Team 1: Predicting Disaster Tweet Using Bidirectional LSTM on Kaggle Tweet Data.

Ishmam Bin Rofi
Brac University
22301229

Mashiyat Mahjabin Eshita
Brac University
22101878

1 Introduction

Social media has become a significant part of our daily lives. We love to express ourselves on social media than any other platforms in recent times. Therefore, using one of the most popular social media platforms Twitter to analyze the situation of natural disasters can be a great help to the people. In our study, we used the popular dataset from Kaggle (Nat). Using the dataset the most important task of this project was to detect that if the data was disaster or non-disaster. In this work, we made these contributions:

1. We cleaned the dataset and predicted if the data was disaster data or not.
2. We evaluated the Bidirectional LSTM model for disaster detection in Twitter.
3. We focused on the F1 score and achieved a score of 71%.

2 Dataset

For our project we collected our dataset from Kaggle's 'Natural Language Processing with Disaster Tweets' (Nat). We used the train.csv as our dataset.

2.1 Dataset Description

Our dataset had 5 columns and 7613 instances. We split our dataset into three segments (train, test, validation). 70% of the total data was used as train data, 20% as test data, and 10% as validation data. The split description is shown in table 1.

Data Label	Percentage	Instances
Train	70	5329
Test	20	1530
Validation	10	754

Table 1: Description of Data

After doing the split and preparing our data, we harnessed multiple plotting techniques to visualize our data properly. We checked the top 20 disasters vs non disaster keywords of our dataset to understand the 'text' column in our test data. The plot showed us that the top keyword in the 'Disaster' class was 'wreckage' and 'typhoon'. While the top keywords in non-disaster tweet were 'siren' and 'electrocute'. The figure is shown in figure 1. Later, we plotted the 'target' column of our train dataset to find the balance and count of disaster and non-disaster tweet. As shown in the Figure 2 we found out that the count of Disaster vs Non-Disaster data in the train dataset is almost balanced.

In the later step, we also plotted the character length of the tweets (Figure 3), while we found out that most of the tweets in our train dataset are around 120-140 characters. This gave us an idea that most of our tweets in the train dataset are moderately long and they are almost half the length of the character limit.

2.2 Dataset Preprocessing

We divided our dataset preprocessing into two parts, in the first part we cleaned our test data and created a new column 'clean_text'. Later we tokenized our data using 'Tokenizer' from 'Keras'.

2.3 Cleaning Dataset

Punctuation Removal We started our dataset cleaning by removing the punctuation from the 'text' of our train data. We used 'string.punctuation' to generate a string full of all punctuations, then we used a python function to remove all the punctuations from the 'text' and appended them in a new column 'clean_text'.

Removing Noise First we collected all the abbreviations of English tweet from a Kaggle code (Sandhyakrishnan, 2022). Then we declared some functions to remove url, HTML tags, mentions,

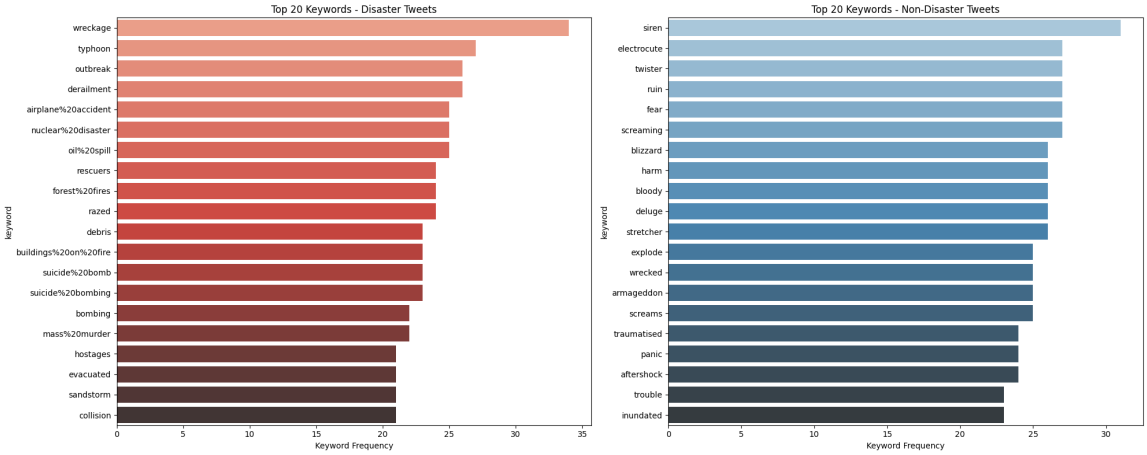

Figure 1: Disaster Vs Non Disaster Tweet in Train Data

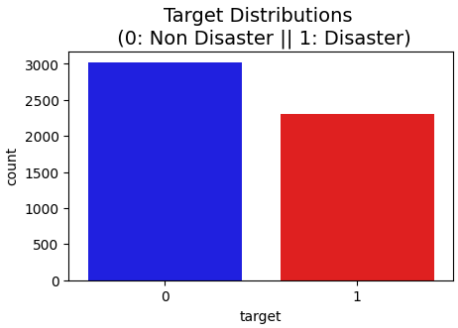

Figure 2: Distribution in Target Column

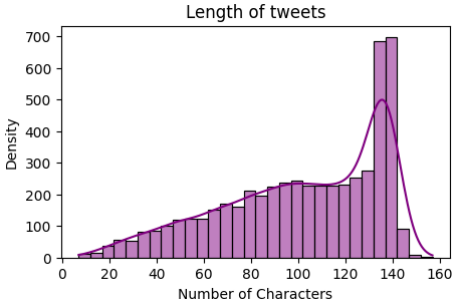

Figure 3: Plot of character length in the train dataset

abbreviations, emojis, emoticons. After removing all these we finally had our clean text in our 'clean_text' column. Finally, we used stop words from our dataset. Stop words are those words which are used frequently in sentences but do not have any significance during prediction (Sarica and Luo, 2021). We used NLTK stopwords to produce our stopwords list.

2.4 Tokenization

We first define a tokenizer with a vocabulary size limit of 3,000 words and fit it on the training dataset's 'clean_text' column. This tokenizer converts the text into sequences of word indices based on the tokenizer's vocabulary. We then apply this process to the training, validation, and test datasets. After converting the text to sequences, we pad the sequences to ensure that all inputs have the same length, preparing them for input into a machine learning model.

3 Model Description

3.1 Embedding of data

We load pre-trained GloVe (Pennington et al., 2014) word embeddings from the file 'glove.6B.100d.txt' into a dictionary called 'embeddings_index', where each word is mapped to its corresponding 100-dimensional vector. We then initialize an 'embedding_matrix' filled with zeros, sized to hold the top 'max_features' (3,000) words from the tokenizer's vocabulary. As we iterate through the tokenizer's word index, if a word exists in the 'embeddings_index' and its index falls within the 'max_features' limit, we insert the pre-trained embedding vector into the 'embedding_matrix' at the

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 20, 100)	300,000
dropout_1 (Dropout)	(None, 20, 100)	0
bidirectional_1 (Bidirectional)	(None, 128)	84,480
dense_1 (Dense)	(None, 1)	129

Total params: 553,829 (2.11 MB)
Trainable params: 84,609 (330.50 KB)
Non-trainable params: 300,000 (1.14 MB)
Optimizer params: 169,220 (661.02 KB)
None

Figure 4: Model Summary

corresponding position. This matrix is later used to initialize the embedding layer of our neural network, enabling the model to benefit from the rich semantic information in the pre-trained word embeddings for more effective text processing and classification.

3.2 Model Description

For our project, we define a sequential model for binary text classification using Keras. In the embedding layer of the model, we take pre-trained GloVe embeddings to represent words in a 100-dimensional space. We have used fixed embeddings using 'trainable = False'which means that the weights won't be updated during the training of the model. In the dropout out layer, we used a dropout rate of 20% to reduce overfitting of the data during training.

In the bidirectional LSTM layer, we use a LSTM layer with 64 output units. We used bidirectional LSTM so that our model can capture dependencies in both forward and backward directions. It also includes an input dropout of 0.2 and recurrent dropout of 0.4 which enhances the generalization by reducing reliance on specific neurons.

In the dense layer, we have only a single neuron and a sigmoid function which is used for binary classification.

The model is compiled using the Adam optimizer with a low learning rate ('1e-5'). The loss function is binary cross-entropy, and the evaluation metrics include recall, accuracy, and precision.

Finally, we implemented an early stopping callback is defined to monitor the validation loss and stop training if the loss doesn't improve for 3 consecutive epochs, helping to prevent overfitting.

The summary of the model is shown in the figure 4.

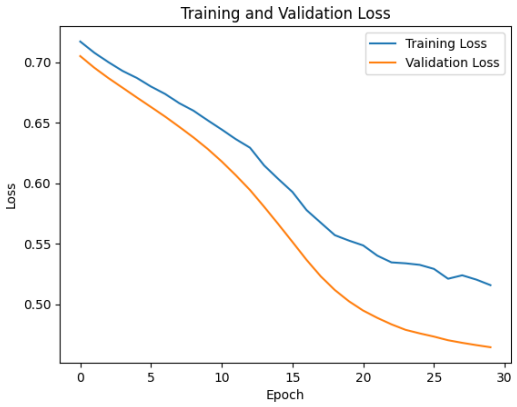

Figure 5: Training Vs Validation Loss

4 Results

4.1 Train Vs Validation Data

4.1.1 Training Vs validation Loss

Figure 5, represents the training and validation loss over 30 epochs during the model's training process. The training loss (blue line) starts high and steadily decreases, indicating that the model is progressively learning and improving its performance on the training dataset. The validation loss (orange line) also begins relatively high but decreases more sharply than the training loss, suggesting the model is generalizing well to unseen validation data. Importantly, the validation loss continues to decline throughout the training process, showing no signs of overfitting, as the model maintains a strong performance on the validation set. Overall, the plot suggests the model is effectively learning from the data without overfitting, which is a positive outcome for this training run.

4.1.2 Training Vs Validation Accuracy

Figure 6 illustrates the training and validation accuracy over 30 epochs during the model's training process. The training accuracy (blue line) initially dips and then begins to increase around the 10th epoch, showing that the model starts learning gradually and improves its performance on the training dataset as the epochs progress. The validation accuracy (orange line) starts high, indicating that the model initially performs well on the validation data, but then it decreases slightly before stabilizing and improving around the 15th epoch. The validation accuracy plateaus near the end, while the training accuracy continues to increase, suggesting that while the model is improving on the training set, it may not be significantly enhancing its performance on the validation data

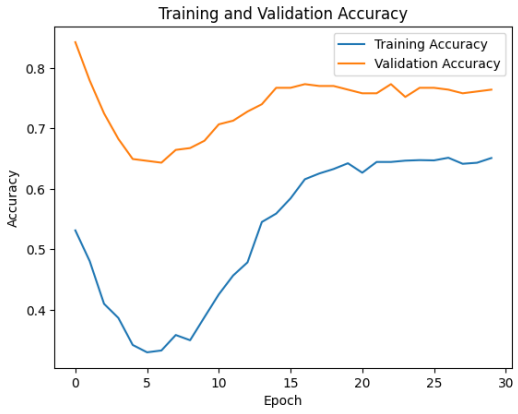

Figure 6: Training Vs Validation Accuracy

4.2 Evaluation on Test Data

The model achieved a Test F1-Score of 0.7055, with an overall accuracy of 77% on the test set of 1,530 samples. Class 0, representing the majority class, had a precision of 0.80, a recall of 0.81, and an F1-score of 0.81, while Class 1 had slightly lower scores with a precision of 0.73, a recall of 0.71, and an F1-score of 0.72. The macro and weighted averages of precision, recall, and F1-score were both 0.76 and 0.77, respectively, indicating balanced performance across both classes, though the model performed slightly better on Class 0. The detailed description is shown in the Table 2,

5 Conclusion

To conclude, after the 10th epoch, our model started learning and showed stability in the 15th epoch.

Data Label	Precision	Recall	F1-score	Support
0	0.80	0.81	.81	895
1	.73	.71	.72	635
Macro Avg	.76	.76	.76	1530
W. Avg	.77	.77	.77	1530

Table 2: Details of Test Result

Then we checked our model using test data and got 77% accuracy with 70% F1 score from our prediction. Consequently, our model predicts both disaster and non-disaster data pretty well indicating that our dataset is very balanced. Consequently, we have also got good accuracy from our model. It shows that our model is working and predicting from the data set pretty well.

6 Acknowledgement

For this project, we took extensive help from outside sources. We took help from Large Language Models like ChatGPT, Copilot. But all AI generated codes and texts were revised manually. We also took extensive help from this particular Kaggle solution. We also took help from a Medium Blog to implement our model properly.

References

[link].

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Sandhyakrishnan. 2022. Nlp with disaster tweets using lstm.

Serhad Sarica and Jianxi Luo. 2021. Stopwords in technical language processing. *Plos one*, 16(8):e0254937.