Leveraging Tweets for Rapid Disaster Response Using BERT-BiLSTM-CNN Model

Satya Pranavi Manthena
San Jose State University

Leveraging Tweets for Rapid Disaster Response Using BERT-BiLSTM-CNN Model

A Project

Presented To

The Faculty of Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Satya Pranavi Manthena

May 2023

The Designated Project Committee Approves the Project Titled

Leveraging Tweets for Rapid Disaster Response Using BERT-BiLSTM-CNN Model

by

Satya Pranavi Manthena

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2023

Dr. Robert Chun	Department of Computer Science
Dr. Fabio Di Troia	Department of Computer Science
Mr. Anudeep Varma Datla	Software Engineer, Google

ABSTRACT

Digital networking sites such as Twitter give a global platform for users to discuss and express their own experiences with others. People frequently use social media to share their daily experiences, local news, and activities with others. Many rescue services and agencies frequently monitor this sort of data to identify crises and limit the danger of loss of life. During a natural catastrophe, many tweets are made in reference to the tragedy, making it a hot topic on Twitter. Tweets containing natural disaster phrases but do not discuss the event itself are not informational and should be labeled as non-disaster tweets. Convolutional layers and domain-specific word embeddings are key to traditional tweet categorization models for crisis response. The objective of our research is to evaluate the efficacy of neural networks in categorizing tweets by utilizing both general-purpose and specific to a domain word embeddings to augment their performance. The prior techniques yield a singular embedding of a word extracted from a specific document.. To address the aforementioned issue, this research offers a classification hybrid model based on Bidirectional Encoder Representations from Transformers (BERT), Bidirectional Long Short-Term Memory (BiLSTM), and Convolution Neural Network (CNN) (BERT-BiLSTM-CNN).

Index terms – **bidirectional encoder representations from transformers, bidirectional long short-term memory, convolutional layers, embeddings, neural networks, convolution neural network**

ACKNOWLEDGEMENT

I would like to take this opportunity to offer my sincere gratitude to Dr. Robert Chun, who serves as my project advisor and has been of tremendous assistance to me during the entirety of this project. His tireless pursuit of excellence and unwavering commitment to his profession have been an excellent source of motivation for me, and they have pushed me to see this project through to completion.

My profound gratitude goes out to Dr. Fabio Di Troia and Mr. Anudeep Varma Datla, two members of my defense committee, for the insightful advice they provided regarding my project.

Finally, I would like to express my deep appreciation to everyone who has helped and encouraged me on this journey to graduation.

TABLE OF CONTENTS

CHAPTER 1 .... **1**

Introduction 1

CHAPTER 2 .... **5**

Background 5

A. Convolutional Neural Network 5

B. Bidirectional Long Short-Term Memory 6

C. Bidirectional Encoder Representation from Transformers Embedding 9

CHAPTER 3 .... **12**

Related Work 12

CHAPTER 4 .... **16**

Methodology 16

CHAPTER 5 .... **18**

Project Description 18

A. PROJECT OVERVIEW 18

B. DATA OVERVIEW 18

a. Dataset 1 18

b. Dataset 2 19

CHAPTER 6 .... **21**

Data Analysis 21

A. DATA EXPLORATION: 21

a. Dataset 1 21

b. Dataset 2 25

B. DATA PRE-PROCESSING: 27
a. Dataset 1 27
i. Glove Embedding 28
ii. BERT embedding 29
b. Dataset 2 31
C. DATA MODELLING: 33
I. Support Vector Machine 34
II. Naive Bayes 35
III. Long Short Term Memory (LSTM) 37
IV. Bidirectional Long Short Term Memory (BiLSTM) 40
V. BERT-BiLSTM-CNN (Proposed Model) 43
CHAPTER 7 47
Results 47
CHAPTER 8 51
Conclusion 51
CHAPTER 9 53
Future Work 53
Bibliography 55

LIST OF FIGURES

Figure 1: CNN Model Architecture........ 6

Figure 2: Schematic Diagram of BiLSTM........ 9

Figure 3: Input Format of BERT........ 11

Figure 4: Architecture Diagram of Proposed Model........ 17

Figure 5: Structure of Dataset 1........ 23

Figure 6: Null Value Count of Dataset 1........ 23

Figure 7: Graphical Representation of Target Column........ 24

Figure 8: Top Recurring Words in a Tweet with Target Column 1........ 25

Figure 9: Structure of Data in Dataset 2........ 26

Figure 10: Null Values in Dataset 2........ 26

Figure 11: Number of Tweets per Crisis in Dataset 2........ 27

Figure 12: Number of On-topic and Off-topic Tweets per Crisis in Dataset 2........ 27

Figure 13: Total Number of Tweets per Category in Dataset 2........ 28

Figure 14: Data Cleaning of Dataset 1........ 29

Figure 15: Tokenizing using GloVe Embedding........ 30

Figure 16: Tokenizing using BERT Embedding........ 31

Figure 17: Cleaning Techniques used in Dataset 2........ 33

Figure 18: Visualization of Resampled Data in Dataset 2........ 34

Figure 19: SVM Classifier........ 35

Figure 20: SVM Confusion Matrix........ 36

Figure 21: Naive Bayes Classifier........ 38

Figure 22: Naive Bayes Confusion Matrix........ 38

Figure 23: LSTM Classifier........ 41

Figure 24: LSTM Confusion Matrix........ 41

Figure 25: BiLSTM Classifier........ 43

Figure 26: BiLSTM Confusion Matrix........ 44

Figure 27: Proposed Model Summary........ 46

Figure 28: Proposed Model Accuracy for Dataset 1........ 47

Figure 29: Proposed Model Accuracy for Dataset 2........ 47

Figure 30: Performance Comparison of Models........ 49

Figure 31: Model Accuracy and Model Loss Graphs for Dataset 2........ 51

LIST OF TABLES

Table 1: Description of Dataset 1 Attribute…………19

Table 2: Description of Dataset 2 Attribute…………21

Table 3: Evaluation metrics on dataset 1………… 48

Table 4: Evaluation metrics on dataset 2………… 50

CHAPTER 1

Introduction

During a crisis, Twitter has become a prominent medium for organizations and individuals to broadcast or gather information. With Twitter being utilized as a method of communication on a daily basis, the volume of information available regarding numerous events is overwhelming [1]. They also use Twitter to seek or provide information source in numerous natural or man-made emergencies like earthquakes, floods, wildfires, and nuclear disasters [2,3,4]. For example, following the 2011 earthquake in Japan, 177 million crisis-related tweets were posted in only one day. A haze that blanketed Singapore in 2013 is another example, at which time more than 23 million educational tweets were sent [5]. People-generated tweets can significantly improve situational awareness. Large-scale disaster response groups can utilize these tweets to make better judgments and respond faster. Humanitarian organizations, on the other hand, cannot manually monitor, analyze, and turn vast amounts of data into usable knowledge. Consequently, social media information, specifically from platforms like Twitter, isn't heavily utilized in their methods for dealing with crises. The ability of deep neural networks to learn intricate relationships between inputs and outputs, using a distributed representation of words without relying on feature engineering, has been proven.

Deep learning algorithms, such as Convolutional Neural Networks (CNN), have also surpassed classical ones in numerous Natural Language Processing (NLP) applications, including tweet categorization. Tweet classification for crisis response is a text classification job that determines if a tweet belongs to a certain type of preset informative class. Tweets, unlike big articles, are short-length text with greater challenges owing to their shortness, sparseness (i.e., different word content), velocity (rapid expansion of short text like SMS and Twitter), and misspellings. For these reasons, determining if a person's statements are announcing a tragedy is extremely difficult. Much earlier research has been undertaken utilizing methods such as word embedding approaches such as Continuous Bag of Words (CBOW) and the Skip-gram model [6]. In the case of a disaster, this can be different; for example, the context of the words in a Twitter message may be a factor in categorization. For instance, consider the following tweet: "*#oldBand amazing performance*! *light*, *color*, *fire on stage*! *lots of people and huge chaos*!" describes a person's experience at a concert, and we may conclude that he appreciated it because of the adjective "*amazing*". Even though it incorporates the term "*f ire*" it does not refer to any danger or urgency; rather, it is used to describe the bright stage decor. Assume another tweet like this: "*California Hwy*. 20 *closed in both directions due to Lake Country f ire*" The term "*fire*"signifies calamity in this context, and the tweet portrays an emergency. The two instances demonstrate how a single word may have numerous meanings depending on its context. As a result, knowing the context of words is critical for determining the meaning of a tweet. To address this issue, Devlin et al. [7] presented Bidirectional Encoder Representations from Transformers (BERT), a contextual embedding learning approach that offers embeddings depending on the context terms of the word. The BERT model outperforms classic embedding learning models in many NLP tasks such as text categorization, text summarization, and entity recognition[8,9]. However, it will be fascinating to see how contextual embeddings might aid in the comprehension of disaster-related text. As per the assertions made, the BiLSTM architecture possesses the capability to apprehend contextual information by means of a bidirectional mechanism, whereas the TextCNN architecture is proficient in extracting significant features for tasks pertaining to text classification. The combination of these two models, in conjunction with an additional module, has the potential to substantially enhance the precision of multi-label classification.

So, in this research, we want to examine the crisis prediction problem from Twitter data utilizing contextual embeddings (BERT) with BiLSTM and Text CNN. As far as I am aware, no previous research in the area of tweet classification has utilized a combination of BERT embedding, Bidirectional Long Short-Term Memory (BiLSTM), and Text Convolutional Neural Network (CNN), making this study unique in its approach.

In the first part of this report, an overview of the various deep neural networks and word embeddings that are used for tweet classification is provided. In the following section, a summary of the previous research that was conducted on Bidirectional Encoder Representations from Transformers (BERT), which is a model for word embedding is given. The next section discusses related work of deep neural networks in contextual tweet classification, namely Bidirectional Long Short-Term Memory (BiLSTM), and Text Convolution Neural Network (TextCNN), and the findings gained using these models. In the next section the Methodology of the research is explained. In Chapter 5 we discuss the project overview with an architecture diagram and also an overview of the data used in experiments. Chapter 6 contains the Data analysis which includes data exploration, pre-processing, and data modeling. In the next sections we talk about the results, conclusion and the possible future work of the research. assistance to others. In recent years, Twitter has also shown to be a significant

CHAPTER 2

Background

A. Convolutional Neural Network

The CNN, a deep learning framework, comprises multiple layers, such as the input and output layers. Convolutional neural networks (CNNs) have been employed in natural language processing (NLP) tasks to encode textual data as a collection of matrices. This is achieved by utilizing pre-trained word embeddings, which facilitate efficient learning. CNNs frequently utilize token sequences as their input, where the CNN filters function as n-grams over contiguous representations. Subsequently, the filters are combined through the incorporation of dense layers. Convolutional neural networks possess the ability to acquire and recognize features autonomously without human intervention, thereby obviating the necessity for prior knowledge or manually designed features.

In contrast to models such as MLP, CNNs have the ability to reduce the number of parameters and mitigate the issue of vanishing or exploding gradients during training. Moreover, Convolutional Neural Networks (CNNs) exhibit weight sharing across all convolutional layers, which results in expedited computation and decreased memory consumption.

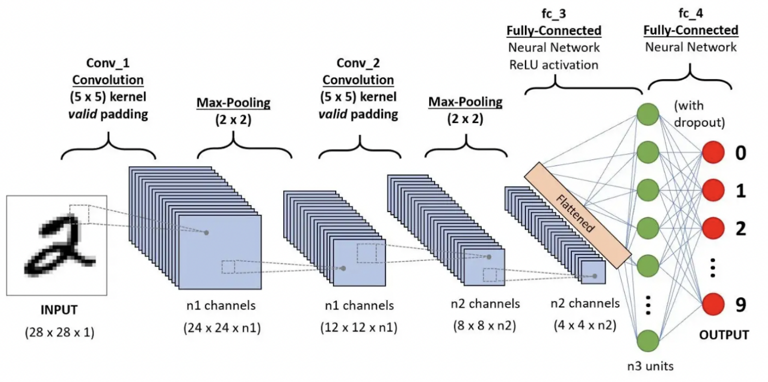


Fig. 1. CNN model architecture

B. Bidirectional Long Short-Term Memory

The recurrent neural network (RNN) architecture is frequently employed for the purpose of evaluating time series data, as it features a feedback loop that enables the optimization of past information utilization. The Recurrent Neural Network (RNN) exhibits certain limitations with respect to its memory and information storage capabilities, leading to challenges in acquiring knowledge of long-term dependencies and the emergence of gradients that tend to diminish. The Long Short-Term Memory (LSTM) model was devised as a remedy for these shortcomings. The Long Short-Term Memory (LSTM) structure comprises memory cells that possess the ability to recollect past data and are regulated by gate mechanisms, namely the input gate, forget gate, and output gate. The modulation of memory cells is accomplished by means of pointwise multiplication and sigmoid function operations within each gate. This process entails the utilization of input data derived from the current state, as well as output from the hidden state of the preceding layer. The forget gate is responsible for deciding the relevance of information and its subsequent retention or discarding. The output value of the forget gate ranges between zero and one, where a value closer to zero signifies the information's forgetfulness, and a value closer to one indicates its retention. The forget gate is determined using the following formula:

$$ft = 0(Wf.\ [ht1, xt] + bf) \qquad (1)$$

The gate module comprises a weight (W) and bias (b) parameters and employs the sigmoid activation function denoted by 0. The function accepts the present input (xt) and the preceding hidden state (ht1). The sigmoidal function plays a crucial role in determining the relevance of information by discretizing input values into binary outputs of either zero or one. The former signifies insignificance while the latter denotes significance. The input gate is written as follows:

$$it = \sigma\,(Wi\ .\ [ht-1, xt] + bi) \qquad (2)$$

Figure 2 depicts a simplified representation of a bidirectional Long Short-Term Memory (LSTM) model. The hyperbolic tangent function (tanh) performs computations on both the current input (xt) and the preceding hidden state (ht1). Following this, the cellular state (Ct) is computed utilizing the aforementioned data, and the revised outcome is retained as the novel cellular state.

$$C^{t} = tanh(Wc.\ [ht-1, xt] + bc) \qquad (3)$$

$$Ct = ft \odot Ct{-1} + it \odot C^{t} \tag{4}$$

The Tanh activation function, which is hyperbolic in nature, is employed in conjunction with the dot product operation to compute the updated memory cell, denoted as Ct. The selection of the subsequent hidden state is determined by the output gate, which subsequently propagates both the Ct and ht to the following time step.

$$ot = \sigma(Wo. [ht1, pt] + bo) \tag{5}$$

$$ht = ot \odot tanh(ct) \tag{6}$$

The conventional Long Short-Term Memory (LSTM) model exhibits limitations in terms of unidirectional data analysis and dependence on prior information. The BiLSTM architecture is composed of two LSTM layers that are situated in the forward and backward directions, respectively. This allows the model to effectively incorporate information from both preceding and succeeding data. The forward Long Short-Term Memory (LSTM) neural network component is designed to capture information from the past context of the input sequence, while the reverse LSTM module is intended to retrieve data from the future context of the input sequence.

$$ht = \rightarrow ht \oplus \leftarrow ht \tag{7}$$

The output that ensues is acquired through the execution of component summation, which is denoted by the symbol ⊕, on the outputs of the two hidden layers. The BiLSTM model demonstrates enhanced efficacy in contrast to the LSTM and RNN models, due to its ability to exploit both preceding and succeeding information. The second figure presents a schematic illustration of the Bidirectional Long Short-Term Memory (BiLSTM) model.

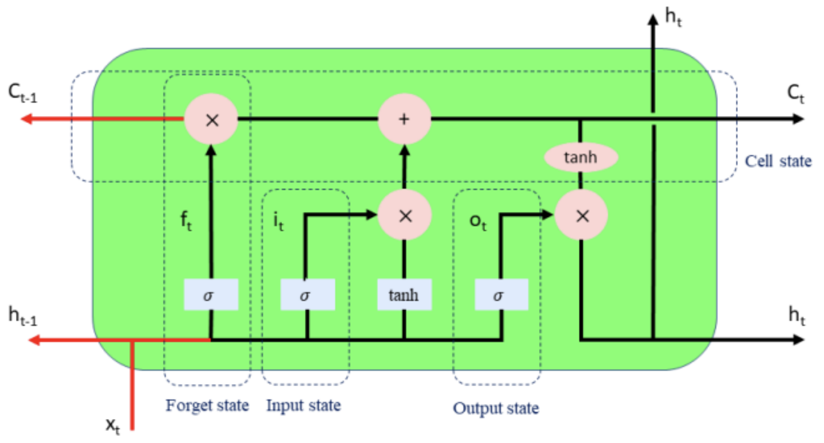


Fig.2 . schematic diagram of bidirectional LSTM

C.Bidirectional Encoder Representation from Transformers Embedding

The BERT embeddings, as described in reference [12], produce a multitude of vectors for a given word across various contexts, thereby distinguishing them from other varieties of word embeddings. Recent developments in the field of natural language processing (NLP) have demonstrated that BERT models outperform conventional embeddings in a range of NLP tasks, including entity recognition and sentence prediction. The process of generating a vector representation is facilitated by BERT through the utilization of a transformer that possesses the ability to scrutinize a complete sequence and make predictions regarding the masked word. In order to produce BERT embedding, it is necessary to first tokenize the text data using the pre-existing BERT tokenizer. This process facilitates the incorporation of the distinctive CLS token at the outset of the text. Following the process of tokenization, the pre-existing BERT model is capable of effectively analyzing textual data, resulting in the generation of feature vectors for the purpose of embedding. Subsequently, the embeddings are classified by the neural network layers. The authors of BERT utilize distinct specialized tokens for the purpose of fine-tuning and training that is specific to a particular task. The following are the special tokens used by BERT writers for fine-tuning and particular task training:

I. CLS: Every sequence's first token. For classification tasks, a classification token is typically used in combination with a softmax layer. It may be safely ignored for anything else.

II. SEP: The "SEP" token serves as a delimiter for sequences during the training of models for tasks that involve pairs of sequences, such as the prediction of the subsequent sentence. The utilization of said token is deemed necessary for tasks involving sequence pairs. However, in cases where a solitary sequence is employed, the token may be appended to the end of said sequence.

III. MASK: A token that denotes a symbolic representation employed to signify hidden elements, exclusively utilized in pre-training activities.

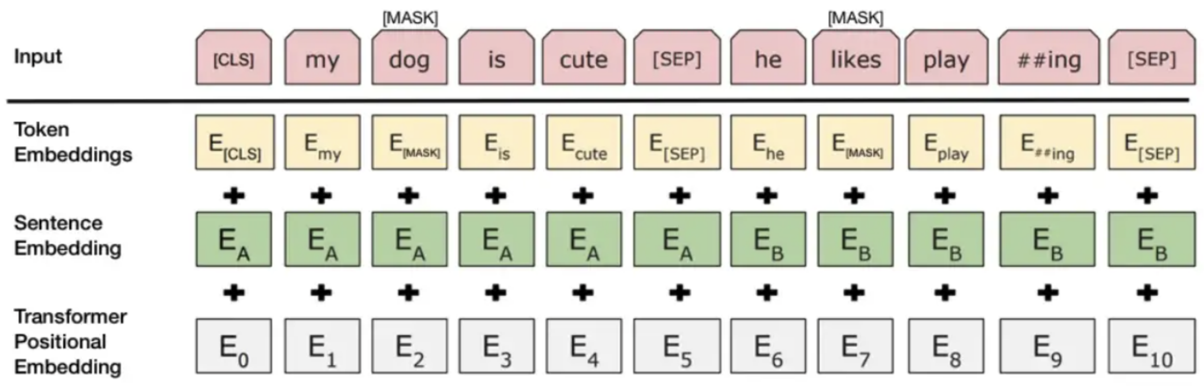


Fig.3 . Input format of BERT

The initial stratum of BERT comprises a series of tokens, encompassing tokens of a distinctive nature. The utilization of the "##ing" token depicted in Figure 3 could potentially result in ambiguity. However, it is noteworthy that BERT utilizes WordPiece [13] tokenization methodology, which dissects words such as "playing" into "play" and "##ing." The objective of this approach is to incorporate a broader range of Out-Of-Vocabulary (OOV) terms.

I. Token embeddings refer to the unique identification numbers assigned to individual tokens within a given vocabulary.

II. Sentence embeddings are a numerical classification method utilized to distinguish between two given sentences, A and B.

III. The positional embeddings of the Transformer model fulfill the function of denoting the precise position of individual words in the provided sequence. Additional information pertaining to this subject matter can be found in citation [14].

CHAPTER 3

Related Work

Twitter has grown in popularity as a platform for collecting various forms of information and employing it as a data source for various purposes. A prior investigation centered on the identification of beneficial tweets pertaining to calamities and the exclusion of extraneous ones through the utilization of a Convolutional Neural Network (CNN) framework. The tweets were classified into two distinct categories, informative and uninformative, and the performance of the CNN model was evaluated against two other supervised learning algorithms, namely SVM and ANN. The present study endeavors to enhance the precision of the Convolutional Neural Network (CNN) algorithm in classifying informative tweets in the context of a crisis. In an independent investigation, scholars analyzed the utilization of Twitter data for the purpose of earthquake detection. The researchers employed supervised learning methodologies, including Support Vector Machine (SVM), Random Forest, and Decision Tree, to analyze a corpus of Twitter posts related to earthquakes in the Indonesian language. The pre-processing stage involved the implementation of various procedures such as tokenization, stop word removal, and normalization. The recall evaluation is used in the study to evaluate which of the suggested models will be better at detecting earthquakes. As a consequence, Random Forest outperformed SVM and Decision Tree. Another study utilized tweet data to categorize important messages in catastrophes [17]. The study's goal was to be able to distinguish between tweets that may be valuable and offer facts about the individuals impacted, catastrophe areas, and etc, and those that are not useful for a disaster.

A proposal was put forth to employ a hybrid approach involving a Convolutional Neural Network (CNN) and Word2Vec model for the purpose of categorizing tweets. In a recent study, Madichetty et al. [18] employed the contextual representation technique of ELMo embedding to categorize tweets related to disasters. The researchers discovered that the deep learning algorithm exhibited notable proficiency in categorization assignments, particularly when incorporating word embeddings. The developed model integrated ELMo embedding with two dense classifiers that employed the ReLU activation function and the SoftMax function. In order to evaluate the efficacy of the model, a comparative analysis was conducted with three other baseline models, namely SVM utilizing Bag-of-words, CNN utilizing crisis word embedding, and MLP-CNN.

Several scholars have investigated the possibility of enhancing emotional analysis in tweet classification by integrating multiple neural networks such as CNN, SVM, and ANN. The ABCDM model, proposed by Basiri Jiang et al. [19], utilizes a bidirectional CNN-RNN architecture with attention mechanisms as a means of achieving its objectives. The ABCDM methodology employs a methodology that entails the utilization of two distinct BiLSTM and GRU layers. This approach enables the analysis of temporal information flow in both past and future directions, thereby facilitating the extraction of pertinent contexts. Furthermore, the results obtained from the bidirectional layer of ABCDM are subjected to an attention mechanism in order to highlight particular words.

Rehman et al. [20] developed an LSTM-depth CNN model. The postscript was embedded using the convolution-produced feature set and a global maximum pool layer. This Word2Vec-trained layer employed long-term dependency. Dropout, normalization, and correcting linear units improved model correctness.

Zhang et al. [21] proposed a novel model, CNN-LSTM, which integrates CNN and LSTM techniques. The researchers performed a comparative evaluation between the proposed model and E-LSTM, that utilizes 3D convolution in lieu of 2D convolution. The authors subsequently presented BiE-LSTM, a model that utilizes bidirectional embedding of long short-term memory. Minaee, Azimi, and Abdolrashidi [22] have presented a sentiment analysis approach that leverages both CNN and BiLSTM models. One approach is employed to capture time-related details, whereas the other method extracts the local structure from the given data. The utilization of this approach has been observed to be comparatively more efficacious than relying solely on the CNN-LSTM model. Li et al. [23] suggested a hierarchical attention BiLSTM model for multimodal sentiment analysis based on cognitive brain (ALCB). Finn, Abbeel, and Levine [24] suggested a sentiment classification approach based on MAML and BiLSTM that uses gradient descent to update parameters. The aspect sentiment data set exhibited superior performance compared to commonly employed models, with 1.68% in accuracy, 2.86% in recall, and 2.27% in F1 value.

A method for analyzing sentiment trends in Chinese texts was proposed by Gan, Feng, and Zhang [25]. Their strategy incorporates a convolutional neural network (CNN)-based recurrent soft thresholding model (BiLSTM) with an attention mechanism based on a modular dilated joint architecture. Enhancing the representation of the term vector can lead to an improvement in the classification performance of the model. The suggested approach for text classification employs deep learning techniques, thereby obviating the requirement for manual feature engineering. Additionally, it can be trained on more extensive datasets to enhance its learning capacity.

Numerous research teams have employed Twitter data to enhance their comprehension of emergency scenarios and forecast calamities. One cohort directed their attention towards the clustering of textual data for the purpose of identifying tweets that pertain to catastrophic events. Meanwhile, another cohort employed text mining and statistical methodologies to scrutinize crises. Various machine learning models have been utilized to analyze Twitter data for the purpose of predicting emergencies. This is achieved by representing tweet phrases as embeddings. Palshikar et al. developed a model with inadequate supervision utilizing a bag of words (BOW) approach, whereas other scholars utilized frequency-based word representation and Markov-based models to forecast disasters. A novel pre-processing approach has been devised for conducting sentiment analysis of tweets using BERT. It would be intriguing to investigate the efficacy of the model on various word embeddings to ascertain the extent to which contextual words can enhance the accuracy of tweet classification as a disaster.

CHAPTER 4

Methodology

This research implements a machine learning model for classification of tweets with the purpose of sorting tweets into those that include actual catastrophe information and those that do not. We use a combination of Bidirectional LSTM and CNN architectures along with the pre-trained BERT model. The model is constructed using TensorFlow and Transformers libraries. It first loads the tweet data from a CSV file using Pandas and then sets up the input and output shape of the model. BERT embeddings are obtained by passing the input tokens to the pre-trained BERT model. A bi-LSTM layer is then set up to learn the temporal relationships between the input tokens, and a CNN layer is set up to learn the local patterns between the output of bi-LSTM. The result of the BiLSTM is combined with the output of the CNN once more in order to merge the features that were extracted by both layers. The CNN layer can identify local patterns in the data and the contextual information can be captured by the BiLSTM layer. The model is able to capture both kinds of information and use them to create predictions when the output of the two layers are concatenated. By merging the features extracted by both layers, the model is able to learn a more comprehensive representation of the input data, which has the potential to lead to increased performance. This improvement is due to the fact that the model is able to learn a more complete representation of the data. The architecture diagram in figure 4 gives a visual representation of the methodology of the research.

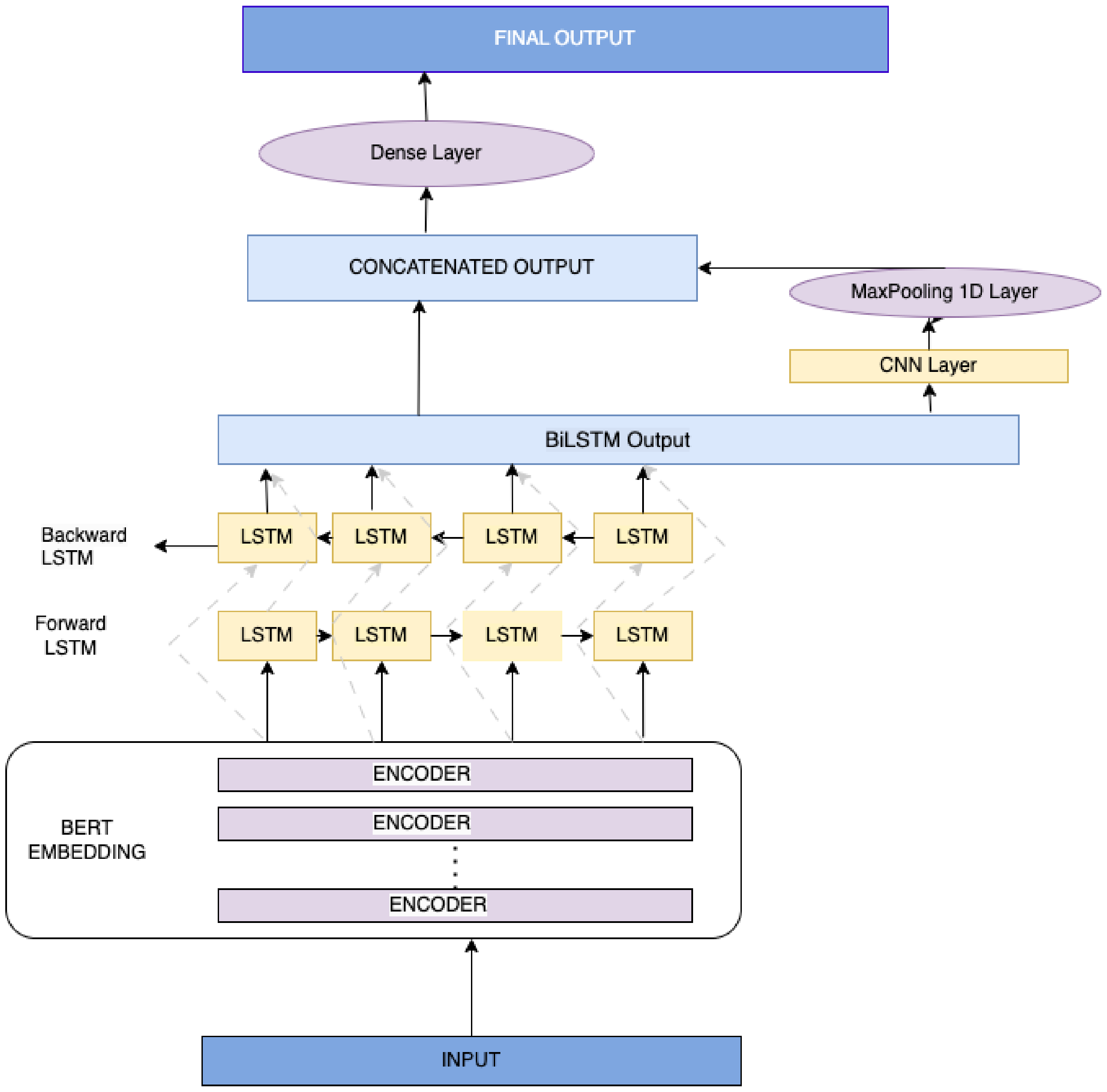


Fig.4 *.Architecture diagram of the proposed model*

CHAPTER 5

Project Description

A. PROJECT OVERVIEW

Several phases will be required to fulfill the project's core aim of developing a classification model that can distinguish between legitimate and tweets containing words that can mean a disaster but are actually not. The initial step of the project will involve conducting an exploratory analysis of data (EDA) and data visualization on the selected datasets. In addition, the implementation of data pre-processing techniques will be carried out to eliminate inaccuracies, address instances of missing data, and improve the overall quality of the dataset prior to the commencement of the data modeling stage. Upon development of the model utilizing the enhanced dataset, its efficacy will be evaluated utilizing suitable metrics for assessment. The execution of all tasks pertaining to data and model will be carried out through the utilization of Google Colab.

B. DATA OVERVIEW

a. Dataset 1

The dataset was procured from Kaggle, a renowned website and virtual community catering to machine learning experts. The subject matter of my research pertains to the challenge titled "Natural Language Processing with Disaster Tweets," which was disseminated via an online platform. The task at hand pertains to two distinct sets of data comprising a cumulative count of 10,876 tweets. The first dataset, referred to as train.csv, comprises 7,613 rows, while the second dataset, referred to as test.csv, comprises 3,262 rows. In order to evaluate the efficacy of my model, I shall exclusively employ the training dataset, partitioning it into distinct subsets for both training and testing purposes.

Dataset Source: https://www.kaggle.com/c/nlp-getting-started/data

The dataset named Train.csv comprises five parameters, as illustrated in the following table 1:

No	**Attribute**	**Attribute Description**	**Format**
1	id	Each tweet is assigned a unique identifier.	double
2	keyword	One specific term extracted from the content of the tweet.	chr
3	location	The geographical origin of the tweet.	chr
4	text	Tweet Content (text)	chr
5	target	The intended result, denoting whether a tweet pertains to an actual calamity (1) or lacks such relevance (0).	double

Table 1: Description of Dataset 1 Attribute

The aforementioned dataset is utilized for four distinct types of data models, each of which is compared to the model that is being proposed for the purpose of evaluating the metrics.

b. Dataset 2

In most cases, the occurrence of natural disasters would cause a big and widespread reaction in the various social media channels. Users will often share their thoughts and activities taken prior to the storm, during the storm, and after the storm. The categorized crisis related tweets collection may be found on CrisisLex.org, which is a repository of data relating to crisis-related social media. We used this collection. We used the CrisisLexT6 dataset, which includes Tweets from six crises that have been labeled according to their degree of relatedness. These crises are as follows: the Hurricane Sandy in 2012, the Oklahoma Tornado Season in 2013, the Explosion in West Texas in 2013, the Floods in Alberta in 2013, the Bombings in Boston in 2013, and the Floods in Queensland in 2013. Around 60,000 tweets were sent on Twitter during six different crisis situations in 2012 and 2013, and those 60,000 tweets were divided into collections of 10,000 tweets each by crowdsourcing employees and subsequently, the items were classified based on their degree of relevance, either as "off-topic" or "on-topic". In order to conduct this research, the data collected from those emergency situations were utilized.

Dataset Source: https://crisislex.org/data-collections.html

The dataset named combined.csv comprises four parameters, as illustrated in table 2:

No	Attribute	Attribute Description	Format
1	tweet id	Each tweet is assigned a unique identifier.	double
2	tweet	Tweet Content (text)	chr
3	label	The desired result, whether or not a tweet is about a true calamity. (on-topic/off-topic)	chr
4	category	The kind of disaster (floods,hurricane,etc)	chr

Table 2: Description of Dataset 2 Attribute

The above dataset is only used on the proposed dataset to evaluate the model's accuracy on a large dataset.

CHAPTER 6

Data Analysis

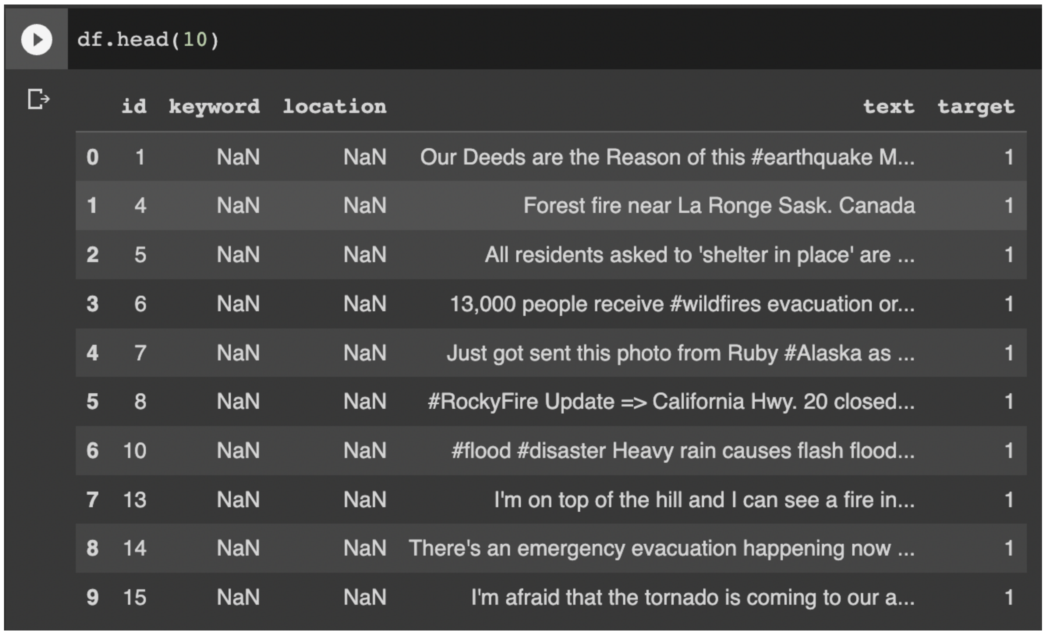
To get started, it is essential to gain an understanding of the dataset that we will be using and to investigate it. This stage is extremely important because it enables us to identify particular characteristics and qualities that are unique to our dataset. Consequently, this step carries a lot of weight. For example, we need to evaluate every one of the five qualities separately, taking into consideration their format, category, frequency of presence, support for missing values, and any other relevant variables. In addition, exploratory data analysis (EDA) will involve charting variables to acquire a better knowledge of their unique properties, such as specific keywords or locations, evaluating the mood or tone of the text, and providing a visual depiction of the words that appear the most frequently in the text.

A. DATA EXPLORATION:

a. Dataset 1

To begin, we will begin by loading all of the necessary libraries into Google Colab in order to carry out exploratory data analysis (EDA), visualization, data pre-processing, data modeling, and assessment. In addition, we will load the working data set.

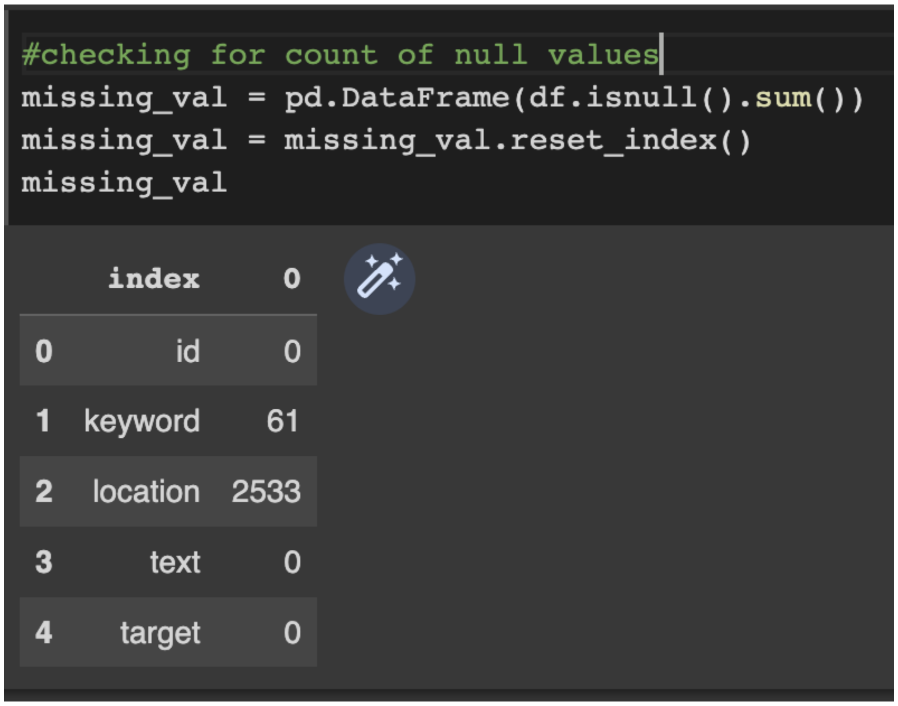
After that, the structure of the data set, as well as its summary and its null value, were discovered:



```
df.head(10)
```

```
   id  keyword  location                                         text  target
0   1      NaN       NaN  Our Deeds are the Reason of this #earthquake M...       1
1   4      NaN       NaN             Forest fire near La Ronge Sask. Canada       1
2   5      NaN       NaN       All residents asked to 'shelter in place' are ...       1
3   6      NaN       NaN       13,000 people receive #wildfires evacuation or...       1
4   7      NaN       NaN       Just got sent this photo from Ruby #Alaska as ...       1
5   8      NaN       NaN       #RockyFire Update => California Hwy. 20 closed...       1
6  10      NaN       NaN       #flood #disaster Heavy rain causes flash flood...       1
7  13      NaN       NaN            I'm on top of the hill and I can see a fire in...       1
8  14      NaN       NaN  There's an emergency evacuation happening now ...       1
9  15      NaN       NaN       I'm afraid that the tornado is coming to our a...       1
```

Fig.5 .*Structure of Dataset 1*



```
#checking for count of null values
missing_val = pd.DataFrame(df.isnull().sum())
missing_val = missing_val.reset_index()
missing_val
```

```
      index     0
0        id     0
1   keyword    61
2  location  2533
3      text     0
4    target     0
```

Fig.6 .*Null value count of dataset 1*

The dataset includes a number of factors, each of which can be used to characterize a single tweet. The first variable is denoted by the letter "id," which refers to a number that uniquely identifies each tweet. The second variable is referred to as "Keyword," and it is a character variable that denotes a particular word or phrase that is seen in every tweet. The third variable is called "location," and like the first two variables, it is a character variable that shows the location from where each tweet was posted. The fourth element, "text," is the most crucial one because it contains the primary content of each tweet. It is also the variable that comes first in the list. The last variable, "target," is an integer that takes the values 1 or 0, depending on the situation. A tweet with a value of 1 indicates that it is referring to a genuine disaster, while a tweet with a value of 0 suggests that it is a hoax tweet about a disaster. It is important to take note that the variables id, text, and target do not have any null values. On the other hand, the variable "Keyword" contains 61 missing values, and the variable "location" contains 2533 null values, as shown in Figure 6. The identification of missing data in the dataset might be aided by checking for null values.

The “target” variable is represented graphically, as demonstrated in the pie chart in figure 7.

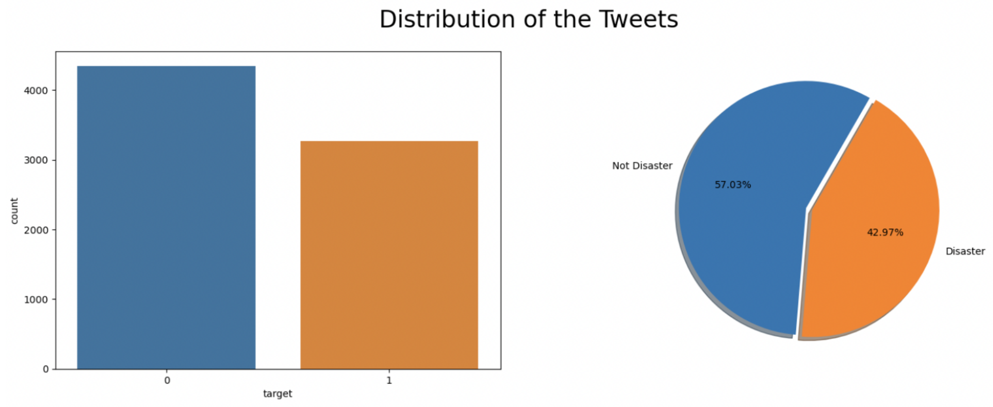


Fig.7 .*Graphical representation of target column*

It has been noticed that the data set has a greater number of records with the "0" label, which relates to phony tweets about disasters, than the number of entries with the "1" label, with the former accounting for 57% of the overall data set. The sampling of an equal number of tweets for both labels is going to be one of the potential solutions that are put to the test. The goal is to create a balanced subset that contains fifty percent of each target variable.

We analyzed the terms that appeared the most frequently for each of the target variables, as indicated in the figure 8. The top recurring words seem to be fire, storm, US, death, flood, like, suicide bomber, etc. They are useful for analyzing the content of Twitter, determining the perspectives of users on a particular issue, and putting an end to hate speech.

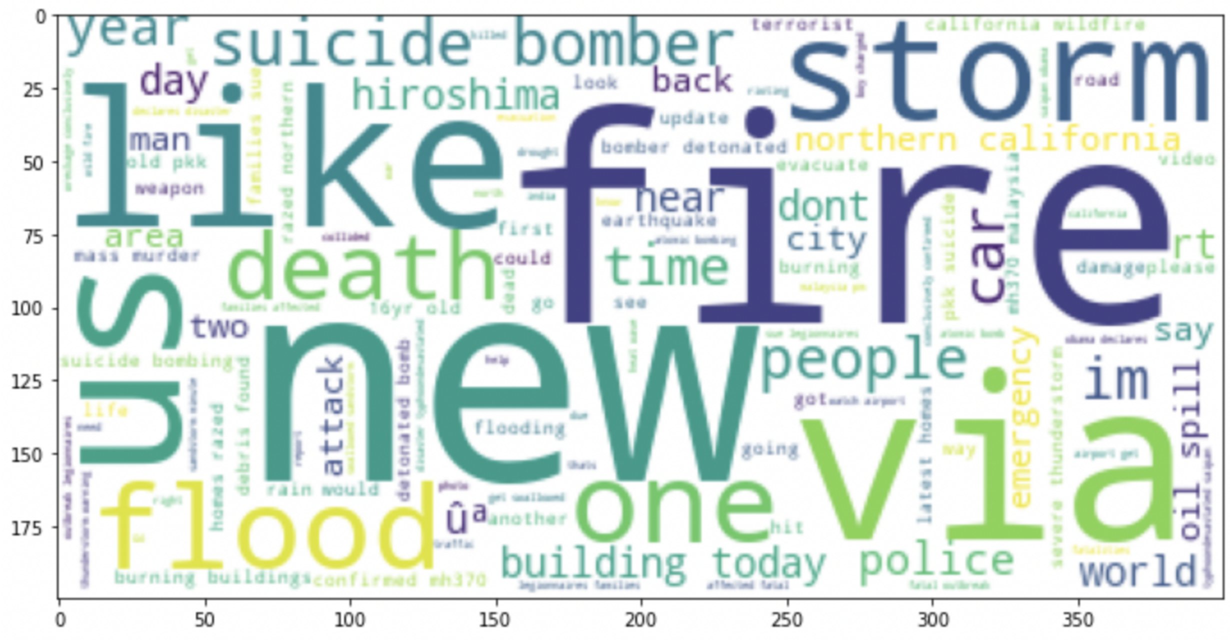
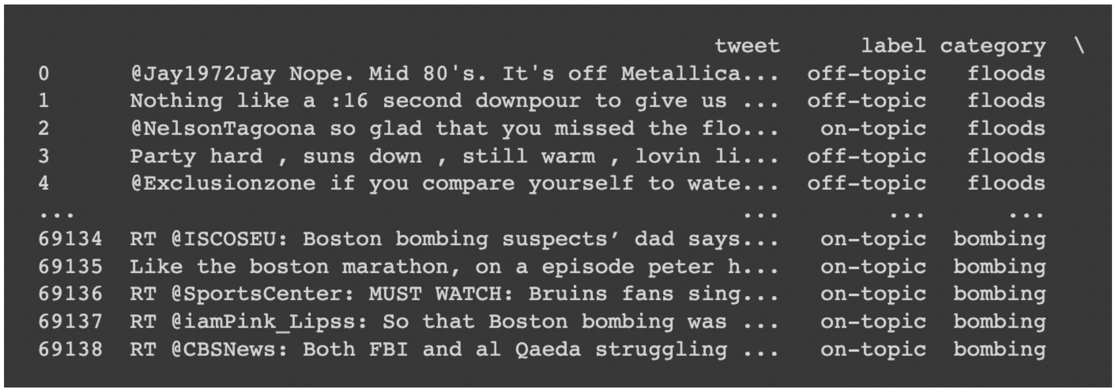


Fig.8 .*Top recurring words in a tweet with target value 1*

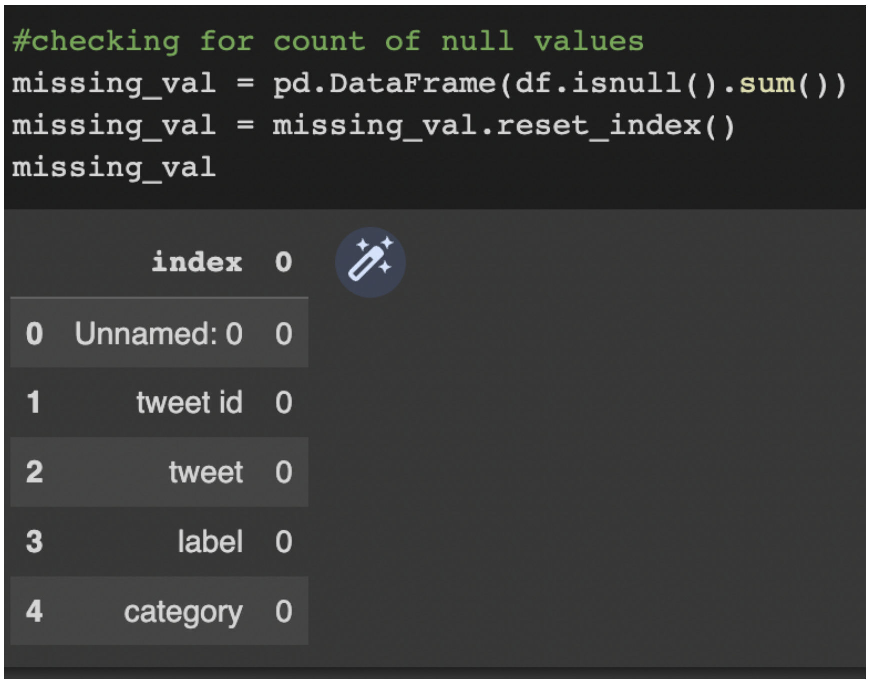
b. Dataset 2

After loading the dataset into Google Colab we analyze the structure of the data and any null values it contains as seen in figures 9 and 10 respectively. The data does not contain any null values in its fields.



```
                                                   tweet      label category  \
0      @Jay1972Jay Nope. Mid 80's. It's off Metallica...  off-topic   floods
1      Nothing like a :16 second downpour to give us ...  off-topic   floods
2      @NelsonTagoona so glad that you missed the flo...   on-topic   floods
3      Party hard , suns down , still warm , lovin li...  off-topic   floods
4      @Exclusionzone if you compare yourself to wate...  off-topic   floods
...                                                  ...        ...      ...
69134  RT @ISCOSEU: Boston bombing suspects' dad says...   on-topic  bombing
69135  Like the boston marathon, on a episode peter h...   on-topic  bombing
69136  RT @SportsCenter: MUST WATCH: Bruins fans sing...   on-topic  bombing
69137  RT @iamPink_Lipss: So that Boston bombing was ...   on-topic  bombing
69138  RT @CBSNews: Both FBI and al Qaeda struggling ...   on-topic  bombing
```

Fig.9 *.Structure of data in dataset 2*

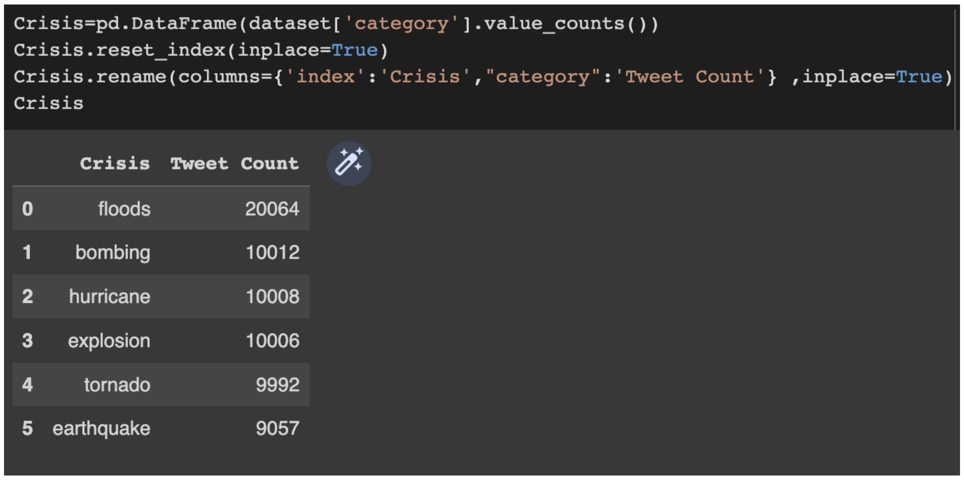


```
#checking for count of null values
missing_val = pd.DataFrame(df.isnull().sum())
missing_val = missing_val.reset_index()
missing_val
```

```
        index  0
0  Unnamed: 0  0
1    tweet id  0
2       tweet  0
3       label  0
4    category  0
```

Fig.10 *.Checking for null values in dataset 2*

We then analyze the number of tweets for each crisis to check if the data is balanced or not. We now have to check the number of tweets that are related to the crisis and how many tweets are out of context.



```
Crisis=pd.DataFrame(dataset['category'].value_counts())
Crisis.reset_index(inplace=True)
Crisis.rename(columns={'index':'Crisis',"category":'Tweet Count'} ,inplace=True)
Crisis
```

	Crisis	Tweet Count
0	floods	20064
1	bombing	10012
2	hurricane	10008
3	explosion	10006
4	tornado	9992
5	earthquake	9057

Fig.11 .*Number of tweets per crisis in dataset 2*

As each category has its own off-topic tweets, the overall number of off-topic tweets from all categories would be far larger than their on-topic tweets. This would skew our database. So, we Plot the total amount of off-topic tweets and on-topic tweets to check if the data is unbalanced.

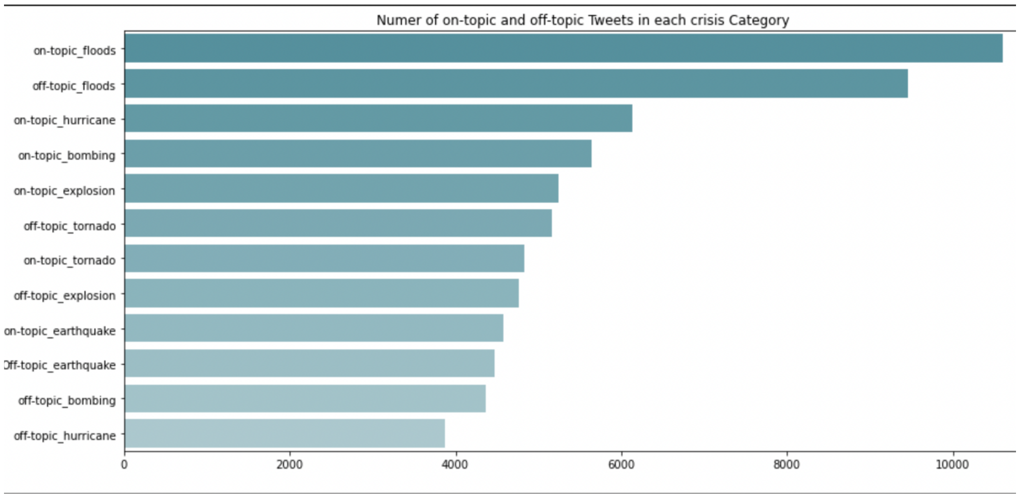


Fig.12 .*Number of on-topic and off-topic tweets per crisis in dataset 2*

As "off-topic" is one of our prediction categories, it should have about as many tweets as the others. So, we label these tweets that are not related as "unrelated" irrespective of crisis.

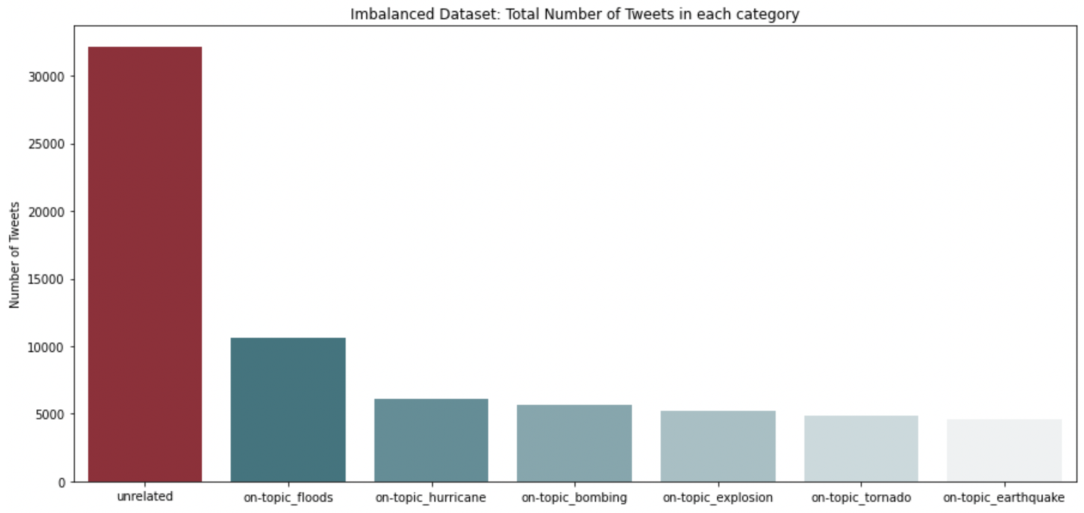


Fig.13 .*Total number of tweets per category in dataset 2*

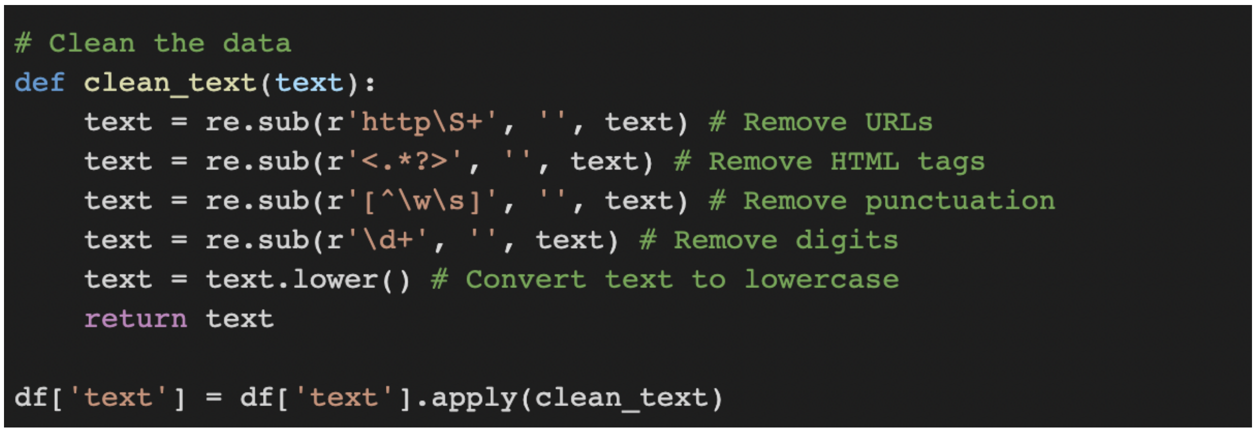
As we can see from figure 13 that the number of unrelated tweets are way higher than the actual on-topic tweets, so we should resample the data after cleaning it.

B. DATA PRE-PROCESSING:

a. Dataset 1

The most interesting variable in the data set is called "text," and it refers to the text of the tweets itself. We are going to investigate this variable further and pre-process it so that it is ready to be used in the modeling stage. The file "stop words english.txt" is imported; it is a text file that contains a large number of stop words (854 words), all of which need to be eliminated from the tweets.

In this step, we delete symbols, change the case of the text to lowercase, remove numerals, stop words, stop words from the imported text file, punctuation, and white spaces as seen in figure 9.



```
# Clean the data
def clean_text(text):
    text = re.sub(r'http\S+', '', text) # Remove URLs
    text = re.sub(r'<.*?>', '', text) # Remove HTML tags
    text = re.sub(r'[^\w\s]', '', text) # Remove punctuation
    text = re.sub(r'\d+', '', text) # Remove digits
    text = text.lower() # Convert text to lowercase
    return text

df['text'] = df['text'].apply(clean_text)
```

Fig.14. *Data cleaning of dataset 1*

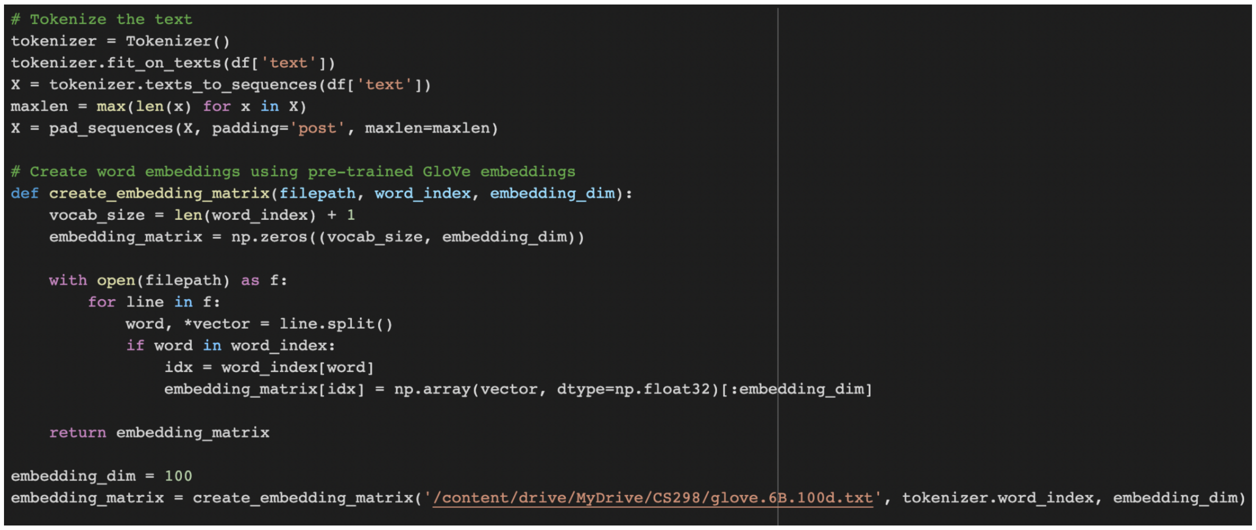
i. Glove Embedding

GloVe embeds words as vectors of real numbers. GloVe analyzes word co-occurrence data in a huge corpus of literature to learn these vector representations.

GloVe assumes that words that co-occur with other words in similar circumstances have comparable meanings. GloVe can develop semantic and syntactic vector representations for words by training on a large corpus of text and optimizing a cost function that captures these co-occurrence patterns.

Sentiment analysis, machine translation, and named entity recognition can leverage GloVe embeddings. GloVe embeddings are pre-trained on a huge corpus of text and may be utilized without further training.

Glove embeddings were used as a tokenization method with the LSTM and BiLSTM models.



```
# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['text'])
X = tokenizer.texts_to_sequences(df['text'])
maxlen = max(len(x) for x in X)
X = pad_sequences(X, padding='post', maxlen=maxlen)

# Create word embeddings using pre-trained GloVe embeddings
def create_embedding_matrix(filepath, word_index, embedding_dim):
    vocab_size = len(word_index) + 1
    embedding_matrix = np.zeros((vocab_size, embedding_dim))

    with open(filepath) as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix[idx] = np.array(vector, dtype=np.float32)[:embedding_dim]

    return embedding_matrix

embedding_dim = 100
embedding_matrix = create_embedding_matrix('/content/drive/MyDrive/CS298/glove.6B.100d.txt', tokenizer.word_index, embedding_dim)
```

Fig.15. *Tokenizing using Glove*

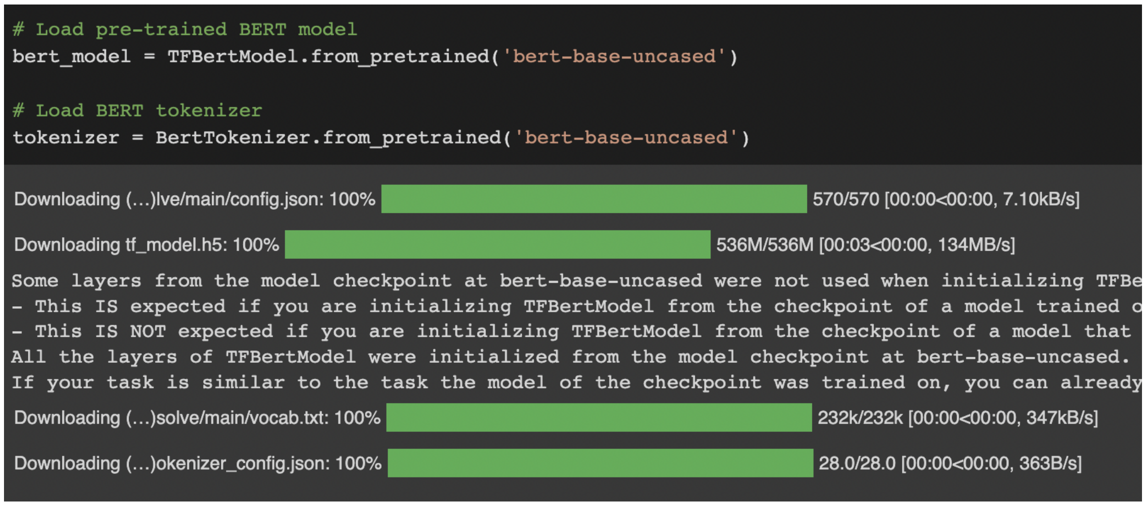
ii. BERT embedding

BERT embeddings were used as the tokenization method in the proposed method BERT-BiLSTM-CNN.

BERT language models can create embeddings for words, phrases, and documents. These embeddings capture the contextual meaning of words by considering the words before and following them in a phrase or document, making them valuable in natural language processing.

Conventional word embeddings like word2vec and GloVe use co-occurrence patterns to represent each word as a fixed-length vector. These embeddings don't consider the word's context, which might cause natural language processing errors.

Using a deep neural network trained on a huge corpus of text, BERT embeddings identify missing words in sentences. BERT learns to encode word meanings depending on context during training, creating more accurate and valuable embeddings for downstream natural language processing tasks and becomes very helpful in scenarios such as the research we are conducting.



```
# Load pre-trained BERT model
bert_model = TFBertModel.from_pretrained('bert-base-uncased')

# Load BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 7.10kB/s]
Downloading tf_model.h5: 100%  536M/536M [00:03<00:00, 134MB/s]
Some layers from the model checkpoint at bert-base-uncased were not used when initializing TFBe
- This IS expected if you are initializing TFBertModel from the checkpoint of a model trained o
- This IS NOT expected if you are initializing TFBertModel from the checkpoint of a model that
All the layers of TFBertModel were initialized from the model checkpoint at bert-base-uncased.
If your task is similar to the task the model of the checkpoint was trained on, you can already
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 347kB/s]
Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 363B/s]
```

Fig.16. *Tokenizing using BERT*

As seen in the figure 16 we are loading the pre-trained bert-base-uncased which has 12 transformer blocks, 110 million parameters, and uncased English text train BERT-base-uncased.

Before tokenizing, the “uncased” tokenizer lowercases every text. This reduces the amount of unique tokens the model must learn and makes it more capitalization-resistant.

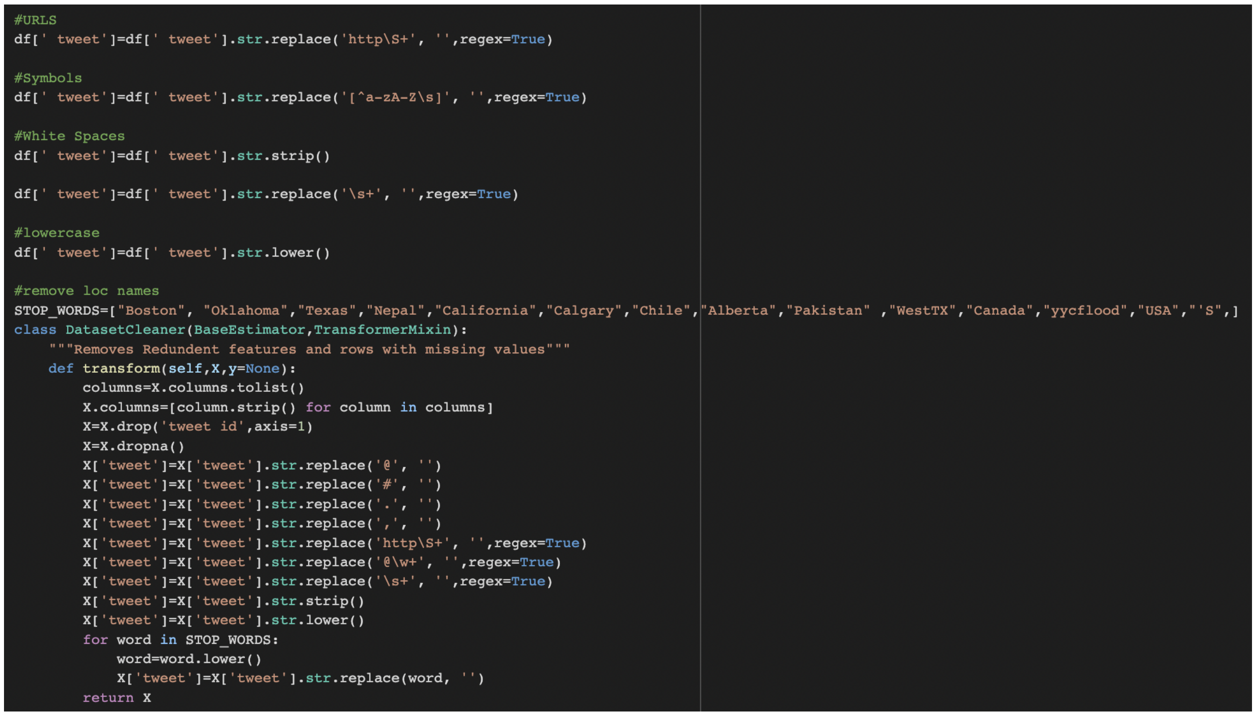
b. Dataset 2

Tweets can contain many different kinds of noise that can negatively affect the performance of the machine learning algorithms . We need to carefully get rid of them. We will use the regular expressions and replace functionality in Pandas to remove the unwanted noise in the data.

- Retweets : They add no real value to the data and can sometimes lead to overfitting
- URL's : They do not deliver any predictive power, The sentiment of a tweet can not be judged by reading an URL. In the worst case scenario they might lead to overfitting.
- Symbols : Hashtags, commas, points and and all kinds of punctuation symbols are removed.
- White Spaces: We also get rid of any additional white spaces in the texts that might be created due to the previous steps.
- Lower case: All texts are transformed to lowercase.
- Location Names: The names of the location which disaster happened were repeated in so many tweets.We want to prevent the model from associating these location names with the crisis and as a result we remove the most frequent ones from the Tweets. The following list of words were removed

from the Tweets: ["Boston", "Oklahoma", "Texas", "Nepal", "California", "Calgary", "Chile", "Alberta", "Pakistan" , "WestTX", "Canada"," yycflood", "USA", "S"]

As seen below in figure 17 we have done the above mentioned cleaning techniques to our data.

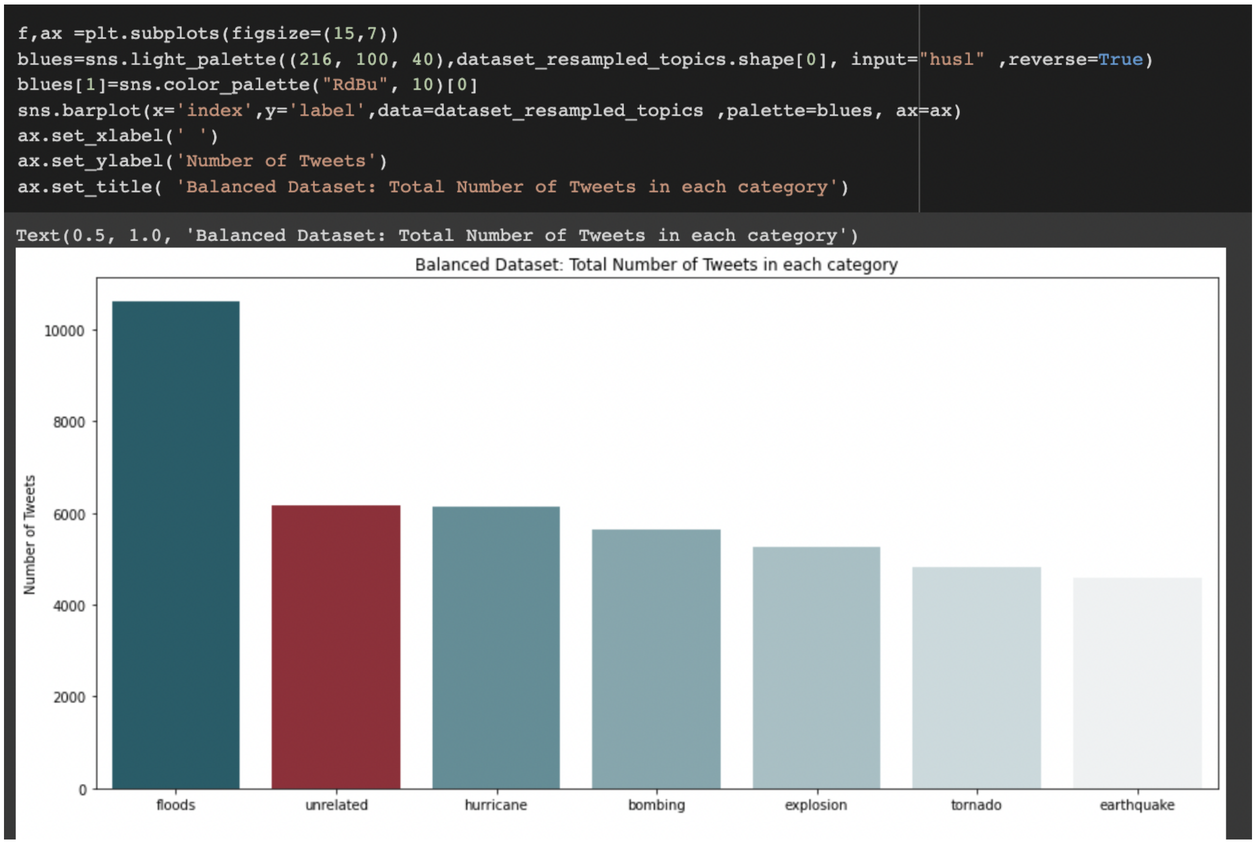


```
#URLS
df[' tweet']=df[' tweet'].str.replace('http\S+', '',regex=True)

#Symbols
df[' tweet']=df[' tweet'].str.replace('[^a-zA-Z\s]', '',regex=True)

#White Spaces
df[' tweet']=df[' tweet'].str.strip()

df[' tweet']=df[' tweet'].str.replace('\s+', '',regex=True)

#lowercase
df[' tweet']=df[' tweet'].str.lower()

#remove loc names
STOP_WORDS=["Boston", "Oklahoma","Texas","Nepal","California","Calgary","Chile","Alberta","Pakistan" ,"WestTX","Canada","yycflood","USA","'S",]
class DatasetCleaner(BaseEstimator,TransformerMixin):
    """Removes Redundent features and rows with missing values"""
    def transform(self,X,y=None):
        columns=X.columns.tolist()
        X.columns=[column.strip() for column in columns]
        X=X.drop('tweet id',axis=1)
        X=X.dropna()
        X['tweet']=X['tweet'].str.replace('@', '')
        X['tweet']=X['tweet'].str.replace('#', '')
        X['tweet']=X['tweet'].str.replace('.', '')
        X['tweet']=X['tweet'].str.replace(',', '')
        X['tweet']=X['tweet'].str.replace('http\S+', '',regex=True)
        X['tweet']=X['tweet'].str.replace('@\w+', '',regex=True)
        X['tweet']=X['tweet'].str.replace('\s+', '',regex=True)
        X['tweet']=X['tweet'].str.strip()
        X['tweet']=X['tweet'].str.lower()
        for word in STOP_WORDS:
            word=word.lower()
            X['tweet']=X['tweet'].str.replace(word, '')
        return X
```

Fig.17. *Cleaning techniques used in dataset 2*

To solve the skewed data problem, we resample a subset of these unrelated Tweets.The total number that we re-sample from these unrelated tweets would be equal to the average number of all tweets in each dataset. We split the dataset into related and unrelated tweets. This ensures that the number of unrelated tweets are not too high and is in reasonable range.

Figure 18 shows the resampled data, and it is no longer skewed and balanced to train models on this dataset.



```
f,ax =plt.subplots(figsize=(15,7))
blues=sns.light_palette((216, 100, 40),dataset_resampled_topics.shape[0], input="husl" ,reverse=True)
blues[1]=sns.color_palette("RdBu", 10)[0]
sns.barplot(x='index',y='label',data=dataset_resampled_topics ,palette=blues, ax=ax)
ax.set_xlabel(' ')
ax.set_ylabel('Number of Tweets')
ax.set_title( 'Balanced Dataset: Total Number of Tweets in each category')
```

```
Text(0.5, 1.0, 'Balanced Dataset: Total Number of Tweets in each category')
```

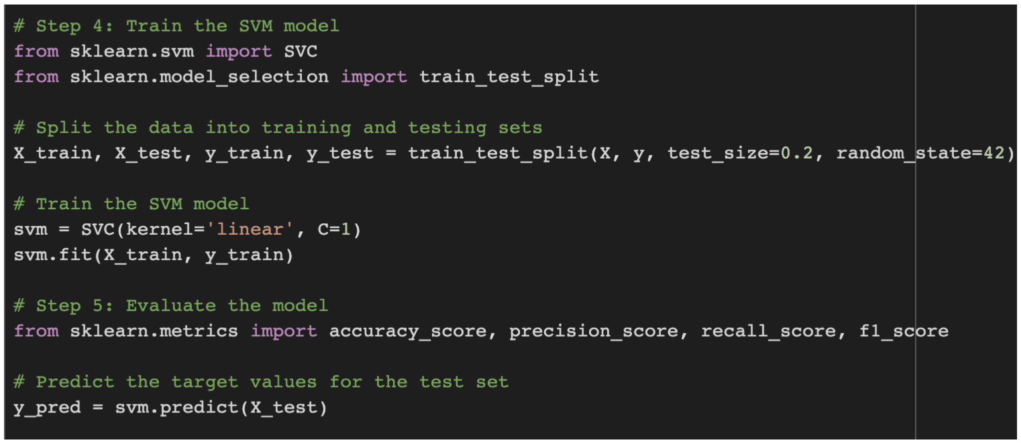
Fig.18. *Visualization of resampled data used in dataset 2*

C. DATA MODELLING:

At the modeling step, the prepared training subset will be fed into the machine learning model's learning process. The confusion matrix will then be used to evaluate the model's performance on the testing subset. This will allow us to assess the model's accuracy, precision, recall, and F1 score and make any required improvements prior to deploying it for practical use.

I. Support Vector Machine

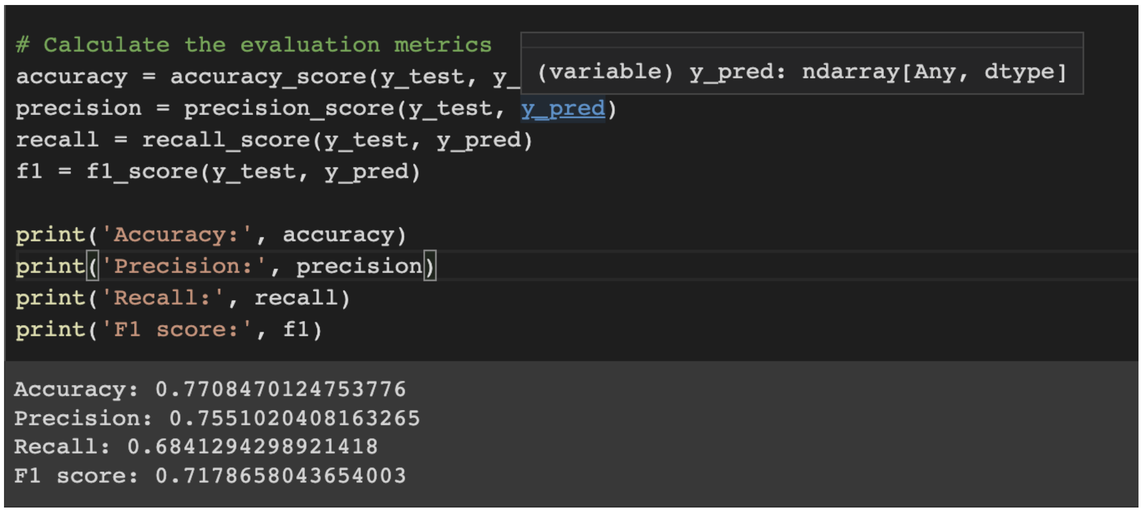
The support vector machine, or SVM, model is helpful for classifying data based on a number of different characteristics. A hyperplane that divides classes in p-dimensional space can be generated using a support vector machine (SVM) by applying the linear function hypothesis to high-dimensional space. In addition to that, it selects the optimal hyperplane for class splitting based on how well it maximizes margins. Two of the most powerful and popular approaches to classifying texts are the Support Vector Machine (SVM) and the Naive Bayes method. If the data can be separated by a significant margin using hypothesis space functions, then support vector machines are able to learn regardless of the dimension of the feature space. This makes it possible for SVMs to generalize even when employing a large number of features. We may generate a hypothesis with the smallest VC-Dimension by picking the optimal parameters. This removes the need for time-consuming and expensive cross-validation and makes it possible to automatically tune the parameters. In the course of our work, we made use of the SVC function that is contained within the Python scikit-learn library as seen in Figure 19.



```
# Step 4: Train the SVM model
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the SVM model
svm = SVC(kernel='linear', C=1)
svm.fit(X_train, y_train)

# Step 5: Evaluate the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predict the target values for the test set
y_pred = svm.predict(X_test)
```

Fig.19 .*SVM Classifier*

After constructing the model using the prediction and confusion matrices, the precision was estimated:



```
# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_  (variable) y_pred: ndarray[Any, dtype]
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1)
```

```
Accuracy: 0.7708470124753776
Precision: 0.7551020408163265
Recall: 0.6841294298921418
F1 score: 0.7178658043654003
```

Fig.20 .*SVM Confusion Matrix*

We can observe from the confusion matrix that the predicted accuracy is 77.08%.

II. Naive Bayes

The Naive Bayes method is characterized by its solid mathematical foundation, which is built on time-honored mathematical concepts. As a consequence, its categorization performance is trustworthy and consistent. Because it is simple to implement, it can be put to a variety of useful practical uses despite the fact that it does not call for a large number of estimated parameters, that it is not overly affected by insufficient data, and that it is difficult to estimate its accuracy. The Naive Bayes classifier can be created in one of two ways: either by using the polynomial model MNB or the multivariate Bernoulli model BNB, which is sometimes referred to as the Binary independent model. Both of these models are known as the Naive Bayes classifier.

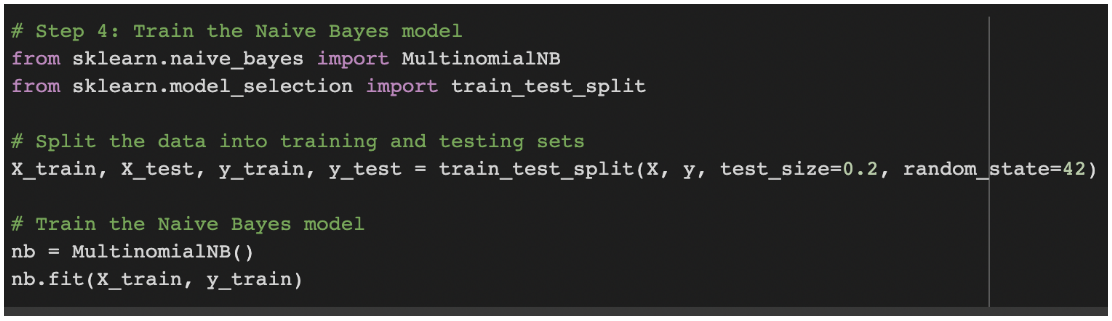
The key differences between these models are :

- Whether to consider a phrase's frequency of recurrence when representing a document. The polynomial MNB model analyzes both the occurrence and frequency of words, whereas the BNB model just considers the occurrence of words.
- There is no evidence of any variations in the way the words are arranged in the report. While the uncommon word "item" is not considered in the classification process of the polynomial MNB model, it should be included in the multivariate Bernoulli BNB model as a factor for calculating the likelihood of the record classification.

Polynomial MNB classification offers the following advantages over BNB model:

- Multinomial MNB can produce a more accurate classification decision when multiple key aspects are combined to form a categorization option.
- In both the training and classification operations, the time complexity is linear.
- The MNB is more resistant to concept drift and noise with polynomial characteristics. So, our research is based on the MNB algorithm.

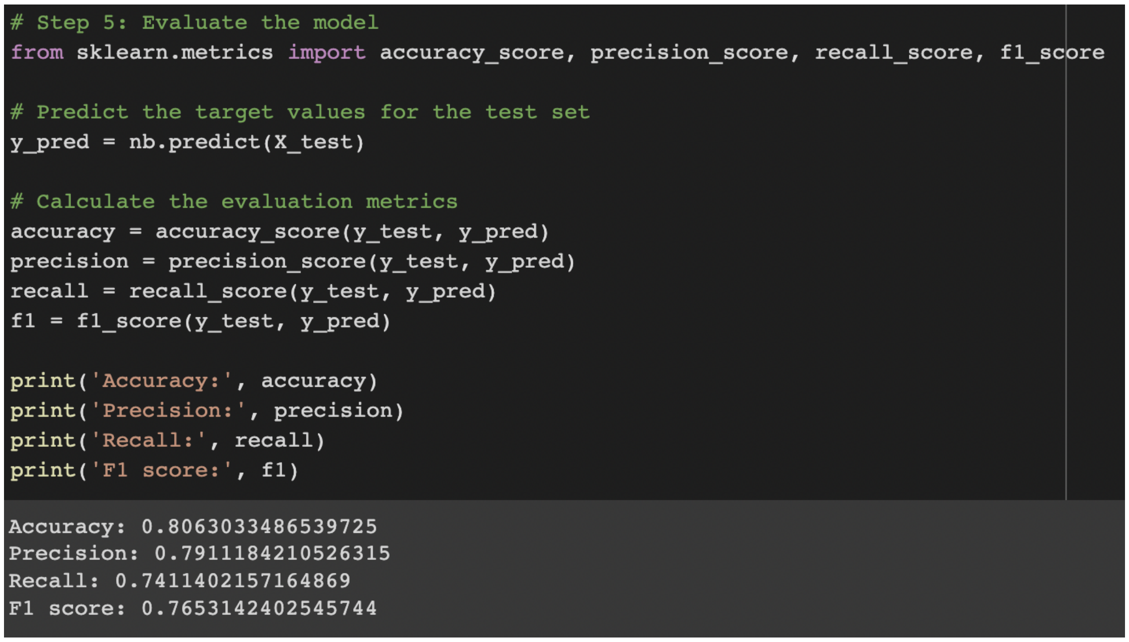
We use the Multinomial NB module from naive bayes in the scikit-learn library in python as seen below:



```python
# Step 4: Train the Naive Bayes model
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Naive Bayes model
nb = MultinomialNB()
nb.fit(X_train, y_train)
```

Fig.21 . *Naive Bayes Classifier*

After constructing the model using the prediction and confusion matrices, the precision was estimated:



```python
# Step 5: Evaluate the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predict the target values for the test set
y_pred = nb.predict(X_test)

# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1)
```

```
Accuracy: 0.8063033486539725
Precision: 0.7911184210526315
Recall: 0.7411402157164869
F1 score: 0.7653142402545744
```

Fig.22 . *Naive Bayes Confusion Matrix*

We can observe from the confusion matrix that the predicted accuracy is 80.63%.

III. Long Short Term Memory (LSTM)

LSTM models are a sort of RNN that can deal with long-term dependencies by incorporating a memory cell and a series of gates that regulate the flow of input. Typical applications include NLP and time series analysis

Since they can analyze input sequences of varying length and capture long-term relationships, LSTMs are particularly useful in NLP. Context is significant in NLP because many language tasks demand comprehension of a word or phrase's context in respect to the words that came before and followed it.

The main differentiation between LSTMs and conventional RNNs is that LSTMs feature memory cells that can retain information for a longer duration of time. Three gates regulate the memory cell: the input gate, the forget gate, and the output gate. The LSTM uses these gates to selectively recall or discard information according to its relevance to the present task.

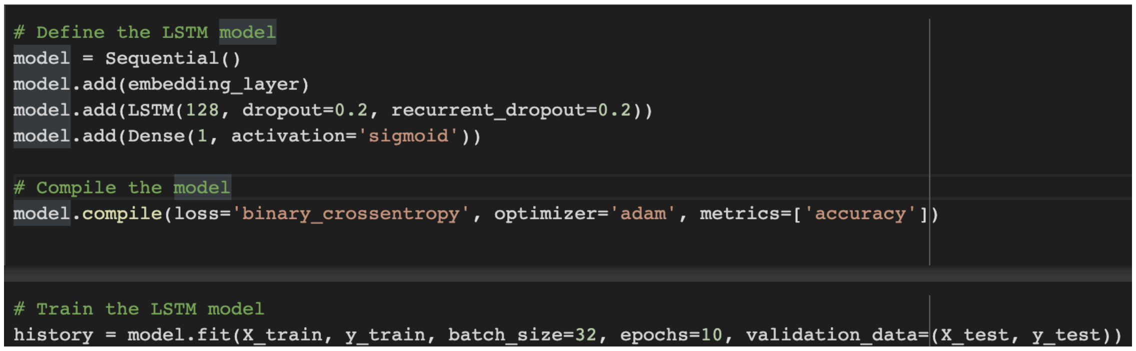
In NLP, an LSTM may accept as input a series of words, with each word represented as a vector. The LSTM would process each word in the sequence, updating its internal state and producing a forecast for the job (such as a sentiment score or a translation). Using techniques like backpropagation across time, the model would be trained on a labeled dataset to update its weights and enhance its performance.

We build an instance of the Sequential class, which is an easily specified and trained linear stack of layers, and then add an embedding layer to the model. This layer transfers a sequence of integer-encoded words to a dense vector space in which words with similar meanings are clustered closer together. The embedding layer is frequently utilized in tasks involving natural language processing.

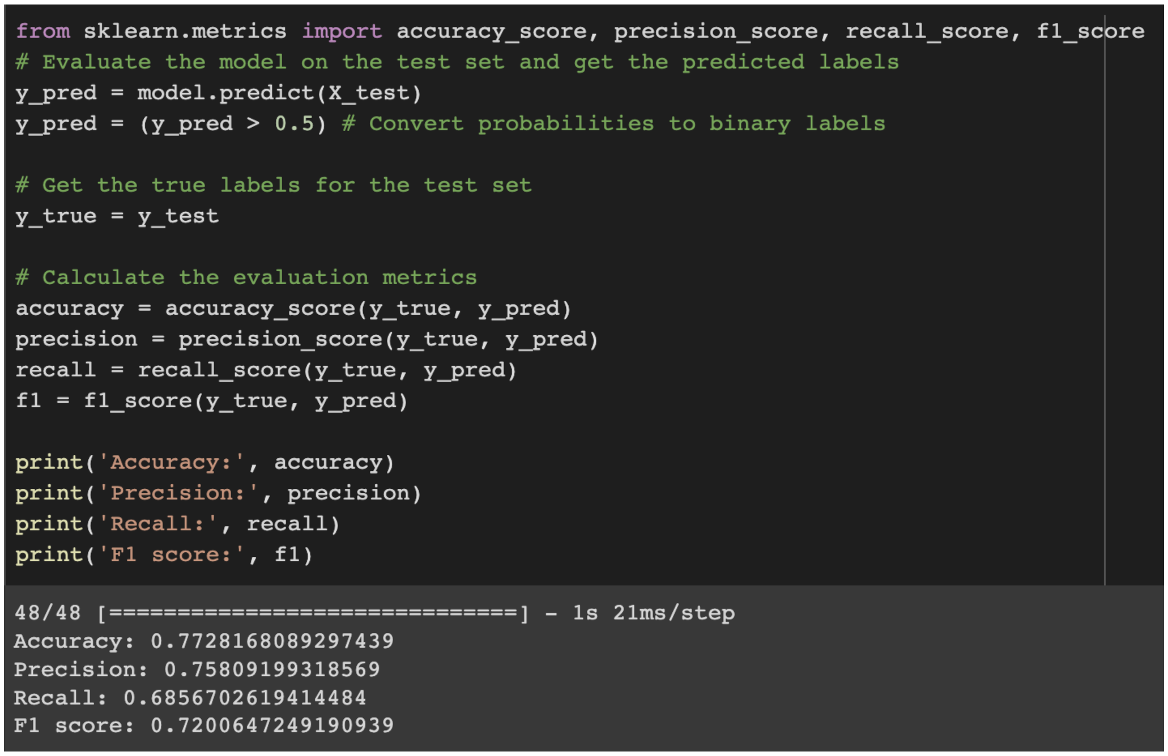
Afterwards, an LSTM layer is added to the model. The LSTM layer includes 128 units, allowing it to record complicated input sequence patterns. To prevent overfitting, the dropout value is set to 0.2, which indicates that 20% of the LSTM units will be randomly dropped out during training. The recurrent dropout parameter is also set to 0.2, indicating that 20% of the connections between LSTM units will be arbitrarily thrown out during training.

Then, a Dense layer with a single output unit and a sigmoid activation function is added to the model. This layer gives a binary classification output, where 0 indicates that the input sequence belongs to a specific class and 1 to the other.

Afterwards, the loss function, optimizer, and evaluation metric are specified in order to compile the model. The binary cross entropy loss function is frequently applied to issues involving binary classification. Adam is a well-known stochastic gradient descent technique that changes the learning rate based on the gradient of the loss function. The accuracy measure is used to evaluate the training and testing performance of the model.


```
# Define the LSTM model
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


# Train the LSTM model
history = model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_test, y_test))
```
Fig.23 . *LSTM Classifier*

We can observe from the confusion matrix that the predicted accuracy is 77.28%.


```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Evaluate the model on the test set and get the predicted labels
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5) # Convert probabilities to binary labels

# Get the true labels for the test set
y_true = y_test

# Calculate the evaluation metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1)
```
```
48/48 [==============================] - 1s 21ms/step
Accuracy: 0.7728168089297439
Precision: 0.75809199318569
Recall: 0.6856702619414484
F1 score: 0.7200647249190939
```
Fig.24 . *LSTM Confusion Matrix*

IV. Bidirectional Long Short Term Memory (BiLSTM)

Bidirectional Long Short-Term Memory (BiLSTM) model is a neural network design that may be used for sequence tagging, sentiment analysis, and machine translation, among other applications.

In NLP, input data frequently comprises word or character sequences. A BiLSTM model handles these sequences in a bidirectional manner, meaning it scans the input sequence in both directions. This allows the model to record both the context preceding and after a single word.

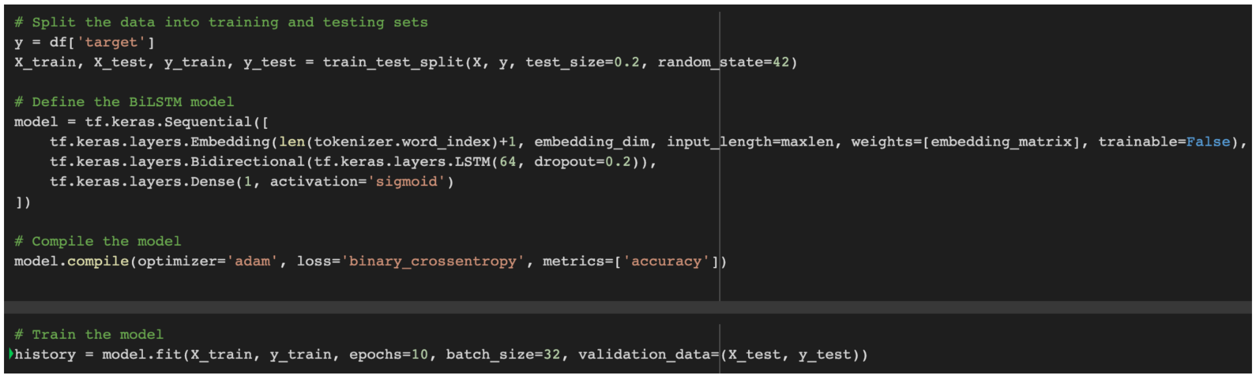
For instance, while processing a phrase, a BiLSTM model would interpret the first word both forward and backward, taking into account the words before and following it. It would then go to the next and repeat the procedure, and so on until the full phrase has been analyzed.

BiLSTM may be utilized for a range of NLP applications, including part-of-speech tagging, named entity identification, and sentiment analysis. In these challenges, the model receives a sequence of words as input and generates a sequence of labels as output. By processing the input sequence in both directions, the BiLSTM model can achieve more precision than a regular LSTM model since it can capture more context and interword relationships.

The BiLSTM model we've constructed using the Sequential class in Keras is composed of three layers:

- A layer of Embedding that transforms the input data (represented as a sequence of integers) into dense, fixed-size vectors. This layer receives the vocabulary size (the length of the word index plus one), the embedding dimension (the size of the dense vectors), the input length, and the embedding matrix (which is pre-trained word embeddings). With the trainable option set to False, the embedding layer weights will not be changed during training.
- A bidirectional LSTM layer capable of processing the input sequence in both ways (forward and backward). This layer receives the output from the embedding layer and consists of 64 units (neurons) with a dropout rate of 0.2.
- A dense layer with a sigmoid activation function and a single unit. This layer receives the output from the LSTM layer and generates a binary output (0 or 1) according to the anticipated class.

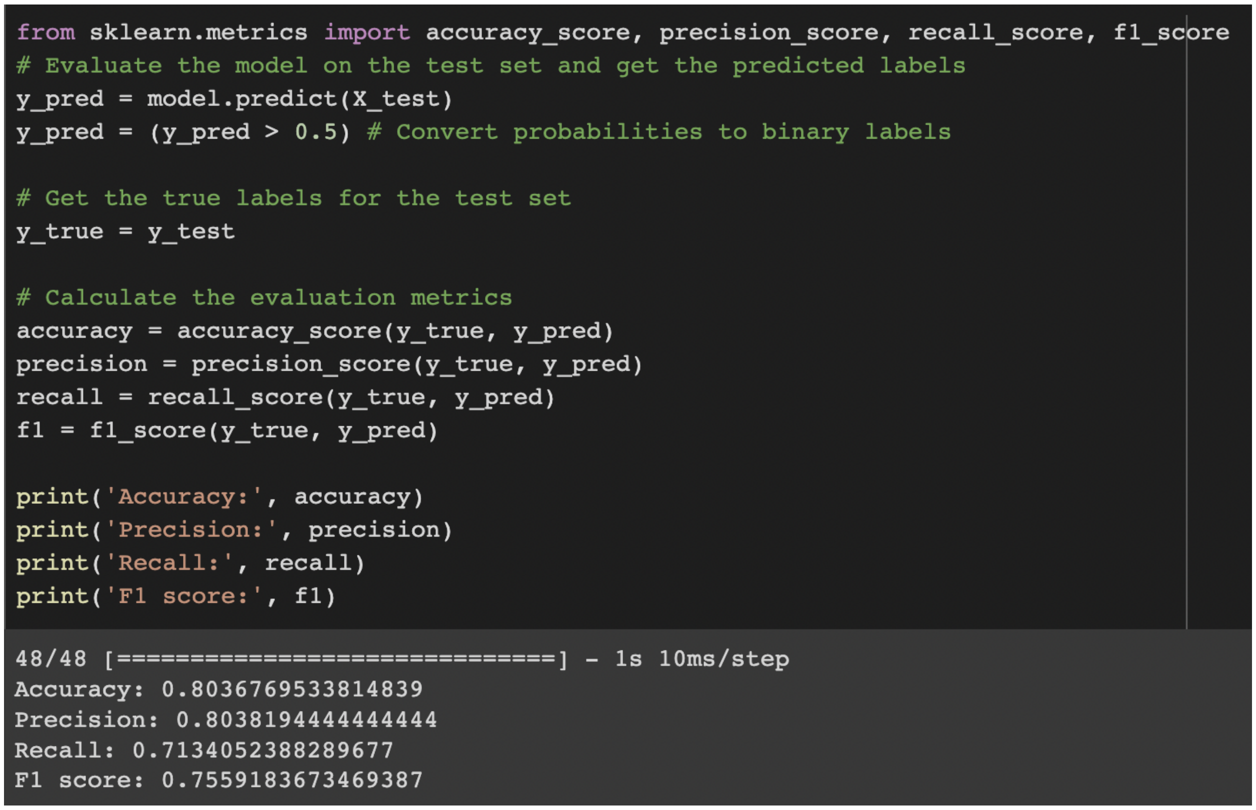
The code then builds the model with the Adam optimizer, the binary cross-entropy loss function, and precision as the evaluation measure.



```
# Split the data into training and testing sets
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the BiLSTM model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(len(tokenizer.word_index)+1, embedding_dim, input_length=maxlen, weights=[embedding_matrix], trainable=False),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, dropout=0.2)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

Fig.25 . *BiLSTM Classifier*

We can observe from the confusion matrix in fig 26 that the predicted accuracy is 80.30%.



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Evaluate the model on the test set and get the predicted labels
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5) # Convert probabilities to binary labels

# Get the true labels for the test set
y_true = y_test

# Calculate the evaluation metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1)
```

```
48/48 [==============================] - 1s 10ms/step
Accuracy: 0.8036769533814839
Precision: 0.8038194444444444
Recall: 0.7134052388289677
F1 score: 0.7559183673469387
```

Fig.26 . *BiLSTM Confusion Matrix*

V. BERT-BiLSTM-CNN (Proposed Model)

The proposed model is constructed using the libraries TensorFlow and Transformers. The following are the stages followed by the code:

- Import tweet information from a CSV file using the Pandas library.
- Load the pre-trained BERT model and BERT tokenizer from the Transformers library.
- Configure the input and output forms for the model. In this instance.

- By sending the input tokens to the pre-trained BERT model, you may obtain BERT embeddings.
- Construct a bi-LSTM layer to discover the temporal correlations between the input tokens and to obtain its output.
- Establish a CNN layer to learn the local patterns between the bi-LSTM output and its output.
- Combine the results of the bi-LSTM and CNN layers.
- Establish thick layers and an output layer to anticipate the tweet's sentiment.
- Define the input and output of the model and build it using an optimizer.
- Convert tweets to BERT embeddings and use the fit() method to train the model using the input tweets and sentiment labels.
- The outcomes of the training are saved in a history variable.
- This model employs a variety of deep learning architectures to discover the underlying patterns in the input text data and classify if the tweet is talking about a disaster or not.

These are the model parameters used in the code:

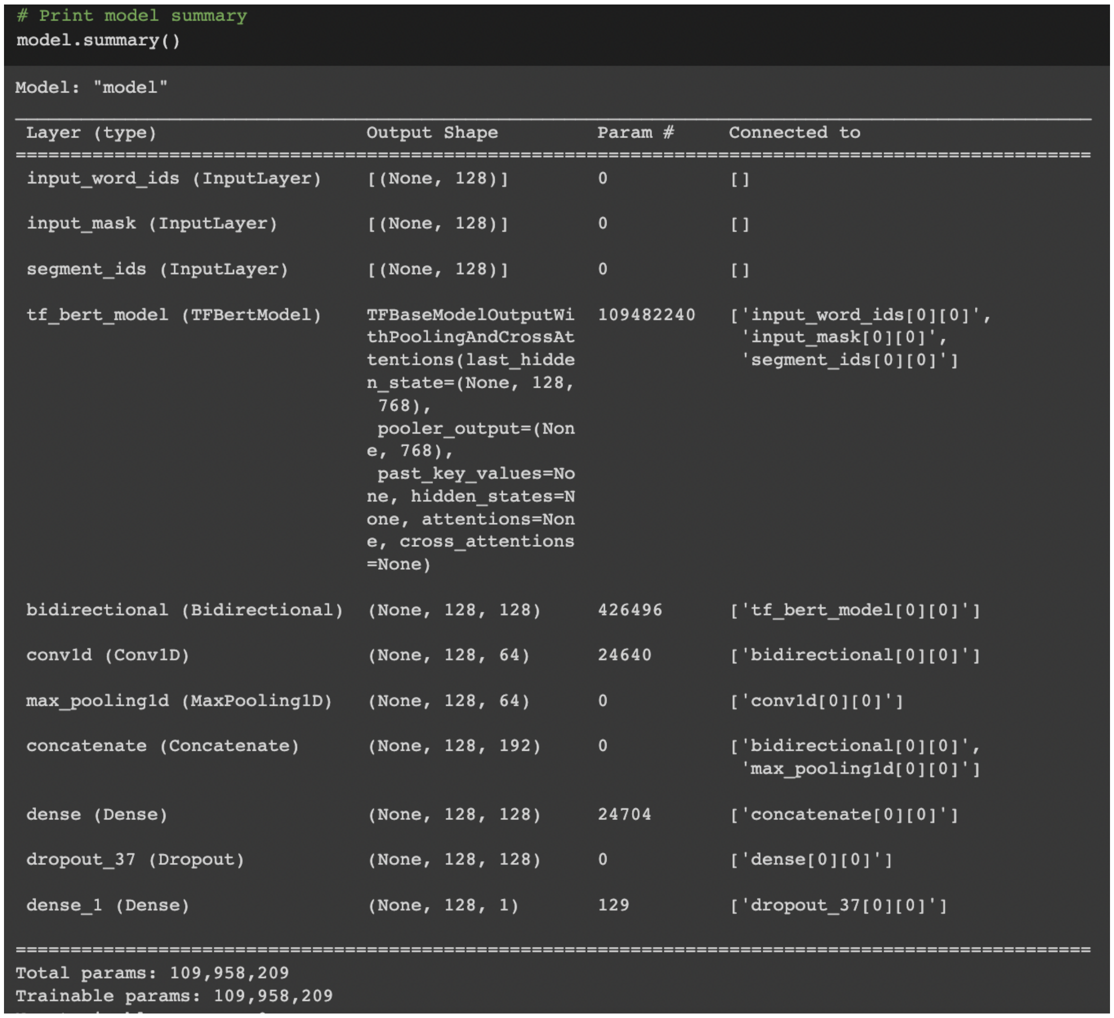
- *Input shape*: The model's input shape has a maximum length of 128 tokens.

Learning rate: The Adam optimizer used to develop the model has its learning rate set to 128.

- The batch size utilized to train the model has been set at 32.
- The number of epochs utilized to train the model has been set to five.

- The number of LSTM units in the Bi-LSTM layer has been set to 64.
- *CNN filters*: 64 filters have been assigned to the CNN layer.
- *CNN kernel size*: The CNN layer kernel size is set at 3.
- *Dense units:* The number of dense layer units has been set to 128.
- The model incorporates a dropout rate of 0.5.

These settings are modifiable and may be adjusted to improve the model's performance on Twitter posts.

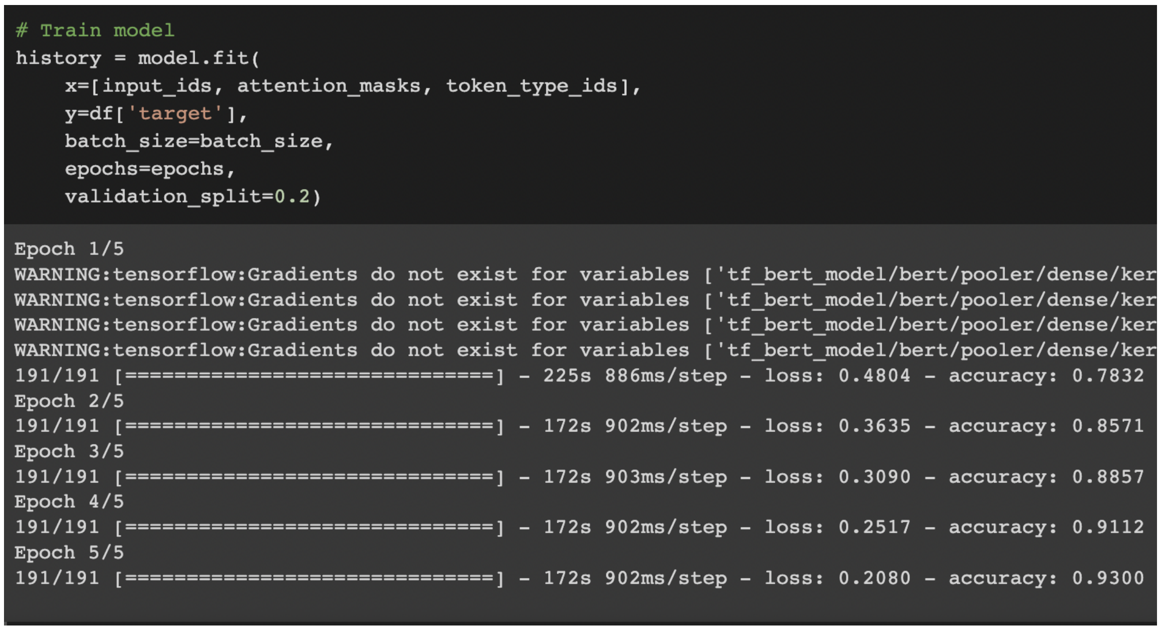


```
# Print model summary
model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_word_ids (InputLayer)    [(None, 128)]        0           []

 input_mask (InputLayer)        [(None, 128)]        0           []

 segment_ids (InputLayer)       [(None, 128)]        0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  109482240   ['input_word_ids[0][0]',
                                thPoolingAndCrossAt               'input_mask[0][0]',
                                tentions(last_hidde               'segment_ids[0][0]']
                                n_state=(None, 128,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 bidirectional (Bidirectional)  (None, 128, 128)     426496      ['tf_bert_model[0][0]']

 conv1d (Conv1D)                (None, 128, 64)      24640       ['bidirectional[0][0]']

 max_pooling1d (MaxPooling1D)   (None, 128, 64)      0           ['conv1d[0][0]']

 concatenate (Concatenate)      (None, 128, 192)     0           ['bidirectional[0][0]',
                                                                  'max_pooling1d[0][0]']

 dense (Dense)                  (None, 128, 128)     24704       ['concatenate[0][0]']

 dropout_37 (Dropout)           (None, 128, 128)     0           ['dense[0][0]']

 dense_1 (Dense)                (None, 128, 1)       129         ['dropout_37[0][0]']

==================================================================================================
Total params: 109,958,209
Trainable params: 109,958,209
```

Fig.27 . *Proposed model summary*

DATASET 1:

As seen below the accuracy of the proposed model is 93%.



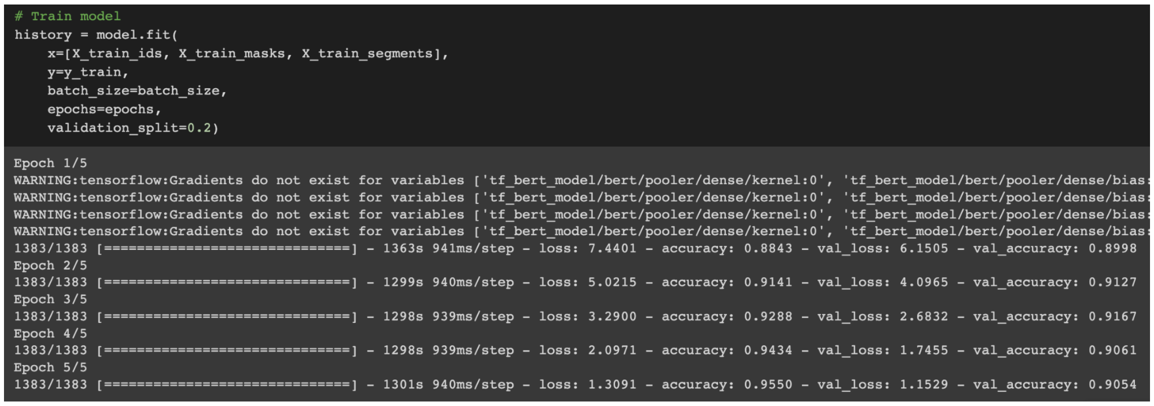
```
# Train model
history = model.fit(
    x=[input_ids, attention_masks, token_type_ids],
    y=df['target'],
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2)
```

```
Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/ker
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/ker
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/ker
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/ker
191/191 [==============================] - 225s 886ms/step - loss: 0.4804 - accuracy: 0.7832
Epoch 2/5
191/191 [==============================] - 172s 902ms/step - loss: 0.3635 - accuracy: 0.8571
Epoch 3/5
191/191 [==============================] - 172s 903ms/step - loss: 0.3090 - accuracy: 0.8857
Epoch 4/5
191/191 [==============================] - 172s 902ms/step - loss: 0.2517 - accuracy: 0.9112
Epoch 5/5
191/191 [==============================] - 172s 902ms/step - loss: 0.2080 - accuracy: 0.9300
```

Fig.28 . *Proposed model accuracy for dataset 1*

DATASET 2:

As seen below the accuracy of the proposed model is 95.5% for the larger dataset 2.



```
# Train model
history = model.fit(
    x=[X_train_ids, X_train_masks, X_train_segments],
    y=y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2)
```

```
Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model/bert/pooler/dense/kernel:0', 'tf_bert_model/bert/pooler/dense/bias:
1383/1383 [==============================] - 1363s 941ms/step - loss: 7.4401 - accuracy: 0.8843 - val_loss: 6.1505 - val_accuracy: 0.8998
Epoch 2/5
1383/1383 [==============================] - 1299s 940ms/step - loss: 5.0215 - accuracy: 0.9141 - val_loss: 4.0965 - val_accuracy: 0.9127
Epoch 3/5
1383/1383 [==============================] - 1298s 939ms/step - loss: 3.2900 - accuracy: 0.9288 - val_loss: 2.6832 - val_accuracy: 0.9167
Epoch 4/5
1383/1383 [==============================] - 1298s 939ms/step - loss: 2.0971 - accuracy: 0.9434 - val_loss: 1.7455 - val_accuracy: 0.9061
Epoch 5/5
1383/1383 [==============================] - 1301s 940ms/step - loss: 1.3091 - accuracy: 0.9550 - val_loss: 1.1529 - val_accuracy: 0.9054
```

Fig.29 . *Proposed model accuracy for dataset 2*

CHAPTER 7

Results

We employed SVM, Naïve Bayes, LSTM-Glove, BiLSTM-Glove, and BERT-BiLSTM-CNN for modeling on dataset 1. The training/testing split was 80/20. BERT-BiLSTM-CNN and Naive Bayes had high accuracy of 93% and 80.63%. We then ran the proposed BERT-BiLSTM-CNN model on dataset 2 which is a larger dataset with ~60k rows, the model has run with training accuracy of 95.5% as seen in table 3. The proposed model was run for 5 epochs on both datasets.

MODELS	Accuracy	F1 Score	Recall	Precision
SVM	77.08%	71.78%	68.41%	75.5%
Naive Bayes	80.63%	76.53%	74.11%	79.11%
LSTM-Glove	77.28%	72%	68.56%	75.80%
BiLSTM-Glove	80.36%	75.5%	71.34%	80.3%
BERT-BiLSTM-CNN	93%	88.15%	85.50%	90.99%

Table 3: Evaluation metrics on dataset 1

The graph in figure 30 presents a comparison of the performance of various models on a given job by making use of the following four assessment metrics: accuracy, F1 score, recall, and precision. SVM, Naive Bayes, LSTM with GloVe embeddings, BiLSTM with GloVe embeddings, and BERT-BiLSTM-CNN are the models that have been used. The evaluation metrics are displayed along the x-axis, and the percentage score is shown along the y-axis.

The graph demonstrates that the BERT-BiLSTM-CNN model beats all other models with respect to all four assessment criteria. It has a score of over 90% for accuracy and precision, as well as a score of over 85% for F1 score and recall. Additionally, the BiLSTM with GloVe embeddings model performs very well, receiving a score of greater than 80% for each of the four evaluation metrics. The SVM and Naive Bayes models both have lower scores when compared to the deep learning models; however, the Naive Bayes model performs marginally better than the SVM model on all metrics, with the exception of precision. The overall performance of the graph reveals that multi-classification deep learning models perform significantly better than typical machine learning models when it comes to text categorization tasks.

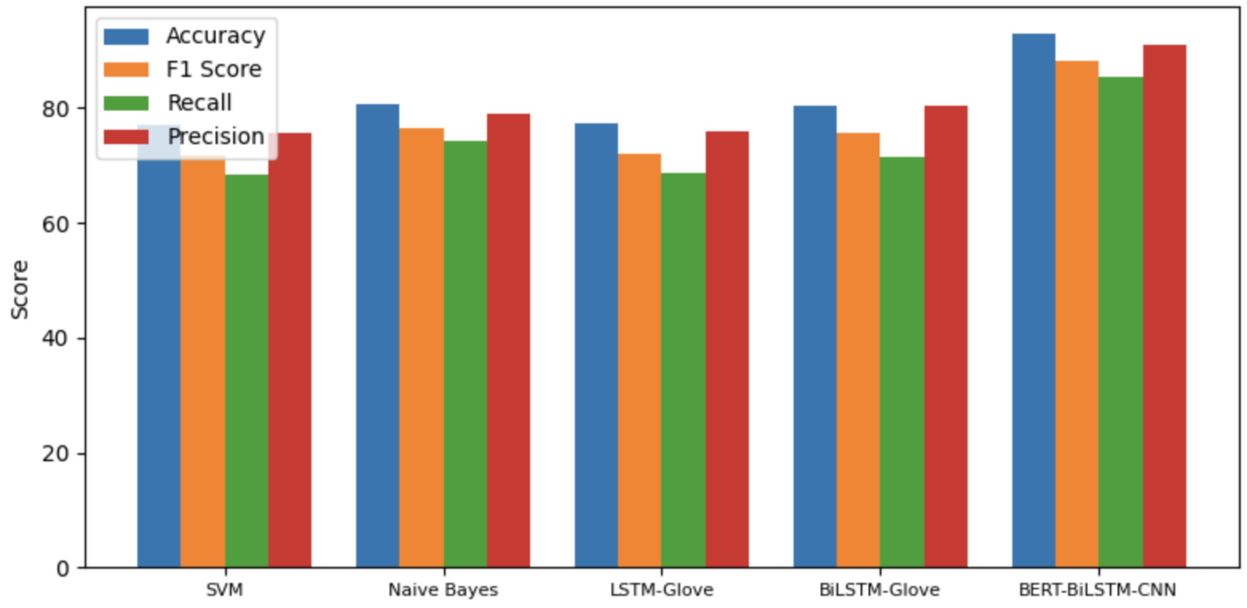


Fig.30 . *Performance Comparison of models*

Figure 30 provides a visual representation of the comparison of the evaluation metrics between the proposed model and the models used in the past.

MODELS	**Accuracy**	**F1 Score**	**Recall**	**Precision**
BERT-BiLSTM-CNN	90.32%	91.15%	93.56%	88.86%

Table 4: Evaluation metrics on dataset 2

As seen in figure 31, the model's performance on both the training and validation data is great, as seen by the high training accuracy of 95.5% and the validation accuracy of 90.54% displayed in the model accuracy graph.

The model does not appear to be overfitting the training data, since the 5% discrepancy between the two sets of accuracy estimates is not excessive. On the other hand, you need to take into account things like the quantity and complexity of the dataset when assessing the model's performance.

The model's performance can be improved with additional experiments like cross-validation or hyperparameter tuning, and a validation accuracy of 90.54% is generally considered good.

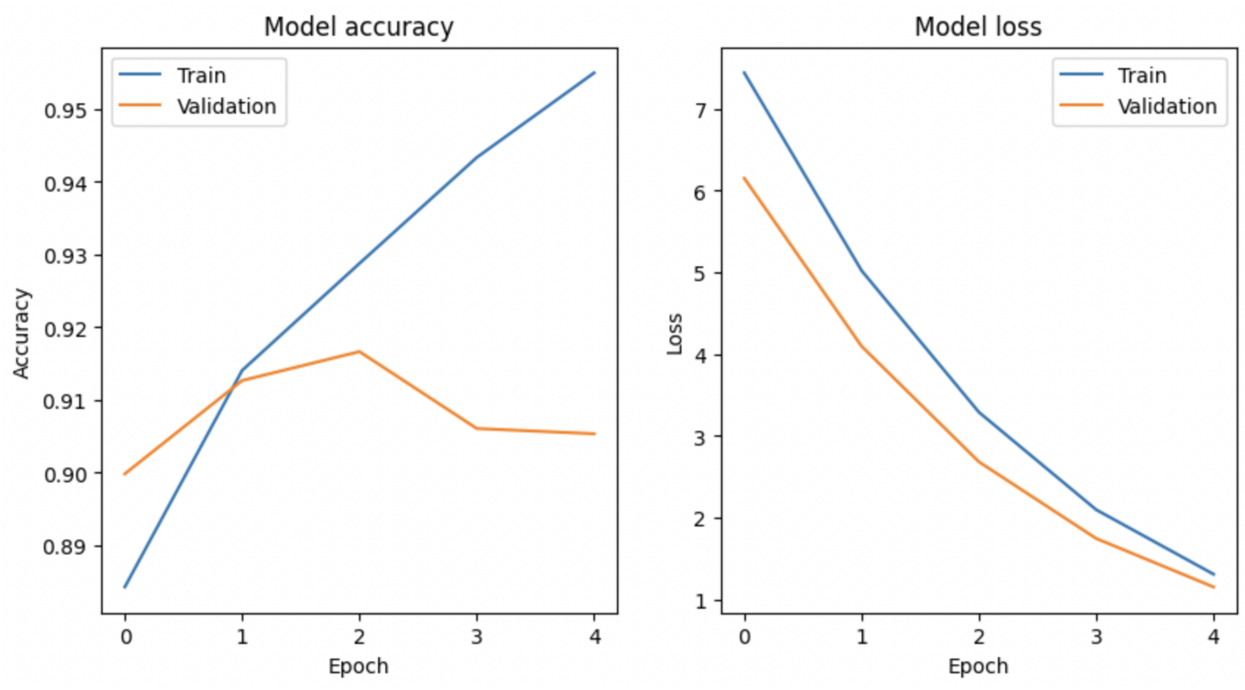


Fig.31 . *model accuracy and model loss graphs for dataset 2*

The model is doing an effective task of minimizing its training error, as indicated by the loss graph. The model is becoming better with time, as shown by the training loss of 1.3091, and it seems to be able to generalize quite well to new, unseen data, as shown by the validation loss of 1.1529.

With a difference of only about 0.15 between training and validation losses, it appears that the model is not overfitting the training data excessively. On the other hand, you need to also take into account things like the quantity and complexity of the dataset when assessing the model's performance.

CHAPTER 8

Conclusion

In conclusion, the objective of the project was to try to develop a model that is able to determine whether or not a certain passage of text is making reference to a genuine catastrophe. Potentially, this model could help distinguish between genuine emergencies and hoaxes, allowing the concerned team to respond more effectively. In the review of the relevant literature, the various strategies for text categorization and analysis that can be applied in practice were dissected, along with the most successful approaches.

By fusing the strength of BERT embeddings with a bidirectional LSTM and CNN layers, a deep learning model for tweet classification was created in this research. This multi-model strategy was developed to take advantage of the capabilities of the pre-trained BERT model, the sequential nature of LSTMs, and the local pattern-recognition capabilities of CNNs.

The hypothesis of the proposed model seems to be positive since it has outperformed all of the other models that were used in previous research as mentioned in the related work with an accuracy of 93% for a smaller dataset and ~95% for a comparatively larger dataset. It can be said that the model can improve in effectiveness as it is exposed to more data and as it is able to understand more complex patterns and relationships.

However, additional information may not automatically result in better results. The performance of the model can be significantly impacted by poor data quality or noise. It's also possible that the model's accuracy plateaus once a certain amount of data has been included. In such circumstances, it may be more fruitful to shift the focus to other areas, such as feature engineering, model design, or hyperparameter tuning.

Overall, this project demonstrates the potential of combining pre-trained language models like BERT with other deep learning architectures to tackle complex natural language processing tasks such as tweet classification.

CHAPTER 9

Future Work

In each of the stages of the machine learning pipeline, there are a number of potential fields of research that could be explored in the near or far future. These include the following:

- Tuning of hyperparameters: To increase the performance of the model, try experimenting with a variety of hyperparameter settings, such as the learning rate, batch size, number of filters, kernel size, and dropout rate. For a more methodical hyperparameter search, one may use grid search, random search, or Bayesian optimization.
- Model architecture: Experiment with different model designs, such as employing multiple convolutional layers with varying kernel sizes, increasing the number of LSTM layers, and so on.
- Preprocessing: Text preprocessing techniques such as stopping, stemming, and lemmatization may be explored to determine if they enhance the model's performance.
- Transfer learning: Experiment with fine-tuned pre-trained models like RoBERTa, DistilBERT, or XLNet to compare their performance with BERT on the task.
- Multilingual Support: By using datasets written in many languages during model training and testing, a more inclusive and robust model can be produced. The 'bert-base-multilingual-cased' and

'bert-base-multilingual-uncased' multilingual BERT variants provide a good place to begin.

- Multi-task learning: To improve the model's efficiency, it can be trained to perform numerous tasks at once, such as sentiment analysis, emotion recognition, and sarcasm detection.
- Parallelized techniques: A potential future approach is to investigate the use of parallelization techniques to run optimal models for each category of disaster such as, floods, earthquake, etc at the same time. Most models are now trained and implemented as a single entity, treating all categories identically. Certain categories, however, may have distinct properties or require specialist modeling methodologies.
- Real-time analysis: Using the trained model, an application programming interface (API) or a web application can be developed that enables real-time analysis of tweets.

Bibliography

[1] S. Madichetty and M. Sridevi, "Improved classification of crisis related data on Twitter using contextual representations," Procedia Comput. Sci., vol. 167, pp. 962–968, 2020.

[2] Y. Qu, C. Huang, P. Zhang, and J. Zhang, "Microblogging after a major disaster in China: a case study of the 2010 Yushu earthquake", Proceedings of the ACM 2011 conference on Computer supported cooperative work , ACM, March, 2011. pp. 25-34.

[3] K. Starbird, L. Palen, A. L. Hughes, and S. Vieweg, "Chatter on the red: what hazards threat reveals about the social life of microblogged information," Proceedings of the ACM 2010 conference on Computer supported cooperative work, ACM, February, 2010. pp. 241-250.

[4] S. Vieweg, A. L. Hughes, K. Starbird and L. Palen, (2010, April). "Microblogging during two natural hazards events: what twitter may contribute to situational awareness," Proceedings of the SIGCHI conference on human factors in computing systems.ACM. April, 2010. pp. 1079-1088.

[5] P. K. Prasetyo, M. Gao, E. P. Lim, and C. N. Scollon, " Social sensing for urban crisis management: The case of singapore haze," In International Conference on Social Informatics, Springer, Cham, November, 2013. pp. 478-491.

[6] P. Bhere, A. Upadhyay, K. Chaudhari, and T. Ghorpade, "Classifying Informatory Tweets during Disaster Using Deep Learning," in ITM Web of Conferences, 2020, vol. 32, p. 3025.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv Prepr. arXiv1810.04805, 2018.

[8] A. Romascanu et al., "Using deep learning and social network analysis to understand and manage extreme flooding," J. Contingencies Cris. Manag., vol. 28, no. 3, pp. 251–261, 2020.

[9] M. S. Jagadeesh and P. J. A. Alphonse, "NIT_COVID-19 at WNUT2020 Task 2: Deep Learning Model RoBERTa for Identify Informative COVID-19 English Tweets.," in W-NUT@ EMNLP, 2020, pp. 450– 454.

[10] Y.Kim, “Convolutional Neural Networks for Sentence Classification” In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1746-1751). 2014.

[11] K. Miao, Q. Hua, and H. Shi, “Short-term load forecasting based on cnn-bilstm with bayesian optimization and attention mechanism,” in International Conference on Parallel and Distributed Computing: Applications and Technologies, pp. 116–128, Springer, 2020.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv Prepr. arXiv1810.04805, 2018.

[13] A. Vaswani et al., “Attention Is All You Need,” 2017, doi: 10.48550/ARXIV.1706.03762.

[14] Y. Wu et al., “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” 2016, doi: 10.48550/ARXIV.1609.08144

[15] C. Caragea, A. Silvescu, and A. H. Tapia, “Identifying informative messages in disaster events using convolutional neural networks,” in International conference on information systems for crisis response and management, 2016, pp. 137–147.

[16] P. Rendra Dwi Lingga, Chastine Fatichah, & Diana Purwitasari. “Deteksi Gempa Berdasarkan Data Twitter Menggunakan Decision Tree, Random Forest, dan SVM”. Jurnal Teknik ITS, 6, 153-158. 2017.

[17] Wu, D., Vidyarthi, A. K., & Muljono, G. (2018). “Classifying tweets during crisis events using deep learning”. Procedia Engineering, 210, 1082-1087. doi:10.1016/j.proeng.2018.02.063

[18] Chitrakar, H., Nepal, D. R., & Nepal, S. (2017). “Machine learning techniques for tweet classification during disasters”. Procedia Computer Science, 105, 148-155. doi:10.1016/j.procs.2017.01.019

[19] M.E. Basiri, S. Nemati, M. Abdar, E. Cambria, and U. R. Acharya, “ABCDM: An Attention-based Bidirectional CNN-RNN Deep Model for sentiment analysis,” Future Generation Computer Systems, vol. 115, pp. 279–294, Feb. 2021, doi: 10.1016/j.future.2020.08.005.

[20] A. U. Rehman, A. K. Malik, B. Raza, and W. Ali, “A Hybrid CNN-LSTM Model for Improving Accuracy of Movie Reviews Sentiment Analysis,” Multimed Tools

Appl, vol. 78, no. 18, pp. 26597–26613, Sep. 2019, doi: 10.1007/s11042-019-07788-7.

[21] Y. Zhang, J. Zheng, Y. Jiang, G. Huang, and R. Chen, "A Text Sentiment Classification Modeling Method Based on Coordinated CNN-LSTM-Attention Model," Chin. j. electron., vol. 28, no. 1, pp. 120–126, Jan. 2019, doi: 10.1049/cje.2018.11.004

[22] S. Minaee, E. Azimi, and A. Abdolrashidi, "Deep-Sentiment: Sentiment Analysis Using Ensemble of CNN and Bi-LSTM Models," 2019, doi: 10.48550/ARXIV.1904.04206

[23] Y. Li, K. Zhang, J. Wang, and X. Gao, "A cognitive brain model for multimodal sentiment analysis based on attention neural networks," Neurocomputing, vol. 430, pp. 159–173, Mar. 2021, doi: 10.1016/j.neucom.2020.10.021

[24] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," 2017, doi: 10.48550/ARXIV.1703.03400

[25] C. Gan, Q. Feng, and Z. Zhang, "Scalable multi-channel dilated CNN–BiLSTM model with attention mechanism for Chinese textual sentiment analysis," Future Generation Computer Systems, vol. 118, pp. 297–309, May 2021, doi: 10.1016/j.future.2021.01.024

[26] Z. Ashktorab, C. Brown, M. Nandi and A. Culotta, "Tweedr: Mining Twitter to Inform Disaster Response", 11th International Conference on Information Systems for Crisis Response and Management, 2014.

[27] A. Karami, V. Shah, R. Vaezi, and A. Bansal, "Twitter speaks: A case of national disaster situational awareness," Journal of Information Science, vol. 46, no. 3, pp. 313–324, Jun. 2020, doi: 10.1177/0165551519828620.

[28] A. Olteanu, C. Castillo, F. Diaz, and S. Vieweg, "CrisisLex: A Lexicon for Collecting and Filtering Microblogged Communications in Crises," ICWSM, vol. 8, no. 1, pp. 376–385, May 2014, doi: 10.1609/icwsm.v8i1.14538.

[29] L. Zou, N. S. N. Lam, H. Cai, and Y. Qiang, "Mining Twitter Data for Improved Understanding of Disaster Resilience," Annals of the American Association of Geographers, vol. 108, no. 5, pp. 1422–1441, Sep. 2018, doi: 10.1080/24694452.2017.1421897.

[30] S. P. Algur and V. S, "Classification of Disaster Specific Tweets - A Hybrid Approach," 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom), 2021, pp. 774-777.

[31] G. K. Palshikar, M. Apte, and D. Pandita, “Weakly Supervised and Online Learning of Word Models for Classification to Detect Disaster Reporting Tweets,” Inf Syst Front, vol. 20, no. 5, pp. 949–959, Oct. 2018, doi: 10.1007/s10796-018-9830-2.

[32] J. P. Singh, Y. K. Dwivedi, N. P. Rana, A. Kumar, and K. K. Kapoor, “Event classification and location prediction from tweets during disasters,” Ann Oper Res, vol. 283, no. 1–2, pp. 737–757, Dec. 2019, doi: 10.1007/s10479-017-2522-3.

[33] M. Pota, M. Ventura, H. Fujita, and M. Esposito, “Multilingual evaluation of pre-processing for BERT-based sentiment analysis of tweets,” Expert Systems with Applications, vol. 181, p. 115119, Nov. 2021, doi: 10.1016/j.eswa.2021.115119.

[34] F. Meng, S. Yang, J. Wang, L. Xia, and H. Liu, “Creating Knowledge Graph of Electric Power Equipment Faults Based on BERT–BiLSTM–CRF Model,” J. Electr. Eng. Technol., vol. 17, no. 4, pp. 2507–2516, Jul. 2022, doi: 10.1007/s42835-022-01032-3.