

# BUDT 737 - Big Data and Artificial Intelligence for Business




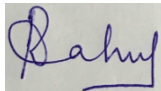
Project Title: NLP with Disaster Tweets using Neural Networks

**Team Members:** Vikash Gujju, Lohith Reddy, Alekya Ghanta, Rahul Murugkar, Jharna Dharaiya

**Team name on kaggle:** BUDT737\_Spring23\_Group17

## **ORIGINAL WORK STATEMENT**

We the undersigned certify that the actual composition of this proposal was done by us and is original work.

Names	Signatures
Vikash Gujju	
Lohith Maddula	
Alekya Ghanta	
Rahul Murugkar	

# 1.Introduction

In this challenge, we will be looking at twitter tweets to build a machine learning model to predict whether tweets are about real disasters or not.

The dataset used here consists of about 10,000 tweets from users who are tweeting about potential disasters, governmental and news agencies are keen to monitor them to gain a better idea of those disasters so that they can get access to real-time data from users on the ground. This would assist them in making more informed decisions and allocating resources appropriately.

There are multiple ways and different approaches one can use to do that. We are primarily focusing on Natural Language Processing (NLP), using Tensorflow and spaCy.

## 2. Data Source

This dataset was created by the company figure-eight, and is located at <https://www.figure-eight.com/data-for-everyone/>.

There are 2 files in the dataset which we will use to build the model - train.csv and test.csv.

Each sample in the train and test set has the following information:

- The text of a tweet
- A keyword from that tweet (this may be blank)
- The location the tweet was sent from (this may be blank)

Columns in the testing and training dataset:

- id - a unique identifier for each tweet
- text - the text of the tweet
- location - the location the tweet was sent from (may be blank)
- keyword - a particular keyword from the tweet (may be blank)
- target - in train.csv only, this denotes whether a tweet is about a real disaster (1) or not (0)

## 2. Code Example

We worked upon improving a pre-existing model - [Introduction to NLP with TensorFlow and spaCy](#). They are versatile, easy-to-use, and efficient.

TensorFlow is a machine learning framework developed by Google which is used in training deep neural networks, and building complex models which is perfect for our use case, natural language processing.

## 3. Methodology

We first performed Text preprocessing and Exploratory Data Analysis from Scratch.

spaCy & nltk are libraries which have been designed for NLP tasks, using which we can use pre-trained models for fleshing out our model. It has a lot of features for use in text processing, and using those algorithms, we can analyze and interpret the data we've chosen with greater accuracy.

We performed the following tasks to build our model:

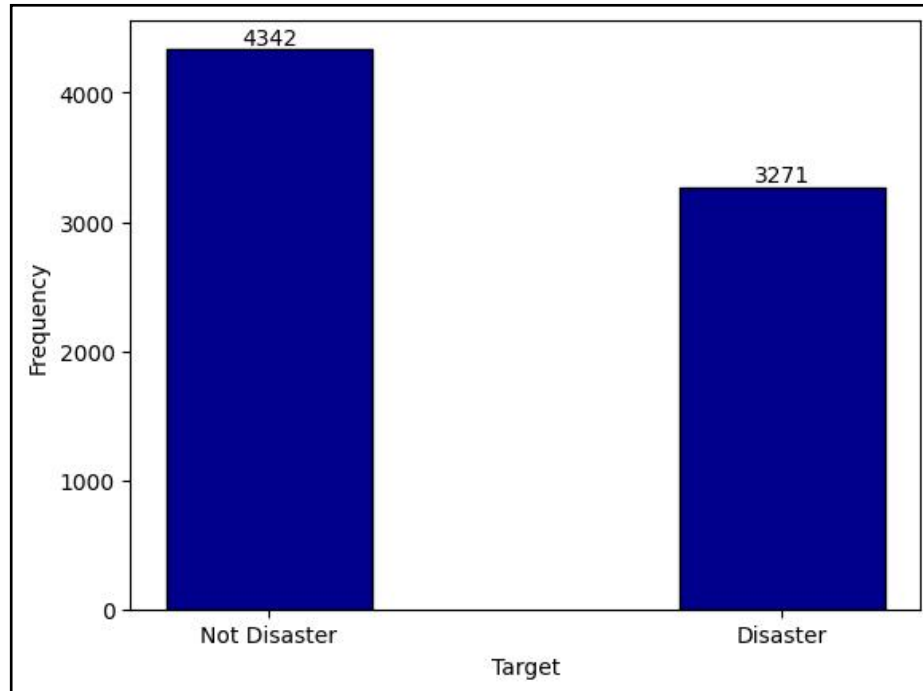
- Load the training and test data.
- Load libraries and packages.
- Preprocess the data.
- Inspect the training data.
- Tokenize and transform data from text to sequences.
- Analyze model performance through hyperparameter tuning.
- Select the best model.
- Investigate and plot our results.

## 4. Data Preprocessing and EDA

We explored the data to understand the underlying issues with it before we could use it to build our model. We started by understanding some basic statistics of our data. These insights from the structure and layout of our data helped us in the next steps of our model building, which is preprocessing. In general, an exploration of the data would help in deciding what the final model would look like and behave.

**How many tweets are classified as Disaster vs Not Disaster in our dataset?**

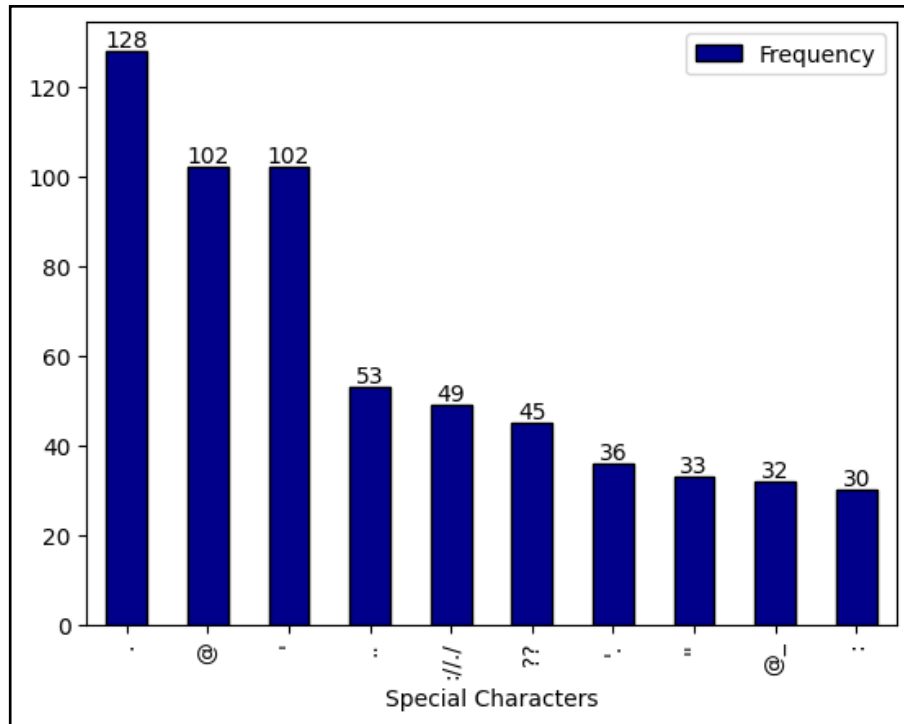
We have about 4,342 Not Disaster tweets vs 3,271 Disaster tweets in the train dataset. It is a relatively even balance of both the classes and hence a good sign for our model training. Having a relatively balanced dataset makes training a model easier because it will help prevent the model from becoming biased towards one class. Just because one class contains more data, the model will not be biased towards it. Hence, we do not need to perform any balancing techniques to build an accurate model.



*Distribution of target classes in training dataset*

### **What data cleaning steps need to be implemented?**

Firstly, we try to identify the non-alphabetic and special characters that would not be beneficial if included in the model. We will remove them from our dataset.



*Top occurring non-alphabetic and special characters in training dataset*

Some of the other cleaning measures we implemented were:

- Removing URLs
- Removing emojis, symbols & pictographs, transport & map symbols, and flags (iOS)
- Lowering the case of the text in the tweets
- Removing extra whitespace
- Removing stop words (articles, prepositions, etc.)

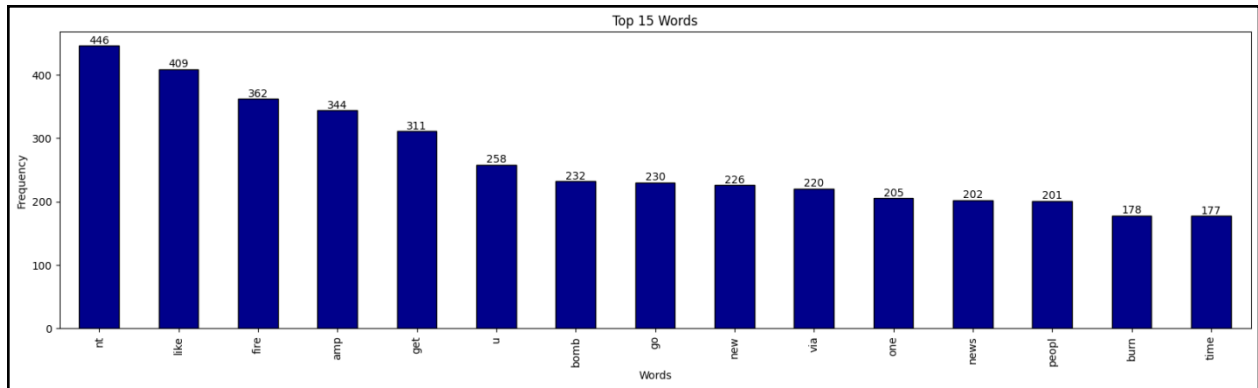
In order to accurately analyze the text and the frequency of words, we performed the following two operations:

- **Lemmatizing** - converting the word to its meaningful base form.
- **Stemming** - removing affixes from a word to retain the stem of that word.

These steps help us remove unnecessary words that don't add value to the analysis or context of the tweet and place emphasis on the words that actually matter to the classification of the tweets.

### **What are some of the frequently occurring words?**

In order to take a look at what the most frequently used words are when describing the tweet, we can see the following words are the top results:



*Top 15 words most frequently appearing in the training dataset*

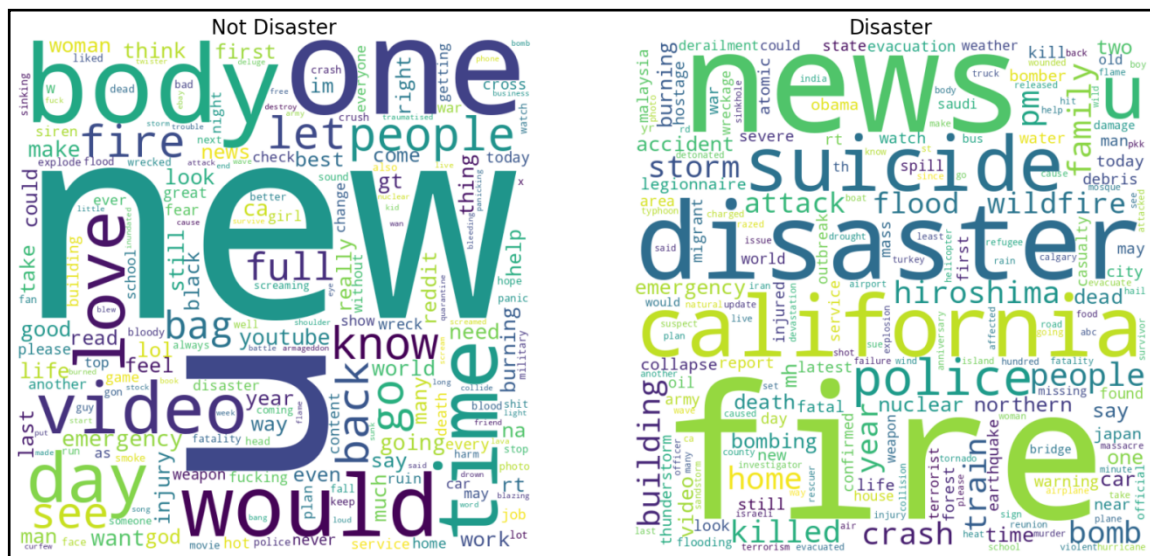
These can help the model determine what words are more likely to be used for a disaster vs. not a disaster. But what are the differences between words used for disaster tweets vs non-disaster tweets?

### Word Clouds for Disaster & Non Disaster Tweets

Word clouds can help visualize the frequency of words being used in a particular dataset. The image below shows us the comparison of the word clouds for disaster vs. non-disasters. The words for disaster tweets mainly include words like new, body, one, time, would, etc. However, they also included words like fire, injury, and wrecked which could signify a disaster tweet.

The disaster word clouds show us frequently used words like disaster, news, fire, california, suicide, etc. The usage of word california could be because the tweet database has been extracted at a time that had one or multiple disasters taking place in the state of California.

Overall, the word clouds helps us see if there is indeed a distinct pattern in the text and usage of words in the tweets between the two classes.

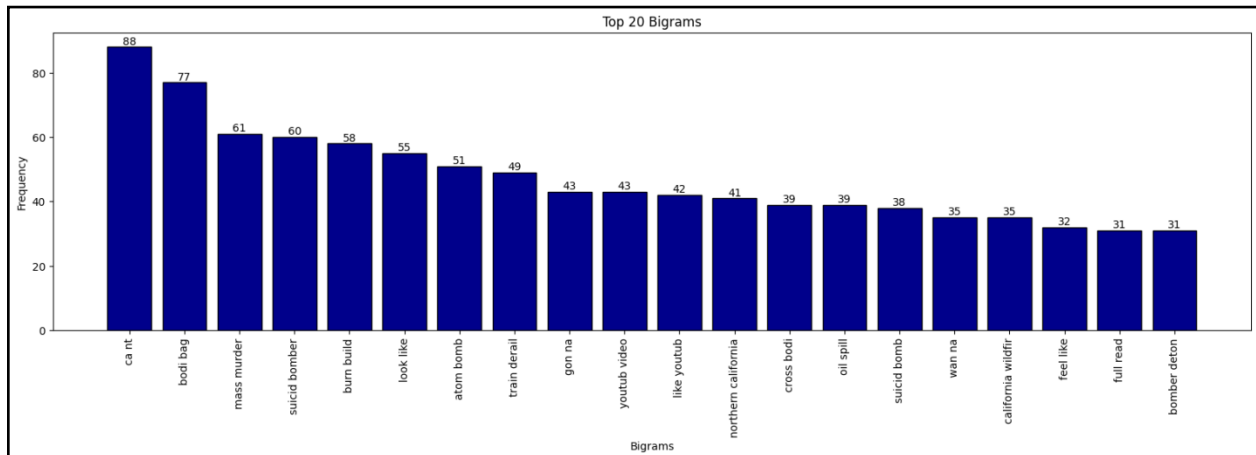


*Word cloud for non disaster vs. disaster tweets*

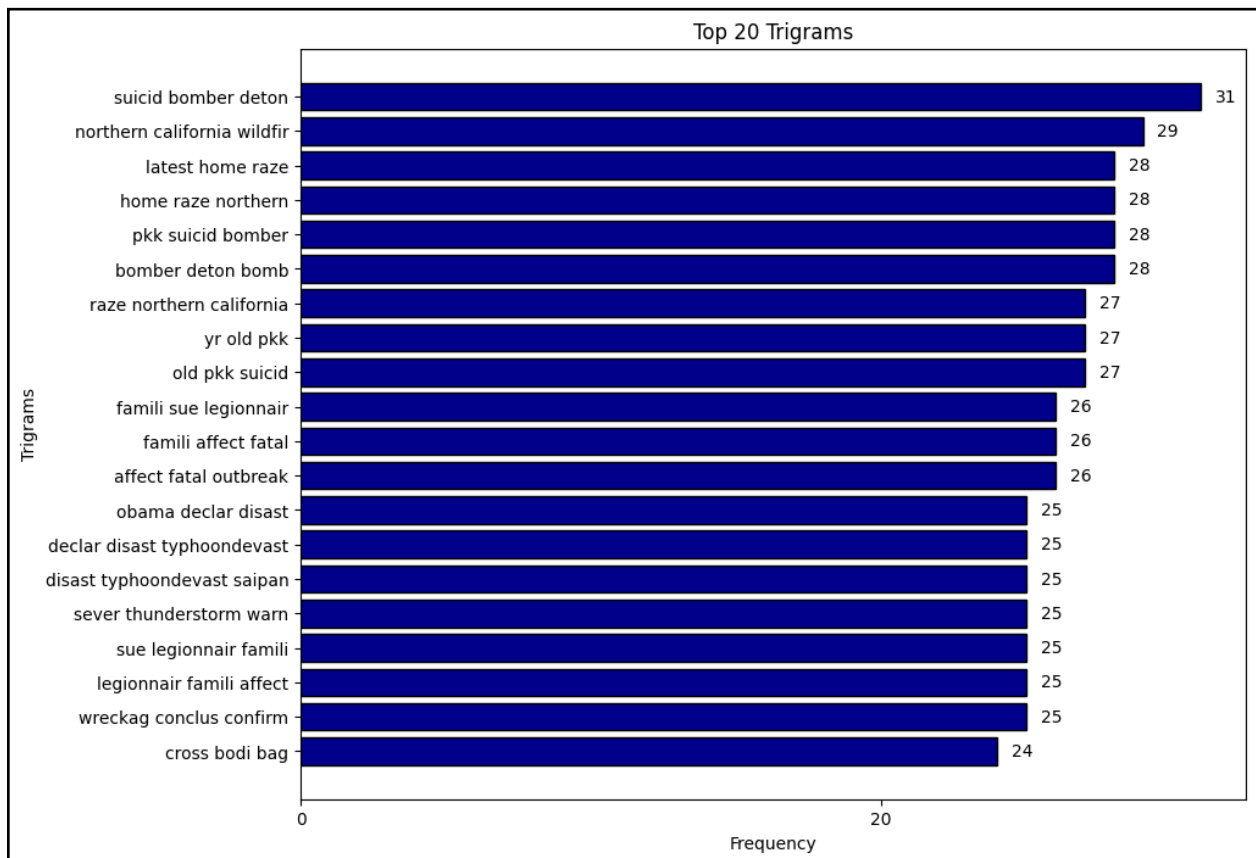
### N-grams

N-grams are one of the most fundamental concepts while working with text data. Bigrams, given the past two words, help to provide the probability of the next word. A Trigram using the past three words and similarly, an N-Gram using a user-defined N number of words.

### What are some of the most frequently used N-grams?



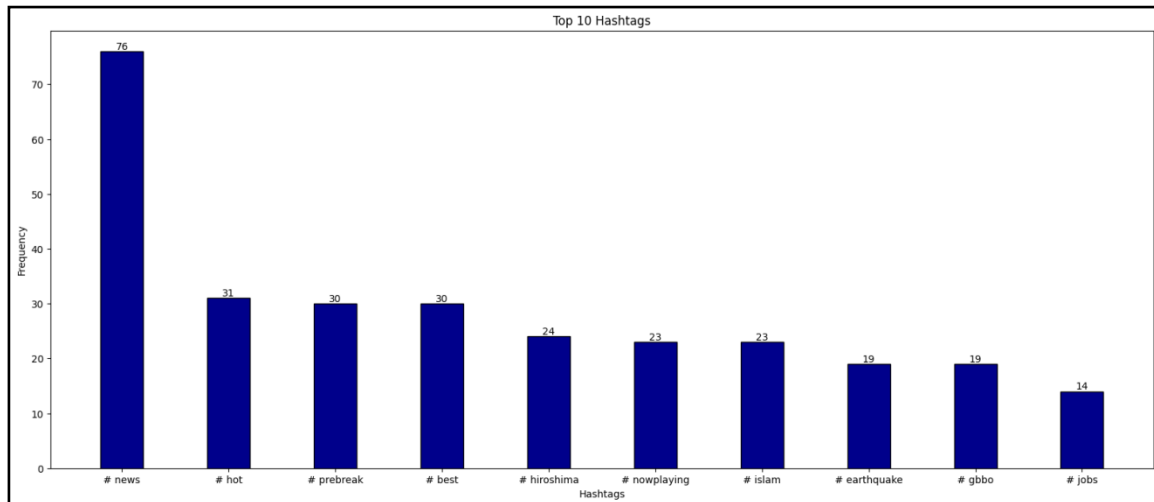
*Top 20 Bigrams by frequency*



*Top 20 Trigrams by frequency*

## Hashtags

Hashtags are an important aspect of Twitter. They help provide additional context to the tweet and help them be more relevant to trending topics. Understanding what hashtags are used within the tweets can be used as an additional input into the model to add context to something that might not make sense without the hashtags.



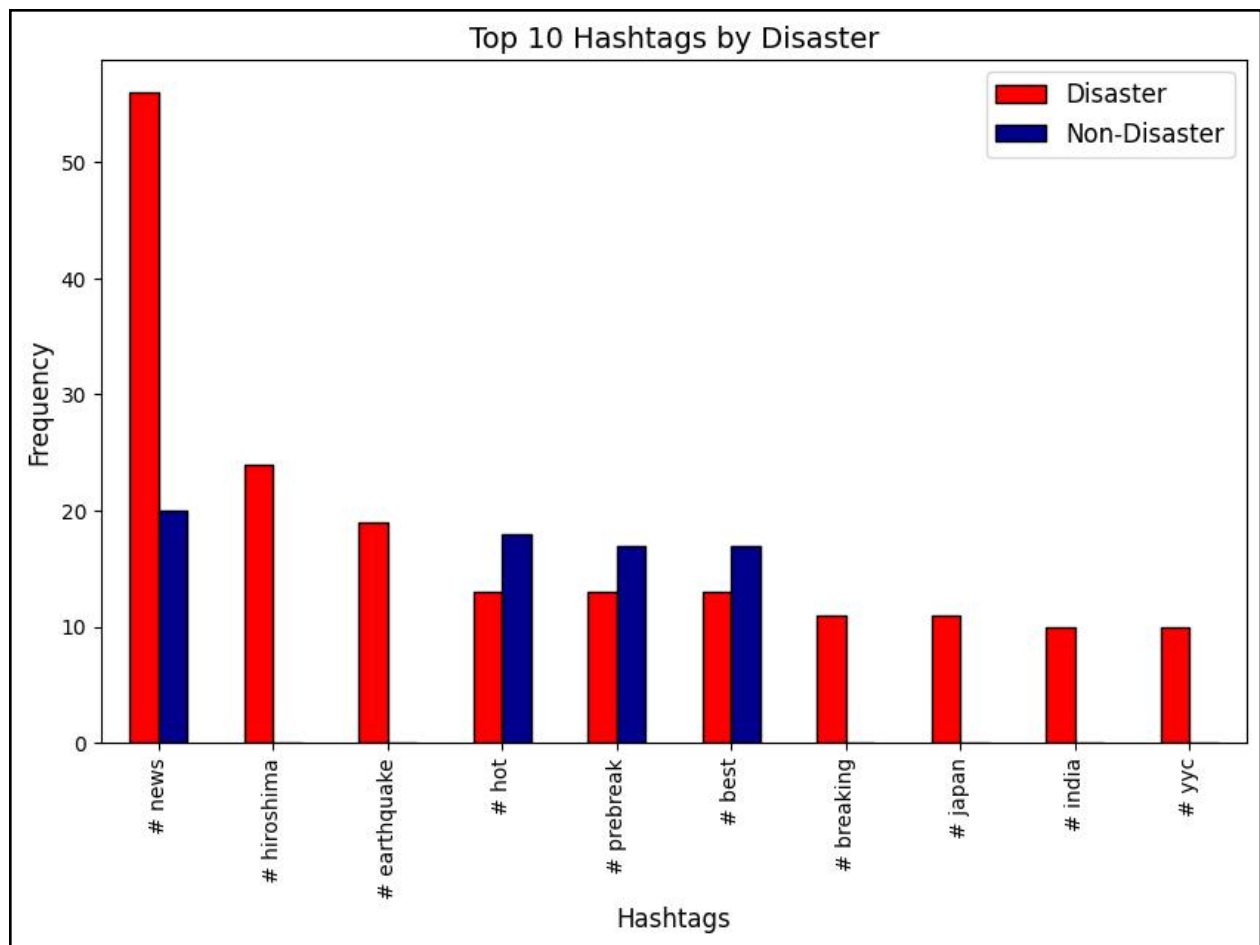
*Top 10 Hashtags by frequency*

In addition, it can help to know what hashtags are more prevalent for disaster vs non-disaster tweets.

The most frequent hashtags used in disaster tweets are:

- #news
- #hiroshima
- #earthquake
- #hot
- #prebreak





*Top 10 hashtags by target variable*

### Term Document Matrix:

In order to have a concise and numerical representation of text data we did the following to get a term document matrix:

1. **Tokenization** - separating the text into smaller units called tokens.
2. **Vectorization** - vectors of real numbers that is input for an ML model.

These steps essentially allow us to standardize the data, which can then be used as inputs in the NLP model. Since inputting raw text is unlikely to give us an accurate model due to the presence of unwanted characters or words that would increase the computational load of the model, these steps can help increase the efficiency of the model.

Additionally, tokenization and vectorization turn the raw text tweets into integer inputs which can be understood by the model. Vectorization can be visualized with the help of the following image:

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Document Vector

Word Vector  
(Passage Vector)

*Vectorization of raw text data by splitting them into individual terms*

For our model specifically, we have chosen the CountVectorizer function from sklearn's feature\_extraction module.

	about	all	cent	cents	money	new	old	one	two
doc	1	1	3	1	1	1	1	1	1

In theory (a)

↓

Index	0	1	2	3	4	5	6	7	8
doc	1	1	3	1	1	1	1	1	1

In practice (b)

*An example of how CountVectorizer creates vectors out of tokens*

We are converting raw text to a numerical vector representation of words by creating a sparse matrix. Each vector is then used as a feature in the NLP model as the algorithms only understand the concept of numerical features irrespective of its underlying type of data (whether it be text, image pixels, numbers, categories, etc.)

Below is a glimpse of the term document matrix:

☞	Tweet:1	Tweet:2	Tweet:3	Tweet:4	Tweet:5
wildfir	0	0	0	1	1
shelter	0	0	2	0	0
place	0	0	2	0	0
order	0	0	1	1	0
evacu	0	0	1	1	0
offic	0	0	1	0	0
reason	1	0	0	0	0
deed	1	0	0	0	0
school	0	0	0	0	1
notifi	0	0	1	0	0
rong	0	1	0	0	0
sent	0	0	0	0	1
canada	0	1	0	0	0
got	0	0	0	0	1
rubi	0	0	0	0	1

*Actual term document matrix sample*

## 5. Model composition:

After preprocessing the text, we first tried running the model on lemmatized & stemmed texts assuming it would be appropriate to avoid too many similar words being considered differently for the model.

However, our observations led to a conclusion that lemmatizing & stemming has been a hindrance to the performance of the model & led to decline in the accuracy. Therefore, we chose to skip Lemmatization & Stemming while preprocessing the data.

The train & test datasets were imported again. The following data cleaning steps were performed for model building:

- Removing URLs
- Removing emojis, symbols & pictographs, transport & map symbols, and flags (iOS)
- Lowering the case of the text in the tweets
- Removing extra whitespace
- Removing stop words (articles, prepositions, etc.)

We then converted the text into sequences of tokens and created a word index using **Tokenizer**, defined parameters like maximum length, maximum number of words in the vocabulary, out-of-vocabulary tokens, padding position, embedding dimension, assigned integer indices to words, and padded the sequences to a fixed length by adding a padding token at the end of the sequences.

### Improvements made on Base Code:

- **Max Length:** The longest text in the train data was 132. Therefore, we thought there would be many texts within the 125-135 range which might not be covered in the base model due to the Max Length being set to a lower value. Therefore, we increased the maximum length to 135.
- **Words in Vocabulary:** The number of words in the vocabulary (words\_in\_voc) determines the size of the vocabulary that the tokenizer will create. It controls the maximum number of unique words that will be assigned integer indices. Therefore, we thought increasing this to 15,000 will be essential for better performance of the model.

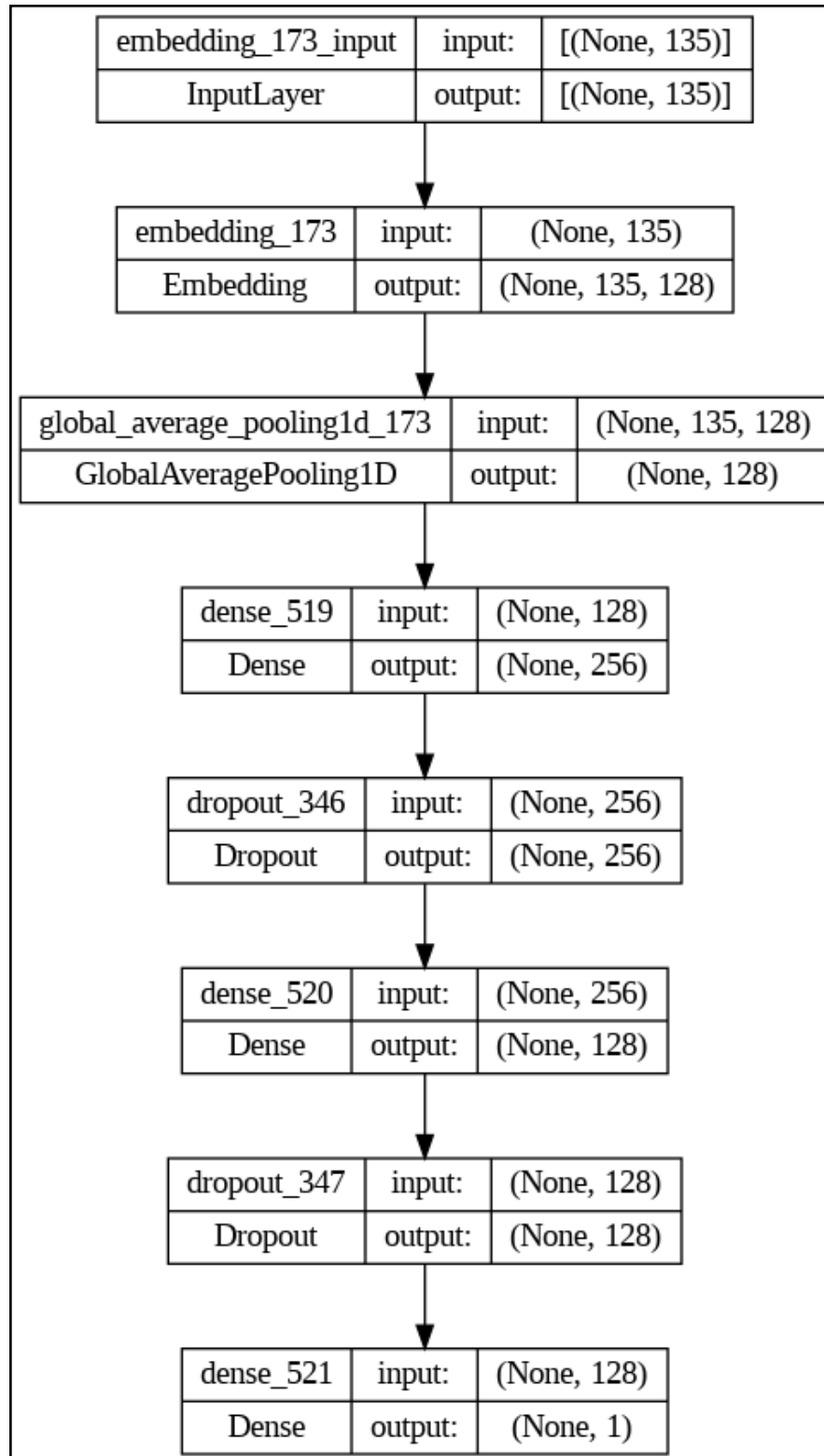
## Model Architecture:

The following layers have been used in the model:

- **Embedding Layer:** In this layer each word index is mapped to 128 embedding dimensions by taking the input sequence. These embeddings can grasp contextual information allowing the model to interpret meaningful representations of the data.
- **GlobalAveragePooling1D Layer:** This layer performs average pooling over the sequence dimension of the input to compute the average presence of a feature. It reduces the dimensionality of the sequence by taking the average across all sequence elements, helping the model to extract main features of importance.
- **Dense Layer(softmax):** Two dense layers of 256 & 128 units have been used with softmax activation to learn non-linear patterns in the data.
- **Dropout Layer:** This layer helps in regularizing the output of the previous layer. It randomly sets a fraction of input units to 0, which helps prevent overfitting and improves generalization.
- **Dense Layer (sigmoid):** This is the output layer that has been used with sigmoid activation function to get the probability of the input belonging to the positive class.

**Callbacks:** We have added EarlyStopping callback to monitor validation loss during training and stop the training process early if there is no further improvement.

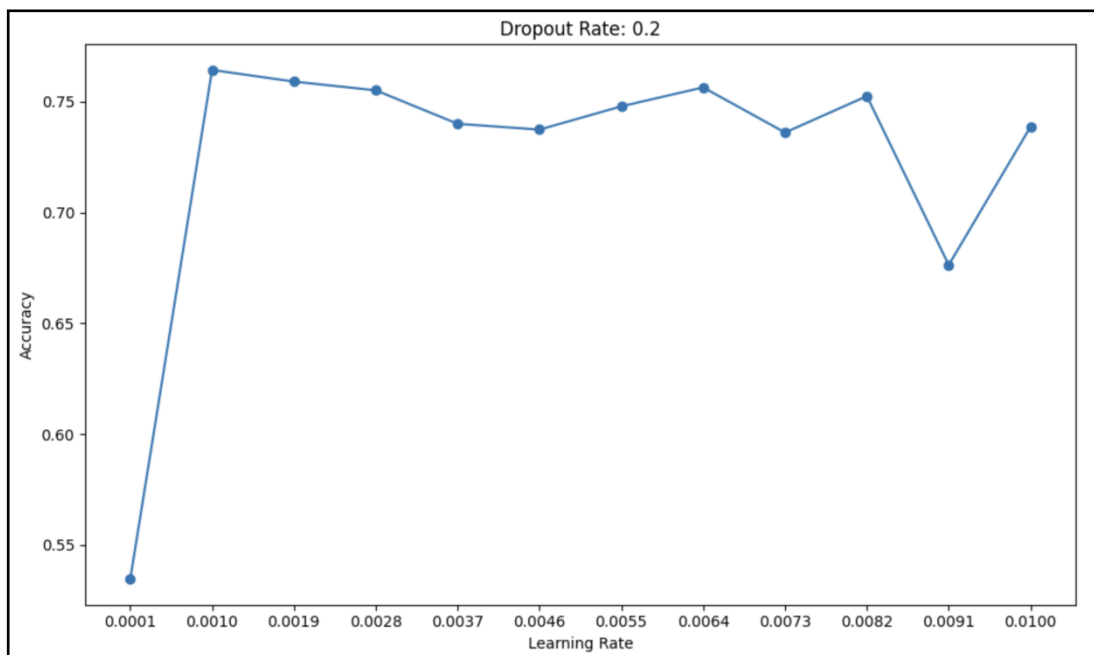
**Optimizer:** Adam optimizer has been used for the model. It updates the learning rate for each network weight individually.



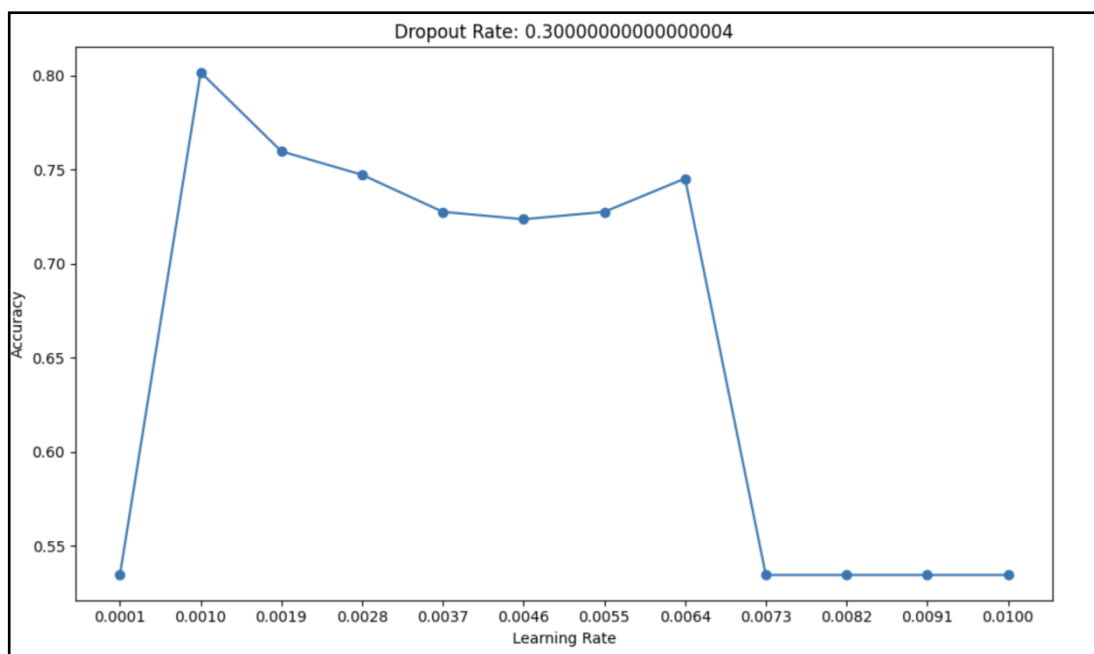
*Model structure*

## Hyperparameter Tuning:

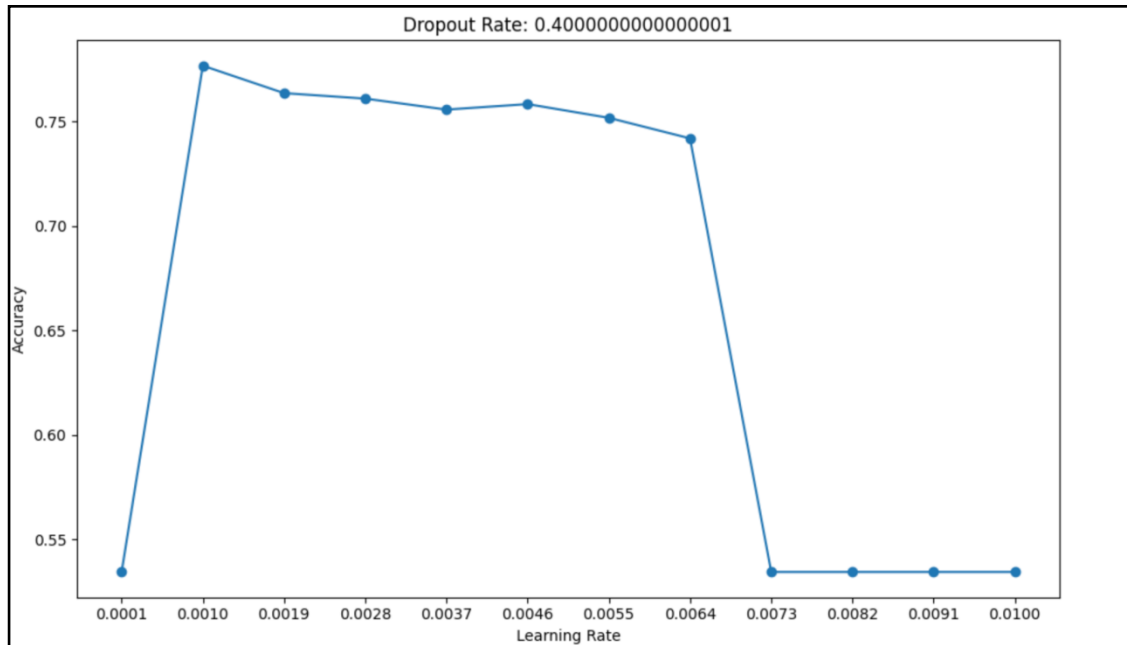
In order to choose the model producing best validation accuracy, we changed the learning rates from 0.0001 to 0.01 with a step size of 0.0009 & dropout rates from 0.2 to 0.5 with a step size of 0.1 simultaneously with the model architecture. Below are the results -



*Accuracy at dropout rate of 0.2*



*Accuracy at dropout rate of 0.3*



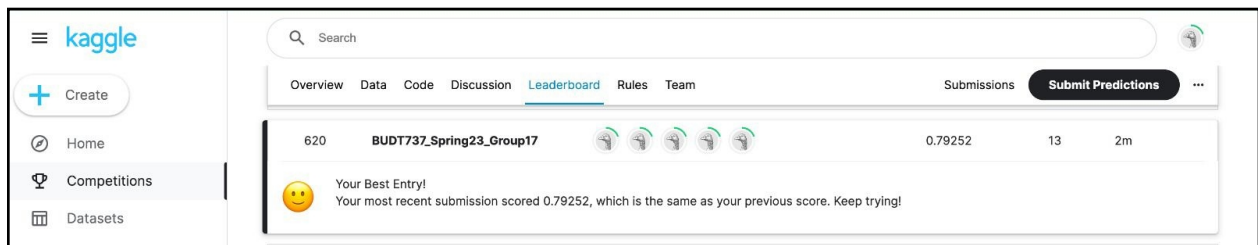
*Accuracy at dropout rate of 0.4*

Model with highest accuracy was observed when the learning rate is 0.001 & dropout is 0.03. The Final model has been built based on this learning rate & dropout.

### Improvements made on Base Code:

- **Activation Function:** The activation function has been changed from relu to softmax. We have tried various other functions like tanh, selu etc & concluded that softmax is the best activation function for our data.
- **Optimizer:** The optimizer has been changed from RMSProp to Adam owing to its computational efficiency and performance on a wide range of deep learning tasks.
- **Learning Rate:** Computed from the hyperparameter tuning.
- **Dropout Rate:** Computed from the hyperparameter tuning.

## 7. Kaggle Score



*Screenshot of Kaggle Submission Score*

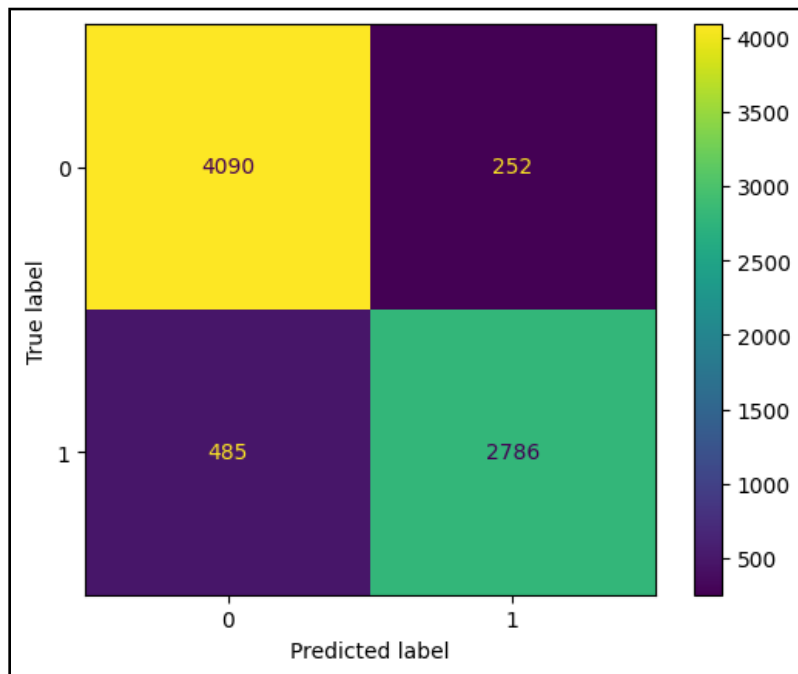
## 8. Results and Conclusion

After running multiple models while playing around with different elements of the model, we have successfully trained a decently accurate model to implement predictions of which tweets in our data are about natural disasters. Our training accuracy is **89.43%**, our validation accuracy is **80.17%**, and kaggle submission accuracy on the test dataset is **79.25%**.

Training Set Accuracy	Validation Set Accuracy	Kaggle Test Set Submission
89.43%	80.17%	79.25%

With a high degree of accuracy, it can assist news and disaster management to allocate resources as they see fit. However, although this data has been picked up in real time from twitter users, the dataset is far too small to accurately determine if a disaster has actually occurred. The data only had about 10,000 tweets, and if the dataset was larger, we would have been able to predict our intended target more accurately.

### Confusion Matrix



*Confusion Matrix of the final model on train + validation data*

### Precision:

Precision is a good measure to calculate when the cost of classifying data as a False Positive is high. For our dataset, classifying too many tweets as disaster tweets can falsely create panic and confusion, while



missing important tweets about actual disasters, if any. Precision is calculated using the following formula:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

For our model, we have a precision of **91.71%**. This is an excellent indicator of the model's ability to precisely classify the target class.

#### **Recall/Sensitivity:**

Recall is calculated when there is a high cost associated with classifying data as False Negatives. In our context, identifying tweets as false negatives has a negative impact on the agencies' ability to quickly identify tweets that could lead them to provide faster and more effective aid to those affected by a disaster. Recall is calculated using the following formula:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

For our model, we have a precision of **85.17%**. Even though it is lower than precision, it is still a good indicator of the model's robustness.

#### **F1 Score**

F1 score is evaluated when a balance is needed between Precision and Recall of the model. F1 Score can be a better measure to use if there is an uneven class distribution (large number of Actual Negatives). Since we have a relatively even class distribution, we do not necessarily need to evaluate the F1 score. However, the model's F1 score is 0.221.

#### **Specificity:**

Specificity is a measure that shows the tendency of a model to designate a record who is class 0 as class 0. A highly sensitive model would mean that there are few false negative results, and thus fewer cases of class 1 are missed. Specificity is calculated using the following formula:

$$\text{Specificity} = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}}$$

For our model, we have a specificity of **94.19%**.

**False Positive Rate** = 1 - Specificity

= 1 - 0.9419

= 0.00580

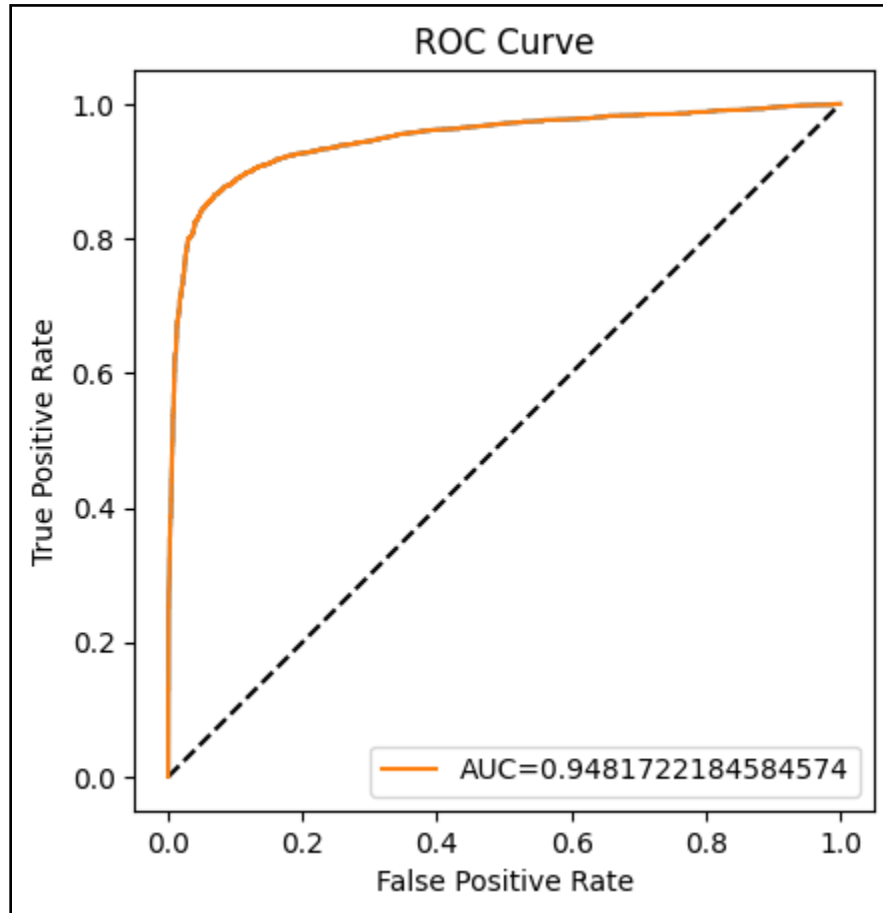
= 5.8%

## AU-ROC Curve

Based on the false positive rate and true positive rate, we can plot the ROC curve. ROC is a curve of probability. An excellent model is expected to have AUC near to the 1, which means that it has a good measure of separability. A poor model on the other hand would have an AUC near 0, which means it has the worst measure of separability. In fact, it would mean that it is reciprocating the result.

Therefore, our model's AUC-ROC curve shows that the AUC value is very close to 1 (0.95). It shows that the model is highly capable of distinguishing between classes accurately.

Ultimately, all metrics point towards a robust model capable of identifying the tweets accurately and efficiently. The model displays a high accuracy even on unseen data and other indicators that point to a useful model for classifying disaster tweets.



*ROC Curve of the model*

## Recommendations

Some of the recommendations for building an even more accurate model would be:

1. Finding or building a larger dataset with a balanced class distribution of both disaster and non-disaster tweets to train the model on.
2. Use of keywords and adding weights to give classification preference to tweets that have the frequently appearing keywords.
3. Changing the learning rates and drop-off rate in order to fine tune the model on newer data, preferably on a larger training dataset.
4. Use of hashtags and mentions to add more context to the tweet which could lead to better classification of tweets and improvement in accuracy.

Overall, this challenge was a great learning experience to learn about the inner workings of neural networks and natural language processing, and we learnt a lot on how the model works on binary classification problems. We also learnt about what makes a model accurate and able to correctly classify data.

## 8. Group contributions

### Vikash Gujju -

I have worked on the data cleaning and data preprocessing steps from scratch. I have tried improving the accuracy score of the base model the team picked up by automating a code to loop through different hyperparameters that would improve the validation and test data accuracy. Additionally, I have helped with team collaboration and setting up regular catch ups with the team members.

### Lohith Maddula -

I collaborated with the team in data preprocessing and building the model through iterative experimentation and analysis, improving the model's accuracy.

### Alekya Ghanta -

I have worked on Exploratory Data Analysis to derive basic insights about the data. I have built various model architectures with different activation functions, batch sizes, epochs, optimizers & tried various regularization techniques like L1 and L2 keras regularizers, dropout, early stopping, learning rate scheduler etc. to see what best suits the model. I have built the learning rate, drop out rate vs accuracy charts to find the best validation accuracy. I also wrote the code to evaluate model performance using Confusion Matrix, ROC Charts etc. I also documented our findings on the best model & EDA in the report.

### Rahul Murugkar -

I worked on standardizing the report, and collaborated with the team on model selection so that we could get the maximum attainable accuracy according to the tenets of the challenge on kaggle. I researched the different models available to us through experimentation so that we could try to become proficient in them. I looked through the workings to see which model gave us the highest accuracy and then proceeded to work on further adding other metrics.

**Jharna Dharaiya -**

I worked on the exploratory data analysis of the data to understand what steps should be taken to preprocess it to make it input ready. I researched about the various pre-processing techniques available to use, especially on vectorization. I collaborated with the team to understand how to tweak the parameters over multiple models to improve the model's accuracy. Lastly, I worked on analyzing the performance metrics of the final model in order to determine its robustness and standardization of the final report.

## 9. Group Learnings

**Vikash Gujju -**

I got to learn how to handle a natural processing language related project end to end. I have understood the process of converting English sentences into words and then assigning numbers that would help us leverage machine learning models on top of them to help us with classification exercises. I have learnt about different NLP libraries like nltk and spacy and how to import them into a machine learning model. I also learned about term document matrix, the concepts of stemming and lemmatization and various layers in the neural networks like GlobalAveragePooling1D, Dense and Dropout and how to use the kaggle platform to participate in competitions.

**Lohith Maddula -**

I learned how to apply several text normalization techniques, such as stemming and lemmatization, as well as how activation functions work. I also learned how changing the learning rates, by adding additional layers and increasing the weights, could improve the model's accuracy.

**Alekya Ghanta -**

Participating in this competition has been a steep learning curve for me. I got to learn about basic text preprocessing in depth like removing Emoji text, Stemming, Lemmatization etc. Even though I had a basic idea about these techniques, I got to know about how these could influence the model in both directions through this competition. I have learned about the concepts of Tokenization & Padding, Neural networks layers like Bidirectional LSTM, GlobalMaxPooling etc even though these could not make it to our final best model. I have learned further about the uses of EarlyCall backs, various Regularization techniques which may or may not help avoiding overfitting. I could see how hyperparameter tuning plays a major role in improving a model's performance. Overall it was a great learning experience for me. I would like to further participate in more Kaggle competitions to develop my knowledge in text processing moving forward.

## Rahul Murugkar -

Working on this challenge was a very insightful look into the workings of natural language processing. I learnt about how agencies respond to natural disasters and where they source their data from for allocating resources. Building, training, and improving upon models greatly enhanced my understanding of the same. I would definitely like to explore different models in the future and look into how I can build better performing models using different methods.

## Jharna Dharaiya -

Actually, working on a natural language processing model has enabled me to get an in-depth look into the steps involved in implementing a robust model. I learnt about how important a role pre-processing plays in ensuring that the model receives an input it can analyze and classify accurately. I got to work on solving the issues with unstandardized text and how different techniques like case changing, lemmatization, stemming, and vectorization can help create a standardized input for an NLP model. I learned about the different methods to standardize input apart from vectorization as well. Finally, I got to experience how actual fine tuning can change the accuracy and performance of a model, and what parameters can affect it. Overall, the project helped me gain a better understanding of such a complex and powerful technique and made me feel more confident in my ability to recreate it to solve real world problems.

## 10. References:

1. [A Gentle Introduction to Pooling Layers for Convolutional Neural Networks](#)
2. [Dropout layer](#)
3. [tf.keras.callbacks.EarlyStopping](#)
4. [Part 5: Step by Step Guide to Master NLP - Word Embedding and Text Vectorization](#)
5. [10+ Examples for Using CountVectorizer](#)