| **IMDB Review Sentiment Analysis** |
| --- |

**Sivagnanam Sakthi Vijay Kumar**

College of Engineering

Northeastern University

Toronto, ON

*sakthivijaykumar.s@northeastern.edu*

**Abstract**

This report analyzes the IMDB reviews dataset which focuses on text analysis and machine learning techniques for sentiment analysis. We will proceed with a comprehensive set of Python libraries and tools for data preprocessing, feature extraction, and model training to classify text data into positive or negative sentiment categories. The dataset used in this analysis is the IMDB Dataset of 50,000 movie reviews, which is split evenly between positive and negative reviews. The notebook demonstrates the application of various machine learning algorithms, including Logistic Regression, Naive Bayes, Support Vector Machines (SVM), and XGBoost, along with deep learning models such as CNN to predict the sentiment of movie reviews. Additionally, it explores advanced text processing techniques such as tokenization, stemming, lemmatization, and the use of TF-IDF vectorization. The report will delve into the methodology, data preprocessing steps, model training and evaluation, and the results obtained from different classifiers. It aims to provide insights into the effectiveness of different machine learning models and text processing techniques in sentiment analysis tasks.

**1 Background**

Sentiment analysis is a subfield of natural language processing (NLP) that involves analyzing text data to determine the sentiment expressed within it, typically categorizing it as positive, negative, or neutral. This Python notebook focuses on sentiment analysis of movie reviews from the IMDB dataset, aiming to automatically classify reviews into positive or negative categories based on their content. The analysis employs several machine learning algorithms and text processing techniques to preprocess the data, extract relevant features, and train models for sentiment classification..

**2 Data Preprocessing**

In the domain of sentiment analysis and natural language processing (NLP), the preprocessing of text data is a crucial step that directly impacts the performance of models. This section of the report discusses a comprehensive text preprocessing pipeline implemented in Python, designed to clean and normalize review texts before they are used for analysis. The preprocessing steps covered include the removal of stopwords, punctuation, HTML tags, URLs, and the application of text normalization techniques such as stemming and lemmatization.

The preprocessing pipeline consists of several functions, each aimed at addressing specific types of noise or irregularities in the text data:

1.Removing Punctuation: Punctuation marks are removed from the text as they are generally not useful for sentiment analysis and can increase the dimensionality of the data.

2.Cleaning Contractions: The text is scanned for contractions (e.g., "isn't", "won't"), and these are expanded to their full form (e.g., "is not", "will not") using a predefined mapping. This standardization helps in reducing the complexity of the text.

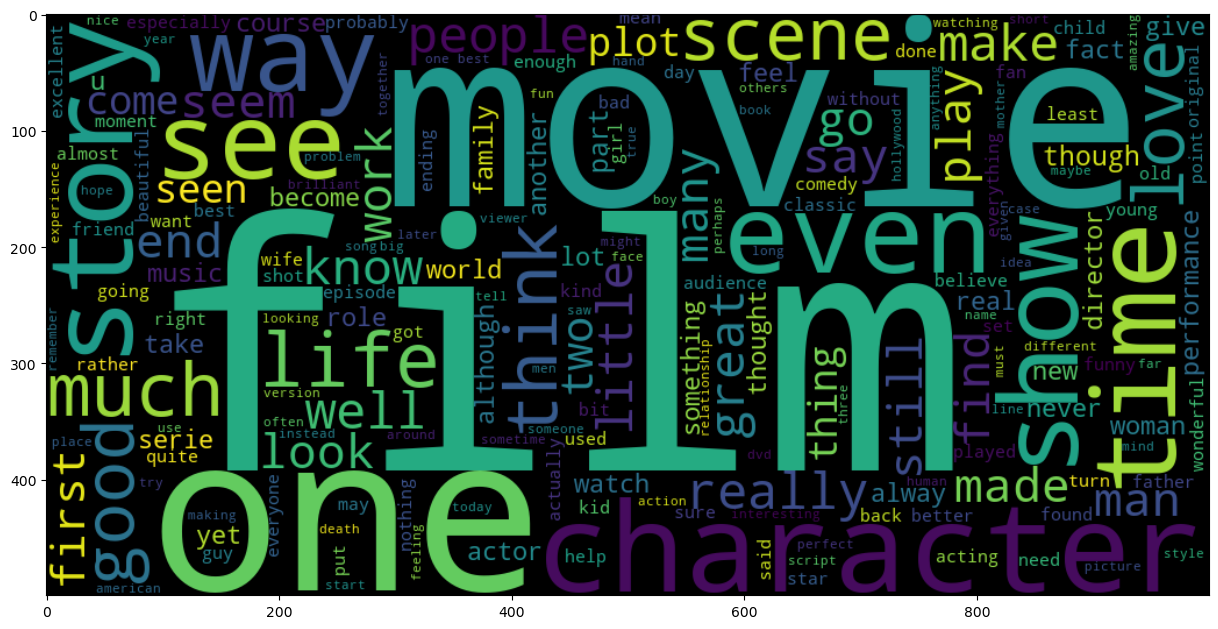
3.Lowercasing: The entire text is converted to lowercase to ensure consistency and to avoid distinguishing between words based solely on their case.

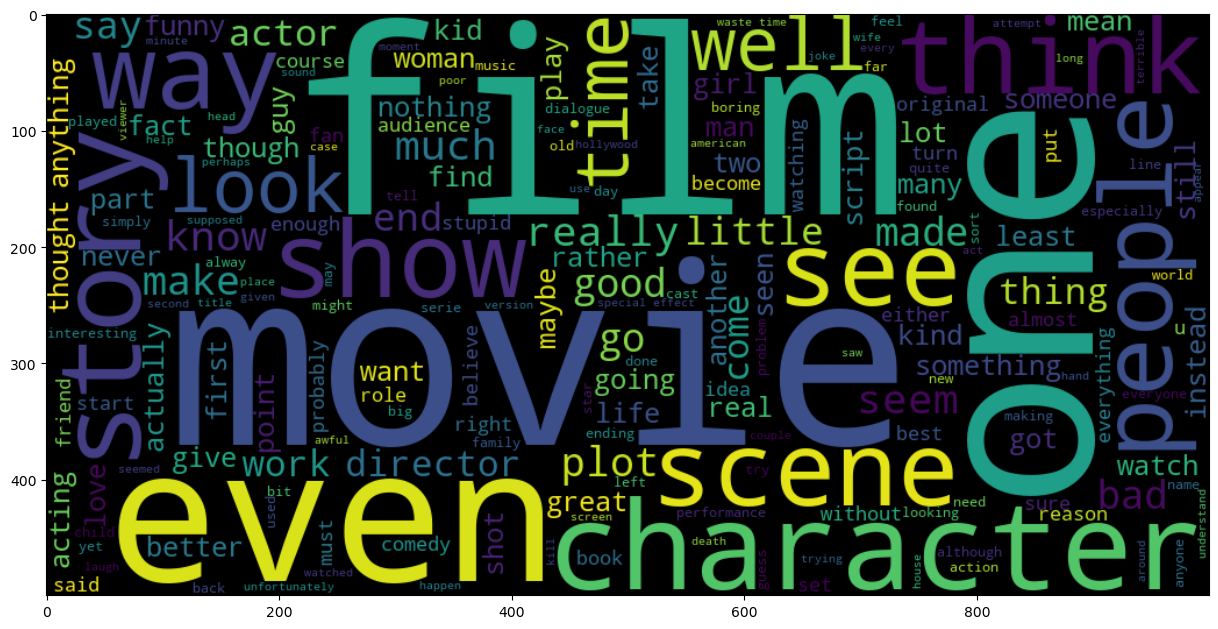
4.Removing HTML Tags: HTML tags, which are irrelevant to text analysis, are removed to clean the data further.

5.Removing URLs: URLs are removed from the text as they do not contribute to understanding the sentiment of the review.

6.Removing Stopwords: Stopwords (common words that offer little value in the analysis) are removed from the text. This step helps in focusing on the words that contribute more significantly to the sentiment.

7.Word Replacement: Specific text patterns (e.g., <br />) are replaced or removed. This step is particularly useful for cleaning data from web sources.

7.Stemming and Lemmatization: These techniques are used to reduce words to their root form. While stemming might lead to the creation of non-existent words, lemmatization ensures that the root word belongs to the language. This process helps in reducing the complexity of the text and in consolidating different forms of a word into a single representation.  
  


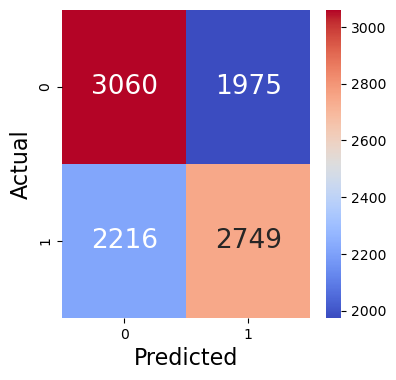


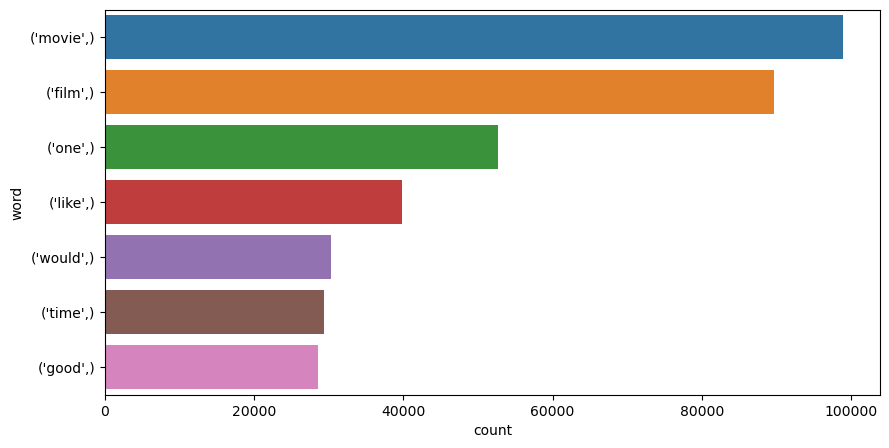
**3. Indirect Features**

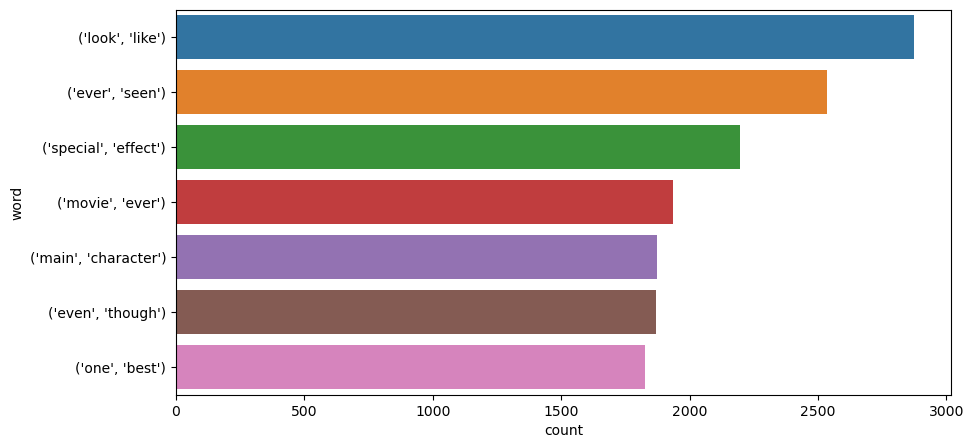
The analysis begins with an exploration of indirect features—attributes derived from the text that do not directly analyze its content. These include metrics like sentence and word counts, the diversity of vocabulary, and punctuation usage, offering insights into the structural and stylistic aspects of the text.

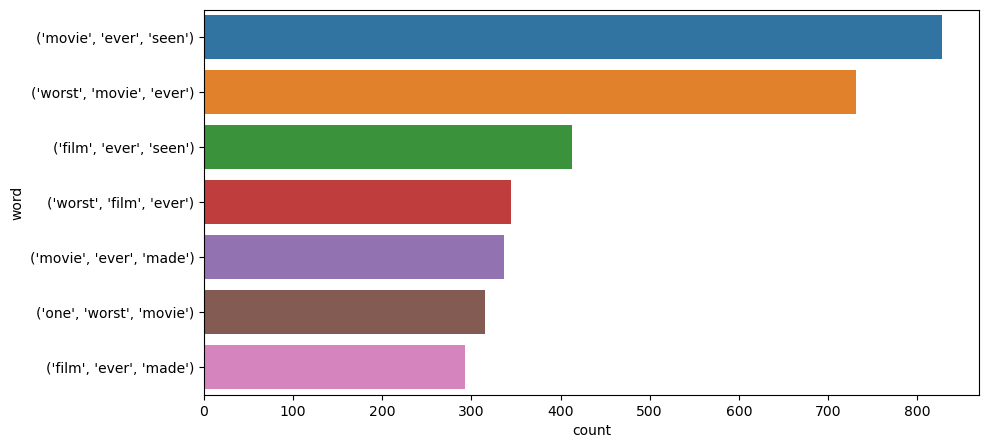
**3.1 Logistic Regression Model**

Utilizing these indirect features, a Logistic Regression model is employed to predict the sentiment of text reviews. This model serves as a foundational analysis, illustrating the potential of structural text attributes to infer sentiment without delving into the textual content. The performance is assessed using accuracy, F1 score, and a detailed confusion matrix visualization.



**3.2 N-gram Analysis**  
  
N-grams are contiguous sequences of 'n' items from a given sample of text or speech. In the context of text analysis, these items are typically words. The variable 'n' denotes the number of words considered in each sequence, leading to unigrams (single words), bigrams (pairs of words), trigrams (triplets of words), and so on. By analyzing these n-grams, we can capture the linguistic patterns that characterize natural language use.The strength of n-gram analysis lies in its simplicity and effectiveness in capturing the local context within text. By analyzing the frequency and arrangement of word sequences, n-grams provide insights into the syntactic patterns that govern language use. This makes n-gram analysis a valuable tool for exploring textual data, enhancing machine learning models, and improving our understanding of language structure and usage.  
  






1)API Request Configuration: The script sets up parameters for the Yelp API request, specifying search term ('Ramen'), location ('Toronto'), and pagination details (offset and limit).

2)Data Retrieval and Pagination: Using a loop, the script iteratively fetches data in batches (of 50 entries per batch) to cover a range of 550 results. This is achieved by adjusting the offset parameter.

3)Error Handling: The script includes an offset ennumerator to manage potential issues during API requests.

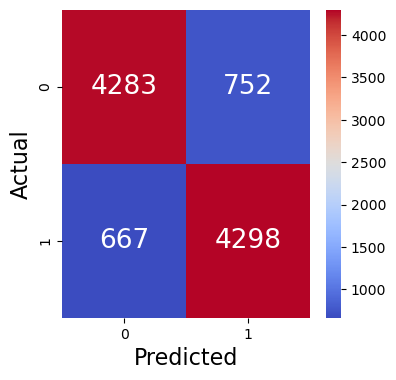
4)Data Aggregation and Storage: The retrieved data is consolidated into a Pandas DataFrame, which is then saved to a CSV file, output\_data.csv, for further analysis or use.

This solution effectively captures and stores relevant data in a structured format, demonstrating the practical application of web scraping and API interaction in Python.

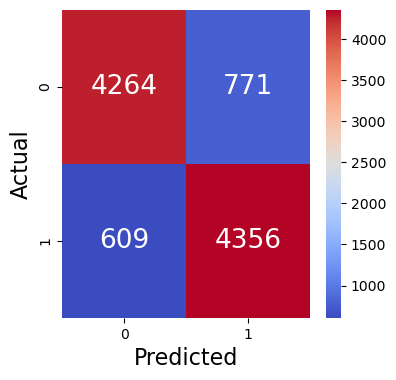
**4 Text Vectorization**

Word Embedding is a sophisticated approach to text processing that seeks to convert words into numerical vectors. This process involves mapping each word to a vector using a predefined dictionary, effectively translating the textual information into a format that ML and DL models can process. The essence of Word Embedding lies in its ability to capture not only the identity of words but also the nuances of their meanings and relationships within the language.Word Embedding offers several advantages over simpler text transformation techniques such as bag-of-words or TF-IDF vectorization. Notably, it captures the semantic relationships between words, meaning that words with similar meanings are represented by vectors that are close to each other in the vector space. This capability allows for a more nuanced understanding and processing of text data, leading to improvements in the performance of ML and DL models on tasks involving natural language.

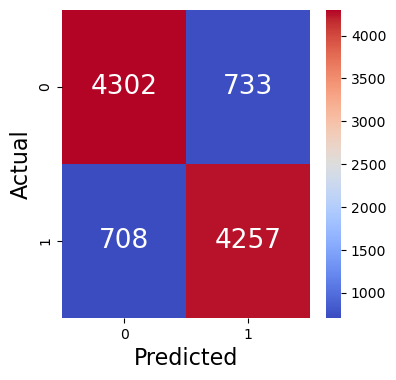
Count Vectorization, also known as the "bag of words" model, is a straightforward yet powerful approach to text representation. This method involves enumerating the occurrence of words within a document, disregarding grammar and word order but maintaining multiplicity. The essence of Count Vectorization lies in its simplicity; it transforms text into a vector where each element represents the frequency of a particular word in the text corpus.

**Naive Bayes with Count  
**

**XGB with Count**

****

**SV with Count**

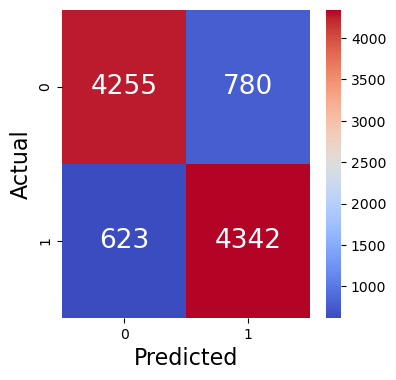
****

.

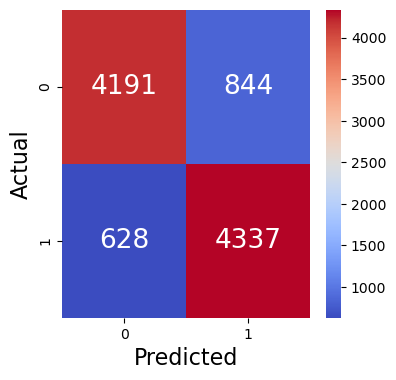
TF-IDF Vectorization addresses the limitations of Count Vectorization by weighting the word frequencies according to their importance in the corpus. The TF component computes the frequency of a word in a document, reflecting its significance within that specific text. The IDF component, on the other hand, inversely weighs the frequency of the word across the entire corpus, diminishing the impact of commonly occurring words and amplifying that of rarer terms.

The combined TF-IDF score represents both the local and global relevance of a word, enabling algorithms to discern and prioritize the most meaningful features of the text. This dual consideration makes TF-IDF an effective tool for tasks requiring nuanced text analysis, such as document classification, search engine indexing, and information retrieval.

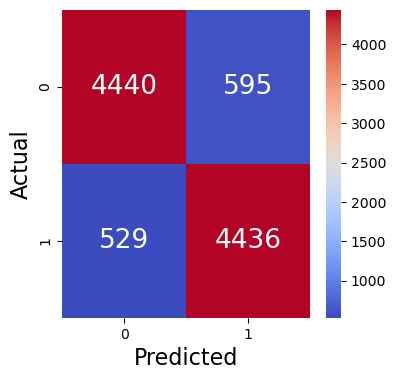
**NB withTFIDF**

****

**XGB with TFIDF**

****

**SV with TFIDF**

****

**5 Deep Learning Approach**

Text Preprocessing and Embedding

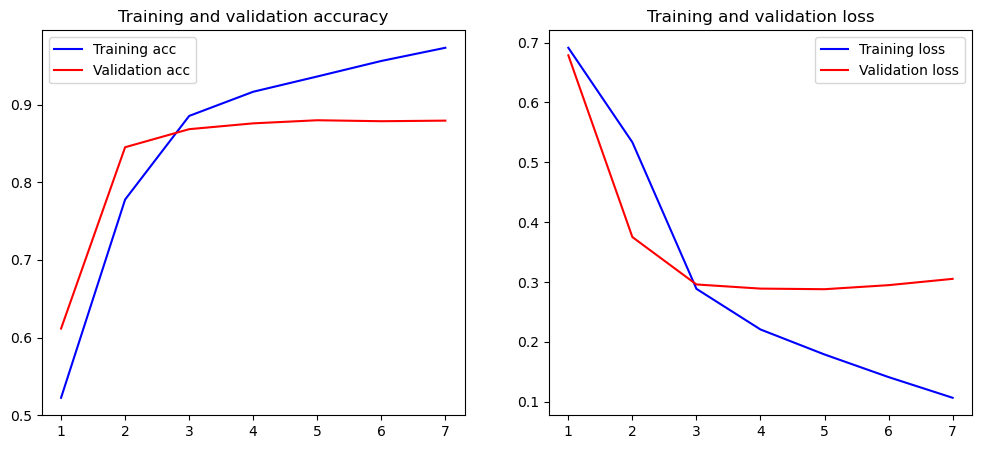
The foundation of any deep learning model for text analysis lies in its preprocessing phase. Utilizing the Keras Tokenizer, the text is first tokenized, converting sentences into sequences of integers, where each integer represents a specific word in a dictionary. To ensure uniformity in input size, these sequences are then padded to a fixed length, facilitating their processing by neural networks.

A crucial aspect of this preprocessing is the application of word embeddings. Unlike simpler vectorization techniques, embeddings map words to high-dimensional vectors, capturing semantic relationships between words. In this exploration, an embedding layer is configured to transform the integer sequences into dense vectors of fixed size, setting the stage for deep learning models to process textual data effectively.

Model Architectures and Training:

Sequential Model with Dense Layers

The first architecture explored is a Sequential model comprising an Embedding layer followed by a Flatten layer and Dense layers. This model aims to capture the semantic essence of text through embeddings and to classify sentiments through its densely connected layers. The training process is closely monitored with callbacks like EarlyStopping to prevent overfitting, ensuring that the model generalizes well to unseen data.

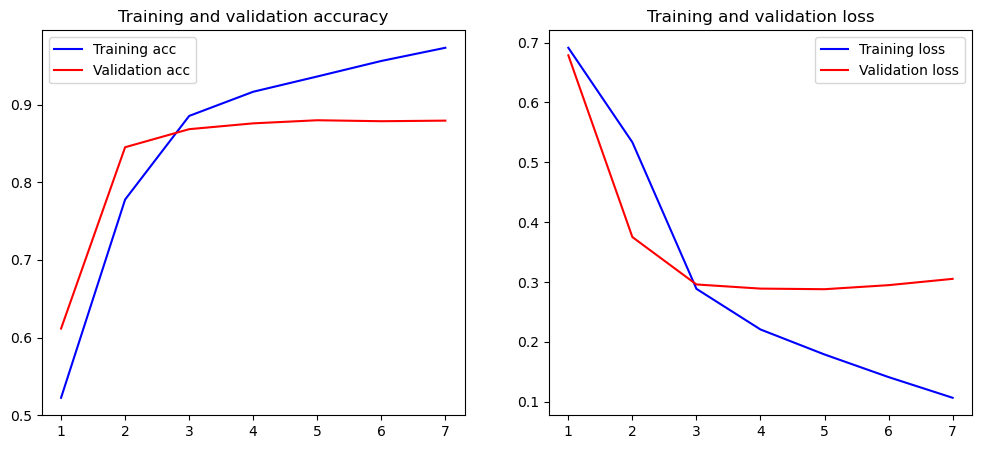


LSTM Model

The exploration progresses to more sophisticated architectures, incorporating Long Short-Term Memory (LSTM) units. LSTMs are adept at capturing long-range dependencies within text, making them particularly suitable for sentiment analysis. The LSTM model includes an Embedding layer, LSTM layers with dropout regularization, and a Dense output layer. This setup is tailored to process sequences effectively, capturing the temporal dynamics of word usage in sentiment expression.

CNN-LSTM Hybrid Model

Finally, a hybrid model combining Convolutional Neural Networks (CNNs) and LSTMs is evaluated. This model integrates the spatial hierarchy of features extracted by CNNs with the sequential processing capabilities of LSTMs. Such an architecture aims to leverage the strengths of both approaches: the CNN's ability to detect local patterns and the LSTM's proficiency in modeling sequence dependencies.



Training Process and Evaluation

Each model undergoes training with a designated training set, validated against a separate validation set to monitor its performance and generalizeability. The EarlyStopping callback is employed across models to halt training when the validation loss ceases to decrease, mitigating the risk of overfitting. The training and validation accuracy and loss are plotted over epochs, providing insight into the models' learning progress and stability.

**6 Conclusion**

The exploration of deep learning techniques for sentiment analysis reveals the potent capabilities of text embeddings and advanced neural network architectures. From dense networks to LSTMs and their hybrids with CNNs, each model offers unique advantages in understanding and classifying sentiments within text. The preprocessing of text into embeddings emerges as a critical step, enabling the nuanced processing of language by deep learning models. As the field progresses, the continuous refinement of these techniques and architectures promises further advancements in the accuracy and efficiency of sentiment analysis.

Through meticulous experimentation and evaluation, this report underscores the importance of tailored preprocessing and model selection in deep learning-based sentiment analysis. The insights garnered pave the way for future research, emphasizing the potential of deep learning in extracting meaningful sentiment insights from textual data, thereby enhancing the understanding of human language and emotions.

**Acknowledgments**

IMDB Dataset  
  
  
  
  
All code is entirely my own.