Sentiment Analysis of IMDb Movie Reviews

Abstract

Sentiment Analysis refers to a technique that utilizes Natural Language Processing techniques in order to determine whether a statement is positive or negative. In this project, we will utilize Sentiment Analysis on IMDb movie reviews in order to judge whether a movie has a generally positive or negative favor among public perception. Employing different classification models (SVM, Random Forest, XG-Boost, LSTM, and BERT), we hope to compare their benchmarks to one another to determine which models are best for Sentiment Analysis in order to see which model would be best to use to determine whether the general sentiment about a movie is positive or negative.

1 Introduction

In this era of modern media, movies are released very frequently, but there is not enough time to watch all of them. Information and opinions about them are many and varied, and it's difficult to tell what the common consensus on a movie is. In this project, we aim to remedy this by using Sentiment Analysis of IMDb movie reviews in order to understand whether the public perception of a movie is generally positive or negative so as to guide someone on whether or not they should watch a movie.

Sentiment Analysis is the act of utilizing NLP techniques in order to determine whether a statement is positive, negative, or neutral. This can be done by extracting features from the text, and training models using these features to determine which classification the statements fall under.

However, sentiment analysis presents several challenges, including ambiguity in sentiment, where reviews contain mixed opinions, and sarcasm detection, where literal meaning differs from intended sentiment. In addition, short vs. long reviews create inconsistency, and feature extraction choices significantly affect performance. Domain dependency also complicates interpretation, requiring careful dataset selection and preprocessing.

In this project, we explore multiple text representation methods—Term Frequency-Inverse Document Frequency (TF-IDF), Count Vectorizer with n-grams, Word2Vec (via GloVe), and BERT embeddings. We then compare how these representations perform when used with different classification models: Support Vector Machines (SVM), Random Forest, XGBoost, LSTM, and BERT. Our evaluation is based on standard metrics including Accuracy, Precision, Recall, and F1-score, to determine the most effective combinations for sentiment classification tasks.

2 Related Work

Sentiment analysis has been widely explored in both academic and industry settings, using a variety of traditional and deep learning approaches. Earlier methods often relied on classical machine learning models such as Support Vector Machines (SVM) and Naïve Bayes, combined with handcrafted features like bag-of-words or TF-IDF, as done by Pang et al. [3]. While these approaches can be effective, they typically require manual feature engineering and lack the ability to capture semantic context.

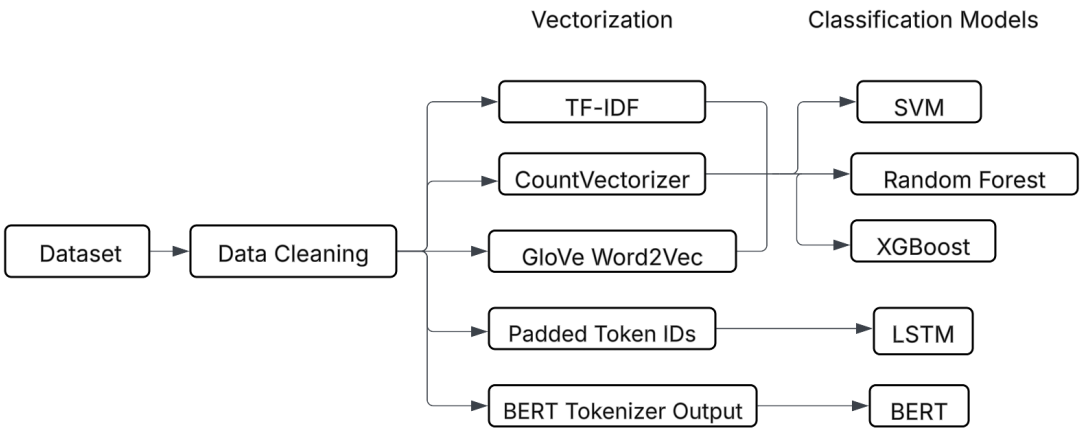
To overcome these limitations, dense word embeddings such as Word2Vec [2] and GloVe [4] were introduced, which represent words in continuous vector spaces and preserve semantic similarity. These representations improved sentiment classification by providing more informative input features. In this project, we use pre-trained GloVe vectors (Twitter-25d) to construct document-level embeddings.

More recently, Devlin et al. [1] proposed BERT, a pre-trained transformer model that achieves state-of-the-art performance on a wide range of NLP tasks, including sentiment analysis. Unlike static word vectors, BERT generates contextualized word embeddings, making it highly effective for handling nuanced and ambiguous expressions.

While prior work has often focused on a single model or vectorization technique, our project aims to provide a broader comparison across multiple models and feature types. Specifically, we evaluate five models—SVM, Random Forest, XGBoost, LSTM, and BERT—paired with TF-IDF, GloVe embeddings, and BERT representations, on the IMDb review dataset to determine which combinations perform best for real-world sentiment classification.

3 Technical Approach

In order to properly evaluate our models, we will follow the methodology outlined in the following figure.



3.1 Dataset

The dataset used in this project is the "IMDb Movie Ratings Sentiment Analysis" dataset available on Kaggle[1]. It consists of 40,000 short movie reviews, each labeled with a sentiment value: 1 for positive and 0 for negative. The dataset is balanced, with approximately 50% of the reviews belonging to each sentiment class. The dataset is provided in a CSV file named `IDMb dataset.csv` with two columns: `review` and `label`.

To build and evaluate our models, we split our dataset into training and testing subsets using an 80/20 stratified split. Stratification ensures that the proportion of positive and negative reviews remains consistent across both the training and testing sets, which is crucial in order to prevent class imbalance and ensure a fair comparison of model performance.

3.2 Data Cleaning and Preprocessing

Before feeding the raw text into machine learning models, we perform a series of preprocessing steps to improve text quality, reduce noise, and ensure uniformity in representation. The preprocessing pipeline consists of two major parts: data cleaning and vectorization.

3.2.1 Data Cleaning

The raw review texts are noisy and unstructured. We apply the following cleaning procedures:

- **HTML tag removal:** Some reviews contain embedded HTML code. We use `BeautifulSoup` to strip all HTML tags and retain only the readable content.
- **Lowercasing:** All text is converted to lowercase to ensure that words like "Good" and "good" are treated as the same token.
- **Removal of special characters and digits:** Using regular expressions, we remove URLs, email addresses, punctuation, and numerical values, as they typically do not contribute to sentiment understanding.
- **Stopword removal:** We eliminate common English stopwords (e.g., "the", "is", "was") using the NLTK stopword list to reduce noise and focus on sentiment-bearing words.
- **Lemmatization:** All remaining tokens are reduced to their base forms using NLTK's `WordNetLemmatizer`. For example, "playing" becomes "play". This step helps unify similar terms and reduce dimensionality.
- **Tokenization:** The cleaned text is tokenized using NLTK's `word_tokenize` into individual words to facilitate later processing.
- **Joined Tokens:** For TF-IDF and count-based methods, we reconstruct the tokenized words into whitespace-separated strings to meet the input format of `TfidfVectorizer`.

3.2.2 Vectorization

To convert the cleaned text into numerical input suitable for machine learning models, we apply four vectorization techniques, each chosen to suit different model architectures:

[1] `https://www.kaggle.com/datasets/yasserh/imdb-movie-ratings-sentiment-analysis`

- **TF-IDF (1-gram):** For traditional machine learning models (SVM, Random Forest, XGBoost), we apply `TfidfVectorizer` using only unigrams. This sparse feature representation captures the importance of individual terms by weighting them inversely by document frequency. We cap the vocabulary at 20,000 terms to limit dimensionality and overfitting.
- **CountVectorizer (1–3-gram):** We also use `CountVectorizer` with an n-gram range of 1 to 3 to extract frequently occurring word sequences (unigrams, bigrams, and trigrams). This approach captures simple phrase patterns without any weighting, and is particularly useful for traditional models that benefit from phrase-level features.
- **GloVe Embeddings:** We use the pre-trained 25-dimensional `glove-twitter-25` embeddings from Gensim to convert each review into a dense vector by averaging the word vectors of all valid (non-stopword) tokens. This embedding-based representation captures semantic similarities between words and provides dense input for traditional classifiers.
- **Deep Learning Tokenization:** For deep learning models, we use model-specific tokenization. The LSTM model uses Keras' `Tokenizer` to convert text into padded sequences of token IDs, which are fed into a trainable `nn.Embedding` layer. The BERT model uses HuggingFace's `bert-base-uncased` tokenizer and directly consumes token IDs and attention masks as input to a transformer-based architecture.

3.3 Classification Models

In this project, we implemented and evaluated five distinct classification models to perform sentiment analysis on the IMDb dataset. Each model was trained and tested on the same splits to ensure consistent comparison. The models include: Support Vector Machine (SVM), Random Forest, XGBoost, Long Short-Term Memory (LSTM), and BERT. The input representations used varied depending on the model type.

3.3.1 Support Vector Machine (SVM)

The SVM classifier was implemented using the `LinearSVC` class from `scikit-learn` with `max_iter=5000`. For TF-IDF input, we used `TfidfVectorizer` with `ngram_range=(1,1)` and a vocabulary size capped at 20,000. The Count-based model used `CountVectorizer` with `ngram_range=(1,3)` to capture unigrams, bigrams, and trigrams. The GloVe-based input was constructed by averaging the 25-dimensional word embeddings from the pre-trained `glove.twitter.27B.25d` model for each token in a review, after removing stopwords.

3.3.2 Random Forest Classifier

We used the `RandomForestClassifier` from `scikit-learn` with 100 trees and `n_jobs=-1` to enable parallel processing. Like the SVM, the Random Forest models were trained using three kinds of feature inputs: TF-IDF with unigrams, Count-based 1–3 gram vectors, and averaged GloVe embeddings. Each input representation followed the same preprocessing and tokenization pipeline as in the SVM setup.

3.3.3 XGBoost Classifier

For gradient boosting, we used the `XGBClassifier` from the `xgboost` Python package. The models were initialized with `use_label_encoder=False`, `eval_metric="logloss"`, and a fixed ran-

dom seed for reproducibility. Similar to the other classifiers, XGBoost was trained on TF-IDF features using unigrams, Count vectors with n-grams ranging from 1 to 3, and dense feature vectors generated from 25-dimensional GloVe embeddings.

3.3.4 LSTM (Long Short-Term Memory)

LSTM networks are a type of Recurrent Neural Network (RNN) designed to solve the vanishing gradient problem through gated mechanisms. Each unit in an LSTM includes a forget gate (f_t), input gate (i_t), candidate cell state ($\tilde{C}_t$), output gate (o_t), and internal cell state (C_t). These gates control the flow of information through time and allow the model to retain or forget past information, making it particularly useful for sequence tasks such as sentiment analysis.

Component	**Symbol**	**Function**
Forget Gate	f_t	Controls whether to discard information from the previous cell state.
Input Gate	i_t	Controls how much of the new input to write to the cell state.
Candidate Cell State	$\tilde{C}_t$	Proposes new candidate values to be added to the state.
Output Gate	o_t	Determines how much of the cell state is exposed to the hidden state h_t.

Table 1: Summary of components in an LSTM unit.

The equations governing the update steps are as follows:

$$
\begin{aligned}
f_t &= \sigma\left(W_f[h_{t-1}, x_t] + b_f\right), \\
i_t &= \sigma\left(W_i[h_{t-1}, x_t] + b_i\right), \\
\tilde{C}_t &= \tanh\left(W_C[h_{t-1}, x_t] + b_C\right), \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \\
o_t &= \sigma\left(W_o[h_{t-1}, x_t] + b_o\right), \\
h_t &= o_t \odot \tanh(C_t).
\end{aligned}
$$

In our implementation, we built a custom LSTM classifier using `PyTorch`. Raw movie reviews were tokenized with NLTK and converted to integer IDs using a vocabulary of the top 20,000 most frequent tokens. Each sequence was padded to a fixed length of 256 tokens. The network consisted of:

- **Embedding layer:** 100-dimensional, learned from scratch
- **LSTM encoder:** 2 layers, bidirectional, hidden size 128
- **Pooling:** Average-pooling and max-pooling combined
- **Classification:** Concatenated pooled features passed to a fully connected layer
- **Dropout:** 0.3 applied after pooling to reduce overfitting
- **Training:** Adam optimizer, learning rate 1e-3, 10 epochs, batch size 64

3.3.5 BERT (Transformer-Based)

BERT stands for Bidirectional Encoder Representations from Transformers. It is a deep learning model developed by Google AI that is used for language understanding using a transformer architecture. Essentially, BERT is multiple Transformer encoders stacked on top of one another, with each layer containing multi-headed self attention and feed-forward networks.

The BERT-based sentiment classifier in this project is implemented using the `bert-base-uncased` model from HuggingFace Transformers. It contains 12 Transformer encoder layers, with a hidden size of 768 and 12 attention heads. It takes as input a tokenized embedding, created using the `bert-base-uncased` tokenizer, which will lowercase all input text, split it into subwords, and then add `[CLS]` and `[SEP]` tokens, which are classification and separation tokens, respectively. The tokenized text and its labels are then wrapped into a PyTorch dataset, which will truncate or pad these inputs to 256 tokens each, and will then also pass along with an attention mask that has 1's where tokens exist, and 0's where padding is in place. Our dataloader then groups the dataset into batches of 16 samples to be fed to the model during training. BERT is pretrained on Masked Language Modeling, where it masks 15% of tokens and attempts to predict the masked token based on the context of the sentence. It is also trained on Next Sentence Prediction, where given two sentences in a sequence, it attempts to predict whether the second sentence would logically follow the first. It uses this pretraining as we finetune it on our task of sentiment analysis in order to have it predict whether the sentiment of a review is positive or negative.

The model architecture we made builds on top of the base BERT encoder and concatenates two representations: (1) the [CLS] token from the last hidden layer and (2) the mean-pooled embedding of all contextualized token vectors, weighted by attention mask. This combined representation is passed through a two-layer classification head with GELU activation and dropout, followed by a linear output layer. We utilized the AdamW optimizer, and a learning rate of 2×10^{-5}.

The model was trained using cross-entropy loss and gradient clipping (max_norm = 1.0) over 10 epochs. Evaluation metrics including Accuracy, Precision, Recall, and F1-score were computed on the 20% test split using `sklearn`. Performance was tracked and visualized across 10 epochs.

4 Results and Discussion

We evaluated every model on the held-out test split (8,000 reviews, 50% positive and 50% negative). Four metrics are reported: **Accuracy**, **Precision**, **Recall**, and **F1-score**, all computed with `sklearn`. The confusion-matrix notation used to derive them is shown in Table 2.

Table 2: Confusion-matrix notation.

	True label	
Pred.	Positive	Negative
Positive	TP	FP
Negative	FN	TN

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

Table 3: Test-set performance of every model / vectorization pair.

Model	Vectorizer / input	Accuracy	Precision	Recall	F1
SVM (Linear)	TF-IDF (1-gram)	0.888	0.89	0.89	0.89
	Count (1–3)	0.863	0.86	0.86	0.86
	GloVe (25-d avg.)	0.756	0.756	0.756	0.756
Random Forest	TF-IDF (1-gram)	0.853	0.85	0.85	0.85
	Count (1–3)	0.850	0.85	0.85	0.85
	GloVe (25-d avg.)	0.750	0.750	0.750	0.750
XGBoost	TF-IDF (1-gram)	0.851	0.85	0.85	0.85
	Count (1–3)	0.853	0.85	0.85	0.85
	GloVe (25-d avg.)	0.753	0.753	0.753	0.753
LSTM	Word-index seq.	0.888	0.866	0.916	0.891
BERT	fine-tuned CLS	0.902	0.907	0.897	0.902

4.1 Impact of feature representation

Sparse TF-IDF vs. Count n-grams. Unigram TF-IDF consistently out-performs 1–3-gram raw counts on all three classical classifiers (SVM : +2.5 pp[2]; RF : +0.3 pp[3]; XGBoost : -0.2 pp[4]). The additional high-order n-grams introduce noise without delivering extra sentiment signal.

Dense GloVe averages. Averaging 25-dimensional GloVe vectors reduces the input to only 25 features, but costs roughly 13–14 pp[5] accuracy compared with TF-IDF. The coarse average evidently obliterates fine-grained polarity cues (e.g. negation) that linear or tree models need.

4.2 Model comparison

- **Linear SVM** is still the strongest traditional classifier on sparse vectors (0.888 accuracy).
- **Tree ensembles** (RF, XGBoost) trail SVM by about 3–4 pp[6], but are less brittle to high-order n-grams and class imbalance.
- **LSTM** overtakes all classical baselines, reaching 0.888 accuracy with only 100-dim embeddings, confirming the benefit of sequence modelling.
- **Fine-tuned BERT** sets the new high-water mark (0.902 accuracy, 0.902 F1), showing that contextual transformer features capture sentiment nuances missed by bag-of-words representations.

Overall, deep contextual models clearly surpass both sparse and static-embedding baselines, but at the cost of $\sim$10$\times$ training time and GPU resources.

[2] 0.888 - 0.863 = 0.025 = 2.5 pp

[3] 0.853 - 0.850 = 0.003 = 0.3 pp

[4] 0.851 - 0.853 = -0.002 = -0.2 pp accuracy

[5] e.g., SVM TF-IDF (0.888) vs SVM GloVe (0.756) $\rightarrow$ 0.132 = 13.2 pp; RF: 0.853 - 0.750 = 10.3 pp; XGBoost: 0.853 - 0.753 = 10.0 pp

[6] SVM (TF-IDF 1-gram) = 0.888; RF = 0.853 $\rightarrow$ diff = 3.5 pp; XGBoost = 0.851 $\rightarrow$ diff = 3.7 pp

5 Conclusion and Future Work

This project conducted a comprehensive evaluation of multiple machine learning and deep learning models for sentiment analysis on the IMDb movie review dataset. We implemented three text representation techniques—TF-IDF (unigrams and n-gram variants), Count-based n-grams, and pre-trained GloVe Word2Vec embeddings—and evaluated their performance across five classifiers: Support Vector Machines (SVM), Random Forests, XGBoost, LSTM, and BERT.

Our experiments demonstrate that traditional models with sparse vectorizers remain highly competitive. Specifically, TF-IDF unigram features paired with a linear SVM achieved the best classical performance at **88.8%** accuracy, while TF-IDF with six-grams yielded **88.7%** and 1–3-gram raw counts **86.3%**. Random Forest and XGBoost reached up to **85.3%** accuracy using TF-IDF, and used 1–3-gram counts they achieved **85.0%** and **85.3%** respectively. In contrast, averaging 25-dimensional GloVe embeddings with an SVM led to only **75.6%** accuracy (Random Forest: **75.0%**, XGBoost: **75.3%**), confirming that static dense averages lose fine-grained polarity cues.

Deep learning models showcased further gains: our LSTM encoder achieved **88.8%** accuracy, matching the best classical learner, while fine-tuned BERT outperformed all baselines with **90.2%** accuracy and a balanced F1 of **0.902**.

Future Work

Building upon these findings, we identify several avenues for future exploration:

- **Hyperparameter and Architecture Tuning:** Systematic search over LSTM layer sizes, learning rates, and BERT fine-tuning schedules to squeeze additional gains.
- **Error Analysis:** Detailed study of misclassified reviews—particularly sarcastic or negated phrases—to guide feature augmentation and architecture design.
- **Model Interpretability:** Visualization of attention weights in BERT and feature importances in tree models to increase transparency.
- **Cross-Domain Generalization:** Validation of our best models on other review datasets (e.g. Amazon, Yelp) to assess robustness.
- **Ensemble Methods:** Combining sparse and contextual representations (e.g. concatenating TF-IDF with BERT embeddings) may yield further improvements.

In summary, our work shows that carefully engineered classical methods can rival deep architectures, but neural sequence and transformer models provide the highest accuracy when resources permit. These results lay the groundwork for more interpretable, robust, and adaptable sentiment analysis systems.

Github Link

https://github.khoury.northeastern.edu/aswilkinson12/CS6140-Final-Project

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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 (NAACL-HLT)*, pages 4171–4186, 2019.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3111–3119, 2013.

[3] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, 2002.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.