



# **NATURAL DISASTER TWEET MANAGEMENT**



A Project Report in partial fulfilment of the degree

**Bachelor of Technology**  
in  
**Computer Science & Engineering**

By

19K41A0576

M.TharunKumar

19K41A0587

P.Tejaswi

19K41A0588

T.Vishwa

Under the guidance of

**D. RAMESH**

Assistant Professor School of CS & AI SRU

**Submitted to**

**DEPARTMENT OF COMPUTERSCIENCE&ENGINEERING**  
**S.R.ENGINEERINGCOLLEGE (A), ANANTHASAGAR,WARANGAL**

(Affiliated to JNTUH, Accredited by NBA) May-2022.



## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

### **CERTIFICATE**

This is to certify that the Project Report entitled “NATURAL DISASTER TWEET MANAGEMENT” is a record of bonafied work carried out by the student(s) Tharun kumar, Tejaswi, Vishwa, bearing Roll No(s) 19K41A0576, 19K41A0587, 19K41A0588 during the academic year 2022-23 in partial fulfillment of the award of the degree of **Bachelor of Technology in Computer Science & Engineering** by the S.R. ENGINEERING COLLEGE, Ananthasagar, Warangal.

**Supervisor**

**Head of the Department**

## **External Examiner**

### **ABSTRACT**

In today's world social media has become an integral part of life. Twitter is an American micro-blogging and social networking portal which provides the users a platform to post news, data and thoughts. Twitter has become an essential mode of communication medium during the occurrence of an emergency or disaster. This information regarding disasters propagated over the social media can save thousands of life by alerting others so that they can take evasive action. Although governments and emergency management agencies work together through their respective national system for response to disasters, the sentiments of the people affected during and after the catastrophe decide the effectiveness of the disaster response and the recovery process. In recent years, sentiment analysis via Twitter-based machine learning has become a common subject. However, the detection of such tweets was often difficult due to the tweets' language structure's uncertainty. Thus, it is not always apparent if the words of a person announce a catastrophe. BERT (Bidirectional Encoder Representations by Transformers) is a profound learning model developed by Google. Since Google opened it, several scientists and companies have embraced it and have applied it to many text classification tasks. Therefore, we use BERT in this paper to some dataset disaster tweets.

## Table of contents

S.No.	Content	Page No.
1	Introduction	5
2	Literature Review	6
3	Design	8
4	Dataset	9
5	Data Pre-processing	10
6	Methodology	11
7	Results	13
8	Conclusion	18
9	References	1

## 1. INTRODUCTION

Twitter was introduced publicly on July 15, 2006 initially as a messaging service within a small group. It grew and became a microblogging and social networking portal service which enables users to post small write-ups, reviews and messages popularly known as “tweets”. Registered users are provided a platform where they can post, like and re-tweet tweets, whereas unregistered users can only view tweets. Twitter has come a long way since its inception days of 2006. As of October, 2019, every second produces around 6000 tweets on an average globally. This figure is 350,000 per minute or 500 million tweets per day or 200 billion tweets per year.

The twitter data is a set of unstructured data because the data is not in any particular prescribed format. Here lies the real challenge in analyzing the twitter data. For this purpose Natural Language Processing (NLP) is required. The tweets are tokenized into words after data cleaning is completed. Then analysis is performed on the tokenized text data.

Various disasters in various parts of the world have raised awareness about internal awareness disaster reduction. The issue of disaster management is one of many countries and organizations. In this regard, research on disaster management is increasing. In fact, the term "natural disaster" has been called a misnomer already in 1976. A disaster is a result of a natural or man-made hazard impacting a vulnerable community. It is the combination of the hazard along with exposure of a vulnerable society that results in a disaster.

Disasters and emergencies are happening globally in a continuous and uninterrupted manner. The ubiquitousness of smartphones, laptops and tablets has enabled people to communicate the occurrence of disasters they are experiencing in real-time. Thus twitter has become an important mode or channel of communication in times of emergencies and disasters. Many data-analytics agencies are trying to programmatically monitor and analyze twitter data. Specifically, news agencies and disaster relief organizations are trying to analyze tweets in real time to identify the occurrence of disasters from tweets. This would help millions of common people in averting the dangers of impending disasters. If people could be informed regarding the occurrence of disasters at a particular location in real-time then they can take evasive action. Government agencies can execute evacuation before the situation goes beyond control.

## 2. LITERATURE REVIEW

Several research have been conducted using various methodologies to identify real or fake disaster tweets.

[1] Meet Rajdev and Dr. Kyumin Lee, in this paper they conducted a case study of the 2013 Moore Tornado and Hurricane Sandy. They have proposed flat and hierarchical classification approaches with our proposed features. Finally, we have developed both flat and hierarchical classifiers for each dataset (disaster) and the combined dataset. The experimental results show that the proposed approaches identify spam and fake messages with 96.43% accuracy .

[2] Meera R.Nair<sup>1</sup>G.R.Ramya<sup>1</sup>P. BagavathiSivakumar ,This paper mainly contains a study regarding how people of Chennai used social media especially twitter, in response to the country's worst flood that had occurred recently. The tweets collected were analysed using machine learning algorithms such as Random Forests, Naive Bayes and Decision Tree. By comparing the performances of all the three it was found that Random Forests is the best algorithm that can be relied on, during a disaster.

[3] Kyle Hunt,Puneet Agarwal,Jun Zhuang; In this work, they filled the research gap by developing a machine learning framework to predict the veracity of tweets that are spread during crisis events. The tweets are tracked based on the veracity of their content as either true, false, or neutral.

[4] Arghya Ray and PRADIP KUMAR BALA Conducted experiments on tweets during the natural disasters or man made disasters. To reduce the impact of these rumours, a technique was proposed that makes use of supervised learning to differentiate between information about an actual event from information about a false one and communicating it effectively to appropriate organizations. For this purpose, 934 social media feeds were analysed using a Naïve Bayes classifier.

[5] Jayashree Domala; Manmohan Dogra; Vinit Masrani; Dwayne Fernandes; Kevin D'souza;The current research suggests A software solution that can help disaster management websites to dynamically show the disaster relevant news which can be shared to other social media handles through their sites. The objective here is to automatically scrape news from English news websites and identify disaster relevant news using natural language processing techniques and machine learning.

S NO	PAPER NAME	PUBLISHED YEAR	MODELS USED	BEST MODEL AND ACCURACY
1	Fake and Spam Messages: Detecting Misinformation During Natural Disasters on Social Media	2016	Hierarchical Classification Flat Classification Algorithm	94.92%
2	Usage and analysis of Twitter during 2015 Chennai flood towards disaster management	2017	Decision Tree Random Forest Naïve Bayes	Random Forest- 96.7%
3	Monitoring Misinformation on Twitter During Crisis Events	2020	Machine Learning framework	83%
4	Social media for improved process management in organizations during disasters	2020	Naïve Bayes Classifier	86.3%
5	Tweets Analysis for Disaster Management	2020	SVM-Support Vector Machine. KNN-K Nearest Neighbor Logistic Regression	93.4%
6	Automated Identification of Disaster News for Crisis Management	2020	Naive Bayes Classifier Logistic Regression Support Vector Machine (SVM)	Logistic Regression 88%
7	Identification of Disaster-related tweets using Natural Language Processing	2020	Decision Tree	71%
8	Using Machine Learning in Disaster Tweets Classification	2022	KNN, SVM, XGBoost, and Naïve Bayes	SVM-80%

**Table 1 :** Literature survey on various journal papers

### **3. DESIGN**

#### **3.1 REQUIREMENT SPECIFICATION(S/W & H/W)**

##### **Hardware Requirements**

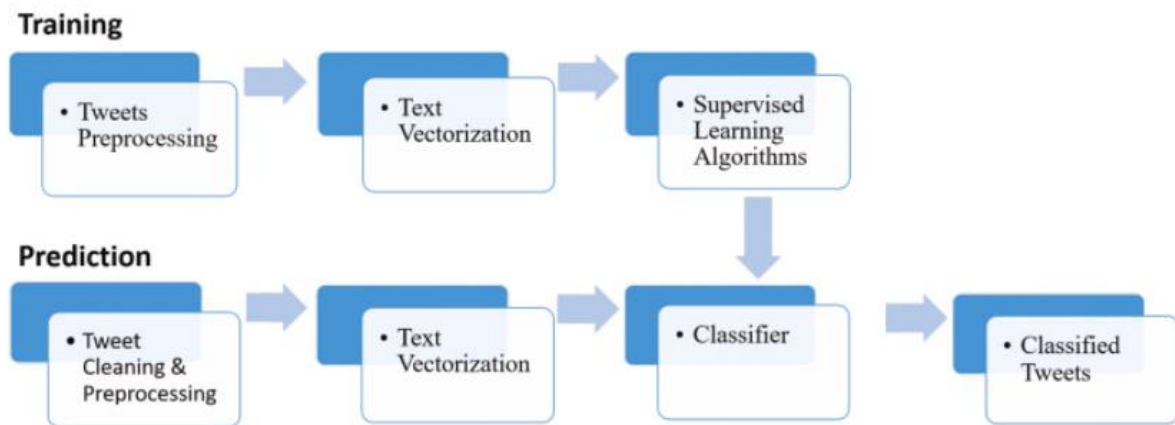
- ✓ **System** : Intel Core i3, i5, i7 and 2GHz Minimum
- ✓ **RAM** : 4GB or above
- ✓ **Hard Disk** : 10GB or above
- ✓ **Input** : Keyboard and Mouse
- ✓ **Output** : Monitor or PC

##### **Software Requirements**

- ✓ **OS** : Windows 8 or Higher Versions
- ✓ **Platform** : Jupyter Notebook, Google Colab
- ✓ **Program Language** : Python

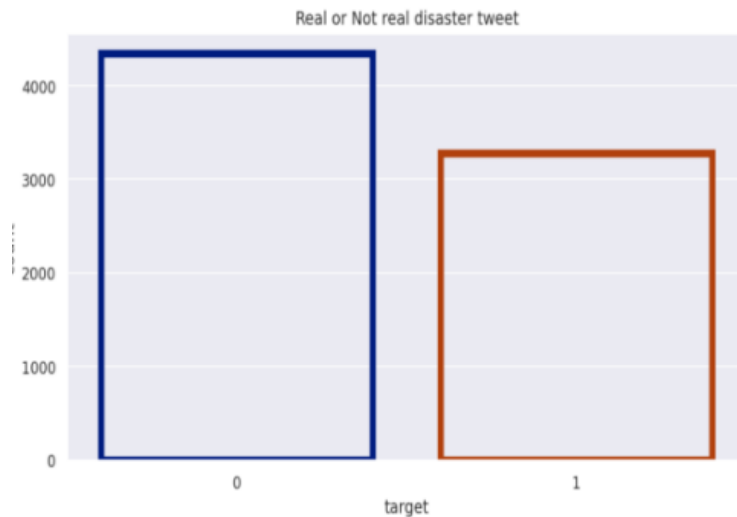


### 3.1 FLOW CHART



### 4. DATA SET

In this study, we used the dataset from Kaggle. The dataset could be downloaded from <https://www.kaggle.com/c/nlp-getting-started/data>. We have 10873 comments, and we estimate whether or not a particular tweet is about a specific catastrophe. From 10,873 data, 57.03% were not real disasters. Figure 1 described the features and specifications. Data set contains: Id – tweet identifier, text - text of the tweet, location - location the tweet was sent from (maybe NaN), keyword - A relevant keyword in the tweet (maybe NaN), target - Output that tells if a tweet is a real disaster (1) or not (0).



**Figure 1.** Emotion and count Graph

Our data set contains:

- 1.ID
- 2.The text of a tweet
- 3.A keyword from that tweet (although this may be blank!)
- 4.The location the tweet was sent from.
- 5.Target(0 or 1)

	id	keyword	location	text	target
3040	4362	earthquake	NaN	#USGS M 0.9 - Northern California: Time2015-08...	1
3041	4363	earthquake	TiÃchira - Venezuela	#SCSeEstaPreparando Light mag. 4.4 earthquake ...	1
3042	4365	earthquake	California, USA	#USGS M 1.2 - 23km S of Twentynine Palms Calif...	1
3043	4366	earthquake	a box	@AGeekyFangirl14 's things she looks in a sign...	0
3044	4368	earthquake	Melbourne, Australia	Nepal earthquake 3 months on: Women fear abuse...	1
3045	4372	earthquake	Barcelona, Spain	ML 2.0 SICILY ITALY <a href="http://t.co/z6hxx6d2pm">http://t.co/z6hxx6d2pm</a> #eu...	0
3046	4373	earthquake	Hawaii, USA	USGS reports a M1.94 #earthquake 5km S of Volc...	1
3047	4374	earthquake	New Zealand	GNS sees unnecessary deaths resulting from ear...	1
3048	4375	earthquake	NaN	'There was a small earthquake in LA but don't ...	1

**Figure.2** Data Set Sample

## 5. DATA PRE-PROCESSING

The collected dataset will be examined and prepared, since the dataset contains tweets text, data cleaning steps will involve removing URLs, symbols, emojis, accounts mention (@user), hash-tags (#), as well as dealing with null values within the dataset. In addition, the dataset will be split into training and testing subsets. The aim is to have a clean dataset prior to the model building step.

In NLP, text pre-processing is the first step in the process of building a model. The various text pre-processing steps are: Tokenization, Lower casing, Stop words removal, Stemming and Lemmatization.

**Tokenization:** Splitting the sentence into words.

**Lower casing:** Converting a word to lower case (NLP -> nlp).

Words like *Book* and *book* mean the same but when not converted to the lower case those two are represented as two different words in the vector space model (resulting in more dimensions).

**Stop words removal:** Stop words are very commonly used words (a, an, the, etc.) in the documents. These words do not really signify any importance as they do not help in distinguishing two documents.

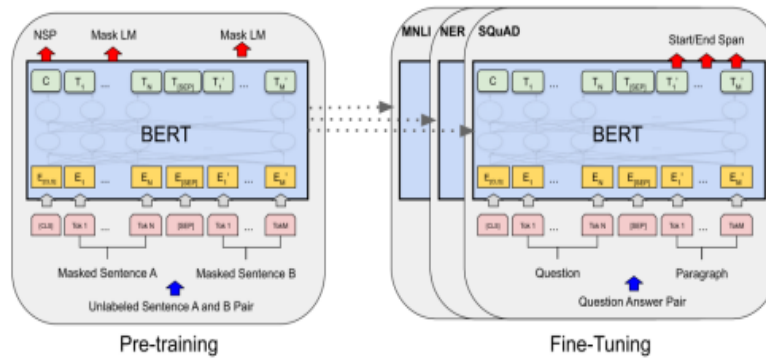
**Stemming:** It is a process of transforming a word to its root form.

**Lemmatization:** Unlike stemming, lemmatization reduces the words to a word existing in the language. For lemmatization to resolve a word to its lemma, part of speech of the word is required. This helps in transforming the word into a proper root form. However, for doing so, it requires extra computational linguistics power such as a part of speech tagger.

Text inputs need to be transformed to numeric token ids and arranged in several Tensors before being input to BERT. TensorFlow Hub provides a matching preprocessing model for each of the BERT models, which implements this transformation using TF ops from the TF.text library. It is not necessary to run pure Python code outside your TensorFlow model to preprocess text.

## 6. METHODOLOGY

BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element. BERT utilizes a transformer, a mechanism of attention that learns the link between words (or sub-words) in a text. The encoder that reads text data and the decoder generates a job forecast in its vanilla form, and the converter comprises two different mechanisms. As BERT's purpose is to construct a language model, it just requires an encoder mechanism. Two steps are used in BERT: pre-training and fine-tuning. During pre-training, the model is conditioned on unlabelled data in separate pre-training activities. Pertained parameters launch BERT models to be fine targeted, and the named downstream data refines all parameters, whose procedure we can see in Figure 1.



**Figure 1.** General pre-training and fine-tuning procedures for BERT[7].

There are two steps in our framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabelled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labelled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The question-answering example in Figure 1 will serve as a running example for this section. A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture. Model Architecture BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017) and released in the tensor2tensor library.<sup>1</sup> Because the use of Transformers has become common and our implementation is almost identical to the original, we will omit an exhaustive background

description of the model architecture and refer readers to Vaswani et al. (2017) as well as excellent guides such as “The Annotated Transformer.”<sup>2</sup> In this work, we denote the number of layers (i.e., Transformer blocks) as  $L$ , the hidden size as  $H$ , and the number of self-attention heads as  $A$ .<sup>3</sup> We primarily report results on two model sizes: BERTBASE ( $L=12$ ,  $H=768$ ,  $A=12$ , Total Parameters=110M) and BERTLARGE ( $L=24$ ,  $H=1024$ ,  $A=16$ , Total Parameters=340M).

	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

**Figure 2.** Trained Dataset.

There are multiple BERT models available.

**BERT-Base, Uncased** and **seven more models** with trained weights released by the original BERT authors.

**Small BERTs** have the same general architecture but fewer and/or smaller Transformer blocks, which lets you explore tradeoffs between speed, size and quality.

**ALBERT:** four different sizes of "A Lite BERT" that reduces model size (but not computation time) by sharing parameters between layers.

**BERT Experts:** eight models that all have the BERT-base architecture but offer a choice between different pre-training domains, to align more closely with the target task.

**Electra** has the same architecture as BERT (in three different sizes), but gets pre-trained as a discriminator in a set-up that resembles a Generative Adversarial Network (GAN). BERT with Talking-Heads Attention and Gated GELU [**base, large**] has two improvements to the core of the Transformer architecture.

## 6.1 BERT MODELS

The BERT models return a map with 3 important keys: `pooled_output`, `sequence_output`, `encoder_outputs`:

**pooled\_output** represents each input sequence as a whole. The shape is `[batch_size, H]`. You can think of this as an embedding for the entire movie review.

**sequence\_output** represents each input token in the context. The shape is `[batch_size, seq_length, H]`. You can think of this as a contextual embedding for every token in the movie review.

**encoder\_outputs** are the intermediate activations of the `L` Transformer blocks. `outputs["encoder_outputs"][i]` is a Tensor of shape `[batch_size, seq_length, 1024]` with the outputs of the `i`-th Transformer block, for  $0 \leq i < L$ . The last value of the list is equal to `sequence_output`.

For the fine-tuning you are going to use the `pooled_output` array.

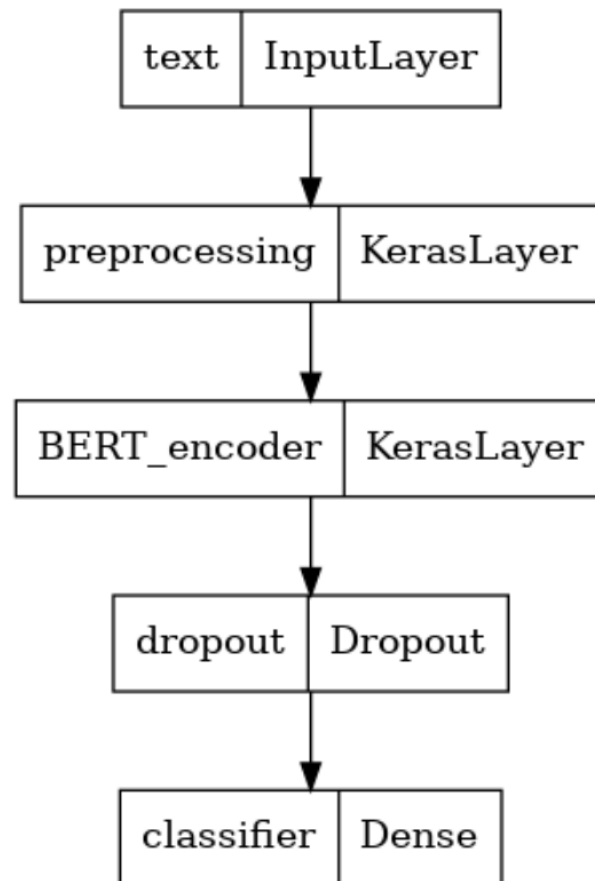
### Define your model

the preprocessing model, the selected BERT model, one Dense and a Dropout layer.

```
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
    return tf.keras.Model(text_input, net)
```

look at the model's structure.

```
tf.keras.utils.plot_model(classifier_model)
```



## Model training

You now have all the pieces to train a model, including the preprocessing module, BERT encoder, data, and classifier.

### Loss function

Since this is a binary classification problem and the model outputs a probability (a single-unit layer), you'll use **losses.BinaryCrossentropy** loss function.

```
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
metrics = tf.metrics.BinaryAccuracy()
```

## Optimizer

For fine-tuning, let's use the same optimizer that BERT was originally trained with: the "Adaptive Moments" (Adam). This optimizer minimizes the prediction loss and does regularization by weight decay (not using moments), which is also known as AdamW.

For the learning rate (`init_lr`), you will use the same schedule as BERT pre-training: linear decay of a notional initial learning rate, prefixed with a linear warm-up phase over the first 10% of training steps (`num_warmup_steps`). In line with the BERT paper, the initial learning rate is smaller for fine-tuning

Using the `classifier_model` you created earlier, you can compile the model with the loss, metric and optimizer.

```
epochs = 5
steps_per_epoch = tf.data.experimental.cardinality(train_ds).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                         num_train_steps=num_train_steps,
                                         num_warmup_steps=num_warmup_steps,
                                         optimizer_type='adamw')
```

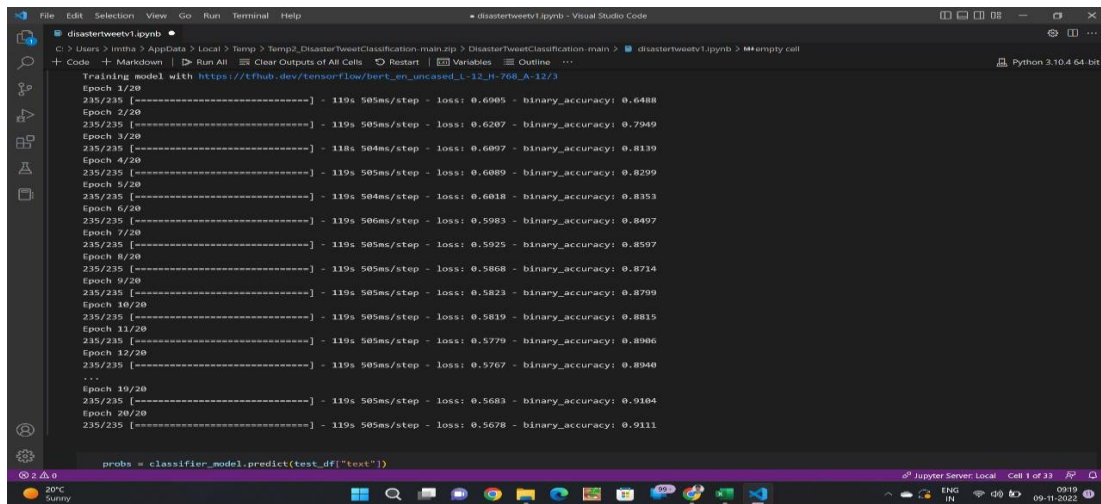
## Loading the BERT model and training

Using the `classifier_model` you created earlier, you can compile the model with the loss, metric and optimizer.

```
classifier_model.compile(optimizer=optimizer,
                        loss=loss,
                        metrics=metrics)
```

```
print(f'Training model with {tfhub_handle_encoder}')
history = classifier_model.fit(x=train_ds,
                              validation_data=val_ds,
                              epochs=epochs)
```





```
disastertweetv1.ipynb - Visual Studio Code
C:\Users\imitha> AppData\Local\Temp\Temp2_DisasterTweetClassification-main.zip\DisasterTweetClassification-main> disastertweetv1.ipynb
Training model with https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3
Epoch 1/20
235/235 [=====] - 119s 505ms/step - loss: 0.6905 - binary_accuracy: 0.6488
Epoch 2/20
235/235 [=====] - 119s 505ms/step - loss: 0.6207 - binary_accuracy: 0.7949
Epoch 3/20
235/235 [=====] - 118s 504ms/step - loss: 0.6097 - binary_accuracy: 0.8139
Epoch 4/20
235/235 [=====] - 119s 505ms/step - loss: 0.6089 - binary_accuracy: 0.8299
Epoch 5/20
235/235 [=====] - 119s 504ms/step - loss: 0.6018 - binary_accuracy: 0.8353
Epoch 6/20
235/235 [=====] - 119s 506ms/step - loss: 0.5983 - binary_accuracy: 0.8497
Epoch 7/20
235/235 [=====] - 119s 505ms/step - loss: 0.5925 - binary_accuracy: 0.8597
Epoch 8/20
235/235 [=====] - 119s 505ms/step - loss: 0.5868 - binary_accuracy: 0.8714
Epoch 9/20
235/235 [=====] - 119s 505ms/step - loss: 0.5823 - binary_accuracy: 0.8799
Epoch 10/20
235/235 [=====] - 119s 505ms/step - loss: 0.5819 - binary_accuracy: 0.8815
Epoch 11/20
235/235 [=====] - 119s 505ms/step - loss: 0.5779 - binary_accuracy: 0.8906
Epoch 12/20
235/235 [=====] - 119s 505ms/step - loss: 0.5767 - binary_accuracy: 0.8940
...
Epoch 19/20
235/235 [=====] - 119s 505ms/step - loss: 0.5683 - binary_accuracy: 0.9104
Epoch 20/20
235/235 [=====] - 119s 505ms/step - loss: 0.5678 - binary_accuracy: 0.9111
probs = classifier_model.predict(test_df["text"])
```

i

## Evaluate the model

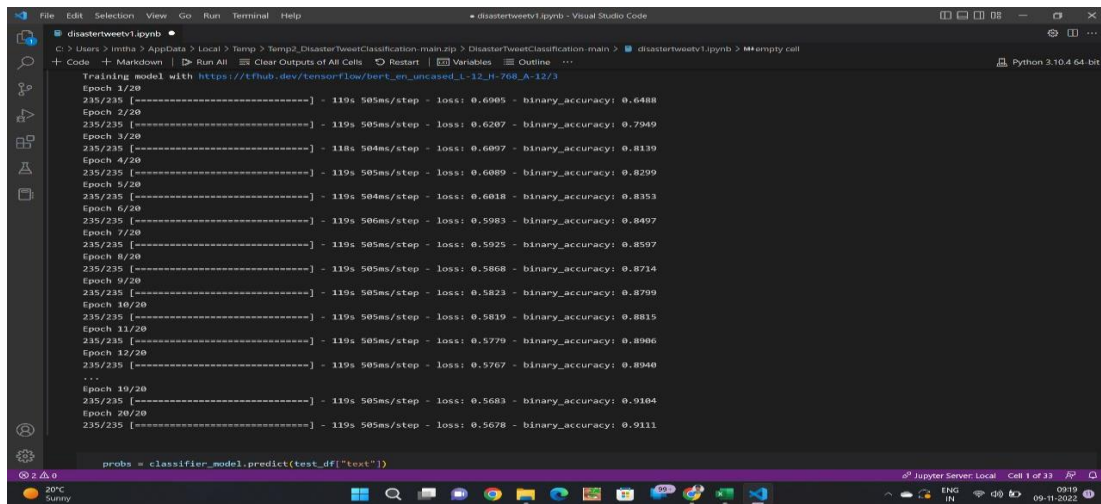
Let's see how the model performs. Two values will be returned. Loss (a number which represents the error, lower values are better), and accuracy.

```
loss, accuracy = classifier_model.evaluate(test_ds)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
```

## 7. RESULTS

In this paper, we have conducted a case study of two natural disasters. First, we have collected tweets posted during the natural disasters on Twitter, and then analyzed distinguishing patterns among legitimate, spam and fake tweets. Our experimental results show that detecting fake and spam tweets during natural disasters is possible with a accuracy of 91%



```
disastertweet1.ipynb
C:\Users\imitha> AppData\Local\Temp\Temp2_DisasterTweetClassification-main.zip\DisasterTweetClassification-main> disastertweet1.ipynb
Training model with https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3
Epoch 1/20 [-----] - 119s 505ms/step - loss: 0.6905 - binary_accuracy: 0.6488
Epoch 2/20 [-----] - 119s 505ms/step - loss: 0.6207 - binary_accuracy: 0.7949
Epoch 3/20 [-----] - 118s 504ms/step - loss: 0.6097 - binary_accuracy: 0.8139
Epoch 4/20 [-----] - 119s 505ms/step - loss: 0.6089 - binary_accuracy: 0.8299
Epoch 5/20 [-----] - 119s 504ms/step - loss: 0.6018 - binary_accuracy: 0.8353
Epoch 6/20 [-----] - 119s 506ms/step - loss: 0.5983 - binary_accuracy: 0.8497
Epoch 7/20 [-----] - 119s 505ms/step - loss: 0.5925 - binary_accuracy: 0.8597
Epoch 8/20 [-----] - 119s 505ms/step - loss: 0.5868 - binary_accuracy: 0.8714
Epoch 9/20 [-----] - 119s 505ms/step - loss: 0.5823 - binary_accuracy: 0.8799
Epoch 10/20 [-----] - 119s 505ms/step - loss: 0.5819 - binary_accuracy: 0.8815
Epoch 11/20 [-----] - 119s 505ms/step - loss: 0.5779 - binary_accuracy: 0.8906
Epoch 12/20 [-----] - 119s 505ms/step - loss: 0.5767 - binary_accuracy: 0.8940
...
Epoch 19/20 [-----] - 119s 505ms/step - loss: 0.5683 - binary_accuracy: 0.9104
Epoch 20/20 [-----] - 119s 505ms/step - loss: 0.5678 - binary_accuracy: 0.9111
probs = classifier_model.predict(test_df["text"])
```

The predictions made are pretty good; the model created can predict accurately determine a real disaster (target) and which one is non-disaster (non-target).

## 8. CONCLUSION

BERT (Bidirectional Encoder Representations from Transformers) is a Google-designed, profound learning model. Since Google opened it, several scientists and companies have embraced it and have applied it to many text classification tasks. Therefore, we use BERT in this paper to some earthquake tweets. This research will help rescue and emergency responders establish effective knowledge management techniques for a rapidly evolving disaster environment. The predictions made are pretty good; the model created can predict accurately determine a real disaster (target) and which one is nondisaster (non-target).

## 9. REFERENCES

[1] Sherine Rady, and Mostafa Aref, “A Deep Learning Architecture with Word Embeddings to Classify Sentiment in Twitter”, Springer Nature Switzerland AG 2021, A. E. Hassanien et al. (Eds.): AISI 2020, AISC 1261, pp. 115-125,0032021

<https://ieeexplore.ieee.org/document/7396773>

<https://www.sciencedirect.com/science/article/pii/S1877050917319038>

<https://onlinelibrary.wiley.com/doi/abs/10.1111/risa.13634>

<https://www.researchgate.net/publication/338493122> Social media for improved process management in organizations during disasters

<https://ieeexplore.ieee.org/document/9156031>

<https://jalammar.github.io/illustrated-transformer/>

[https://www.tensorflow.org/text/tutorials/classify\\_text\\_with\\_bert](https://www.tensorflow.org/text/tutorials/classify_text_with_bert)

---