[Explore](https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/)

[S](https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/)

[Master Generative AI: Your clear, step-by-step guide to become a certified GenAI expert!](https://www.analyticsvidhya.com/genaipinnacle/genai-roadmap?utm_source=blog_india&utm_medium=desktop_flashstrip&utm_campaign=29-Apr-2024||&utm_content=roadmap)

[Download Roadmap](https://www.analyticsvidhya.com/genaipinnacle/genai-roadmap?utm_source=blog_india&utm_medium=desktop_flashstrip&utm_campaign=29-Apr-2024||&utm_content=roadmap)

[Home](https://www.analyticsvidhya.com/blog/)

[Advanced](https://www.analyticsvidhya.com/blog/category/advanced/)

Manual for the First Time Users: Google BERT for Text Classification

**Manual for the First Time Users: Google BERT for Text Classification**

[A](https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/)

[Abhishek Jaiswal](https://www.analyticsvidhya.com/blog/author/datanuts/)31 Dec, 2021 • 9 min read

This article was published as a part of the [Data Science Blogathon](https://datahack.analyticsvidhya.com/contest/data-science-blogathon-15/)



                                                                                       Source: huggingface.io

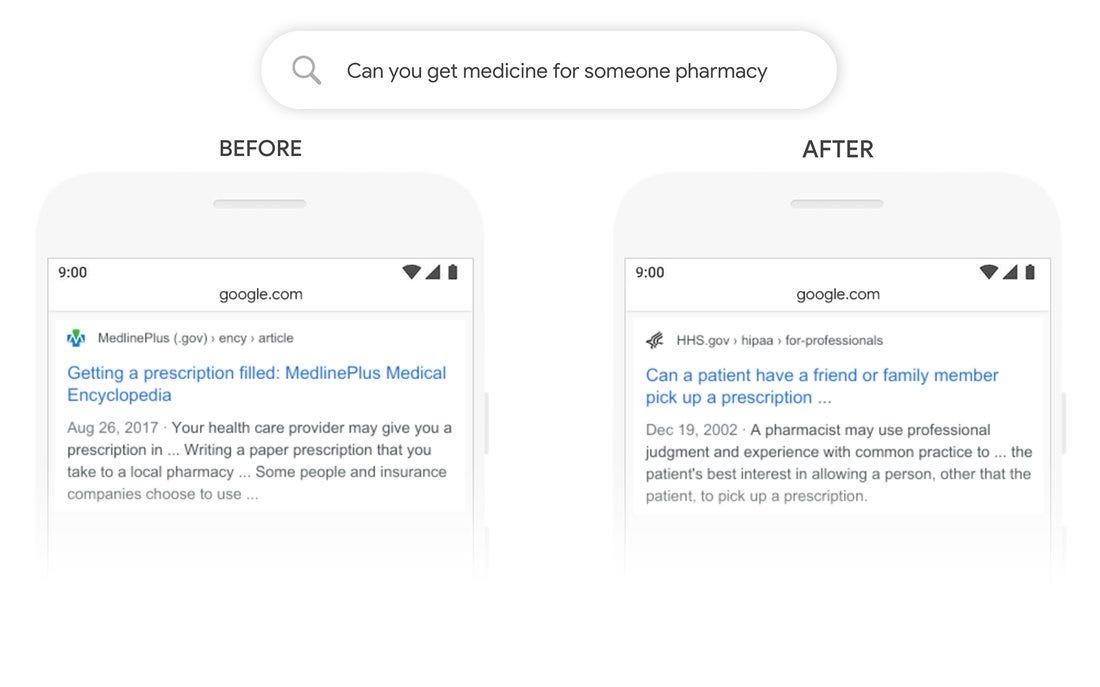
Hey Folks!

In this article, we will understand BERT from absolute zero and we will see BERT in action and its implementation

**we will Understand Bert By answering these Questions**

**1. what is the core idea behind its Working?2. Why do we need BERT?3. How does it work?4. How can we use it?5. Fine Tuning of BERT Using BERT for Text Classification**

Have you ever wondered how Google search has improved in the past few years?



                                                                                            Source: OpenSource

the above result shows how BERT Implementation on Google’s Search has improved its search results.

In the past few years, we have started implementing the Deep learning concepts in the field of NLP and we performed many NLP tasks using Deep Learning.

**Core Idea of BERT**

BERT(Bidirectional Encoder Representation of Transformers)

Bert works on language modelling. Language modelling basically means the understanding of a particular language and this is done by “fill in the blanks” based on context.

for example,

A Boy Went to School and took \_\_\_\_\_ bag with him.

a language model will be able to fill the blank by “school”.BERT uses MASKED LM (MLM), it’s a very powerful training mechanism where BERT randomly masks (hide) words and tries to predict in the sentence.

Since BERT is a Bidirectional Model, it tries to look from both directions left-right + right-left.

to predict a masked word BERT takes both the next token and previous token of the masked word into consideration for prediction.

**Why BERT is so powerful?**

1. *Bert is based on the Transformer model architecture with attention layers.*

2. *it’s a context-based language representation. it means the meaning of the word will be based on the meaning of the sentence.*

**Transformer Mechanism**

A transformer uses an attention mechanism to learn the relationship between all words in a sentence.

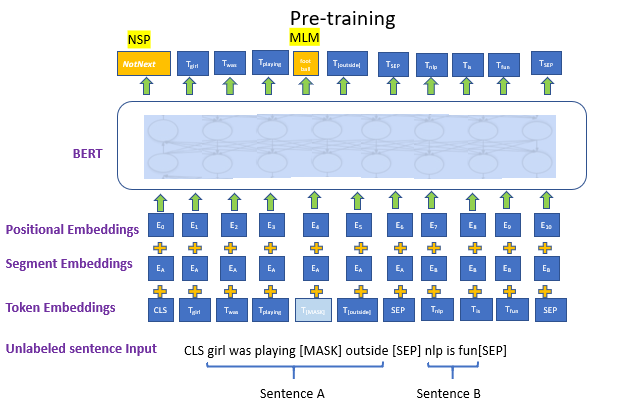
**Why do we need BERT?**

In NLP our biggest challenge was to create a big enough dataset that can be trained on a model to understand a language.

Training on a Language is not an easy task, and BERT is trained on trillions of wiki pages, which make it super-powerful to understand the context of any sentence.

Bert can be used for free, it can be easily fine-tuned and easily implementable in our use case.

**Working of BERT**



Source: OpenSourceBERT works on Transformer’s attention mechanism that helps BERT to understand the relationships between words in a sentence.

A Basic Transformer contains an encoder part ( reads the input ) and a decoder part (produces prediction). Bert only makes use of the encoder part. The encoder part of Bert takes the tokens sequence as input after converting tokens into some vector form.

The detailed work on Transformers is published in a [paper](https://arxiv.org/pdf/1706.03762.pdf) by Google Team.

**How to Implement BERT**

**steps involved**

**1.Getting the BERT model from the TensorFlow hub**

**2.Build a Model according to our use case using BERT pre-trained layers.**

**3.Setting the tokenizer**

**4.Loading the dataset and preprocessing it**

**5.Model Evaluation**

Getting the Bert

there are multiple ways to get the pre-trained models, either Tensorflow hub or hugging-face’s transformers package.

loading model from the [TensorFlow hub](https://www.tensorflow.org/hub" \t "_blank).

Tensorflow hub provides a wide range of pre-trained models

**accessing Tensorflow-hub**

!pip install --upgrade tensorflow\_hub

import tensorflow\_hub as hub

import numpy as np

Load the BERT model

*## loading bert from tensorhub*

module\_url = "https://tfhub.dev/tensorflow/bert\_en\_uncased\_L-24\_H-1024\_A-16/1"

bert\_layer = hub.KerasLayer(module\_url, trainable=False)

trainable = False freezing the pre-trained Bert layers as we don’t want to retrain Bert layers.

BERT Model Version**bert\_en\_uncased\_L-24\_H-1024\_A-16 model**

**L=24 hidden layers** (Transformer blocks),

**H=1024**Hidden Layers

**A=16** attention heads.

This model is trained on the Wikipedia and BooksCorpus Dataset. **en\_uncased** signifies that the model is pre-trained for the English language and its case insensitive.

Loading the tokenizer

for the training, we need to parse our textual dataset into BERT-supported input format. In order to do this, we first tokenize our dataset and then convert it into features (encoding into some numbers)

**Splitting a sentence into its individual words is called tokenization.**

Import tokenizer file

!wget --quiet <https://raw.githubusercontent.com/tensorflow/models/master/official/nlp/bert/tokenization.py>

import tokenization

**Setting up the tokenizer**

vocab\_file = bert\_layer.resolved\_object.vocab\_file.asset\_path.numpy()

do\_lower\_case = bert\_layer.resolved\_object.do\_lower\_case.numpy()

tokenizer = tokenization.FullTokenizer(vocab\_file, do\_lower\_case)

**vocab\_file** it’s a vocabulary file for mapping our dataset into features.

**do\_lower\_case** lowering the generated tokens

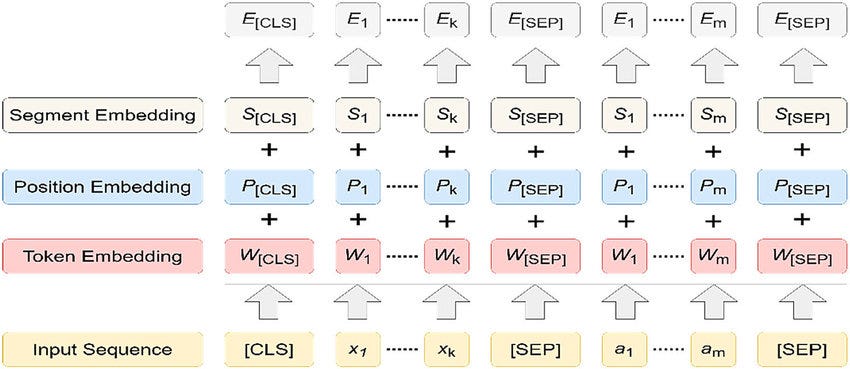
The **FullTokenizer**class takes **vocab\_file** as input parameters.

**calling tokenizer:**

tokenizer.tokenize('Where are \* you going ?')

https://cdn-images-1.medium.com/max/1500/1*msNje1Ou9ilmiAU7VlLy2w.png

Understanding Input Data Format

                                                                                       Source: OpenSource

BERT inputs a combination of 3 different data format

**Token Embeddings**

Token Embedding holds the information of our dataset. it’s a number assigned to each unique words tokens

[CLS] token is attached at the beginning of every sentence that indicates the starting

[SEP] token is attached at the end of each and every sentence that indicates the ending of a sentence.

**Position Embeddings**

It is used to indicate the position of tokens in a sentence.

this helps BERT to capture the sequence or order of information given in a sentence.

**Segment Embeddings**

The model must know whether a particular token belongs to Sentence 1 or sentence 2.

In BERT. This is done by generating a fixed token, called the segment embedding

Till now we have discussed BERT, its input format, how to load the BERT model.

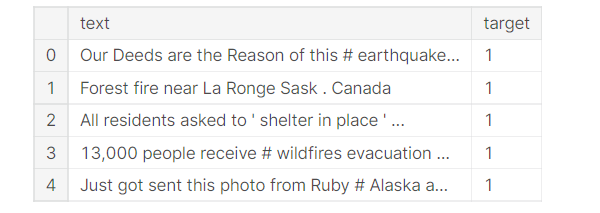
Loading the dataset

we will be using the Disaster Tweets dataset, download dataset [link](https://www.kaggle.com/vbmokin/nlp-with-disaster-tweets-cleaning-data).

this dataset contains training and testing files.

train = pd.read\_csv("../input/nlp-with-disaster-tweets-cleaning-data/train\_data\_cleaning.csv", usecols=['text','target'])

test = pd.read\_csv("../input/nlp-with-disaster-tweets-cleaning-data/test\_data\_cleaning.csv", usecols = ['text'])



if the target is 1 then Disastrous Tweet otherwise normal tweet

Pre-Processing Dataset into BERT Format

as we know BERT inputs the data for training is a combination of 3 /2 embeddings. so in this step, we will prepare our dataset in BERT input Format.

**Required Libraries:**

from tensorflow.keras.layers import Dense, Input

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.models import Model

the function **bert\_encoder**takes textual data and tokenizer and creates token\_embeddings,positional\_embeddings, and segment\_embedding which will be passed in our model for training

Bert supports max length up to 512 only

def bert\_encoder(texts, tokenizer, max\_len=512):

*# here we need 3 data inputs for bert training and fine tuning*

all\_tokens = []

all\_masks = []

all\_segments = []

for text **in** texts:

text = tokenizer.tokenize(text)

text\_sequence = text[:max\_len-2] *# here we are trimming 2 words if they getting bigger than 512*

input\_sequences = ["[CLS]"] + text\_sequence + ["[SEP]"]

pad\_len = max\_len - len(input\_sequences)

tokens = tokenizer.convert\_tokens\_to\_ids(input\_sequences)

tokens += [0] \* pad\_len

pad\_masks = [1] \* len(input\_sequences) + [0] \* pad\_len

segment\_ids = [0] \* max\_len

all\_tokens.append(tokens)

all\_masks.append(pad\_masks)

all\_segments.append(segment\_ids)

return np.array(all\_tokens), np.array(all\_masks), np.array(all\_segments)

**bert\_encoder**takes tokenizer and text data as input and returns 3 different lists of mask/position embedding, segment embedding, token embedding.

**convert\_tokens\_to\_ids** it maps our unique tokens to the vocab file and assigns unique ids to the unique tokens.

**max\_length = 512**, the maximum length of our sentence in the dataset

**Note:**Token Embedding and Positional Embedding are necessary to pass for BERT Training

**Calling the encoding function:**

train\_input = bert\_encoder(train.text.values, tokenizer, max\_len=160)

**max\_len = 160**since the length of most tweets is within 150 words.

the **train\_input** contains a list of 3 arrays (all\_tokens, all\_masks,all\_segments)

Building model using BERT layers

We need to design a model according to our use case using BERT pre-trained model by adding some CNN layers which will give us end prediction.

def build\_model(bert\_layer, max\_len=512, num\_class):

input\_word\_ids = Input(shape=(max\_len,), dtype=tf.int32, name="input\_word\_ids")

input\_mask = Input(shape=(max\_len,), dtype=tf.int32, name="input\_mask")

segment\_ids = Input(shape=(max\_len,), dtype=tf.int32, name="segment\_ids")

\_, sequence\_output = bert\_layer([input\_word\_ids, input\_mask, segment\_ids])

clf\_output = sequence\_output[:, 0, :]

out = Dense(num\_class, activation='sigmoid')(clf\_output)

model = Model(inputs=[input\_word\_ids, input\_mask, segment\_ids], outputs=out)

model.compile(Adam(lr=2e-6), loss='binary\_crossentropy', metrics=['accuracy'])

return model

the function **build\_model** takes Bert layer, **max\_len** and **num\_class** as input and returns the final model

default **max\_len = 512 .**

**num\_class = 1** the final dense layer with 1 output will predict the probability of tweets to be disastrous.

BERT layers take an array of 3 /2 embeddings for training[[input\_words\_tokens][input\_maks][segement\_ids]] hence we need to create 3 input layers of the size equal to **max\_len.**

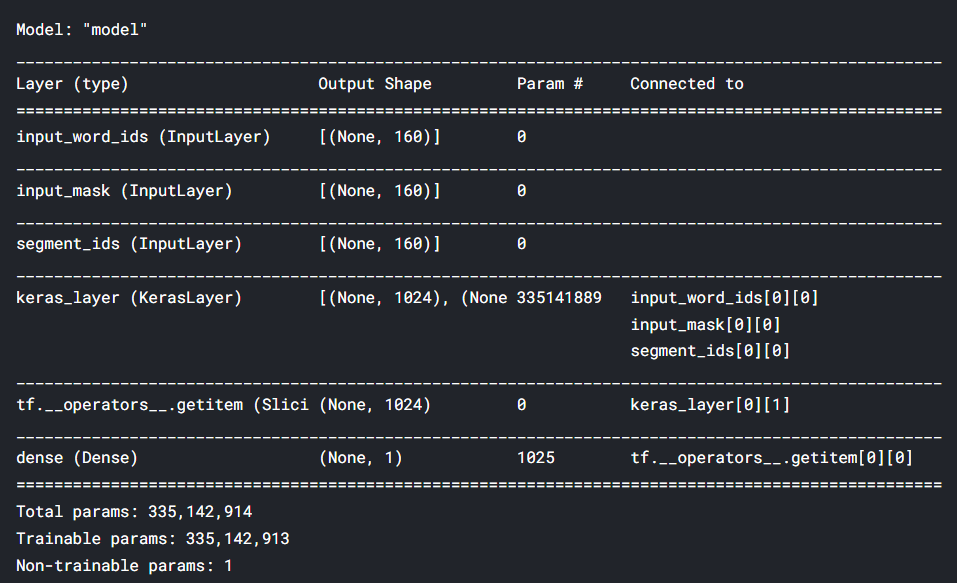
binary\_cross\_entropy for binary classification

**sequence\_output[:, 0, :]** intermediate hidden states.

the **model\_final** will be our final model which we will use for training.

model\_final = build\_model(bert\_layer, max\_len=160, num\_class = 1)

model\_final.summary()

                                                                                                                                                                 Source. [Kaggle.com](https://www.kaggle.com/preatcher)

Training Step

So far we have built our model and the data embeddings to be passed for training.

It’s time to begin the training.

train\_history = final\_model.fit(

train\_input, train\_labels,

validation\_split=0.2,

epochs=3,

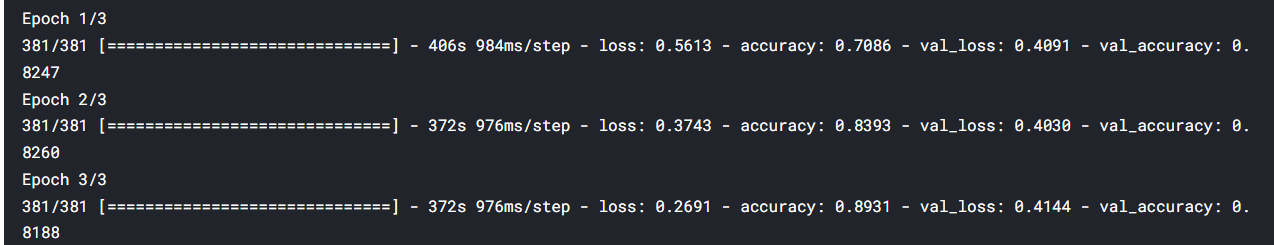
batch\_size=16

)

final\_model.save('model.h5')

**validation split = 0.2** signifies that 20 % of the training data will be used as validation data.

**train\_label** is the target array

                                                                                Source: [Kaggle.com](https://www.kaggle.com/preatcher)

Awesome!!

we just ran 3 epochs and got a validation accuracy of 82%

Testing and validation

for the testing and prediction, the test data must be in the same format as training data.

Calling the **bert\_encoder** function on the test data will convert it into 3 embeddings and that will be passed to the **model.predict** method.

test\_input = bert\_encoder(test.text.values, tokenizer, max\_len=160)

test\_pred = final\_model.predict(test\_input)

prediction = np.where(test\_pred>.5, 1,0)

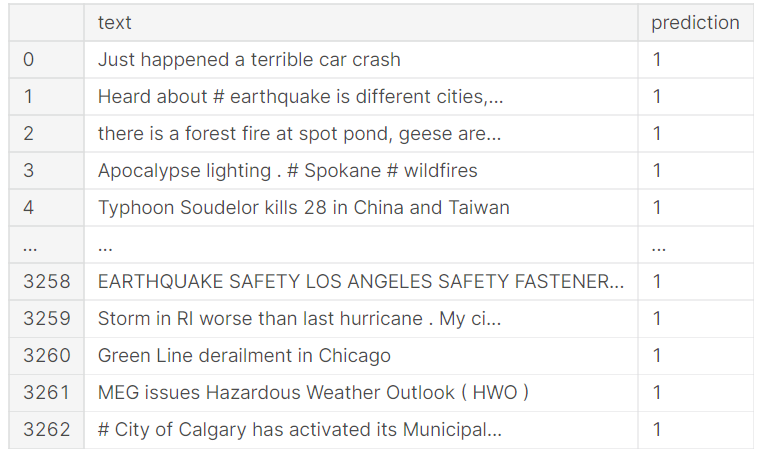
prediction is an array containing the probability of a tweet to be disastrous. and if the probability is greater than 0.5 we will categorize that as disastrous and label that as 1

test['prediction'] = prediction

**results:**

filtering tweets according to our prediction.

test[test.prediction == 1]

                                                                       Source: [Kaggle](https://www.kaggle.com/preatcher" \t "_blank)

Perfect!!

all our tweets predicted to be disastrous reads to be disastrous.

Improving the Result

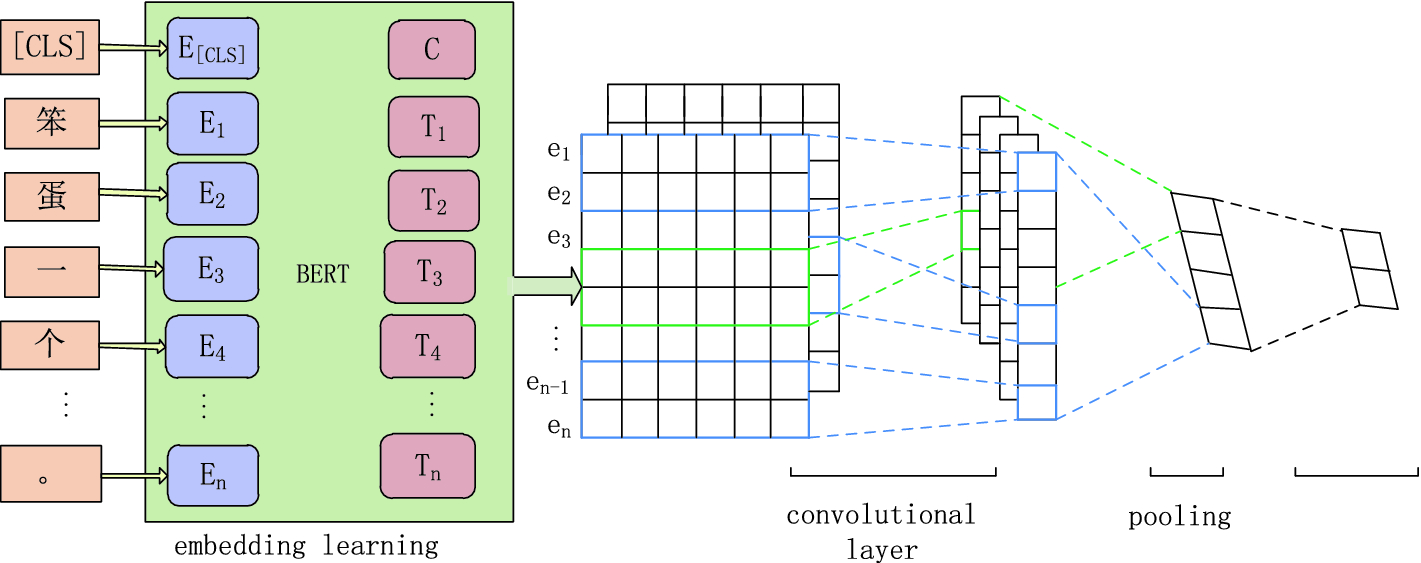
the pre-trained model gives awesome results in a few epochs. but you can further improve the results by doing some tweaking:

Use callbacks and dynamic learning rates for efficient training.

Use a deeper BERT architecture ie. **bert\_large** has more layers and it can learn comparatively more information

Use Stacked BERT layers

Add multiple CNN layers on top of BERT layers

                                                                                                                                                                Source: Springer Link

Conclusion

BERT is an advanced and very powerful language representation model that can be implemented for many tasks like question answering, text classification, text summarization, etc.

in this article, we learned how to implement BERT for text classification and saw it working.

Implementing BERT using the transformers package is a lot easier. in the next article, we will discuss implementing NLP models in no time using the transformers package.

Download the Source code using the [link](https://www.kaggle.com/preatcher/bert-disaster-tweets).

if you have something to ask me write to me on [Linkedin](http://www.linkedin.com/in/iamabhishek26102001" \t "_blank)

**The media shown in this article is not owned by Analytics Vidhya and are used at the Author’s discretion.**

[BERT](https://www.analyticsvidhya.com/blog/tag/bert/)[blogathon](https://www.analyticsvidhya.com/blog/tag/blogathon/)[Google BERT](https://www.analyticsvidhya.com/blog/tag/google-bert/)[text classification](https://www.analyticsvidhya.com/blog/tag/text-classification/)

[A](https://www.analyticsvidhya.com/blog/author/datanuts/)

[Abhishek Jaiswal](https://www.analyticsvidhya.com/blog/author/datanuts/)31 Dec 2021

[Advanced](https://www.analyticsvidhya.com/blog/category/advanced/)[Classification](https://www.analyticsvidhya.com/blog/category/classification/)[NLP](https://www.analyticsvidhya.com/blog/category/nlp/)[Python](https://www.analyticsvidhya.com/blog/category/python-2/)[Text](https://www.analyticsvidhya.com/blog/category/text/)

Responses From Readers

Top of Form



Submit reply

Bottom of Form