistic Regression are already sufficient, and deeper models offer limited additional benefit without richer input features.

Figure 1 shows the word clouds generated from the Fully Connected Network (FCN) model trained on TF-IDF features. The left side corresponds to words most associated with the negative sentiment class, while the right side shows words most indicative of positive sentiment.

Figure 1: Word clouds for negative (left) and positive (right) sentiment classes generated from the FCN model trained on TF-IDF representations.

5.5 Fully Connected Networks with Word Vector Embeddings

Building on the previous experiments with Fully connected Networks, we extended our analysis by incorporating an explicit embedding layer into the Fully Connected Network (FCN) architecture to work directly with word vector representations.

In this setup, each input document was first tokenized into a sequence of word indices, and the model included an embedding layer to map these indices into dense vector representations. The ways we initialize this embedding layer was previously explained in section 4.2.1.

For aggregation, we applied simple pooling techniques such as mean pooling across token embeddings to produce a fixed-size input to the FCN. The remainder of the network architecture mirrored the previous FCN setup, consisting of two hidden layers with ReLU activations, dropout for regularization, and a final softmax output layer for binary sentiment classification. The results of these experiments are presented in Table 7.

Table 7: Fully Connected Network results using different word vector initialization strategies

Embedding Initialization	**Froze Embedding Layer**	**Accuracy**	**Precision**		**Recall**		**F1**	
			P	**N**	**P**	**N**	**P**	**N**
Pretrained Word2Vec	✓	0.8367	0.8537	0.8212	0.8126	0.8679	0.8327	0.8405
Pretrained Word2Vec		0.8658	0.8778	0.8544	0.8498	0.8817	0.8636	0.8583
Trained Word2Vec (100D)	✓	0.8272	0.8289	0.8254	0.8245	0.8298	0.8267	0.8276
Trained Word2Vec (100D)		0.8326	0.8326	0.8326	0.8326	0.8326	0.8326	0.8326
Trained Word2Vec (300D)	✓	0.8388	0.8510	0.8274	0.8214	0.8562	0.8359	0.8416
Trained Word2Vec (300D)		0.8458	0.8604	0.8324	0.8257	0.8660	0.8427	0.8489
Trained Word2Vec (500D)	✓	0.8360	0.8120	0.8640	0.8744	0.7976	0.8421	0.8295
Trained Word2Vec (500D)		0.8414	0.8191	0.8671	0.8764	0.8064	0.8468	0.8356
Random Initialization (100D)		0.7252	0.7309	0.7198	0.7130	0.7374	0.7218	0.7285
Random Initialization (300D)		0.7862	0.7805	0.7922	0.7965	0.7760	0.7884	0.7840
Random Initialization (500D)		0.8059	0.7966	0.8158	0.8216	0.7902	0.8089	0.8028

5.6 LSTM with Embedding Layer: Baseline and Enhancements

In this experiment, we extend our investigation into sequential modeling to capture temporal relationship between inputs. To this goal, we implemented Long Short-Term Memory (LSTM) networks using an embedding layer constructed from pretrained Word2Vec word vectors. Initially, a standard unidirectional LSTM was trained as the baseline model.

To explore architectural improvements, we introduced the following enhancements individually and jointly:

- **Bidirectional LSTM:** Allowing information to flow in both forward and backward directions.
- **Attention Mechanism:** Enabling the model to focus more selectively on important parts of the input sequence.
- **Bidirectional + Attention:** Combining both enhancements to assess synergistic effects.
- **Bidirectional + Attention + Learnable:** Same as previous entry, with out freezing the embdding layer.

Each model variant was trained from scratch and evaluated based on accuracy, F1-score, precision, and recall, consistent with previous experiments. Table 8 summarizes the performance across these configurations.

Table 8: Performance of LSTM variants with various enhancements

Model Variant	**Accuracy**	**Precision**		**Recall**		**F1**	
		P	**N**	**P**	**N**	**P**	**N**
Baseline LSTM	0.5024	0.7239	0.5012	0.0078	0.9970	0.0154	0.6671
+ Attention	0.8607	0.8350	0.8907	0.8991	0.8223	0.8659	0.8552
+ Bidirectional	0.8579	0.8460	0.8707	0.8752	0.8406	0.8603	0.8554
+ Bidirectional + Attention	0.8662	0.8534	0.8801	0.8845	0.8480	0.8686	0.8638
+ Bidirectional + Attention + Learnable Embedding	0.8549	0.8673	0.8432	0.8379	0.8718	0.8524	0.8573

Across all configurations, the baseline unidirectional LSTM struggled to capture sentiment effectively, achieving an accuracy close to random guessing. Introducing an attention mechanism or bidirectional processing individually led to substantial improvements, reflecting their ability to better model long-range and contextually important information within the reviews. Combining bidirectional LSTM with attention further boosted performance. Finally, allowing the embedding layer to be trainable resulted in a slight decrease in performance, suggesting that fine-tuning embeddings on relatively small datasets may introduce noise rather than meaningful adaptations.

5.6.1 LSTM with Learnable Word Vector Embeddings

In our previous experiments, we observed that making the embedding layer learnable resulted in a decrease in model performance. In this experiment, we extend our analysis by training a series of bidirectional LSTM models with attention mechanisms, using either an embedding layer populated with Word2Vec vectors trained on our dataset or an embedding layer initialized randomly. Our goal is to evaluate the difference in performance between using domain-trained embeddings and learning embeddings entirely from scratch. In are these experiments, embedding layer weights are kept trainable. Results are provided in Table 9

Overall, the results indicate that using domain-trained Word2Vec embeddings consistently outperforms random initialization across all tested dimensions. Models initialized with trained embeddings achieved higher accuracy, precision, recall, and F1-scores compared to their randomly initialized counterparts. Notably, embedding dimensions of 100 and 300 led to the strongest performance, while

Table 9: Bidirectional LSTM with attention results using different word vector initialization strategies

Embedding Initialization	Accuracy	Precision		Recall		F1	
		P	N	P	N	P	N
Trained Word2Vec (100D)	0.8743	0.8872	0.8622	0.8576	0.8910	0.8721	0.8763
Trained Word2Vec (300D)	0.8675	0.8630	0.8721	0.8737	0.8613	0.8683	0.8667
Trained Word2Vec (500D)	0.8626	0.8642	0.8611	0.8605	0.8648	0.8623	0.8629
Random Initialization (100D)	0.8247	0.8943	0.7760	0.7364	0.9130	0.8077	0.8389
Random Initialization (300D)	0.8498	0.8405	0.8596	0.8634	0.8362	0.8518	0.8477
Random Initialization (500D)	0.8471	0.8601	0.8351	0.8291	0.8651	0.8443	0.8498

gains from increasing dimensionality beyond 300 were marginal. Randomly initialized embeddings, although able to learn meaningful representations during training, showed lower performance, suggesting that pretraining embeddings on domain-specific data provides a substantial advantage for LSTM-based sentiment classification tasks on relatively small datasets.

5.6.2 Transformer with Learnable Word embeddings

To further explore sequence modeling architectures, we developed a Transformer-based model. Based on our experiments with LSTM networks, we observed that although LSTMs were able to capture sequential dependencies, they struggled to effectively train strong embedding layers on our dataset. Transformers, by contrast, are known for their superior ability to model long-range dependencies and learn robust representations through self-attention mechanisms. Motivated by these observations, we adopted a Transformer encoder architecture to better capture the complex relationships present within movie reviews.

Our Transformer model was designed to be lightweight. Since we only want to classify each input, it is an encoder only transformer with two layers, two attention heads, a feed-forward dimensionality of 256, and sinusoidal positional encodings.

Similar to previous experiments, we initialized the embedding layer in 3 different ways, reflected in section 4.2.1. The results of these experiments are provided in Table 10

Table 10: Transformer using different word vector initialization strategies

Embedding Initialization	Accuracy	Precision		Recall		F1	
		P	N	P	N	P	N
Pretrained Word2Vec	0.8692	0.8700	0.8683	0.8680	0.8703	0.8690	0.8693
Trained Word2Vec (100D)	0.8698	0.8880	0.8531	0.8462	0.8933	0.8666	0.8728
Trained Word2Vec (300D)	0.8294	0.8923	0.7839	0.7493	0.9096	0.8146	0.8421
Trained Word2Vec (500D)	0.8358	0.8969	0.7910	0.7588	0.9128	0.8221	0.8475
Random Initialization (100D)	0.8404	0.8313	0.8500	0.8542	0.8267	0.8426	0.8382
Random Initialization (300D)	0.8607	0.8625	0.8589	0.8582	0.8632	0.8604	0.8611
Random Initialization (500D)	0.8498	0.8631	0.8375	0.8316	0.8681	0.8471	0.8525

The results demonstrate that initializing the embedding layer with pretrained Word2Vec vectors yields strong performance, achieving an accuracy of 86.92%. However, embeddings trained directly on the IMDB dataset also achieved comparable performance, particularly for the 100-dimensional setup. Interestingly, random initialization also led to competitive results, with 300-dimensional random embeddings achieving the highest accuracy among random strategies (86.07%). These findings highlight the Transformer's strong capacity to learn meaningful representations even without any pretrained word vectors. Overall, while pretrained embeddings provide a slight advantage, the self-attention mechanism inherent to Transformers enables effective learning of the initial embedding quality.

5.7 DistilBert

To evaluate the benefits of large-scale pretrained representations, we leveraged `distilbert-base-uncased`, a lightweight Transformer-based model pretrained on a masked language modeling objective. We explored several fine-tuning strategies to adapt the model for sentiment classification on the IMDB dataset. First, we trained a classification head on top of a frozen DistilBERT encoder directly on IMDB. To assess the impact of intermediate task transfer, we also experimented with `distilbert-base-uncased-finetuned-sst-2-english`, a version of DistilBERT already fine-tuned on the Stanford Sentiment Treebank (SST-2) dataset. In this setup, we either froze the SST-2-tuned encoder and trained a new classification head, or fully fine-tuned the model further on IMDB.

The results of these experiments are summarized in Table 11. Across all configurations, full model fine-tuning significantly outperformed training only the classification head, highlighting the importance of allowing the pretrained encoder layers to adapt to domain-specific characteristics. Sequential fine-tuning from SST-2 to IMDB led to marginal gains over direct fine-tuning on IMDB alone, suggesting that task-specific intermediate fine-tuning can slightly enhance model adaptation for sentiment classification.

Table 11: Performance of DistilBERT models under different fine-tuning strategies

Training Strategy	**Accuracy**	**Precision**		**Recall**		**F1**	
		P	**N**	**P**	**N**	**P**	**N**
IMDB (Frozen Encoder)	0.8428	0.8417	0.8441	0.8444	0.8413	0.843	0.843
SST-2 $\rightarrow$ IMDB (Frozen Encoder)	0.9038	0.9004	0.9072	0.9081	0.8994	0.904	0.904
IMDB (Fully Fine-tuned)	0.9307	0.9268	0.9346	0.9352	0.9261	0.930	0.930
SST-2 $\rightarrow$ IMDB (Fully Fine-tuned)	0.9348	0.9317	0.9380	0.9384	0.9313	0.935	0.935

5.8 RoBERTa

To further investigate the impact of large-scale pretrained models on sentiment classification, we leveraged `roberta-base`, a Transformer-based model pretrained on a masked language modeling objective over a larger corpus than DistilBERT. Similar to our experiments with DistilBERT, we explored several fine-tuning strategies to adapt RoBERTa for the IMDB dataset. First, we trained a classification head on top of a frozen RoBERTa encoder directly on IMDB. We also utilized `textattack/roberta-base-SST-2`, a version of RoBERTa already fine-tuned on the Stanford Sentiment Treebank (SST-2) dataset. In this setup, we either froze the SST-2-tuned encoder and trained a new classification head, or fully fine-tuned the model further on IMDB.

The results of these experiments are summarized in Table 12. Similar trends were observed as with DistilBERT: full model fine-tuning significantly outperformed training only the classification head, and sequential fine-tuning from SST-2 provided additional gains. Additionally, RoBERTa achieved higher performance than DistilBERT across all configurations, which is expected given RoBERTa's larger model capacity and richer pretraining corpus.

Table 12: Performance of RoBERTa models under different fine-tuning strategies

Training Strategy	**Accuracy**	**Precision**		**Recall**		**F1**	
		P	**N**	**P**	**N**	**P**	**N**
IMDB (Frozen Encoder)	0.8412	0.8420	0.8403	0.8406	0.8417	0.8413	0.8413
SST-2 $\rightarrow$ IMDB (Frozen Encoder)	0.9056	0.9041	0.9070	0.9074	0.9037	0.9060	0.9050
IMDB (Fully Fine-tuned)	0.9501	0.9487	0.9514	0.9517	0.9485	0.9502	0.9500
SST-2 $\rightarrow$ IMDB (Fully Fine-tuned)	0.9527	0.9511	0.9542	0.9545	0.9510	0.9528	0.9526

6 Discussion

Table 13 showcases the best performing models in each category.

Table 13: Best-performing models in each category

Category	Configuration	Accuracy
Logistic Regression (BoW/TFIDF)	TF-IDF + L2 penalty	0.8816
Logistic Regression (Word Vectors)	Pretrained Word2Vec	0.8459
Random Forest	Count Vectorizer	0.8492
FCN (BoW/TFIDF)	TF-IDF Representation	0.8796
FCN (Word Vectors)	Pretrained Word2Vec (Learnable)	0.8658
LSTM Variants	+ Bidirectional + Attention	0.8662
LSTM with Word Vectors	Trained Word2Vec (100D)	0.8743
Transformer with Word Vectors	Trained Word2Vec (100D)	0.8698
DistilBERT	SST-2 $\rightarrow$ IMDB (Fully Fine-tuned)	0.9348
RoBERTa	SST-2 $\rightarrow$ IMDB (Fully Fine-tuned)	0.9527

Classical machine learning algorithms, such as logistic regression, perform surprisingly well using bag-of-words and TF-IDF representations. However, their overall performance is constrained by the limited semantic information of these feature types and the models' inability to capture complex semantic structures in the data. Even with feature engineering, classical models plateau around an accuracy of 85–88%, indicating the limits of their modeling capacity.

Deep learning models, such as fully connected networks and LSTMs, were not able to consistently outperform the strong logistic regression baseline trained on TF-IDF features. While these models have the capacity to learn richer representations and capture sequential dependencies, their advantage is not fully realized without access to extremely large datasets or specialized architectures. The use of word embeddings, particularly pretrained vectors, helps these models better encode semantic relationships that bag-of-words models cannot capture. However, the quality of these embeddings is very important.

Transformer-based architectures significantly can outperform classical and early deep learning models as we saw with DistilBert and RoBERTa. Even our lightweight transformer was able to achieve a comparable score, but more importantly, it showcase strong capability to learn good embeddings from the dataset itself without relying on any previously learnt word vectors. Fine-tuning large pretrained language models yields the best results, with RoBERTa achieving over 95% accuracy when fine-tuned sequentially from SST-2 to IMDB.

Despite their superior performance, Large Language Models (LLMs) come with notable costs. Their training and inference processes are slower and require more computational resources compared to classical and shallow deep learning models. This tradeoff between accuracy and computational efficiency must be considered when selecting models for real-world applications.

In conclusion, while classical models offer a strong and efficient baseline, state-of-the-art performance in sentiment classification is achieved through the use of pretrained Transformers. Word embeddings bridge the gap between classical and deep learning models, emphasizing the importance of semantic feature representations in achieving high performance.

Code and Data availability

All the scripts and codes that were used in this project are available in github repository `SentimentAnalysis_IMDBMovieReviews`. The datasets that were used in this project are publicly accessible at this link.

Contributions

N.G.E. contributed to the conceptualization of preprocessing steps and machine learning models, as well as writing and editing the report. P.D.K. contributed to the conceptualization of preprocessing steps and deep learning models, as well as writing and editing the report.

Experiment setup

All experiments were conducted using an NVIDIA V100-SMX2 GPU with Python 3.12 and PyTorch 2.6.0+cu118, under fixed random seeds (Torch: 42, NumPy: 42) with deterministic training enabled.

References

[1] Jamin Rahman Jim, Md Apon Riaz Talukder, Partha Malakar, Md Mohsin Kabir, Kamruddin Nur, and Mohammed Firoz Mridha. Recent advancements and challenges of nlp-based sentiment analysis: A state-of-the-art review. *Natural Language Processing Journal*, page 100059, 2024.

[2] Bing Liu. *Sentiment analysis and opinion mining*. Springer Nature, 2022.

[3] Mayur Wankhade, Annavarapu Chandra Sekhara Rao, and Chaitanya Kulkarni. A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 55(7): 5731–5780, 2022.

[4] Tejaswini M. Untawale and G. Choudhari. Implementation of sentiment classification of movie reviews by supervised machine learning approaches. In *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1197–1200, 2019. doi: 10.1109/ICCMC.2019.8819800.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[8] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5 (64-67):2, 2001.

[9] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1631–1642. Association for Computational Linguistics, 2013.

[10] Yelp. Yelp open dataset, 2015. URL `https://www.yelp.com/dataset`. Accessed: 2024-04-27.

[11] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 188–197, 2019.

[12] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[15] Oumaima Bellar, Amine Baina, and Mostafa Ballafkih. Sentiment analysis: predicting product reviews for e-commerce recommendations using deep learning and transformers. *Mathematics*, 12(15):2403, 2024.

[16] Peng Jin, Yue Zhang, Xingyuan Chen, and Yunqing Xia. Bag-of-embeddings for text classification. In *IJCAI*, volume 16, pages 2824–2830, 2016.

[17] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[18] Michael P LaValley. Logistic regression. *Circulation*, 117(18):2395–2399, 2008.

[19] Steven J Rigatti. Random forest. *Journal of Insurance Medicine*, 47(1):31–39, 2017.

[20] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4580–4584. Ieee, 2015.

[21] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.