Project Report: IMDB Movie Review Sentiment Analysis

Omer Seyfeddin Koc [1]

[1]Northeastern University
360 Huntington Ave, Boston, MA 02115 USA

Abstract

This project aims to classify movie reviews from the IMDb dataset as positive or negative using sentiment analysis. The approach includes data preprocessing steps, such as cleaning and vectorizing the data, and trying out 11 different machine learning models. The best performing model was a CNN-LSTM hybrid model with an accuracy of 0.90, demonstrating the effectiveness of deep learning techniques in natural language processing tasks.

Introduction

In the contemporary digital age, peoples increasingly rely on internet reviews to make decisions, movie reviews are one of them. Given the vast number of movies available, viewers may find it challenging to determine which movies to watch. Sentiment analysis can assist in resolving this problem by categorizing reviews as positive or negative, allowing viewers to quickly and easily evaluate a movie's quality.

This project aims to conduct sentiment analysis on movie reviews from the IMDB database using various natural language processing techniques. The dataset used in this project consists of 50,000 reviews with an equal number of positive and negative reviews, prepared by Stanford University's AI Lab.

To prepare the data for analysis, various data cleaning techniques will be implemented, such as removing HTML, URLs, stop words, punctuation, and stemming. These steps are crucial to transforming raw data into a form suitable for analysis. Python's Natural Language Toolkit (NLTK) library will be utilized to carry out these operations.

Following data preprocessing, a vectorizer will be applied to convert the text data into a feature matrix. The Term Frequency-Inverse Document Frequency (TF-IDF) model was selected over the Bag of Words (BoW) model as it accounts for how often a word appears throughout the document and assigns higher weight to more significant words. This feature is particularly useful in our dataset, where commonly used words such as 'movie' or 'actor' are less important.

Several models will be utilized in this project, ranging from simpler models such as Logistic Regression to more complex models such as CNN-LSTM. Also, unlike the other examples in Kaggle, the text-davinci-002 model, which is integrated with OpenAI, was tried on our project.

The primary objective of this project is to achieve high accuracy in sentiment analysis, enabling viewers to quickly and accurately assess a movie's quality before watching it.

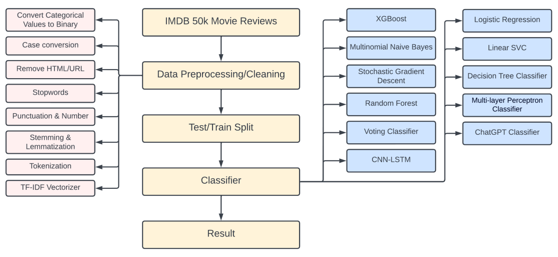


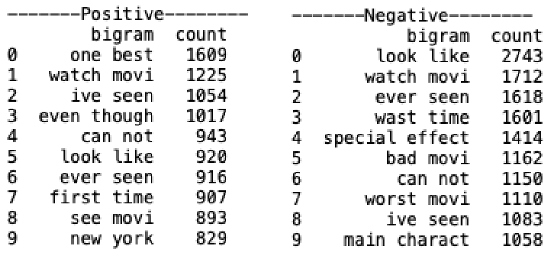
Figure 1: Problem Formulation

Background

Dataset

Sentiment classification is an essential task in natural language processing and has numerous applications in different fields. One of the most well-known and widely used datasets in this area is the iMDB Large Movie Review Dataset, which provides a collection of movie reviews along with their binary sentiment polarity labels. The dataset has been divided into two sets of 25,000 positive and 25,000 negative reviews, with an overall balanced label distribution. Moreover, to prevent any correlation in the ratings, the dataset has been restricted to contain no more than 30 reviews for each movie.

As part of this project, an analysis was conducted on a dataset of movie reviews to determine the sentiment of each review. The dataset consisted of both positive and negative reviews. To further analyze the reviews, a unigram analysis was performed using a count vectorizer object to transform the reviews into count vectors, which were then used to determine the top 10 most common words in positive and negative reviews. Additionally, a bigram analysis was conducted by defining a count vectorizer with a bigram range and analyzing the top 10 most common bigrams in positive and negative reviews. The length distribution of the reviews was also calculated and visualized as a histogram.



```
-------Positive--------           -------Negative--------
         bigram  count                     bigram  count
0      one best   1609            0      look like   2743
1    watch movi   1225            1     watch movi   1712
2      ive seen   1054            2      ever seen   1618
3  even though   1017             3      wast time   1601
4       can not    943            4 special effect   1414
5     look like    920            5      bad movi    1162
6     ever seen    916            6       can not    1150
7    first time    907            7    worst movi    1110
8      see movi    893            8       ive seen   1083
9      new york    829            9  main charact   1058
```

Figure 2: Bigram Analysis

In addition, word clouds were created separately for positive and negative reviews to provide a visual representation of the most common words used in each sentiment category.

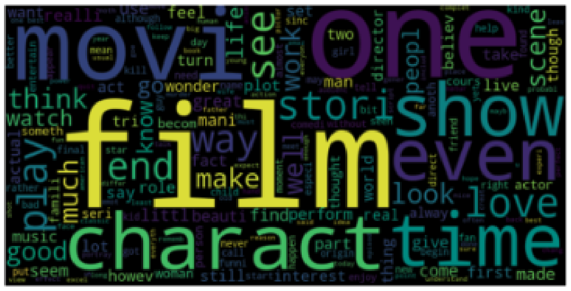


Figure 3: Positive Reviews Word Cloud

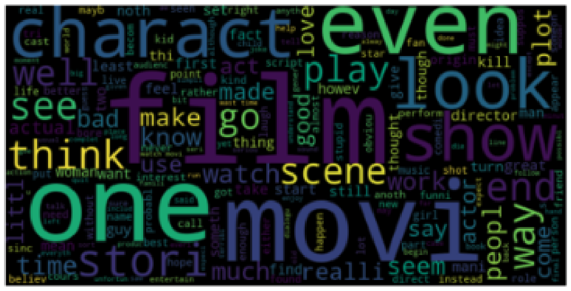


Figure 4: Negative Reviews Word Cloud

Overall, this analysis provides insights into the language used in positive and negative movie reviews, which is crucial for binary classification analysis of the dataset, and can be used to inform future analysis and decision-making.

Metric

For this binary classification problem, I will evaluate the performance of the models based on their accuracy. Accuracy measures the percentage of correctly classified instances out of the total number of instances. In other words, it shows how often the model makes the correct prediction.

Accuracy is a commonly used metric for binary classification problems and is easy to interpret. However, it may not be the best metric to use if the classes are imbalanced, meaning one class has significantly more instances than the other. In such cases, other metrics such as precision, recall, or F1-score may be more appropriate to evaluate the performance of the models.

Nevertheless, since our classes are balanced with an equal number of positive and negative reviews, accuracy is an appropriate metric to use in this project.

$$Accuracy = \frac{Correct\ prediction}{Dataset\ size}$$

The classification performance of a total of 11 different models was calculated with the Accuracy metric.

Data Preprocessing/Cleaning

Before using the data in the models, it is crucial to preprocess and clean the data. In this project, a total of 9 different techniques were used to convert the raw data into a final usable form. Firstly, the positive and negative labels were converted to 1 and 0, respectively, which is a common step in binary classification tasks such as sentiment analysis.

Next, all the reviews were converted to lowercase to standardize the text and simplify the data. HTML tags and URLs were then removed from the data, which is an essential step to eliminate unwanted text and reduce noise in the data.

Stopwords, which are words that do not carry significant meaning, were removed from the data using the Natural Language Toolkit (NLTK) to further reduce noise. Additionally, punctuation and numbers were also removed to clean the text and reduce the dimensionality of the data.

Two techniques were used to normalize words in the data: stemming and lemmatization. Stemming involves reducing words to their base form or stem, while lemmatization involves reducing words to their base form or lemma based on their part of speech. These techniques help in reducing the dimensionality of the data and improving the accuracy of the analysis.

Finally, text tokenization was performed, which involves breaking down the text into smaller units or tokens such as words or phrases. Tokenization is an essential step to prepare the data for further analysis by converting it into a format that can be easily processed by the algorithms.

Overall, these preprocessing and cleaning techniques are crucial for effective sentiment analysis and play a significant role in improving the accuracy of the models used in the project.

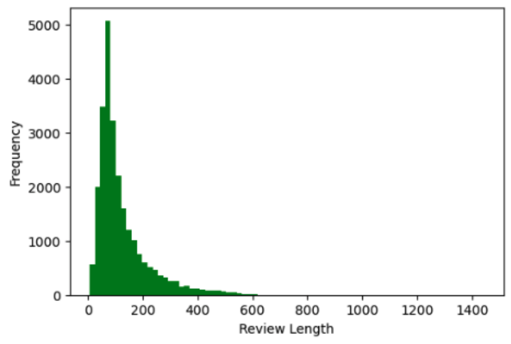


Figure 5: Review Length Distribution - Positive Reviews

Models

Following the data preprocessing and cleaning, the prepared data is ready to be utilized for the training of the models. The purpose of this training is to determine the best model based on the performance metric (Accuracy). To accomplish this, several models will be trained using the processed data. The ultimate goal is to select the most suitable model for our specific application. The selection process will be guided by the chosen performance metric, which will be used to evaluate the models. Through the application of rigorous testing and analysis, I aim to identify the optimal model that can accurately classify sentiment in the text data (classification).

In classification tasks, traditional machine learning algorithms such as Multinomial Naive Bayes, Logistic Regression, Linear SVC, Stochastic Gradient Descent (SGD), Decision Tree Classifier, Random Forest, and MLPClassifier are commonly used. Multinomial Naive Bayes is a probabilistic algorithm that calculates the probability of a given data point belonging to each class based on the frequency of its features. Logistic Regression is another probabilistic algorithm that models the relationship between the features and the class probabilities. Linear SVC is a linear classifier that tries to find the hyperplane that best separates the classes. Stochastic Gradient Descent (SGD) is an optimization algorithm that can be used to train various linear models, including linear regression, logistic regression, and linear SVM.

Decision Tree Classifier is a non-parametric algorithm that builds a tree-like model by recursively splitting the data based on the features. Random Forest is an ensemble algorithm that combines multiple decision trees to reduce overfitting. MLPClassifier stands for Multi-Layer Perceptron and is a neural network model that is often used for classification tasks.

The Voting Classifier is an ensemble model that combines the predictions of multiple models to improve the overall accuracy. OpenAI Classifier (text-davinci-002) and XGBoost are a natural language processing models that can be used for text classification tasks. Finally, the CNN-LSTM Hybrid Model is a deep learning model that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to capture both spatial and temporal features in the data.

In summary, the methods used in this project include both traditional machine learning algorithms and deep learning models, each with their own strengths and weaknesses. The choice of method depends on the specific characteristics of the data and the task at hand.

Related Work

Prior to commencing the project, I conducted extensive research on sentiment analysis projects available on online platforms, such as Kaggle and Github. I also reviewed articles on medium and other related sources to gain sufficient knowledge on the topic. Through this research, I identified several techniques and practices that could be applied to the project, which helped to define the project's overall scope and approach. Notably, the project was not directly influenced or based on any specific existing project.

For my IMDB sentiment analysis project, I have implemented a range of machine learning algorithms. I avoided using more advanced machine learning models, such as Bidirectionnal Encoder Representations for Transformers (BERT) and Recurrent Neural Networks (RNNs) due to my lack of confidence in implementing them without proper guidance.

Moreover, the computational resources and training time required to use these models were significantly higher than the models I implemented. It is worth noting that the CNN-LSTM Hybrid Model is also a complex and difficult model. However, I included it in my project as I am proficient in using it, having applied it in my undergraduate project.

Project Description

Within the scope of the project, I have performed several processes to prepare the data for model training and evaluation. These processes include data pre-processing and cleaning, model training and evaluation.

Data Preprocessing/Cleaning

Label Encoding for Sentiment Classification

The review labels in the dataset are converted from strings to binary values, where positive reviews are represented by the value 1 and negative reviews by the value 0. This conversion is necessary in order to enable the models to classify the sentiment of the reviews accurately. This step is a crucial part of the data preprocessing phase as it ensures that the data is in a format that can be effectively used in the subsequent training of machine learning and deep learning models.

Lowercasing all reviews

```
# Convert all reviews to lowercase
data['review'] = data['review'].str.lower()
```

This process helps to standardize the text data and make it more consistent. It is important to convert all reviews to lowercase to avoid any discrepancies in the analysis that may occur due to variations in capitalization. By applying this process, I ensure that all reviews are in the same format, making it easier to analyze and process the data. This is an important step in the data preprocessing phase of the project as it helps to ensure the quality and consistency of the data.

Removing HTML Tags

```
# Function to remove HTML tags from a text string
def remove_html(text):
    clean_text = re.sub('<.*?>', '', text)
    return clean_text

# Apply the function to the 'review' column in the DF
data['review']=data['review'].apply(remove_html)
```

This function uses the regular expression library re to substitute all instances of HTML tags in a text string with an empty string, effectively removing them. Then, the defined function is applied to the 'review' column in the DataFrame using the $apply()$ method. This process is important as

HTML tags can contain formatting and other extraneous information that is not relevant to the sentiment classification task and may introduce noise into the model.

Removing URLs

```
# Function to remove URLs from a text string
def remove_urls(text):
    clean_text = re.sub(r'http\S+', '', text)
    clean_text = re.sub(r'www\S+', '', clean_text)
    return clean_text

# Apply the function to the 'review' column in the DF
data['review']=data['review'].apply(remove_urls)
```

The function uses regular expressions to match patterns of URLs that start with 'http' or 'www' and then replaces them with an empty string. This is done to clean up the text data and remove any irrelevant information that may affect the analysis. After defining the function, it is applied to the 'review' column in the DataFrame using the 'apply' method. This step is important in cleaning and preprocessing the text data, which is necessary for accurate analysis and modeling. Therefore, it is essential to remove URLs to prevent any bias in the data.

Removing Stopwords Using NLTK

```
# Download the NLTK stopwords
nltk.download('stopwords')
# Get the English stopwords
stop_words = set(stopwords.words('english'))

# Function to remove stopwords from a text string
def remove_sw(text):
    words = text.split()
    words = [word for word in words if word.lower()
    not in stop_words]
    return ' '.join(words)

# Apply the function to the 'review' column in the DF
data['review'] = data['review'].apply(remove_sw)
```

This code downloads the stopwords from the Natural Language Toolkit ($NLTK$) library and saves them in a set. Stopwords are common words like "the", "is", "and", etc. that do not carry much meaning in a sentence. A function is defined to remove these stopwords from a given text string by first splitting the string into individual words, then keeping only those words that are not in the stop_words set. Finally, the cleaned words are joined back together to form a text string again. The function is then applied to the 'review' column of the IMDb data to remove stopwords from all the reviews. This process reduces the number of unnecessary words in the data and can help improve the accuracy of text analysis models.

Removing Punctuation and Numbers

```
# Function to remove punctuation and numbers
def remove_pn(text):
    return text.translate(str.maketrans('', '',
    string.punctuation + '0123456789'))

# Apply the function to 'review' column
data['review'] = data['review'].apply(remove_pn)
```

This function takes a text string as input and returns the same text string with all punctuation marks and numbers removed. It uses two string methods to achieve this: $str.translate$ with $str.maketrans$ to remove numbers, and $string.punctuation$ to remove punctuation. Then, the function is applied to the 'review' column of the DataFrame to remove punctuation and numbers from all reviews.

Stemming and Lemmatization

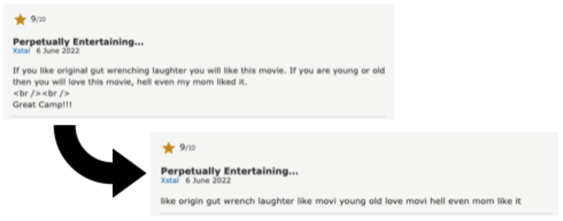
```
# Initialize PorterStemmer and WordNetLemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Function to perform stemming and lemmatizing
def s_l(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # Perform stemming on the tokens
    stemmed_tokens = [stemmer.stem(token)
    for token in tokens]
    # Perform lemmatizing on the stemmed tokens
    lemmatized_tokens = [lemmatizer.lemmatize(token)
    for token in stemmed_tokens]
    # Join the lemmatized tokens back into a text
    lemmatized_text = ' '.join(lemmatized_tokens)
    return lemmatized_text

# Apply to the 'review' column
data['review'] = data['review'].apply(s_l)
```

In this step, I am performing stemming and lemmatizing on the text data in the 'review' column of a DataFrame. Stemming and lemmatizing are common techniques used in Natural Language Processing (NLP) to reduce words to their root form. This helps to standardize the text and reduces the number of unique words in the dataset. I start by initializing a PorterStemmer and a WordNetLemmatizer from the NLTK library. These are the two most commonly used tools for stemming and lemmatizing in NLP.

Next, I define a function named 's_l' which takes a text as input, tokenizes it, performs stemming on the tokens, and then lemmatizes the stemmed tokens. Finally, the function joins the lemmatized tokens back into a text string and returns it. I then apply this 's_l' function to the 'review' column of the DataFrame using the 'apply' method. This updates the 'review' column with the stemmed and lemmatized version of the text data.


9/10

Perpetually Entertaining...
Xstal 6 June 2022

If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it.
<br /><br />
Great Camp!!!

9/10

Perpetually Entertaining...
Xstal 6 June 2022

like origin gut wrench laughter like movi young old love movi hell even mom like it

Figure 6: Before:After Review

Text Tokenization

```
# Tokenize the 'review' column in your IMDb data
data['tokens'] = data['review'].apply(word_tokenize)
```

Text tokenization is performed on the 'review' column of a DataFrame that contains IMDb movie reviews. Text tokenization is a crucial step in Natural Language Processing (NLP) preprocessing as it breaks down the text into smaller units that can be analyzed or processed further. The 'word_tokenize' function from the NLTK library is applied to the 'review' column of the DataFrame using the 'apply' method. This function takes a text string as input and returns a list of tokens, where each token represents a word in the text. A new column named 'tokens' is created in the DataFrame, which contains the list of tokens obtained from the 'review' column. This enables the tracking of the tokenized version of the text data alongside the original text.

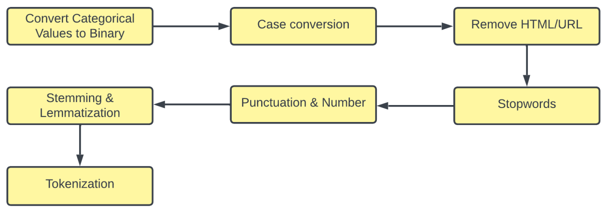


Figure 7: Data Preprocessing Diagram

Models and Applications

You can access the codes of all used models from the Github repository. Link to my Github repository. In this section, only complex or tuned models will be explained.

TF-IDF Vectorizer

In this part of the project, the text data was prepared for a sentiment analysis task by converting the tokenized text into numerical vectors using the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique. TF-IDF is a statistical measure that reflects the importance of a word in a document or corpus. It is calculated by multiplying the frequency of a term in a document by the inverse document frequency, which is the logarithmically scaled fraction of the total number of documents that contain the term.

Multinomial Naive Bayes

In this part, hyperparameter tuning is performed on the Multinomial Naive Bayes classifier to optimize its performance for sentiment analysis on text data. A parameter grid is defined, and an instance of the classifier is created along with a GridSearchCV object to perform a grid search over the parameter grid. The number of cross-validation folds is set to 5, and accuracy is used as the scoring metric. The GridSearchCV object is then fit on the training data to find the best hyperparameters based on the highest cross-validation accuracy. A new instance of the Multinomial Naive Bayes classifier is created with the best hyperparameters and fit on the training data. Finally, predictions are made on the test data and the accuracy score is calculated using the 'accuracy_score()' method from the scikit-learn library.

Logistic Regression

I define a parameter grid is defined for hyperparameter tuning by specifying a range of values for the 'penalty', 'C', and 'max_iter' hyperparameters of the logistic regression classifier. Grid Search Cross Validation is then performed to find the best hyperparameters, which involves training the classifier with different combinations of hyperparameters and evaluating their performance using cross-validation. The best hyperparameters are then obtained and the classifier is updated with the best hyperparameters. Best Hyperparameters: 'C': 10, 'max_iter': 200, 'penalty': 'l2'

Random Forest

The Random Forest Classifier is a machine learning algorithm that constructs a set of decision trees and aggregates their predictions to generate the final output. The code first imports the 'RandomForestClassifier' class from the scikit-learn library. Then, it creates an instance of the classifier object with 100 trees and sets the 'random_state' to 42 to ensure the reproducibility of the results.

Next, the classifier is trained on the training data using the 'fit' method, which takes the training data as input and learns to make predictions based on the features and labels of the data. Finally, the performance of the classifier is evaluated on the test data using the 'score' method, which calculates the mean accuracy of the classifier on the test data. Accuracy measures how well the classifier can predict the correct label for the test data.

MLPClassifier

Multi-Layer Perceptron (MLP) Classifier is created and trained for sentiment analysis on iMDB data. The MLP Classifier is a type of artificial neural network that consists of multiple layers of interconnected neurons and is often used for classification tasks. First, an instance of the MLPClassifier class is created with 2 hidden layers, each containing 50 neurons. The 'max_iter' parameter is set to 100, which determines the maximum number of iterations the MLP algorithm will run during training.

Next, the MLPClassifier is trained on the training data using the 'fit' method, which takes the training data as input and learns to make predictions based on the features and labels of the data. Overall, the MLP Classifier is a powerful algorithm that can handle non-linear relationships between the features and labels of the data. It is also able to learn complex decision boundaries, making it a popular choice for classification tasks.

Voting Classifier

In this method, a Voting Classifier is being implemented for sentiment analysis on iMDB data. A Voting Classifier is an ensemble learning algorithm that aggregates the predictions of multiple individual models to make a final prediction. First, three base models are defined: Multinomial Naive Bayes Classifier, Logistic Regression Classifier, and Linear Support Vector Classifier. These models are initialized with different hyperparameters and can have different strengths and weaknesses.

Next, a Voting Classifier object is created with the specified estimators and voting method. The 'estimators' parameter takes a list of tuples, where each tuple contains the name and the corresponding model object. The 'voting' parameter specifies the aggregation method for the individual model predictions. Here, 'hard' voting is used, meaning the predicted class is the mode of the predicted classes of the individual models.

Finally, the Voting Classifier is trained on the training data using the 'fit' method. The classifier combines the predictions of the individual models to make a final prediction for each data point in the test set. Overall, the Voting Classifier is a powerful algorithm that can improve the accuracy of classification tasks by combining the strengths of multiple individual models.

Open AI: text-davinci-002

In this method, the OpenAI API is used to classify the sentiment of a given text. Then, a function is defined to classify the sentiment of a given text using the OpenAI API. The function sends a prompt to the API with the given text and waits for a response, which is then parsed and returned as a label indicating either positive or negative sentiment. Finally, the function is applied to the selected subset of the IMDb dataset and a new column is added to the dataset indicating the predicted sentiment for each review using the OpenAI API.

It is worth noting that the use of the OpenAI API for sentiment analysis is a relatively new technology, and it was expected to yield better results than it did. However, the results obtained were worse than expected.

XGBoost

In this code, the parameters for the XGBoost model are set using a Python dictionary. The 'max_depth' parameter specifies the maximum depth of each tree in the boosted model, 'eta' is the learning rate, 'objective' specifies the loss function to be minimized and 'eval_metric' is the evaluation metric to be used during training. A value of 'binary:logistic' for the 'objective' parameter means that the model will output probabilities for binary classification tasks.

The XGBoost model is then created using these parameters and trained on the training data using the 'fit' method of the XGBClassifier class. The trained model can then be used to predict the classes of new data.

CNN-LSTM

This method implements a deep learning model architecture that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to perform sentiment analysis on a iMDB reviews. The model takes in sequences of integers (encoded words) as inputs, and returns a binary classification label indicating positive or negative sentiment.

First, the sequences are padded to a fixed length of 150 using the pad_sequences() function from Keras. This ensures that all sequences are of the same length, a requirement for the CNN layer. The model architecture starts with an input layer that takes in the padded sequences. An embedding layer is then added to convert the integer-encoded words into dense vectors of fixed size. The Embedding() function takes in the input dimension, which is the size of the vocabulary (in this case, 5000), the output dimension of the embedding layer (150), and the length of the input sequences.

A 1D convolutional layer (Conv1D()) is added with 64 filters and a kernel size of 3. The activation function used is ReLU. This layer is used to extract local features from the input sequences. Next, a max pooling layer (MaxPooling1D()) is added to downsample the output of the convolutional layer by taking the maximum value in each feature map. This layer helps to reduce the dimensionality of the feature maps and makes the model more computationally efficient.

A Bidirectional LSTM layer (Bidirectional()) is then added with 64 units. The LSTM layer processes the sequence data in both forward and backward directions, allowing it to capture the temporal dependencies in the data. A dropout layer (Dropout()) is added to reduce overfitting by randomly setting a fraction of the input units to 0 at each update during training. A dropout rate of 0.5 is used, meaning that 50% of the inputs will be randomly set to 0 during training.

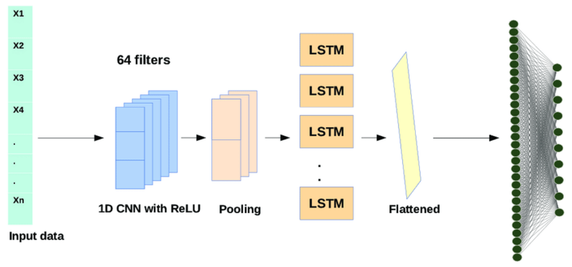


Figure 8: Architecture of the Hybrid 1D CNN-LSTM model

Finally, a dense output layer (Dense()) is added with 1 unit and a sigmoid activation function, which outputs a probability value between 0 and 1, indicating the likelihood of a positive sentiment. The model is then compiled with a binary cross-entropy loss function (loss='binary_crossentropy') and the Adam optimizer (optimizer='adam'). The accuracy metric is used to evaluate the model during training (metrics=['accuracy']).

The model is trained using the fit() function, with a batch size of 64 and for 3 epochs. During training, the validation data is provided to evaluate the performance of the model on unseen data. Once the training is completed, the model is evaluated using the evaluate() function, which returns the loss value and accuracy score on the test data.

Empirical Results

Several classification algorithms were trained and evaluated using the dataset. The first algorithm used was Multinomial Naive Bayes. The best alpha parameter was found to be 0.86, and the accuracy on the test set was 0.8615. The algorithm performed reasonably well, but not as well as some of the other algorithms.

Next, logistic regression was used, and the hyperparameters were tuned using grid search. The best hyperparameters found were a regularization parameter C of 10, maximum iterations of 200, and L2 penalty. The logistic regression model achieved an accuracy of 0.8947 on the test set, which is better than that of Multinomial Naive Bayes.

Linear SVC was the next algorithm used, which achieved an accuracy of 0.8923. This result is comparable to that of logistic regression. Stochastic Gradient Descent (SGD) was also tested, and an accuracy of 0.8898 was achieved. This result is slightly worse than that of linear SVC and logistic regression.

A decision tree classifier was used next, with a maximum depth of 145. The accuracy achieved was 0.7185, which is considerably worse than that of the other algorithms. Random forest classifier was also tested, and an accuracy of 0.8483 was achieved. This result is better than that of SGD, but worse than that of logistic regression and linear SVC.

The MLPClassifier achieved an accuracy of 0.8793, which is comparable to that of linear SVC and better than that of SGD. A voting classifier was also tested, which combined the predictions of logistic regression, linear SVC, and MLPClassifier. The accuracy achieved was 0.8944, which is comparable to that of logistic regression. A ChatGPT classifier was also tested, which used the da-vinci-002 model. The accuracy of sentiment classification achieved was 0.7834. This result is worse than that of most of the other algorithms tested.

XGBoost was used, which achieved an accuracy of 0.8595. This result is better than that of Multinomial Naive Bayes, but worse than that of logistic regression, linear SVC, MLPClassifier, and the CNN-LSTM hybrid model. The CNN-LSTM hybrid model achieved the highest accuracy of all the algorithms tested, with an accuracy of 0.9018. This result is likely due to the ability of the model to capture both local and global dependencies in the text data.

In summary, the best performing algorithms on the iMDB Review dataset were the CNN-LSTM hybrid model, logistic regression, and the voting classifier. The decision tree classifier performed the worst out of all the algorithms tested.

The accuracy values and names of the models are as follows:

Model	Accuracy
Multinomial Naive Bayes	0.8615
Logistic Regression	0.8947
Linear SVC	0.8923
Stochastic Gradient Descent (SGD)	0.8898
Decision Tree Classifier	0.7185
Random Forest	0.8483
MLPClassifier	0.8793
Voting classifier	0.8944
OpenAI Classifier (da-vinci-002)	0.7834
XGBoost	0.8595
CNN-LSTM Hybrid Model	0.9018

Conclusions / Future Directions

In this project, I explored various machine learning algorithms for sentiment analysis on the iMDB Review dataset. I found that the CNN-LSTM hybrid model achieved the highest accuracy of 0.9018, while the decision tree classifier performed the worst with an accuracy of 0.7185. The logistic regression and voting classifier also performed well with accuracies of 0.8947 and 0.8944, respectively.

Overall, I learned that deep learning models can achieve high accuracy in sentiment analysis tasks, but they require more computational resources and longer training times than traditional machine learning models. Additionally, I found that feature engineering can significantly impact the performance of traditional models like logistic regression and decision trees. I would have explored more deep learning models like transformer-based models such as BERT or GPT, which have shown state-of-the-art performance in various NLP tasks.

For future directions, there are several potential directions that could be explored for further development of this project. Firstly, one could investigate the use of more advanced natural language processing techniques, such as contextualized word embeddings like BERT or GPT, to potentially improve the model's performance. Secondly, it may be worthwhile to explore the use of ensemble methods, such as bagging or boosting, to combine multiple models and improve the overall accuracy of the sentiment analysis. Thirdly, the project could be expanded to include more diverse datasets, such as reviews from different domains (e.g. books, restaurants, products) or different languages, to evaluate the generalizability of the model. Finally, one could explore the use of explainability techniques to better understand how the model is making its predictions, such as feature importance analysis or generating example text that strongly influences the model's prediction.

As a recommendation for future DS5220 students, I highly suggest exploring a variety of models for their projects and performing a thorough comparison of results to determine the most suitable model for the task. It is important to avoid getting stuck on a particular model and to engage in hyperparameter tuning when necessary. Additionally, possessing theoretical knowledge of these methods can greatly enhance the success of the project. By employing these strategies, students can maximize the effectiveness of their projects and achieve good results.

References

Bengfort, B., Bilbro, R., Ojeda, T. (2018). Applied text analysis with Python: Enabling language-aware data products with machine learning. Sebastopol, CA: O'Reilly Media.

Géron, A. (2020). Hands-on machine learning with scikit-learn, Keras, and tensorflow concepts, tools, and techniques to build Intelligent Systems. Beijing: O'Reilly.

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

Maas, A. (n.d.). Large Movie Review Dataset. Retrieved April 25, 2023, from https://ai.stanford.edu/~amaas/data/sentiment/

N. (2019). IMDB dataset of 50K movie reviews. Retrieved April 1, 2023, from www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Pang, B., Lee, L. (2008). Opinion Mining and Sentiment Analysis. Foundations and Trends® in Information Retrieval, 2(1-2), 1-135.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830.

Raschka, S., Mirajalili, V. (2019). Python machine learning: Machine learning and deep learning with python, scikit-learn, and tensorflow. Birmingham, UK: Packt Publishing.

Shang, L., Sui, L., Wang, S., Zhang, D. (2020, May 01). Sentiment analysis of film reviews based on CNN-BLSTM-attention. Journal of Physics: Conference Series, 1550(3), 032056. doi:10.1088/1742-6596/1550/3/032056