Natural Language Processing, Report

Introduction:

Sentiment analysis (SA), also referred to as opinion mining, is a branch of natural language processing (NLP) focused on detecting and extracting subjective information from text. This involves determining the sentiment conveyed in the text, which is usually categorized as positive, negative, or neutral. My project is essentially a binary sentiment classification based on the well-known "IMDB" data set. One prominent application of sentiment analysis is in the entertainment industry, specifically in analyzing movie reviews.

Domain Specific Area:

The demand for sentiment analysis has increased due to the growing need to analyze and structure hidden information from social media, which comes in the form of unstructured data. [1] Additionally:

- **Consumer Insights:** By analyzing sentiment in movie reviews, production companies, distributors, and marketers can gain a better understanding of audience reactions and preferences. This feedback can inform future productions and marketing strategies. [2], [3] This field is so widespread that it even had a significant impact on the 2016 US Presidential Elections. [4]
- **Recommendation Systems:** Efficient sentiment analysis enhances recommendation systems on platforms like IMDb and Netflix, enabling them to suggest movies that closely match users' preferences, despite potentially sparse datasets. Furthermore, sentiment analysis has broad applications in other domains such as commodity markets, social media filtering, and various other recommendation systems. [5]
- **Market Prediction:** Sentiment analysis can forecast box office performance based on pre-release reviews and social media sentiment, providing valuable insights for industry stakeholders. [6]

There are several challenges specific to this dataset.

- **Subjectivity and Opinion Variability:** Movie reviews exhibit a wide range of language, style, polysemy, and sentiment expression, making word sense disambiguation crucial for understanding the correct meaning within context. Recurrent Neural Networks (RNN) are particularly effective in addressing this challenge. Additionally, some reviews feature nuanced opinions or sarcasm, often indicated by capitalization and punctuation, which complicates accurate sentiment interpretation for analysis algorithms. [7] Unfortunately, my preprocessing step did not account for this complexity, as I converted all text to lowercase and ignored punctuation.
- **Domain-Specific Language:** Movie reviews frequently include domain-specific terminology, slang, or references that are uncommon in other text genres. To accurately interpret sentiment, algorithms must adapt to this specialized language. Fortunately, pretrained word embeddings with large vocabularies are extremely useful in these situations. [8], [9]
- **Handling Mixed Sentiments:** Reviews often contain mixed sentiments, with some aspects of a movie being praised and others criticized within the same review. Algorithms need to identify and accurately classify these mixed sentiments, particularly when using a binary classifier, as many reviews may actually be neutral.

Objectives

The primary objective of this project is to investigate the effectiveness and applicability of both statistical and embedding-based models for text classification tasks, utilizing machine learning (ML) for the former and deep learning (DL) for the latter. Specifically, the project focuses on the IMDB dataset, which contains 25,000 positive and 25,000 negative reviews, and involves significant preprocessing to clean the text data. The project is divided into three parts: a baseline model using Naive Bayes, a statistical approach employing TF-IDF (Term Frequency-Inverse Document Frequency) and ensemble methods like Support Vector Machines (SVM) and logistic regression, and an embedding-based approach using Word2Vec embeddings [10] with a bidirectional Long Short-Term Memory (LSTM) model [11]. By comparing these approaches, the aim is to identify which yields superior performance and in which scenarios each method is most appropriate. Furthermore, we are experimenting with the hypothesis that using embeddings, especially in conjunction with Recurrent Neural Networks (RNNs) that have a sense of the positioning of the text, may be a preferable way to generalize more effectively given a small corpus of text like the IMDB dataset [12]. While certain papers may argue that SVM is superior. [13]

The findings will contribute to a deeper understanding of text classification techniques and their practical applications in NLP.

Goals of Exploring Statistical and Embedding-Based Models

1. **Compare Performance:**
    - **Statistical Models:** Evaluate traditional models like Naive Bayes and TF-IDF combined with ensemble methods (e.g., SVM, logistic regression). These models are typically easier to interpret and require less computational power. Even though creating hand-crafted features may require additional time for implementation. [13]
    - **Embedding-Based Models:** Assess modern approaches using pre-trained Word2Vec embeddings with bidirectional LSTM networks. These models can capture more complex patterns and contextual information in the text.
2. **Understand Applicability:**
    - Identify scenarios where statistical models perform well, particularly in cases with simpler text data and smaller datasets.
    - Determine conditions where embedding-based models excel, such as when handling more complex texts or capturing nuanced semantic relationships.
3. **Identify Best Practices:**
    - Establish guidelines for preprocessing, feature extraction, and model selection that can be applied to other text classification problems.
    - Provide insights into the trade-offs between model complexity, interpretability, and computational requirements.
4. **Improve Generalization:**
    - Test the hypothesis that embedding-based models, especially those utilizing RNNs, can generalize more effectively on smaller datasets like the IMDB dataset. RNNs' ability to capture word positioning and contextual dependencies might offer significant advantages over traditional methods

Dataset Description

The chosen dataset is the IMDB reivew dataset. This dataset is widely used in sentiment analysis tasks and provides a balanced collection of movie reviews labeled as positive or negative. The dataset consists of 50,000 movie reviews, with 25,000 reviews designated for training and 25,000 for testing.

Dataset Characteristics:

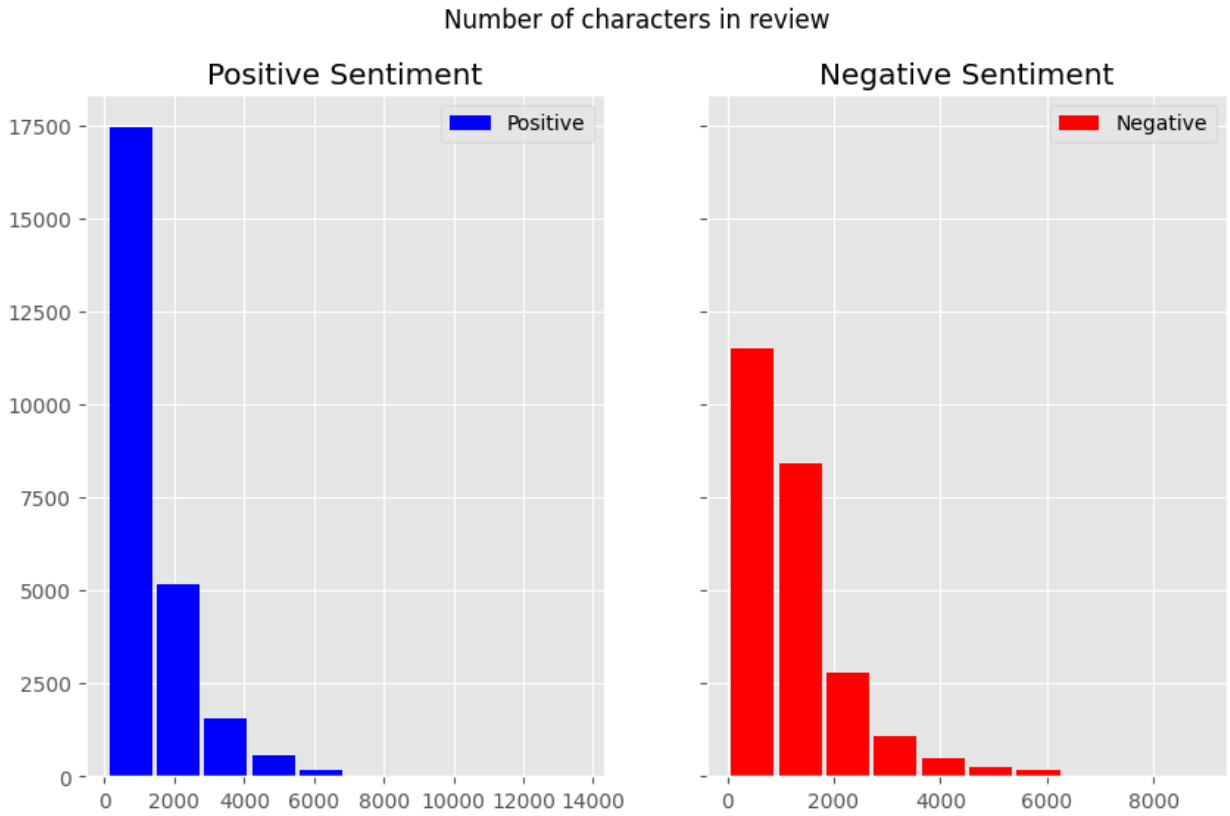
- **Size:** The IMDB Reviews dataset contains 50,000 reviews in total.
- **Data Types:** Each review is a text sequence, and each sentiment label (positive or negative) is assigned as a binary classification task.
- **Acquisition:** The dataset was originally compiled by researchers from Stanford University and is publicly available through TensorFlow Datasets (TFDS) . It was acquired by aggregating reviews from the IMDB website, where users provide their sentiment towards movies they have watched.
- **Platform:** IMDB website (Internet Movie Database), is a popular platform where users submit reviews and ratings for movies.

Relevance to the Project

- **Representative Challenge:** The IMDB Reviews dataset represents a challenging task in sentiment analysis, where the goal is to classify whether a movie review expresses a positive or negative sentiment.
- **Attention to Steps:** The dataset requires attention to preprocessing steps such as text cleaning (removing HTML tags, punctuation, URLs), tokenization, and possibly further normalization (lowercasing, stop word removal).
- **Suitability for LSTM and Embeddings:** The dataset is suitable for exploring the effectiveness of LSTM models in capturing sequential dependencies in text data, augmented by Word2Vec embeddings to enhance the model's ability to generalize across varying review lengths and linguistic nuances.

Visualization of the dataset

Bar Chart

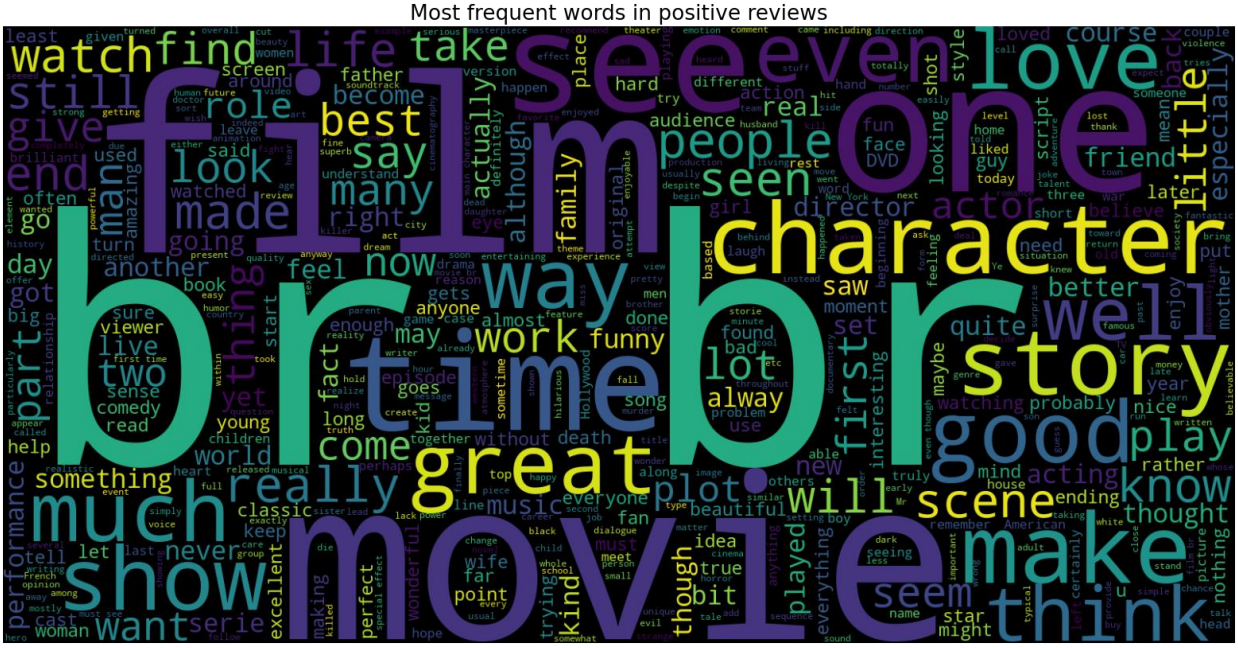


Observations

- Both histograms show that most reviews, regardless of sentiment, tend to be relatively short, with the highest concentration under 2,000 characters.
- Although most reviews, whether positive or negative, are relatively short, there seems to be a pattern where positive reviews are generally even shorter. This might be because those who write negative reviews are often film enthusiasts who take reviewing seriously and provide a thorough analysis of the movie's issues. In contrast, casual viewers like me tend to write brief, positive reviews without much detail.
- Both distributions are right-skewed, indicating that longer reviews are less common.

Word Cloud

Most frequent words in positive reviews



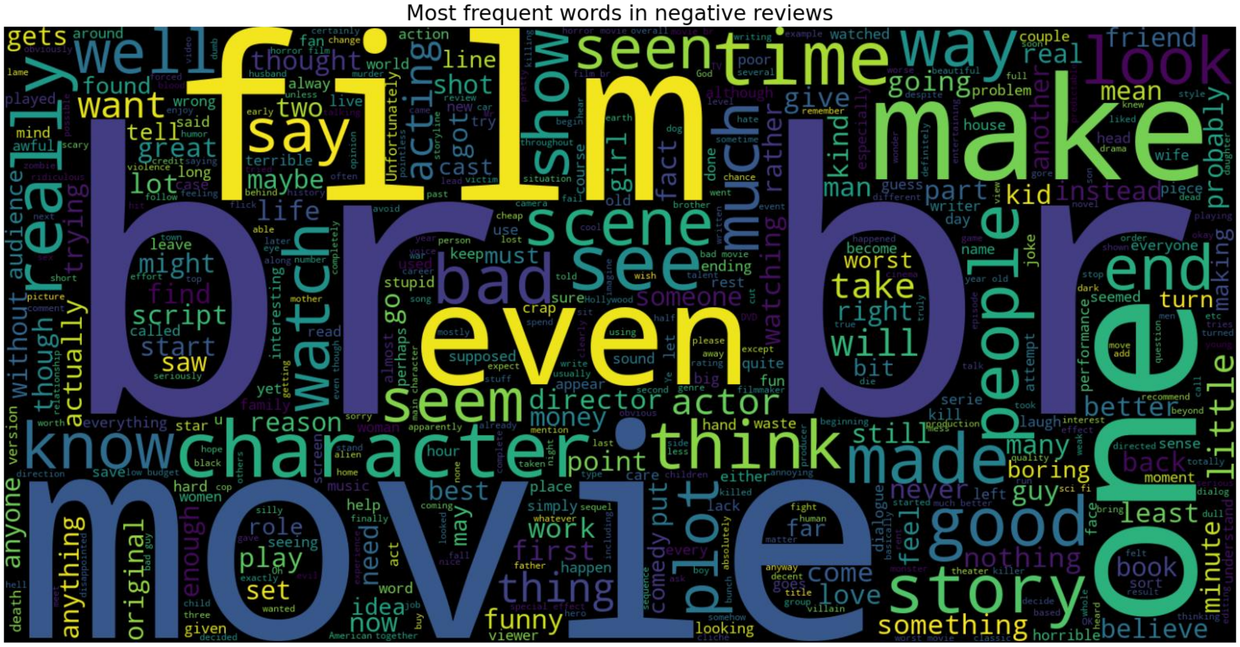
The larger the word appears in the cloud, the more frequently it occurs in the reviews. This helps in identifying common themes or sentiments in the positive reviews at a glance.

Common Words

Some of the most prominent words in the word cloud include:

- **"movie"**
- **"film"**
- **"character"**
- **"watch"**
- **"story"**
- **"good"**
- **"time"**
- **"see"**
- **"plot"**
- **"life"**

Most frequent words in negative reviews



Common Words

Some of the most prominent words in the word cloud include:

- **"movie"**
- **"film"**
- **"one"**
- **"make"**
- **"character"**
- **"even"**
- **"time"**
- **"seen"**
- **"story"**
- **"really"**

These words likely represent common themes or elements that are frequently mentioned in negative reviews.

Evaluation Methodology

When evaluating machine learning models, especially for binary classification tasks like sentiment analysis on the IMDB dataset, several key metrics are commonly used: Accuracy, Precision, Recall, F1-Score, and Area Under the Curve (AUC). Additionally, a Confusion Matrix can provide a detailed breakdown of the model's performance. Below is a comprehensive description of these metrics and how they can be applied to compare the two methodologies in the project.

1. Accuracy

- **Definition:** Accuracy is the ratio of correctly predicted instances to the total instances in the dataset.
- **Formula:** $Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$
  - TP: True Positives
  - TN: True Negatives
  - FP: False Positives
  - FN: False Negatives
- **Application:** Accuracy is a straightforward metric that gives a quick overview of the model's performance. In the deep learning part of the project, accuracy was used during training to monitor the model's progress. Since the dataset is balanced, with equal number of positive and negative reviews. Despite being a basic metric, the balanced nature of the dataset is the reasons it was used as the primary metric.

2. Precision

- **Definition:** Precision is the ratio of correctly predicted positive instances to the total predicted positives.
- **Formula:** Precision $= \frac{TP}{TP + FP}$
- **Application:** Precision evaluates the accuracy of the model's positive predictions by measuring the proportion of correct positive predictions out of all the positive predictions made by the model. Precision is particularly important when the cost of false positives is high. Consider a classifier used to filter explicit content for a search engine intended for minors. In this scenario the classifier needs to have a high precision even at the cost of a lower recall.

3. Recall (Sensitivity)

- **Definition:** Recall is the ratio of correctly predicted positive instances to all actual positives.
- **Formula:** Recall $= \frac{TP}{TP + FN}$

- **Application:** Recall is important in scenarios where capturing all positive instances is crucial. It helps measure how well the model can identify positive cases. In other words, it measures the proportion of actual positive instances (or true positives) that are correctly identified by the model. Consider Google Scholar's search engine, where recall is the more favorable metric.
- **Note:** The precision-recall tradeoff refers to the inverse relationship between precision and recall in a classification model. As you increase the threshold for predicting positive instances to improve precision, recall tends to decrease because fewer actual positives are identified. Conversely, lowering the threshold to capture more positives and improve recall usually reduces precision, as more false positives are included. Therefore, these two metrics are somewhat complementary.

4. F1-Score

- **Definition:** The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that balances both concerns.
- **Formula:** $\text{F1-Score} = 2 \cdot \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$ OBJ
- **Application:** The F1-Score is valuable for striking a balance between Precision and Recall, particularly in scenarios with uneven class distributions. It is also beneficial when both Precision and Recall hold equal significance.

5. Confusion Matrix

- **Definition:** A Confusion Matrix is a table that is used to evaluate the performance of a classification algorithm by comparing the actual versus predicted classifications.
- **Structure:**

	Predicted Positive	Predicted Negative
Actual Positive		
Actual Negative		

- **Application:** The Confusion Matrix offers detailed insights into the model's performance across different classes, facilitating the identification of specific areas where the model may underperform. This is particularly useful for error analysis and refining manually crafted features in machine learning applications.

6. Area Under the Curve (AUC)

- **Definition:** AUC is the area under the Receiver Operating Characteristic (ROC) curve. It provides a measure of the model's ability to distinguish between positive and negative classes.
- **Formula:** The AUC is determined from the ROC curve, which graphs the True Positive Rate (Recall) against the False Positive Rate (1 - Specificity). In scikit-learn,

it is computed using the trapezoidal rule, an approximation method for calculating the area under a curve.

- **Application:** AUC is valuable for model comparison since it offers a unified metric that encapsulates the model's performance across various classification thresholds. It contrasts the model's performance with that of a simple dummy classifier, depicted by a straight diagonal line in ROC curve plots, which has 0.5 AUC as expected.

Applying These Metrics to Compare Methodologies

After fitting and fine-tuning the three models, we assess their performance using the previously mentioned metrics. The evaluation process is greatly facilitated by the user-friendly interface of the scikit-learn library.

Hypothesis

The project aims to demonstrate that embedding-based models, specifically those using Word2Vec and bidirectional LSTMs, can generalize more effectively even with a relatively small corpus like the IMDB dataset. This effectiveness is attributed to their ability to capture context and positional information within the text. The validity of this hypothesis can be easily verified or refuted using the aforementioned metrics.

Implementation

Preprocessing methods

Stemming

Pros

1. Dimensionality Reduction:

- **Explanation:** Stemming reduces the number of distinct words in the text data by converting different forms of a word to a common base form.
- **Benefit:** This can reduce the complexity of the model and improve its performance by grouping similar words together.

2. Standardization:

- **Explanation:** Words like "running", "runs", and "ran" are reduced to a common stem ("run").
- **Benefit:** This helps in standardizing the text data, making it easier to analyze and compare.

3. Improved Model Performance:

- **Explanation:** By reducing the vocabulary size, stemming can help in improving the efficiency of machine learning algorithms.
- **Benefit:** This can lead to faster training times and potentially better generalization. Regular statistical methods often generate a high-dimensional, sparse feature space, complicating training and increasing the risk of overfitting. Stemming can help reduce this risk.

4. Simplicity and Ease of Use:

- **Explanation:** The code is straightforward and easy to understand, utilizing the PorterStemmer from the NLTK library.
- **Benefit:** Easy to implement and integrate into existing text processing pipelines.

Cons

1. Loss of Context:

- **Explanation:** Stemming can be aggressive, sometimes reducing words to stems that are not actual words (e.g., "running" to "run", "better" to "bet").
- **Drawback:** This can result in loss of meaning and context, which might negatively impact the performance of some models.

2. Reduced Accuracy:

- **Explanation:** Stemming does not consider the context in which a word is used, leading to potential inaccuracies (e.g., "organiz" from "organization" and "organizing").
- **Drawback:** This can reduce the accuracy of models that rely on precise word forms, such as language models.

3. Over-Simplification:

- **Explanation:** The stemmed words might not represent the full semantic meaning of the original words.
- **Drawback:** This oversimplification can lead to information loss, which might be crucial for certain text analysis tasks.

4. Ambiguity:

- **Explanation:** Different words might be reduced to the same stem even if they have different meanings (e.g., "arm" (body part) and "army" (military force) both reduced to "arm").
- **Drawback:** This can introduce ambiguity and confusion in the text data, potentially impacting model performance.

Comparison with Other Methods

1. Lemmatization:

- **Explanation:** Lemmatization reduces words to their base or dictionary form (lemma) considering the context (e.g., "better" to "good"). Suitable for morphologically complex languages like Arabic and German.
- **Pros:** More accurate in preserving meaning and context; less likely to produce non-words.
- **Cons:** More computationally intensive and slower than stemming; requires more resources and complex implementation.

2. N-grams:

- **Explanation:** N-grams involve creating combinations of adjacent words or characters (e.g., bigrams, trigrams).
- **Pros:** Captures contextual information and word order; useful for certain tasks like language modeling and text generation.
- **Cons:** Can result in a very large feature space; more complex and resource-intensive.

3. Word Embeddings (e.g., Word2Vec, GloVe):

- **Explanation:** Word embeddings map words to dense vectors in a continuous vector space, capturing semantic relationships and context.
- **Pros:**
    - Provides rich semantic meaning and context.

    - Supports various downstream tasks such as clustering, classification, and recommendation.
    - Embeddings can capture complex relationships and analogies between words.
    - Pre-trained models are available, which can save time and computational resources.
- **Cons:**
    - Requires significant computational resources for training from scratch.
    - Complex implementation and integration compared to stemming.
    - Might require fine-tuning for specific tasks or domains to achieve optimal performance.
    - Potentially large memory footprint, especially for very large vocabularies.

4. Sub word Tokenization (e.g., BPE, SentencePiece):

- **Explanation:** Breaks words into sub word units, capturing morphological features (e.g., "unhappiness" to "un", "happi", "ness").
- **Pros:**
    - Handles out-of-vocabulary words effectively.
    - Preserves semantic meaning better than simple stemming.
    - Allows for more granular understanding of word components.
- **Cons:**
    - More complex than stemming.
    - Requires pre-training and substantial computational resources.
    - May introduce additional complexity in the text preprocessing pipeline.

TF-IDF

Pros

1. Feature Extraction:

- **Explanation:** ML and DL models require numerical input. Text data, being raw and unstructured, needs to be converted into a numerical format.
- **Benefit:** TF-IDF effectively transforms text into numerical representations, considering both term frequency (TF) and inverse document frequency (IDF). This dual consideration helps capture the importance of words within individual documents and across the entire dataset.
- **Formula:**

A) $TF(t,\ d)\ =\ \frac{Count\ of\ term\ 't'\ in\ document\ 'd'}{total\ terms\ in\ 'd'}$

- t = Term (word)
- d = Document
- total terms in d = Total number of terms (words) in document d

B) $IDF(t)\ =\ \log\left(\frac{total\ number\ of\ documents}{nuber\ of\ documents\ contating\ term\ 'd'}\right)$

- t = Term (word)
- total terms in d = Total number of terms (words) in document d
- number of documents containing term t = number of documents that contain at least of term t

C) $TF-IDF(t,\ d)\ =\ TF(t,\ d)\times IDF(t)$

2. Dimensionality Reduction:

- **Explanation:** By using TF-IDF, the text is transformed into a vector space model where each document is represented as a vector of weights.
- **Benefit:** This representation reduces the dimensionality of the text data while retaining the most important information. It compresses the data into a more manageable form, which can enhance the efficiency of subsequent machine learning algorithms.

3. Improves Model Performance:

- **Explanation:** TF-IDF helps in identifying the most important words in each document.
- **Benefit:** Words that are common across documents get lower weights, whereas unique and important words get higher weights. This differentiation helps improve the performance of machine learning models by focusing on terms that are more informative and relevant for the given task.

4. Easy Implementation:

- **Explanation:** TF-IDF is a well-established method available in many machine learning libraries (e.g., scikit-learn in Python).
- **Benefit:** This makes it easy to implement and integrate into existing data processing pipelines. The simplicity of the algorithm also means it requires less computational resources compared to more complex methods like word embeddings.

Cons

1. Sparsity:

- **Explanation:** The TF-IDF matrix can be very sparse, especially for large vocabularies.
- **Drawback:** Sparse matrices can lead to increased memory usage and computational overhead, potentially slowing down model training and inference.

2. Lack of Context:

- **Explanation:** TF-IDF treats each word independently and does not consider the context in which words appear.
- **Drawback:** This can result in loss of meaning, as TF-IDF does not capture relationships between words (e.g., "New York" as a single entity versus "new" and "york" independently). Unlike N-gram tokenization.

3. Fixed Vocabulary:

- **Explanation:** TF-IDF relies on a fixed vocabulary determined at the time of training.
- **Drawback:** This can be problematic when dealing with new, unseen words or when the vocabulary evolves, as the model may not effectively handle out-of-vocabulary terms. Unlike embeddings where this issue is not a problem at all.

4. Sensitivity to Noise:

- **Explanation:** Common words that appear in many documents are down-weighted, but rare words (which might be noise) can get high weights.
- **Drawback:** This sensitivity can lead to overemphasis on rare terms that might not be meaningful, potentially degrading model performance.

5. Scalability Issues:

- **Explanation:** For very large datasets, calculating TF-IDF can become computationally expensive.

- **Drawback:** This can limit the scalability of the method, making it less suitable for extremely large text corpora without significant computational resources. Although this is not the case in our project since IMDB is a relatively small dataset.

6. Limited to Bag-of-Words Model:

- **Explanation:** TF-IDF is based on the bag-of-words model, which ignores the order and structure of words.
- **Drawback:** This limitation can be a disadvantage for tasks where word order and syntax are important, like our sentiment analysis.

Some other preprocessing methods used in the project

Lowercasing:
- **Pros**: Converts text to lowercase to ensure uniformity and reduces complexity by avoiding case sensitivity issues.
- **Cons**: May lose information where case carries meaning (e.g., proper nouns, sarcasm).

**Removing
 Tags and URLs**:
- **Pros**: Cleans the text by removing HTML tags and URLs, which are often irrelevant for sentiment analysis.
- **Cons**: None significant for IMDB dataset, where HTML tags and URLs don't carry sentiment information.

Removing @ and # Symbols:
- **Pros**: Cleans the text of mentions and hashtags which are irrelevant in the IMDB review context.
- **Cons**: None significant for IMDB dataset. Mostly # symbols are used in social media application like tweeter.

Removing Punctuation:
- **Pros**: Simplifies text by removing punctuation which is often noise in sentiment analysis.
- **Cons**: May lose sentiment nuances (e.g., exclamations, question marks, sarcasms).

Tokenization:
- **Pros**: Breaks text into individual words (tokens), which is essential for most NLP tasks.
- **Cons**: No significant cons; it's a standard procedure.

- **Alternatives:** Could also use more advanced tokenizers (e.g., spaCy, BERT tokenizer).

Stop words Removal:

- **Pros**: Removes common words that usually do not carry significant sentiment (e.g., "the", "is").
- **Cons**: May remove words that could be important in certain contexts, but generally they are not.

Frequency Table:

- **Pros**: Generates a frequency distribution of words, which is useful for understanding the importance of different words in the dataset.
- **Cons**: Basic frequency counts do not capture context or semantics.

Embedding method: Word2vec

Origin and Creation: Word2Vec is a seminal word embedding technique developed by Tomas Mikolov et al. at Google in 2013. It was created to address the limitations of traditional sparse vector representations like TF-IDF, which fail to capture semantic meanings and relationships between words. Word2Vec uses neural network models to learn continuous vector representations (embeddings) that encode semantic similarities based on word co-occurrence patterns in large text datasets.

How It Works: Word2Vec operates using two primary architectures:

1- **Continuous Bag of Words (CBOW)**: Predicts the current word given its context (surrounding words).
2- **Skip-gram**: Predicts surrounding words given the current word.

Both architectures learn embeddings by adjusting the model's weights to minimize the prediction error, effectively capturing semantic relationships among words.

Pros:

- **Semantic Meaning**: Word2Vec embeddings capture semantic relationships between words, enabling nuanced understanding in natural language processing tasks.
- **Efficiency**: Once trained, embeddings are computationally efficient to use compared to sparse representations like TF-IDF.

- **Generalization**: They generalize well even with smaller datasets, making them versatile for various NLP tasks.
- **Pre-trained Models**: Pre-trained Word2Vec models are available for direct use in different applications, saving training time. Like this project.
- **Fine-tuning:** Word2Vec models could potentially get trained on a particular dataset, enabling them to capture domain-specific semantics and enhance the robustness of sentiment analysis. This frequently requires a huge dataset as well, which was not the case in this project

Cons:

- **Fixed Vocabulary**: Unable to handle out-of-vocabulary words effectively unless supplemented with techniques like subword embeddings.
- **Context Window**: Performance can be sensitive to the choice of context window size and embedding dimensions.
- **Computationally Intensive Training**: Training from scratch can be computationally expensive for large datasets.

Comparison to Other Embedding Methods:

- **TF-IDF**: TF-IDF calculates a weight representing the importance of a term in a document relative to a corpus. It is sparse and interpretable but lacks semantic meaning and context awareness compared to Word2Vec.
- **GloVe (Global Vectors for Word Representation)**: Like Word2Vec, GloVe captures word semantics based on global word-word co-occurrence statistics, but it optimizes directly for word vectors using matrix factorization techniques. The choice between Word2Vec and GloVe often depends on the specific task and dataset size. Word2Vec is favored in tasks requiring context-sensitive embeddings, while GloVe is preferred for tasks benefiting from a global understanding of word semantics.
- **FastText**: FastText extends Word2Vec by considering subword information, allowing it to generate embeddings for out-of-vocabulary words and improve performance on morphologically rich languages. This approach enhances robustness in scenarios where vocabulary is dynamic or incomplete.

Summary:

Word2Vec pioneered the generation of dense, context-aware word embeddings, transforming natural language processing tasks by capturing semantic relationships effectively. Its reliance on neural network models and training through co-occurrence patterns has set a standard for embedding techniques. However, its limitations include

sensitivity to data quality, fixed vocabulary constraints, and computational demands for training. Compared to TF-IDF and similar sparse representations, Word2Vec excels in semantic understanding and efficiency, yet falls short in adaptability to out-of-vocabulary terms. In contrast, newer models like FastText offer advancements by incorporating subword information or contextualized embeddings, pushing the boundaries of NLP capabilities beyond what Word2Vec initially achieved.

Models Used

Baseline Performance

Naive Bayes

I decided to use the Naive Bayes algorithm as the baseline of my work due to its simplicity and computational efficiency. Below, I have outlined the main pros and cons of this algorithm in its role as a baseline algorithm.

Formulas:

A) Posterior Probability:

$$P(C \mid d) \propto P(C) \cdot i = 1 \prod_{i=1}^{n} nP(wi \mid C)$$

B) Class Prior Probability:

$$P(C) = \frac{Number\ of\ documents\ in\ class\ 'C'}{Total\ number\ of\ documents}$$

C) Likelihood of a Word Given a Class (with Laplace Smoothing):

$$P(wi \mid C) = \frac{Count\ of\ 'wi'\ in\ documents\ of\ class\ 'C\ +\ 1'}{Total\ number\ of\ words\ of\ class\ 'C'\ +\ Vocabulary\ size}$$

Pros:

- **Simplicity and Efficiency**: Naive Bayes is easy to implement and computationally efficient, making it suitable for large datasets.
- **High-Dimensional Data Handling**: Works well with high-dimensional data, particularly effective in text classification where data is typically high-dimensional.

Cons:

- **Strong Independence Assumption**: Assumes that features are independent, which is rarely true in real-world data, leading to suboptimal performance.
- **Poor Performance with Correlated Features**: Can perform poorly if the features (words) are highly correlated.
- **Less Robust to Imbalanced Data**: May struggle with imbalanced datasets, though this is mitigated by the balanced dataset.

Reasons for Using as Baseline:

- **Simplicity and Interpretability**: Naive Bayes is a straightforward and interpretable model, making it an excellent starting point for text classification tasks.
- **Baseline Performance**: Establishing a baseline with Naive Bayes allows me to demonstrate the effectiveness of more sophisticated models like TFIDF with ensemble methods and embedding-based models (e.g., bidirectional LSTM with Word2Vec). This comparison can highlight the value added by more advanced techniques.
- **Comparative Analysis**: Naive Bayes often performs reasonably well on text data, particularly when features are relatively independent.
- **From White Box to Black Box**: Including a simple model like Naive Bayes helps illustrate the progression from basic to advanced models. It shows how machine learning techniques evolve in complexity and performance.

Models Used for Statistical Approach:

Logistic Regression

Logistic Regression serves as a foundational method for understanding and benchmarking against more complex models.

Formula:

$$Z = \theta^T . X$$

$$P(y = 1 \; given \; X) = \frac{1}{1 + e^{-Z}}$$

- θ is the vector of coefficients (including the intercept term θ0).
- X is the vector of input features.
- θT denotes the transpose of θ.
- e is Euler's number, the base of the natural logarithm.

Pros:

- **Interpretability**: Coefficients provide insights into the importance of each feature.
- **Robust Performance**: Performs well in binary classification tasks.
- **Probability Outputs**: Can output probabilities, useful for certain applications based on confidence levels.

Cons:

- **Assumes Linearity**: Assumes a linear relationship between features and the log-odds of the outcome.
- **Requires Scaling**: Frequently features need to be scaled, adding a preprocessing step.

Despite its robustness, the accuracy of Logistic Regression might not be sufficient for our task of sentiment analysis. The algorithm demonstrates strong performance with an AUC of 0.9, indicating excellent discriminative ability. The precision, recall, and F1-scores for both classes are high, showing that the model is both precise and capable of capturing the majority of actual instances of each class. The overall accuracy of 89% further confirms the model's robustness. This means that the logistic regression model is well-suited for the task, effectively balancing precision and recall, and accurately distinguishing between the two classes in the IMDB review dataset.

Confusion Matrix:

	Predicted Positive	Predicted Negative
Actual Positive	6779	734

Actual Negative	927	6434

Multinomial Naive Bayes

Multinomial Naive Bayes is tailored for text data, making it particularly effective for text classification tasks with word frequency features. It's computationally efficient and can handle large vocabularies.

Formula:

$$P(Ck \mid X) \propto P(Ck) \prod_{i=1}^{n} P(Xi \mid Ck)^{Xi}$$

Pros:

- **Tailored for Text Data**: Particularly effective for text classification tasks with word frequency features.
- **Computational Efficiency**: Can handle large vocabularies efficiently.

Cons:

- **Assumes Feature Independence**: Like all Naive Bayes variants, assumes feature independence, which may not perform well if the classes have significant feature overlap.

The Multinomial Naive Bayes algorithm shows good performance with an AUC of 0.86, indicating a strong discriminative ability. The precision, recall, and F1-scores for both classes are high and well-balanced, reflecting the model's ability to accurately classify both positive and negative reviews. The overall accuracy of 86% further confirms the model's reliability. Compared to the logistic regression model, MNB has slightly lower performance metrics, but it still provides a robust and efficient solution for text classification tasks like sentiment analysis of IMDB reviews. This suggests that while logistic regression might be slightly better for this dataset, MNB remains a viable and effective alternative.

Confusion Matrix:

	Predicted Positive	Predicted Negative

Actual Positive	6387	1126
Actual Negative	975	6386

Support Vector Machine (SVM)

Support Vector Machine (SVM) often achieves high accuracy, especially with text classification. It handles high-dimensional data well and can be effective when the number of features is greater than the number of samples. With the right kernel and regularization, SVMs can be robust to overfitting.

Pros:

- **High Accuracy**: Often achieves high accuracy, especially with text classification.
- **Handles High-Dimensional Data**: Effective when the number of features is greater than the number of samples.
- **Robust to Overfitting**: With the right kernel and regularization.

Cons:

- **Slow Training**: Training can be slow, especially with large datasets.
- **Parameter Tuning**: Requires careful tuning of parameters like the regularization parameter and kernel parameters.
- **No Probability Estimates**: Standard SVMs do not provide probability estimates, though this can be mitigated with methods like Platt scaling.

The Support Vector Classifier algorithm demonstrates outstanding performance with an AUC of 0.9, indicating excellent discriminative ability. The precision, recall, and F1-scores for both classes are high and well-balanced, showing the model's effectiveness in accurately classifying both positive and negative reviews. The overall accuracy of 90% further confirms the model's robustness. Compared to the logistic regression and Multinomial Naive Bayes models, the SVC model performs slightly better in terms of AUC and overall metrics. This suggests that the SVC is a very effective model for this text classification task, providing a strong balance of precision, recall, and overall accuracy. It is a highly reliable choice for analyzing IMDB reviews in an NLP context.

Confusion Matrix:

	Predicted Positive	Predicted Negative
Actual Positive	6747	766
Actual Negative	916	6445

Ensemble Method:

The Voting Classifier is an ensemble method that combines the predictions of multiple machine learning models to improve accuracy and robustness. In this case, it combines Logistic Regression (LR) and Support Vector Classifier (SVC) using hard voting, which means the final prediction is based on the majority vote from the individual classifiers. This is the final model used for Statistical approach. While it's true that the individual models in an ensemble should ideally be independent to maximize the benefits of ensembling, the Voting Classifier can still offer improvements even when the models are not entirely independent, like this project, since they are trained on the same training set.

Pros:

- **Improved Accuracy**: By combining multiple models, the Voting Classifier often achieves higher accuracy than any individual model.
- **Robustness**: It reduces the risk of overfitting, as it balances the strengths and weaknesses of the component models.

Cons:

- **Complexity**: More complex to implement and understand compared to a single model.
- **Slower Prediction**: The prediction process can be slower since it involves running multiple models.
- **Dependency on Base Models**: The performance heavily depends on the chosen base models and their individual performances.

Confusion Matrix:

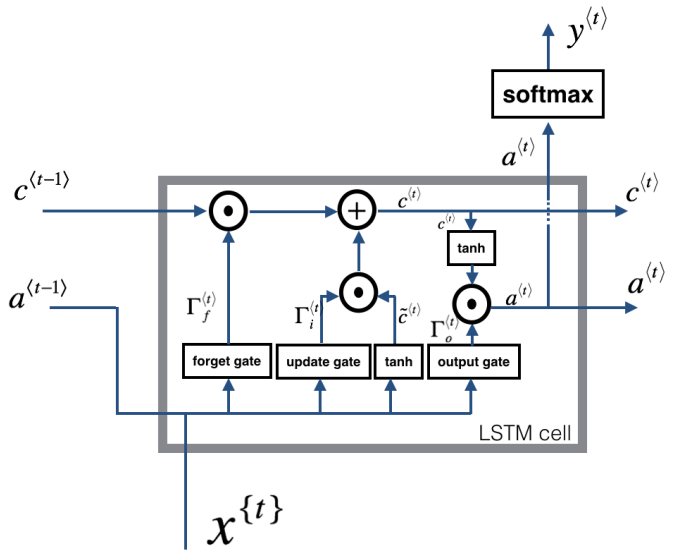
	Predicted Positive	Predicted Negative
Actual Positive	6777	736
Actual Negative	891	6470

Model Used for Embedding approach:

Long Short-Term Memory (LSTM)

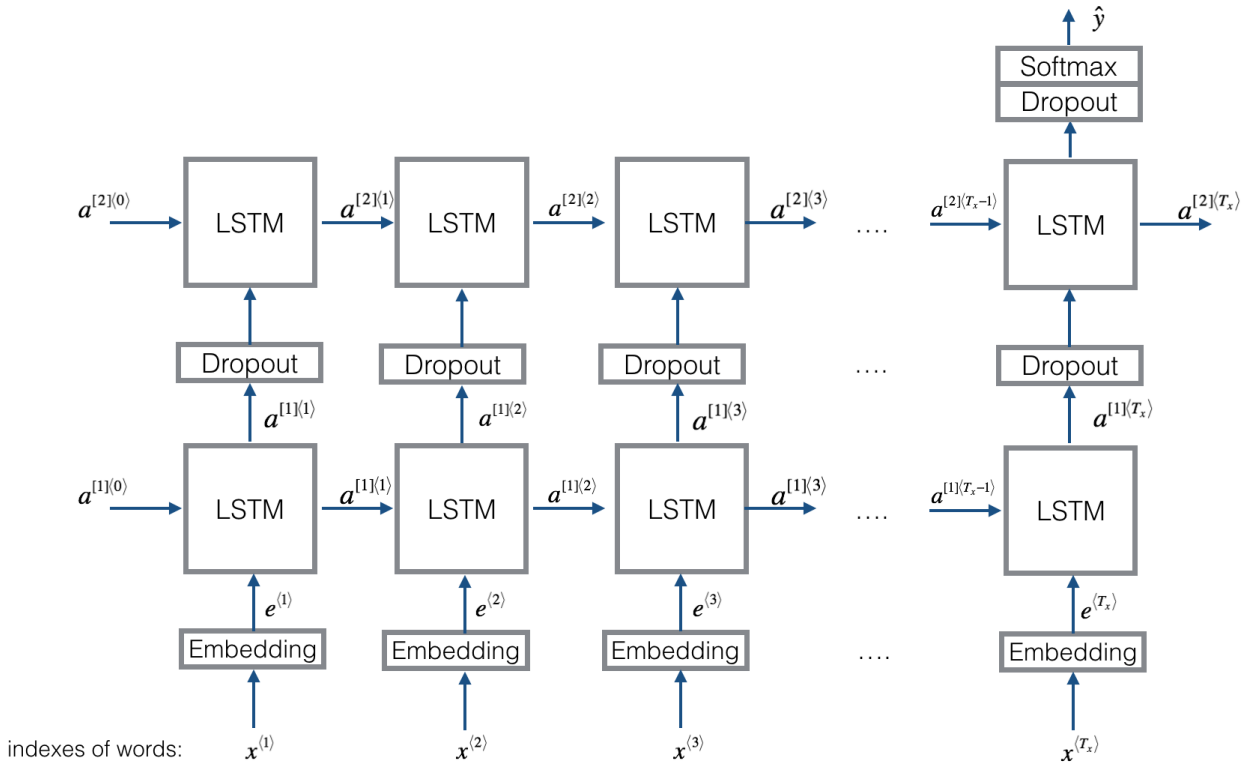
LSTM networks are a type of RNN capable of learning long-term dependencies. They are designed to combat the vanishing gradient problem, which happens in the normal RNNs having to process long sequences. Therefore, LSTMs are effective for a variety of sequential data tasks.

Visualization of a single LSTM cell:



$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$
$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$
$$\tilde{c}^{\langle t \rangle} = \tanh(W_C[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_C)$$
$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} \circ \tilde{c}^{\langle t \rangle}$$
$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$
$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} \circ \tanh(c^{\langle t \rangle})$$

Visualization of Architecture of the Deep Learning Model:



Given that the number of LSTM layers is fixed, padding was applied to all shorter inputs based on the longest input in the examples, ensuring uniform input length. Embedding was performed using Word2Vec, where the dot product of each one-hot encoded vector for each word in the example was computed with the embedding matrix to capture the semantics of even unknown words. The resulting embedding matrix (et) was then passed to the first LSTM layer, which consists of 128 units. Each cell output was subjected to a 50% dropout regularization technique to prevent overfitting. The output was then passed to another LSTM layer with 128 units, where only the output of the last layer was used. This was followed by a single Dense layer with one unit, utilizing a sigmoid activation function (despite the diagram incorrectly showing a SoftMax function) for binary classification.

Pros:

Long-Term Dependency Learning:

LSTMs are designed to remember information over long periods. They mitigate the vanishing gradient problem, which is common in traditional RNNs, allowing them to capture long-range dependencies in sequential data.

Handling Sequence Data:

LSTMs are particularly well-suited for tasks involving sequential data, such as time series prediction, language modeling, and speech recognition.

Flexible Input and Output Lengths:

They can handle input sequences and output sequences of varying lengths, making them versatile for different types of sequential tasks.

Cell State and Gating Mechanisms:

The cell state and gating mechanisms (input, forget, and output gates) help regulate the flow of information, enabling the network to maintain and update relevant information while discarding irrelevant data.

Cons:

Computationally Intensive:

LSTMs require significant computational resources for training, which can be a bottleneck, especially with large datasets or very long sequences.

Complexity:

The internal architecture of LSTMs is more complex compared to simple RNNs, which can make them harder to implement and tune.

Training Time:

Training LSTMs can be slow, especially when dealing with long sequences, due to the recurrent nature of the network and the need to process each time step sequentially.

Overfitting:

They are prone to overfitting, particularly when dealing with small datasets, necessitating the use of regularization techniques such as dropout.

Benefits of Stacking Two LSTM Layers

Enhanced Feature Extraction:

Stacking multiple LSTM layers allows the network to learn more complex representations of the data. The first LSTM layer might capture basic patterns, while the second layer can capture higher-level abstractions.

Increased Capacity:

A deeper network has more capacity to model complex relationships in the data, which can lead to better performance on tasks requiring nuanced understanding of the sequence.

Improved Generalization:

By capturing more complex patterns, stacked LSTM networks can potentially generalize better to unseen data, especially when trained on large and diverse datasets.

Hierarchical Learning:

Each layer can learn different aspects of the data hierarchy. For example, in language modeling, lower layers might learn syntactic structures, while higher layers capture semantic information.

Challenges and Considerations with Stacking LSTMs

Increased Computational Cost:

Stacking LSTM layers increases the number of parameters, which can significantly increase training time and memory requirements.

Risk of Overfitting:

With more parameters, there is an increased risk of overfitting, especially if the amount of training data is limited.

Gradient Vanishing/Exploding Problems:

While LSTMs are designed to mitigate these issues, they can still occur, particularly in very deep networks. Proper initialization, gradient clipping, and advanced optimization techniques are often required.

Need for Careful Tuning:

Deeper architectures require careful tuning of hyperparameters (e.g., learning rate, dropout rates, number of units) to achieve optimal performance.

The use of LSTM networks, and specifically stacking them, offers powerful advantages for sequential data tasks due to their ability to capture complex, long-range dependencies. However, these benefits come with increased computational demands and potential challenges in training and overfitting. Balancing these factors through careful design and tuning is essential for leveraging the full potential of stacked LSTM networks in NLP and other sequence-based applications.

Confusion Matrix of the Model:

	Predicted Positive	Predicted Negative
Actual Positive	4289	711
Actual Negative	1460	3540

Conclusion

Performance Analysis and Comparative Discussion:

Here is a comparison of three models: a baseline model using Naive Bayes, a statistical model in an ensemble method, and a model using word embeddings. The comparison utilizes metrics described in the Introduction section.

The embedding model's underperformance is surprising, especially since it performed worse than the simple baseline model. Several factors could contribute to this outcome:

- **Learning Rate**: The learning rate might have been too high, as evidenced by higher accuracy in some mini-batches. Implementing a learning rate decay function could help.
- **Dropout Regularization**: The dropout regularization probability may have been too high and should be lowered below 0.5.
- **Number of Epochs**: Training for only 10 epochs might have been insufficient. More epochs could be necessary to fully train the model on the entire dataset.
- **Embedding Matrix Size**: The Word2Vec embedding matrix might have been too large for the available hardware. Using a simpler embedding method like GloVe or self-trained embedding layers could have produced a more accurate model in fewer epochs, albeit with reduced generalization power.
- **Fine-Tuning Embeddings**: Fine-tuning the embedding could have been beneficial. However, backpropagation through the LSTM layers was slow, and updating the embedding matrix weights added to this challenge.

Based on the accuracy results on the training and test sets, which are somewhat equal, we can conclude that our model has high bias but lower variance. This means that there is still plenty for the algorithm to learn from the data.

However, the LSTM model shows a higher AUC compared to base model despite lower pointwise metrics like accuracy, precision, recall, and F1-score, even with a balanced dataset. This suggests that the LSTM excels in distinguishing between classes by confidently ranking instances based on their predicted probabilities. Its ability to capture complex sequential patterns and dependencies likely contributes to this performance advantage over Naive Bayes, which relies on simpler probabilistic assumptions. Thus, while pointwise metrics may be lower, the LSTM's strength lies in its capability to utilize sequential data effectively for classification tasks, enhancing overall predictive performance as measured by AUC.

Training the ensemble method was significantly more time-consuming compared to the other two models. It took approximately 2 days to complete just 10 epochs of training. Even though I utilized a pre-trained Word2Vec model, fine-tuning the embedding layer and incorporating it into the model with backpropagation would have demanded significant additional time and effort. This stands as one of the primary drawbacks of using embedding layers with conjunction to DL. However, based on current trends, it is my belief that with the right hyperparameters and architectures, embeddings integrated with LSTMs or transformers could surpass the performance of conventional machine learning models.

The ensemble method employing statistical approaches appears the most promising among the models tested. It has surpassed all individual models it comprises, underscoring the effectiveness of the "wisdom of the crowd" principle in various aspects. However, it's important to acknowledge that training and prediction times for these ensemble methods are significantly longer compared to each of the three individual models within it. This could potentially hinder performance in real-time applications.

Moving forward, future research could explore more advanced deep learning architectures such as transformers, which have demonstrated substantial advancements in natural language understanding tasks. Additionally, fine-tuning embedding layers or exploring alternative embeddings like GloVe could optimize model performance and computational efficiency.

In conclusion, while my initial exploration into DL and ML methodologies yielded valuable insights and performance metrics, further experience and refinement are necessary to fully leverage the capabilities of advanced models like LSTMs for sentiment analysis tasks.

	Base	Statistical				Embedding
Model	**Naive Bayes**	Logistic Regression	Multinomial Naive Bayes	SVM	**Ensemble**	**LSTM**
Accuracy	85.2%	89%	86%	89%	90%	82%
Precision	85%	89%	86%	89%	89%	80%
Recall	85%	87%	86%	89%	89%	80%
F1-Score	85%	89%	86%	89%	89%	78%
AUC	85%	90%	86%	89%	89%	87%
Time to train	fast	fast	fast	moderate	moderate	Extremely slow
Time to predict	fast	fast	fast	fast	fast	slow

Project Summary and Reflections

Reflecting on the learning experience from this project, it is evident that each model type—Naive Bayes, statistical ensemble methods, and embedding-based LSTM—offered unique insights into the practicality and application of machine learning and deep learning techniques in sentiment analysis.

Preprocessing plays a crucial role in ensuring the quality and effectiveness of models in sentiment analysis. Initially, the IMDB dataset underwent comprehensive text cleaning to remove HTML tags, punctuation, and special characters, which could otherwise introduce noise and hinder model performance. Following this, text normalization techniques such as lowercasing were applied to standardize the text, ensuring consistency in tokenization and subsequent analysis. Tokenization further segmented the cleaned text into individual words or tokens, facilitating the extraction of meaningful features for analysis. Stop words, commonly occurring words like "the" and "and", were subsequently removed to reduce noise and improve computational efficiency during model training.

Furthermore, text stemming was employed to reduce words to their root forms, thereby consolidating variations of words with similar meanings. This step aimed to enhance the efficiency of feature extraction and reduce the dimensionality of the dataset. Additionally, for embedding-based models, including the LSTM model using Word2Vec embeddings, the preprocessed text was mapped to pre-trained word embeddings. This process embedded each word into a high-dimensional vector space, capturing semantic relationships between words based on their context in the dataset. These embeddings served as input features for deep learning models, facilitating the extraction of contextual and positional information critical for understanding sentiment in textual data. Overall, the preprocessing steps ensured that the models received clean, standardized input data, thereby enhancing their ability to learn meaningful patterns and achieve optimal performance in sentiment analysis tasks. A thorough comparison of each preprocessing method was done in here

Starting with Naive Bayes, it served as a foundational model that introduced fundamental concepts such as feature independence assumptions and probabilistic classification. Its simplicity and computational efficiency make it practical for tasks where real-time prediction and interpretability are critical. However, its reliance on strong feature independence assumptions limits its ability to capture complex relationships in data, particularly in natural language processing tasks where contextual nuances play a significant role.

The statistical ensemble methods, leveraging TF-IDF with SVM and logistic regression and multinomial naive bayes, provided a robust alternative. By combining multiple base learners, ensemble methods demonstrated improved performance in accuracy, precision, recall, and F1-score metrics compared to individual models. This approach highlighted the effectiveness of aggregating diverse models to mitigate biases and variance, enhancing overall predictive capability. The interpretability of logistic regression coefficients further facilitated understanding of feature importance, making it suitable for applications where model transparency is essential.

In contrast, the embedding-based LSTM model aimed to harness the contextual information embedded within Word2Vec representations and sequential dependencies captured by LSTM layers. Despite its potential to generalize well with sequential data like text, its performance in this project was hindered by challenges in hyperparameter tuning, such as learning rate and dropout regularization. The computational intensity and longer training times associated with deep learning models underscore the need for scalable infrastructure and optimization strategies.

In practical terms, each model type contributes uniquely to the problem of sentiment analysis. Naive Bayes remains a viable baseline for quick prototyping and initial model evaluation. Ensemble methods offer robust performance improvements through model aggregation, suitable for scenarios requiring high accuracy and robustness. Deep learning models like LSTMs excel in capturing intricate patterns in sequential data, paving the way for applications demanding nuanced understanding of context and dependencies.

The transferability of these solutions to other domain-specific areas depends on the nature of the data and the specific requirements of the task. For instance, sentiment analysis in customer reviews may benefit from ensemble methods for their interpretability and accuracy, while deep learning models could be leveraged in analyzing streaming social media data for real-time sentiment tracking.

Looking ahead, improvements and future research directions include:

1- Model Optimization: Further exploration of hyperparameter tuning strategies, such as learning rate schedules and advanced regularization techniques, to enhance deep learning model performance.
2- Data Augmentation: Techniques to augment training data and improve model generalization, especially for deep learning models with limited datasets like the IMDB dataset.
3- Interpretable Deep Learning: Development of methods to enhance interpretability of deep learning models, enabling better understanding of model predictions and feature importance.
4- Transfer Learning: Application of transfer learning techniques to pre-trained embeddings and models, leveraging knowledge from large datasets to improve performance on smaller, domain-specific datasets.

In conclusion, this project has provided valuable hands-on experience in selecting, implementing, and evaluating different models for sentiment analysis. It underscores the importance of understanding model strengths and limitations, optimizing model performance, and critically assessing applicability to real-world scenarios across various domains.

I condensed the report's detailed comparison and included a link for reviewers to explore the method's explanation if desired.

References and Bibliography

[1] Q. T. Ain *et al.*, “Sentiment Analysis Using Deep Learning Techniques: A Review,” 2017. [Online]. Available: www.ijacsa.thesai.org

[2] F. Luo, C. Li, and Z. Cao, “Affective-feature-based Sentiment Analysis using SVM Classifier.”

[3] J. S. Krauss, D. Simon, and K. Fischbach, “Predicting Movie Success and Academy Awards through Sentiment and Social Network Analysis,” 2008. [Online]. Available: https://www.researchgate.net/publication/200038418

[4] R. Ahmad, A. Pervaiz, H. Mannan, and F. Zaffar, “Aspect Based Sentiment Analysis for Large Documents Aspect Based Sentiment Analysis for Large Documents with Applications to US Presidential Elections 2016.”

[5] C. N. Dang, M. N. Moreno-García, and F. De la Prieta, “An approach to integrating sentiment analysis into recommender systems,” *Sensors*, vol. 21, no. 16, Aug. 2021, doi: 10.3390/s21165666.

[6] D. D. Gaikar, B. Marakarkandy, and C. Dasgupta, “Using twitter data to predict the performance of bollywood movies,” *Industrial Management and Data Systems*, vol. 115, no. 9, pp. 1604–1621, Oct. 2015, doi: 10.1108/IMDS-04-2015-0145.

[7] E. Cambria, S. Poria, A. Gelbukh, I. P. Nacional, and M. Thelwall, “AFFECTIVE COMPUTING AND SENTIMENT ANALYSIS Sentiment Analysis Is a Big Suitcase,” 2017. [Online]. Available: www.computer.org/intelligent

[8] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning Word Vectors for Sentiment Analysis.”

[9] X. Glorot, A. Bordes, and Y. Bengio, “Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach,” 2011. [Online]. Available: http://www.cs.jhu.edu/

[10] D. Meyer, “How exactly does word2vec work?,” 2016.

[11] J.-H. Wang, T.-W. Liu, X. Luo, and L. Wang, “An LSTM Approach to Short Text Sentiment Classification with Word Embeddings.”

[12] S. M. Rezaeinia, R. Rahmani, A. Ghodsi, and H. Veisi, “Sentiment analysis based on improved pre-trained word embeddings,” *Expert Syst Appl*, vol. 117, pp. 139–147, Mar. 2019, doi: 10.1016/j.eswa.2018.08.044.

[13] M. Al-Smadi, O. Qawasmeh, M. Al-Ayyoub, Y. Jararweh, and B. Gupta, “Deep Recurrent neural network vs. support vector machine for aspect-based sentiment analysis of Arabic hotels’ reviews,” *J Comput Sci*, vol. 27, pp. 386–393, Jul. 2018, doi: 10.1016/j.jocs.2017.11.006.