IMDB-Movie-Review-Classifier

About the project:

This project focuses on sentiment analysis which is the process of analyzing text data of different reviews to determine if its tone was positive or negative. The IMDb dataset was used which had collections of data related to the reviews and sentiments of movies and TV shows available on the Internet Movie Database (IMDb).

Import and Analysis:

After obtaining a dataset with 50,000 rows of data, I have conducted a thorough analysis to understand its properties. The dataset included two essential columns: 'sentiment' which contained categorical values indicating positive or negative sentiment, and 'reviews' which included textual reviews. The text reviews and the sentiments that went along with them were extracted by utilizing Python's flexible pandas library. I have put the reviews into a variable called "reviews" and assigned the variable "labels" with binary values of 1 for positive sentiment and 0 for negative sentiment. The careful preprocessing set the stage for further dataset manipulations and explorations.

Introducing Glove:

I have made use of GloVe which is a precomputed word embedding in 100-space that has been trained on a very large corpus consisting of almost all the words in the English language. GloVe captures semantic relationships between words and achieves this with a co-occurrence matrix. I have loaded GloVe word embeddings from a text file using the ‘load_glove_embeddings’ function which was stored in the ‘glove_embeddings’ dictionary. The function reads the GloVe embeddings file line by line and then splits each line into words and its corresponding vector components. Each word is stored as a key of the dictionary while its corresponding value represents its vector components which are converted into a NumPy array. This ‘glove_embeddings’ dictionary can be used to search the word vectors for any words that are going to be used for our sentiment analysis.

Tokenize and Pad Sequences:

The ‘Tokenizer’ and ‘pad_sequences’ functions were used from the Keras library to tokenize and pad the text sequences as part of the preprocessing step. The parameter ‘num_words’ was set to 5000 which will only consider the most frequent 5000 words in the dataset while other words will be ignored to limit the vocabulary size for greater efficiency and focus on the most relevant words. The method ‘fit_on_texts’ was used to process the texts of the variable ‘reviews’ and update the vocabulary of the tokenizer based on the word counts in the provided texts. The method ‘texts_to_sequences’ will convert each of the texts into a sequence of integers based on the vocabulary learned during the ‘fit_on_texts’ step. The method ‘word_index’ was also used which is a dictionary containing words as keys and their corresponding integer indices as values based on their frequency that was created during the ‘fit_on_texts’ step. Finally, The ‘pad_sequences’ function was used to ensure that all sequences have the same length with the specification of the maximum length of the sequences in the parameter.

Preparation of GloVe embedding matrix:

An embedding matrix was prepared by using pre-trained GloVe word embeddings by initializing zeros to the NumPy array. It has rows with the size of the number of unique words in the vocabulary and a column of size 100 which represents the dimensionality of the GloVe word embeddings. I have updated the rows in the ‘embedding_matrix’ with the GloVe embedding vector for each word in our vocabulary. Finally, this matrix was ready to be used as weights of an embedding layer in the neural network.

Embedding Coverage Analysis:

The code calculates the percentage of words in our dataset that have pre-trained GloVe embeddings. The ‘vocab_size’ is the count of unique words, and ‘nonzero_elements’ is the count of words with available embeddings. By dividing the latter by the former, the percentage of words for which I have pre-trained embeddings was found. This percentage helps us understand how well our dataset aligns with the existing word meanings in GloVe. A higher percentage indicates a better match, potentially improving our model's ability to understand and analyze text, especially in tasks like sentiment analysis.

Train Test Split:

Our data was split into two parts 80% for training and 20% for testing using the function ‘train_test_split’ by setting the ‘test_size’ parameter to 0.2. The variables ‘X_train’ and ‘X_test’ consist of the training and testing features which are the padded sequences of integers obtained from text data while the variables ‘y_train’ and ‘y_test’ represent the training and testing labels which consist of the positive and negative sentiments.

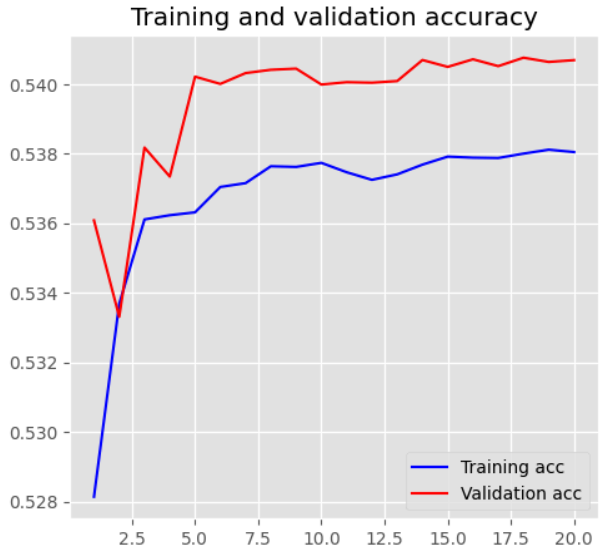
Simple Machine Learning Model:

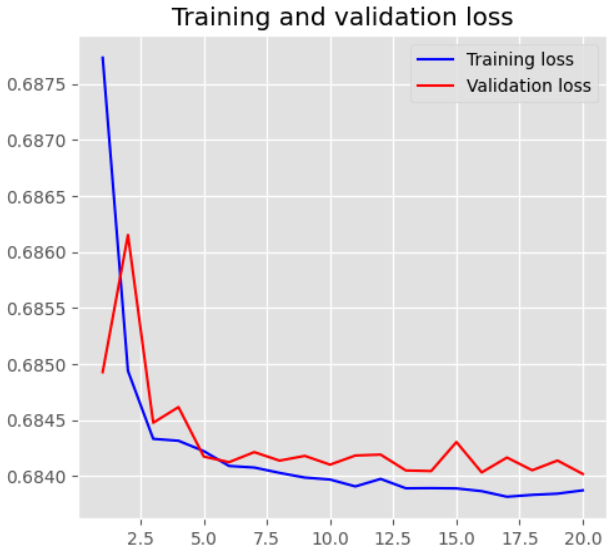
For training a machine learning model on the training part of the data, I have used logistic regression for this text classification task in which the model learns the relationship between the input features and the target variable. Its performance on the test part of the data was found by calculating the mean accuracy of the model by comparing its predictions with the true labels of ‘y_test’. It obtained an accuracy of approximately 52% which indicates that the model's predictions are correct only around half of the times.

Shallow model:

A three-layer shallow neural network was trained where the initial layer comprises an embedding layer utilizing pre-trained GloVe embeddings with a 100-length vector output. Notably, I have set 'trainable' to True, enabling fine-tuning of the pre-trained embeddings during training. The second layer, a hidden dense layer with a ReLU activation function and an output shape of 10 contributes to the model's representation learning. The final layer, facilitating binary classification with a sigmoid activation function that outputs probabilities. The model is compiled using binary cross-entropy loss, Adam optimizer, and accuracy as the monitoring metric. After training for 20 epochs with a batch size of 10, the model achieves a test accuracy of 53.22%. The training and validation accuracies stabilized early in training, suggesting a relatively quick convergence. The model's accuracy varied based on different settings. When the batch size

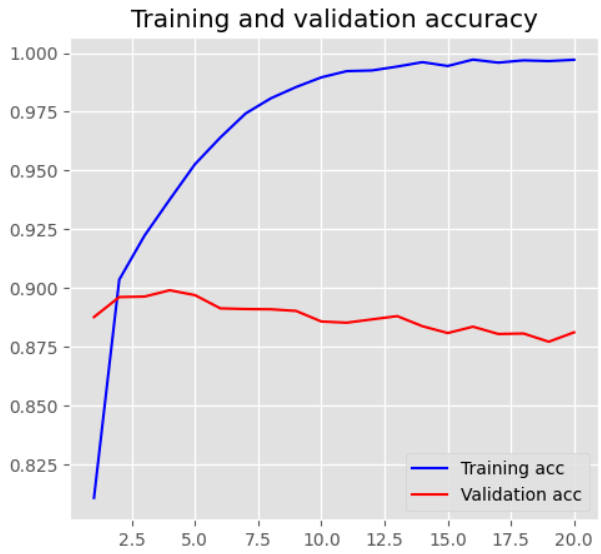
was set to 30 and the model fine-tuned its pre-trained embeddings, it achieved the highest accuracy at 54%. This highlights the impact of adjusting parameters like batch size and training settings on the model's performance. I have also plotted a graph for further analysis where it was observed that the accuracy was slightly better during validation compared to training while there was a slight drop in the loss for both cases.

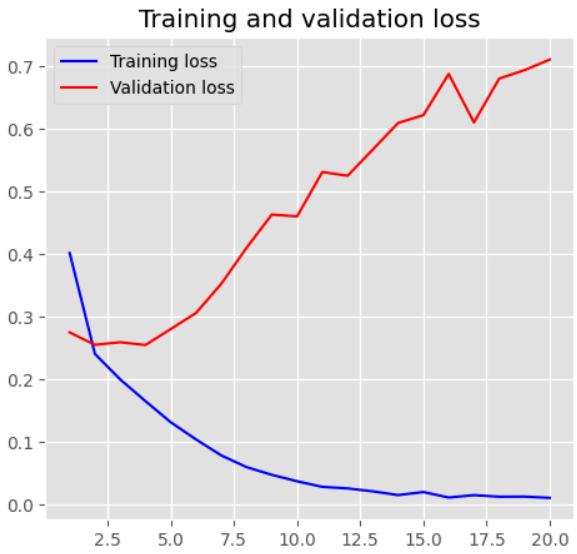




Unidirectional LSTM:

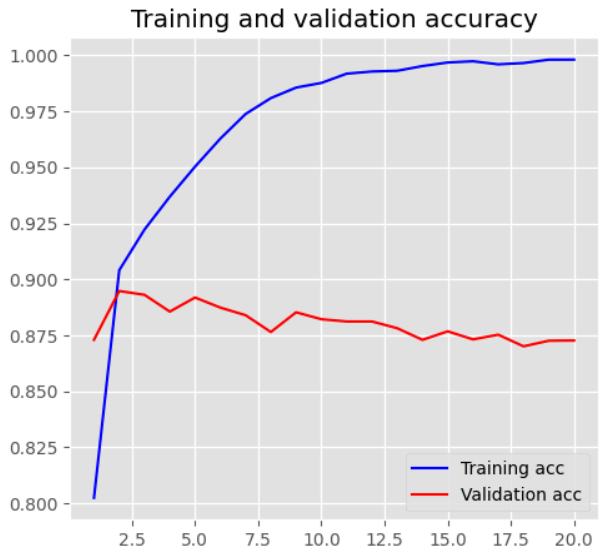
The model was made better by adding a special layer called LSTM which helps the model understand and remember information over longer sequences. This change involved replacing the first dense layer with this LSTM layer, which has 10 memory units. The structure still starts with the Embedding layer for word representations and ends with a Dense layer for binary classification. Like before, I have used binary cross-entropy loss and the Adam optimizer to train and fine-tune the model for improved performance and train it for 20 epochs. First, with a batch of 32 and fixed parameters, it had an accuracy of 85%. Keeping the same batch size but allowing some adjustments led to a slightly better accuracy of 86%. The most improvement happened when I reduced the batch size to 25 and kept the parameters fixed, resulting in the highest accuracy of 88%. Again, I have plotted a graph and observed that the accuracy for training was much higher than validation. It was also seen that the loss was decreasing during training but was increasing significantly during validation.

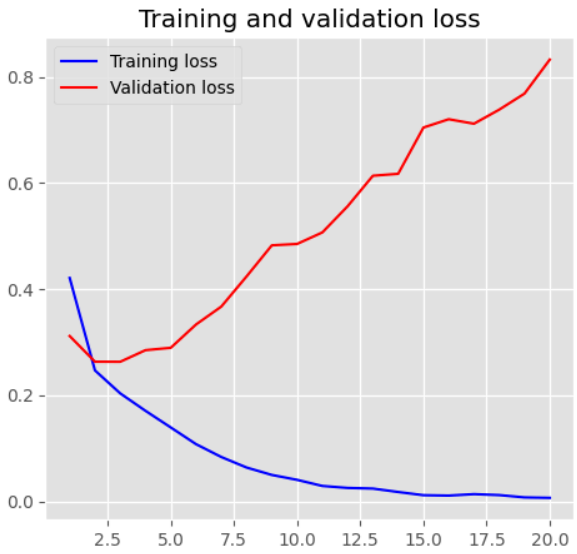




Bidirectional LSTM:

The model was further enhanced by making it even more aware of patterns by replacing the first dense layer with a special bidirectional LSTM layer. It looks at the input sequence in both forward and backward direction to catch more patterns. With 10 units in the LSTM, the bidirectional layer effectively has 20. The final layer stays a Dense layer with one unit and a sigmoid activation that is perfect for binary tasks. Like before, I have compiled the model with binary cross-entropy loss and the Adam optimizer. This added feature helps the model understand sequences better, potentially improving its ability to predict binary outcomes. The Bidirectional LSTM model showed a strong performance during training, achieving an accuracy of 99.9% which indicates its ability to learn and adapt well to the training data. When tested on new, unseen data (the test set), the model maintained a high accuracy of 87.3%. These accuracy values suggest that the Bidirectional LSTM model is effective in both learning from the provided data and making accurate predictions on previously unseen data. After plotting the graph, I have again observed that the accuracy for training was much higher than validation with decreasing loss during training and increasing loss during validation.





Comparing Sentiment Analysis Models and Suggestions for Improvement:
After analyzing the performances of all the models with the original one, both the models with unidirectional and bidirectional LSTM layers performed very well compared to the machine learning and shallow models. The reason behind this is because LSTMs have a more complex architecture than simple RNNs by having memory cells which enables it to capture and store information over long sequences that is essential for tasks where dependencies exist. Moreover, bidirectional LSTMs are able to capture the contextual information from both the past and the future which is essential for understanding the meaning of words in a sentence. However, the LSTM models were prone to overfitting as it did not perform very well during validation despite having high accuracy in training. Therefore, it is important to apply dropouts to the neural network layers during training which will prevent the network from relying too much on specific neurons. Moreover, another improvement can be the augmentation of the dataset by applying techniques such as paraphrasing or adding synonyms to increase the variety of expressions. Furthermore, the model can be made better by improving the performance and reducing runtime by implementing GRU as it has a simpler architecture and fewer parameters compared to LSTMs with one less gate, which can make it computationally more efficient and easier to train.