Predict the sentiment of Reviews Using LSTM Model

Problem Statement:

The aim is to predict the Type of the Sentiment By Analyzing the Reviews of the Customers.

Dataset Overview:

The dataset consists of 1,62,449 rows with 2 features:

1. **Tweets:** This column typically contains text data from individual tweets posted on Twitter. Each entry in this column would be a string of text representing the content of a single tweet. The data could include text, emojis, hashtags, mentions, and URLs, depending on how the data was collected and preprocessed.
2. **Labels:** This column usually contains categories or classifications associated with each tweet. The labels could represent various classes such as sentiment Good or Bad.

Data Pre-Processing:

Essential libraries such as NumPy and Pandas for data manipulation, and Matplotlib and Seaborn for visualization, were imported. No null values or duplicates were found. The dataset was then split into categorical and numeric variables for detailed univariate analysis.

Univariate Analysis:

As Both the columns are categoric we conducted value counts and found out that some of the tweets are repeating and the reviews contain url links.

As we look at the value count of our target column ‘labels’ we clearly observe that we have imbalance in the target variable. as one sub-class is superior than other. We have to perform some oversampling techniques.

Text-Preprocessing:

As our task is to Analyze text we have to pre-process the tweets before feeding it to the model. As for the Pre-Process we have to remove the url links that we observe in univeriate Analysis. We build a function to remove the url links from the tweets.

Then, It first creates a PorterStemmer instance and an empty list ('corpus') to store the preprocessed tweets. The code then iterates over each tweet, removing non-alphabetic characters, converting to lowercase, and splitting into words. It applies stemming to each word using the PorterStemmer, and removes English stopwords. The processed words are rejoined into a single string. Finally, the preprocessed tweet is appended to the 'corpus' list. This preprocessing helps reduce text data to its essential meaning, improving the performance of text analysis.

Model Building:

As for model to build we have to convert our text data into numeric form as machines only understand numeric values. It uses the one_hot function from TensorFlow's Keras library to convert each word in the corpus into a one-hot vector of length 'voc_size' (5000). The length of the longest one-hot vector is determined. Then, 'sent_length' is set to 55, defining the maximum sentence length. The pad_sequences function is used to ensure all sentences have the same length ('sent_length') by adding zeros at the start of shorter sequences. This results in 'embedded_docs', a list of equal-length sequences, ready for input into a neural network model. This preprocessing step is crucial for models that require fixed-size inputs.

Then the embedded tweets (embedded_docs) and their corresponding labels (y) are converted to numpy arrays. These arrays are then split into training and testing sets using scikit-learn's train_test_split function, with 33% of the data allocated for testing and a fixed random state for reproducibility. The model is compiled with binary cross-entropy loss, Adam optimizer, and accuracy as the evaluation metric. the model is trained on the training data for 10 epochs with a batch size of 64. Validation data is provided to monitor the model's performance on unseen data during training.

The predictions, initially probabilities, are converted to binary labels (0 or 1) based on a threshold of 0.5. The performance of the model is then evaluated using the classification_report function from scikit-learn, which provides detailed metrics including precision, recall, and F1-score. This report helps in assessing the model's effectiveness in distinguishing between the two classes.

```
1676/1676 [==============================] - 6s 3ms/step
              precision    recall  f1-score   support

           0       0.94      0.94      0.94     35230
           1       0.88      0.89      0.88     18379

    accuracy                           0.92     53609
   macro avg       0.91      0.91      0.91     53609
weighted avg       0.92      0.92      0.92     53609
```