

🌋 Natural Language Processing with Disaster Tweets



Source: [EarthDailyAnalytics](#)

Competition Description

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programmatically monitoring Twitter (i.e. disaster relief organizations and news agencies).

But, it's not always clear whether a person's words are actually announcing a disaster. Take this example:



12:43 AM · Aug 6, 2015 · Twitter for Android

The author explicitly uses the word “ABLAZE” but means it metaphorically. This is clear to a human right away, especially with the visual aid. But it’s less clear to a machine.

In this competition, we’re challenged to build a machine learning model that **predicts which Tweets are about real disasters and which one's aren't**. We’ll have access to a dataset of **10,000 tweets** that were hand classified.

Disclaimer: The dataset for this competition contains text that may be considered profane, vulgar, or offensive.

Acknowledgments

This dataset was created by the company figure-eight and originally shared on their ‘Data For Everyone’ website [here](#).

Tweet source: <https://twitter.com/AnyOtherAnnaK/status/629195955506708480>

Dataset Description

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

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

What am I predicting?

You are predicting whether a given tweet is about a real disaster or not. If so, predict a 1. If not, predict a 0.

Evaluation

Submissions are evaluated using F1 between the predicted and expected answers.

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP+FP}$$

$$recall = \frac{TP}{TP+FN}$$

and:

True Positive [TP] = your prediction is 1, and the ground truth is also 1 - you predicted a positive and that's true!
False Positive [FP] = your prediction is 1, and the ground truth is 0 - you predicted a positive, and that's false.
False Negative [FN] = your prediction is 0, and the ground truth is 1 - you predicted a negative, and that's false.

From Kaggle to Colab environment

```
In [1]: # IMPORTANT: SOME KAGGLE DATA SOURCES ARE PRIVATE
# RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES.
import kagglehub
import os
import shutil
def download_dataset():
    for dir in ["kaggle", "kaggle/input", "kaggle/input/nlp-getting-started"]
        if not os.path.exists(dir):
            os.mkdir(dir)

    nlp_getting_started_path = kagglehub.competition_download('nlp-getting-started')
    rtatman_english_word_frequency_path = kagglehub.dataset_download('rtatman/rtatman_english_word_frequency')
    yk1598_479k_english_words_path = kagglehub.dataset_download('yk1598/479k_english_words')

    target_dir = "kaggle/input/nlp-getting-started/"
    file_names = os.listdir(nlp_getting_started_path)
    for file_name in file_names:
```

```

        shutil.move(os.path.join(nlp_getting_started_path, file_name), target)

    target_dir = "kaggle/input/479k-english-words/"
    file_names = os.listdir(yk1598_479k_english_words_path)
    for file_name in file_names:
        shutil.move(os.path.join(yk1598_479k_english_words_path, file_name), target)

    target_dir = "kaggle/input/english-word-frequency/"
    file_names = os.listdir(rtatman_english_word_frequency_path)
    for file_name in file_names:
        shutil.move(os.path.join(rtatman_english_word_frequency_path, file_name), target)

    print("Data source import complete.")
    return

```

download_dataset()

Downloading from https://www.kaggle.com/api/v1/competitions/data/download-all/nlp-getting-started...

100%|██████████| 593k/593k [00:00<00:00, 29.0MB/s]

Extracting files...

Downloading from https://www.kaggle.com/api/v1/datasets/download/rtatman/english-word-frequency?dataset_version_number=1...

100%|██████████| 2.13M/2.13M [00:00<00:00, 23.9MB/s]

Extracting files...

Downloading from https://www.kaggle.com/api/v1/datasets/download/yk1598/479k-english-words?dataset_version_number=1...

100%|██████████| 1.40M/1.40M [00:00<00:00, 19.6MB/s]

Extracting files...

Data source import complete.

Installing some packages

In []:

```
!pip install wordninja num2words mplcyberpunk spacy

!python3 -m spacy download en_core_web_lg
```

Loading packages

In []:

```

import numpy as np
import pandas as pd
import warnings
pd.options.mode.copy_on_write = True
warnings.simplefilter(action='ignore', category=FutureWarning)

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.data import Dataset
from tensorflow.keras import Model, Input, Sequential

```

```
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Dropout,
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, CSVLogger
from tensorflow.keras.regularizers import L2
from tensorflow.keras.metrics import F1Score, BinaryAccuracy
from tensorflow.keras.utils import plot_model
import tensorflow_text as text

from keras_nlp.models import BertPreprocessor, BertBackbone

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score

import spacy
from spacy.matcher import Matcher
from spacy import displacy
import wordninja as wn
from num2words import num2words
from collections import OrderedDict
import string
import re
import itertools
from pprint import pprint

import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS

import cloudpickle

import mplcyberpunk
plt.style.use("cyberpunk")

nlp = spacy.load("en_core_web_lg")

src_path = "./kaggle/input/"
dst_path = "./kaggle/working/"
```

Setting for reproducibility

```
In [4]: import random

seed = 42
os.environ['PYTHONHASHSEED'] = str(seed)
random.seed(seed)
np.random.seed(seed)
```

Creating a word dictionary

Counting the words in the dataset English Words 479k

```
In [7]: with open(src_path+'479k-english-words/english_words_479k.txt', 'r') as txt:  
    words = txt.read().split('\n')  
    words = [x.lower() for x in words]  
  
print(f"Number of words: {len(words)}")
```

Number of words: 466545

```
In [ ]: words[0:8]
```

```
Out[ ]: ['2', '1080', '&c', '10-point', '10th', '11-point', '12-point', '16-point']
```

Counting the words in the dataset English Word Frecuency

```
In [ ]: words2 = pd.read_csv(src_path+'english-word-frequency/unigram_freq.csv')[ 'wo  
  
print(f"Number of words: {len(words2)}")
```

Number of words: 333333

```
In [ ]: words2[0:10]
```

```
Out[ ]: ['the', 'of', 'and', 'to', 'a', 'in', 'for', 'is', 'on', 'that']
```

Cleaning the word list

```
In [ ]: def solve(s):  
    d = dict()  
    for c in s:  
        if c not in d:  
            d[c] = 0  
        d[c] += 1  
    return ''.join(d.keys())  
  
print(f"Total number of words before cleaning: {len(words) + len(words2)}")  
  
all_words = words + [str(x).lower() for x in words2]  
all_words = sorted(list(set(words)))  
  
all_words = [x for x in all_words if (len(x) > 2  
                                         and x.isalpha()  
                                         and len(x.replace("a", "").replace("e", ""))<2  
                                         and len(solve(x)) > 2  
                                         and x not in nlp.Defaults.stop_words)]  
  
print(f"Total number of words after cleaning: {len(all_words)}")
```

```
Total number of words before cleaning: 799878  
Total number of words after cleaning: 414698
```

Let's create it

```
In [ ]: dict_words = {}  
for letter1 in string.ascii_lowercase:  
    dict_words[letter1] = {}  
    for letter2 in string.ascii_lowercase:  
        dict_words[letter1][letter2] = [x for x in all_words if x[0:2] == le  
dict_words
```

Exploratory data analysis

Take a look in the training dataset

```
In [ ]: df = pd.read_csv(src_path+"nlp-getting-started/train.csv", index_col="id")  
df.head()
```

```
Out [ ]:
```

	keyword	location	text	target
id				
1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

```
In [ ]: df_shape = df.shape  
print(f"There {df_shape[1]} features and {df_shape[0]} instances")
```

```
There 4 features and 7613 instances
```

Working on the null data

```
In [ ]: df.isna().sum()
```

```
Out [ ]:
```

keyword	61
location	2533
text	0
target	0
dtype:	int64

```
In [ ]: print(f'The missing values in location feature represent the {round(df.isna().sum().sum())}')  
print(f'The missing values in keyword feature represent the {round(df.isna().sum().sum())}')
```

```
The missing values in location feature represent the 33.27% of our dataset  
The missing values in keyword feature represent the 0.8% of our dataset
```

```
In [ ]: df_cl_nl = df.drop(df[df["keyword"].isna()].index)  
df_cl_nl = df_cl_nl.drop(["location"], axis=1)
```

```
In [ ]: df_cl_nl.isna().sum()
```

```
Out[ ]: keyword      0  
text          0  
target        0  
dtype: int64
```

```
In [ ]: df_shape = df_cl_nl.shape  
print(f"There {df_shape[1]} features and {df_shape[0]} instances")
```

```
There 3 features and 7552 instances
```

Working on the duplicated data

```
In [ ]: mask = df_cl_nl.duplicated(subset=["text"])  
df_cl_nl.loc[mask]
```

```
Out[ ]:
```

	keyword	text	target
id			
68	ablaze	Check these out: http://t.co/rOl2NSmEJJ http:/...	0
165	aftershock	320 [IR] ICEMOON [AFTERSHOCK] http://t.co/vA...	0
172	aftershock	320 [IR] ICEMOON [AFTERSHOCK] http://t.co/TH...	0
238	airplane%20accident	Experts in France begin examining airplane deb...	1
898	bioterrorism	To fight bioterrorism sir.	0
...
10071	typhoon	abcnews - Obama Declares Disaster for Typhoon...	1
10080	typhoon	(#LosDelSonido) Obama Declares Disaster for Ty...	1
10220	volcano	USGS reports a M1.94 #earthquake 5km S of Volc...	1
10771	wreckage	Wreckage 'Conclusively Confirmed' as From MH37...	1
10776	wreckage	Wreckage 'Conclusively Confirmed' as From MH37...	1

```
105 rows × 3 columns
```

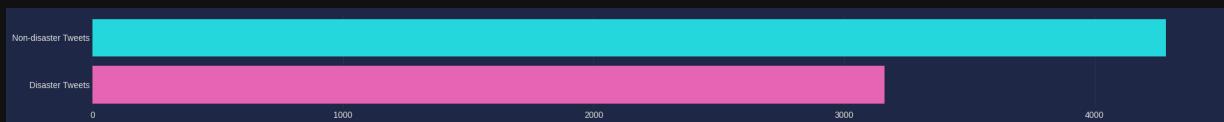
```
In [ ]: df_cl = df_cl_nl.drop_duplicates(subset=["text"])  
  
df_shape = df_cl.shape  
print(f"There {df_shape[1]} features and {df_shape[0]} instances")
```

```
There 3 features and 7447 instances
```

Checking the balance of the target

This dataset is an imbalance dataset that contains 3161 disaster tweets and 4286 non-disaster tweets.

```
In [ ]: labels = ["Non-disaster Tweets", "Disaster Tweets",]
values = [df_cl[df_cl['target'] == 0].shape[0], df_cl[df_cl['target'] == 1].shape[0]]  
  
plt.figure(figsize=(20, 2))
sns.barplot(y=labels, x=values)
plt.tight_layout();
```



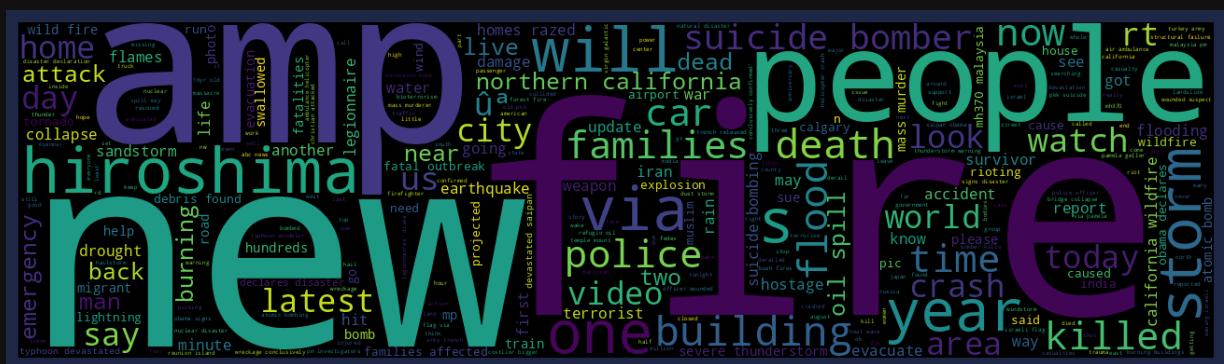
Word clouds

Most frequent words in Disasters Tweets

```
In [ ]: aux = STOPWORDS.copy()
aux.update(["t"])
aux.update(["co"])
aux.update(["https"])
aux.update(["w"])
aux.update(["_"])
aux.update(["Û"])
aux.update(["U"])
aux.update(["^a"])
aux.update(["ò"])

alltext = ' '.join(df_cl.loc[df_cl['target'] == 1, "text"])
word_cloud = WordCloud(width=1280, height=360, stopwords=aux, max_words=300)

plt.figure(figsize=(10, 20), dpi=2000)
plt.imshow(word_cloud)
plt.axis("off")
plt.tight_layout();
```



We can see that the most common words in disaster tweets are related to natural disasters, accidents, and fatalities, such as fire, storm, Hiroshima, and suicide bomber.

Most frequent words in Non-disasters Tweets



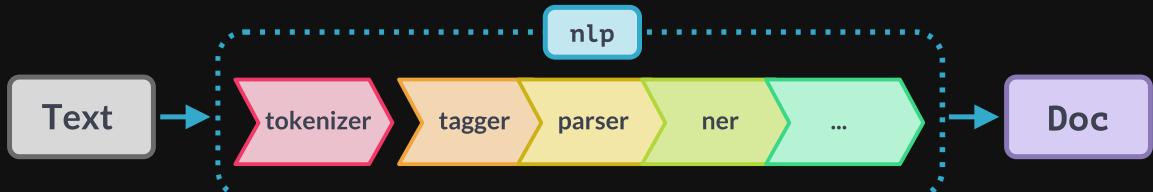
We can see that the most common words in non-disaster tweets are related to cotidian words.

Feature engineering

spaCy for NLP

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. It can be used to build information extraction or natural language understanding systems.

The processing pipeline consists of one or more pipeline components that are called on the Doc in order. The tokenizer runs before the components.



Working on the text of the tweets

Explaining the preprocessing pipeline that we will use in this project

1. Digits to words
2. Dealing with the Twitter mentions
3. Dealing with the hashtags
4. Dealing with the emoticons
5. Extracting, replacing and removing special and numerical characters, puntuactions, mentions, URLs and hashtags
6. Lemmatizing
7. Named-Entity Recognition
8. Lowercasing
9. Stopword removal
10. Tokenization
11. Padding sequence

```
In [ ]: raw_text="""Floods brought on by days of heavy rainfalls have been ravaging  
the Czech Republic and Austria, with thousands of people being e  
Austrian Chancellor Karl Nehammer on Monday said in a post on so  
that was translated by CNBC, that two more people had died follo  
This is in addition to a firefighter who died Sunday."""  
  
text1 = nlp(raw_text)  
displacy.render(text1, style="ent")
```

Floods brought on by days of heavy rainfalls have been ravaging countries including Poland
GPE , Romania GPE , the Czech Republic GPE and Austria GPE , with
thousands CARDINAL of people being evacuated as the death toll rises. Austrian NORP
Chancellor Karl Nehammer PERSON on Monday DATE said in a post on social media
platform X, that was translated by CNBC ORG , that two CARDINAL more people had
died following floods in the country. This is in addition to a firefighter who died Sunday DATE

Creating a example text

```
In [ ]: example_text = "Hello, my name is Alexander Daniel Rios. I'm 30 years old, a
```

Digits to words

```
In [ ]: matcher = Matcher(nlp.vocab)  
  
pattern = [{"IS_DIGIT": True}]  
matcher.add("NUMBERS", [pattern])
```

```

def digits2words(text, to_print=False):
    doc = nlp(text)
    matches = matcher(doc)
    for match_id, start, end in matches:
        if doc.vocab.strings[match_id] == "NUMBERS":
            if to_print:
                text = text.replace(doc[start:end].text, '\u033[4m\u033[32m' + n
            else:
                text = text.replace(doc[start:end].text, num2words(doc[start:end]))
    return text

print(f"ORIGINAL TEXT\n{example_text}\n")
print(f"MODIFIED TEXT\n{digits2words(example_text, to_print=True)}\n")

```

ORIGINAL TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

MODIFIED TEXT

Hello, my name is Alexander Daniel Rios. I'm thirty years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

Dealing with the Twitter mentions

```

In [ ]: pattern = [{"TEXT": {"REGEX": r"@[\w+]"}}]
matcher.add("TWITTER_MENTIONS", [pattern])

def replace_mentions(text):
    doc = nlp(text)
    matches = matcher(doc)
    for match_id, start, end in matches:
        if doc.vocab.strings[match_id] == "TWITTER_MENTIONS":
            text = text.replace(doc[start:end].text, "") # "@PERSON"
    return text

print(f"ORIGINAL TEXT\n{example_text}\n")
print(f"MODIFIED TEXT\n{replace_mentions(example_text)}\n")

```

ORIGINAL TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

MODIFIED TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username . #DataScientist #MLZoomCamp ;) :-)

Dealing with the hashtags

```
In [ ]: pattern = [{ 'ORTH': '#' }, {"TEXT": { "REGEX": r"\w+" }}]
matcher.add("HASHTAGS", [pattern])

def replace_hashtags(text, to_print=False):
    doc = nlp(text)
    matches = matcher(doc)
    for match_id, start, end in matches:
        if doc.vocab.strings[match_id] == "HASHTAGS":
            list_words = wn.split(doc[start+1:end].text)
            hashtag_converted = ' '.join(list_words)
            if to_print:
                text = text.replace(doc[start:end].text, '\u033[4m\u033[32m' + hashtag_converted + '\u033[4m\u033[39m')
            else:
                text = text.replace(doc[start:end].text, '#' + hashtag_converted)
    return text

print(f"ORIGINAL TEXT\n\u033[100m{example_text}\u033[0m\n")
print(f"MODIFIED TEXT\n\u033[100m{replace_hashtags(example_text, to_print=True)}\u033[0m\n")
```

ORIGINAL TEXT
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

MODIFIED TEXT
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #Data Scientist #ML Zoom Camp ;) :-)

Dealing with the emoticons

```
In [ ]: EMOTICONS = {':)': 'smile', ':-)': 'smile', ';d': 'wink', ':-E': 'vampire',
           ':-(': 'sad', ':-<': 'sad', ':P': 'raspberry', ':O': 'surprise',
           ':-@': 'shocked', ':@': 'shocked', ':-$': 'confused', ':\\"': 'a',
           ':#': 'mute', ':X': 'mute', ':^)': 'smile', ':-&': 'confused',
           '@@': 'eyeroll', ':-!': 'confused', ':-D': 'smile', ':-0': 'ye',
           '<(-_-)>': 'robot', 'd[-_-]b': 'dj', ":-)": 'sadsmile', ';)': ';-)': 'wink',
           'O:-)': 'angel', 'O*-)': 'angel', '(:-D': 'gossip'

def replace_emojis(text, to_print=False):
    for emoji in EMOTICONS.keys():
        if to_print:
            text = text.replace(emoji, '\u033[4m\u033[32m' + EMOTICONS[emoji] + '\u033[4m\u033[39m')
        else:
            text = text.replace(emoji, EMOTICONS[emoji])
    return text

print(f"ORIGINAL TEXT\n\u033[100m{example_text}\u033[0m\n")
print(f"MODIFIED TEXT\n\u033[100m{replace_emojis(example_text, to_print=True)}\u033[0m\n")
```

ORIGINAL TEXT

```
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Arge  
ntina. Currently, I am studying at National Technological University. You can  
find my GitHub profile at the following link: https://github.com/aletbm or with  
my username @aletbm. #DataScientist #MLZoomCamp ; ) :-)
```

MODIFIED TEXT

```
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Arge  
ntina. Currently, I am studying at National Technological University. You can  
find my GitHub profile at the following link: https://github.com/aletbm or with  
my username @aletbm. #DataScientist #MLZoomCamp wink smile
```

Extracting, replacing and removing special and numerical characters, puntuactions, mentions, URLs and hashtags

On the Tweet texts

```
In [ ]: def get_list(text, c='#'):  
    regex = ""  
    if c == "https":  
        regex = r"(" + c + "?://" + "[^\s]+)"  
    elif c == "punct":  
        regex = r"![\$%&`!'\\"\\(\()*, -./:;=>?\\^_`\\[\\"{}|}~+]"  
    else:  
        regex = r"(" + c + "[^\s]+)"  
    return re.findall(regex, text)  
  
def clear_tweet(text, list_c):  
    for c in list_c:  
        if c in ["#", "@"]:  
            lista = c  
        else:  
            lista = get_list(text, c)  
        for item in lista:  
            text = text.replace(item, " ") .replace(" ", " ")  
text = re.sub(r'[\x00-\x7F]+', '', text)  
return text  
  
def cleaner(text):  
    hashtags = get_list(text, c='#')  
    mentions = get_list(text, c='@')  
    urls = get_list(text, c='https')  
    clean_text = clear_tweet(text, ["#", "@", "https"])  
    punctuations = get_list(clean_text, c="punct")  
    clean_text = clear_tweet(clean_text, ["punct"])  
  
    return clean_text, len(clean_text.split()), len(clean_text), len(hashtags)  
  
#df_cl["clean_text"], df_cl["n_words"], df_cl["n_characters"], df_cl["n_hashtags"],  
clean_text, n_words, n_characters, n_hashtags, n_mentions, n_urls, n_punctuations  
print(f"ORIGINAL TEXT\n{example_text}\n")
```

```
print(f"MODIFIED TEXT\n{clean_text}\n")
print(f"Number of words: {n_words}")
print(f"Number of characters: {n_characters}")
print(f"Number of hashtags: {n_hashtags}")
print(f"Number of mentions: {n_mentions}")
print(f"Number of URLs: {n_urls}")
print(f"Number of punctuations: {n_punctuations}")
```

ORIGINAL TEXT

```
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: https://github.com/aletbm or with my username @aletbm. #DataScientist #MLZoomCamp ; ) :-)
```

MODIFIED TEXT

```
Hello my name is Alexander Daniel Rios I m 30 years old and I live in Argentina Currently I am studying at National Technological University You can find my GitHub profile at the following link or with my username aletbm DataScientist MLZoomCamp
```

```
Number of words: 42
Number of characters: 246
Number of hashtags: 2
Number of mentions: 1
Number of URLs: 1
Number of punctuations: 14
```

On the keyword

```
In [ ]: df_cl.loc[df_cl.keyword.str.contains("%20"), "keyword"].unique()
```

```
Out[ ]: array(['airplane%20accident', 'blew%20up', 'blown%20up', 'body%20bag',
       'body%20bagging', 'body%20bags', 'bridge%20collapse',
       'buildings%20burning', 'buildings%20on%20fire',
       'burning%20buildings', 'bush%20fires', 'chemical%20emergency',
       'cliff%20fall', 'dust%20storm', 'emergency%20plan',
       'emergency%20services', 'fire%20truck', 'first%20responders',
       'forest%20fire', 'forest%20fires', 'heat%20wave', 'loud%20bang',
       'mass%20murder', 'mass%20murderer', 'natural%20disaster',
       'nuclear%20disaster', 'nuclear%20reactor', 'oil%20spill',
       'radiation%20emergency', 'structural%20failure', 'suicide%20bomb',
       'suicide%20bomber', 'suicide%20bombing', 'violent%20storm',
       'war%20zone', 'wild%20fires'], dtype=object)
```

```
In [ ]: def replace_spaces(text):
    text = text.replace("%20", " ")
    return text
```

```
df_cl.loc[df_cl.keyword.str.contains("%20"), "keyword"].apply(replace_spaces)
```

```
Out[ ]: array(['airplane accident', 'blew up', 'blown up', 'body bag',
   'body bagging', 'body bags', 'bridge collapse',
   'buildings burning', 'buildings on fire', 'burning buildings',
   'bush fires', 'chemical emergency', 'cliff fall', 'dust storm',
   'emergency plan', 'emergency services', 'fire truck',
   'first responders', 'forest fire', 'forest fires', 'heat wave',
   'loud bang', 'mass murder', 'mass murderer', 'natural disaster',
   'nuclear disaster', 'nuclear reactor', 'oil spill',
   'radiation emergency', 'structural failure', 'suicide bomb',
   'suicide bomber', 'suicide bombing', 'violent storm', 'war zone',
   'wild fires'], dtype=object)
```

Lemmatizing

On the Tweet texts

```
In [ ]: def lemmatize(text):
    doc = nlp(text)
    for token in doc:
        text = text.replace(token.text, token.lemma_)
    return text

print(f"ORIGINAL TEXT\n{example_text}")
print(f"MODIFIED TEXT\n{lemmatize(example_text)}")
```

ORIGINAL TEXT

```
Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: https://github.com/aletbm or with my username @aletbm. #DataScientist #MLZoomCamp ; ) :-)
```

MODIFIED TEXT

```
hello, my nbee be Alexander Daniel Rios. Ibe 30 year old, and I live in Argentina. currently, I be study at National Technological University. you can find my GitHub profile at the follow link: https://github.com/aletbm or with my use rnbee @aletbm. #DataScientbet #MLZoomCbep ; ) :-)
```

On the keyword

```
In [ ]: df_cl.keyword.apply(replace_spaces).unique()[-10:]
```

```
Out[ ]: array(['weapons', 'whirlwind', 'wild fires', 'wildfire', 'windstorm',
   'wounded', 'wounds', 'wreck', 'wreckage', 'wrecked'], dtype=object)
```

```
In [ ]: df_cl.keyword.apply(replace_spaces).apply(lemmatize).unique()[-10:]
```

```
Out[ ]: array(['volcano', 'war zone', 'weapon', 'whirlwind', 'wild fire',
   'wildfire', 'windstorm', 'wound', 'wreck', 'wreckage'],
   dtype=object)
```

Named-Entity Recognition

```
In [ ]: def replace_entities(text, to_print=False):
    doc = nlp(text)
    for ent in doc.ents:
        if to_print:
            text = text.replace(ent.text, '\u033[4m\u033[32m'+ent.label_+'\u033[0m')
        else:
            text = text.replace(ent.text, ent.label_)
    return text

print(f"ORIGINAL TEXT\n\u033[100m{example_text}\u033[0m\n")
print(f"MODIFIED TEXT\n\u033[100m{replace_entities(example_text, to_print=True)}\u033[0m")
```

ORIGINAL TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

MODIFIED TEXT

Hello, my name is PERSON. I'm DATE, and I live in GPE. Currently, I am studying at ORG. You can find my ORG profile at the following link: <https://github.com/aletbm> or with my username ORG. MONEYMLZoomCamp ;) :-)

Lowercasing

```
In [ ]: print(f"ORIGINAL TEXT\n\u033[100m{example_text}\u033[0m\n")
print(f"MODIFIED TEXT\n\u033[100m{example_text.lower()}\u033[0m")
```

ORIGINAL TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

MODIFIED TEXT

hello, my name is alexander daniel rios. i'm 30 years old, and i live in argentina. currently, i am studying at national technological university. you can find my github profile at the following link: <https://github.com/aletbm> or with my username @aletbm. #datascientist #mlzoomcamp ;) :-)

Stopword removal

```
In [ ]: def remove_stopwords(text):
    doc = nlp(text)
    filtered_sentence = []
    for token in doc:
        if (token.is_stop == False
            and len(token) > 2
            and token.is_ascii
            and token.is_alpha
            and token.text in dict_words[token.text[0]][token.text[1]]):
            filtered_sentence.append(token.text)
```

```

        return ' '.join(filtered_sentence)

print(f"ORIGINAL TEXT\n{example_text.lower()}")
print(f"MODIFIED TEXT\n{remove_stopwords(example_text.lower())}")

```

ORIGINAL TEXT

hello, my name is alexander daniel rios. i'm 30 years old, and i live in argentina. currently, i am studying at national technological university. you can find my github profile at the following link: https://github.com/aletbm or with my username @aletbm. #datascientist #mlzoomcamp ;) :-)

MODIFIED TEXT

hello alexander daniel years old live argentina currently studying national technological university find profile following link

Tokenization

```

In [ ]: def tokenize(text, tokenizer, fit=False):
    if fit == True:
        tokenizer.fit_on_texts(text)
    return tokenizer.texts_to_sequences(text)

tokenizer = Tokenizer(oov_token=<OOV>)
tokenized_text = tokenize([example_text], tokenizer, fit=True)

print(f"ORIGINAL TEXT\n{example_text}")
print(f"TOKENIZED TEXT\n{tokenized_text}")
print(f"DICTIONARY OF CONVERSION\n{dict(itertools.islice(tokenizer.i

```

ORIGINAL TEXT

Hello, my name is Alexander Daniel Rios. I'm 30 years old, and I live in Argentina. Currently, I am studying at National Technological University. You can find my GitHub profile at the following link: https://github.com/aletbm or with my username @aletbm. #DataScientist #MLZoomCamp ;) :-)

TOKENIZED TEXT

[[7, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 3, 18, 19, 20, 21, 3, 22, 23, 4, 24, 25, 26, 27, 28, 29, 2, 5, 30, 4, 31, 32, 33, 34, 5, 35, 6, 36, 37, 2, 38, 6, 39, 40]]

DICTIONARY OF CONVERSION

```

{1: '<OOV>', 2: 'my', 3: 'i', 4: 'at', 5: 'github', 6: 'aletbm', 7: 'hello', 8: 'name', 9: 'is', 10: 'alexander', 11: 'daniel', 12: 'rios', 13: "i'm", 14: '30', 15: 'years', 16: 'old', 17: 'and', 18: 'live', 19: 'in', 20: 'argentin a', 21: 'currently', 22: 'am', 23: 'studying', 24: 'national', 25: 'technolog ical', 26: 'university', 27: 'you', 28: 'can', 29: 'find', 30: 'profile', 31: 'the', 32: 'following', 33: 'link', 34: 'https', 35: 'com', 36: 'or', 37: 'wi th', 38: 'username', 39: 'datascientist', 40: 'mlzoomcamp'}

```

Padding sequence

```

In [ ]: max_length_tweet = 21*3

padded_text = pad_sequences(tokenized_text, maxlen=max_length_tweet, padding

```

```

print(f"TOKENIZED TEXT\n{tokenized_text}\n")
print(f"TEXT WITH PADDING\n{padded_text}\n")

TOKENIZED TEXT
[[7, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 3, 18, 19, 20, 21, 3, 22, 23,
4, 24, 25, 26, 27, 28, 29, 2, 5, 30, 4, 31, 32, 33, 34, 5, 35, 6, 36, 37, 2,
38, 6, 39, 40]]

TEXT WITH PADDING
[[ 7  2   8   9  10  11  12  13  14  15  16  17   3  18  19  20  21   3  22  23  4  24  25  26
 27 28 29   2   5  30   4  31  32  33  34   5  35   6  36  37   2  38   6  39  40   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]

```

All preprocessing steps in a single function

Cleaning the Tweet text

```

In [ ]: def init_configure():
    matcher = Matcher(nlp.vocab)

    pattern = [{"IS_DIGIT": True}]
    matcher.add("NUMBERS", [pattern])

    pattern = [{"TEXT": {"REGEX": r"@\\w+"}}]
    matcher.add("TWITTER_MENTIONS", [pattern])

    pattern = [{"ORTH": '#'}, {"TEXT": {"REGEX": r"\w+"}}]
    matcher.add("HASHTAGS", [pattern])

    tokenizer = Tokenizer(oov_token=<OOV>)

    max_length_tweet = 21*5
    max_length_keyword = 3*2
    return matcher, tokenizer, max_length_tweet, max_length_keyword

def preprocessing_text(text):
    doc = nlp(text)
    matches = matcher(doc)
    for match_id, start, end in matches:
        if doc.vocab.strings[match_id] == "NUMBERS":
            text = text.replace(doc[start:end].text, num2words(doc[start:end]))
        #if doc.vocab.strings[match_id] == "TWITTER_MENTIONS":
        #    text = text.replace(doc[start:end].text, "") # "@PERSON"
        #if doc.vocab.strings[match_id] == "HASHTAGS":
        #    list_words = wn.split(doc[start+1:end].text)
        #    hashtag_converted = ' '.join(list_words)
        #    text = text.replace(doc[start:end].text, '#' + hashtag_converted)

        #text = replace_emojis(text)
    text, n_words, n_characters, n_hashtags, n_mentions, n_urls, n_punctuation = lemmatize(text)
    #text = replace_entities(text)
    text = text.lower()

```

```

    clean_text = remove_stopwords(text)

    return clean_text, n_words, n_characters, n_hashtags, n_mentions, n_urls

def preprocessing_keyword(text):
    text = replace_spaces(text)
    clean_text = lemmatize(text)
    return clean_text

matcher, tokenizer, max_length_tweet, max_length_keyword = init_configure()

df_cl["clean_text"], df_cl["n_words"], df_cl["n_characters"], df_cl["n_hashtags"], df_cl["clean_keyword"] = df_cl.keyword.apply(lambda text: preprocessing_keyword(text))
df_cl.head()

```

Out []:

	keyword	text	target	clean_text	n_words	n_characters	n_hashtags	n_urls
	id							
48	ablaze	@bbcmtd Wholesale Markets ablaze http://t.co/l...	1	wholesale markets ablaze	4	33	0	
49	ablaze	We always try to bring the heavy. #metal #RT h...	0	try bring heavy metal	9	42	2	
50	ablaze	#AFRICANBAZE: Breaking news:Nigeria flag set a...	1	break news nigeria flag set ablaze	9	58	1	
52	ablaze	Crying out for more! Set me ablaze	0	cry set ablaze	7	33	0	
53	ablaze	On plus side LOOK AT THE SKY LAST NIGHT IT WAS...	0	plus look sky night ablaze	12	54	0	

Tokenization and Padding

In []:

```

def get_tokens(text, max_length, fit=False, padding=False):
    preprocessed_text = tokenize(text, tokenizer, fit)
    if padding:
        preprocessed_text = pad_sequences(preprocessed_text, maxlen=max_length)
    return preprocessed_text

df_cl["tokenized_text"] = list(get_tokens(df_cl["clean_text"].tolist(), max_length))
df_cl["tokenized_keyword"] = list(get_tokens(df_cl["clean_keyword"].tolist()))
df_cl.head()

```

Out []:	keyword	text	target	clean_text	n_words	n_characters	n_hashtags	n_
id								
		@bbcmtd Wholesale Markets ablaze http://t.co/l...	1	wholesale markets ablaze	4	33	0	
48	ablaze	We always try to bring the heavy. #metal #RT h...	0	try bring heavy metal	9	42	2	
49	ablaze	#AFRICANBAZE: Breaking news:Nigeria flag set a...	1	break news nigeria flag set ablaze	9	58	1	
50	ablaze	Crying out for more! Set me ablaze	0	cry set ablaze	7	33	0	
52	ablaze	On plus side LOOK AT THE SKY LAST NIGHT IT WAS...	0	plus look sky night ablaze	12	54	0	
53	ablaze							

Vocabulary size

```
In [ ]: vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out []: 8826

Let's take a look at the vocabulary

```
In [ ]: pprint(dict(itertools.islice(tokenizer.index_word.items(), 20)))
```

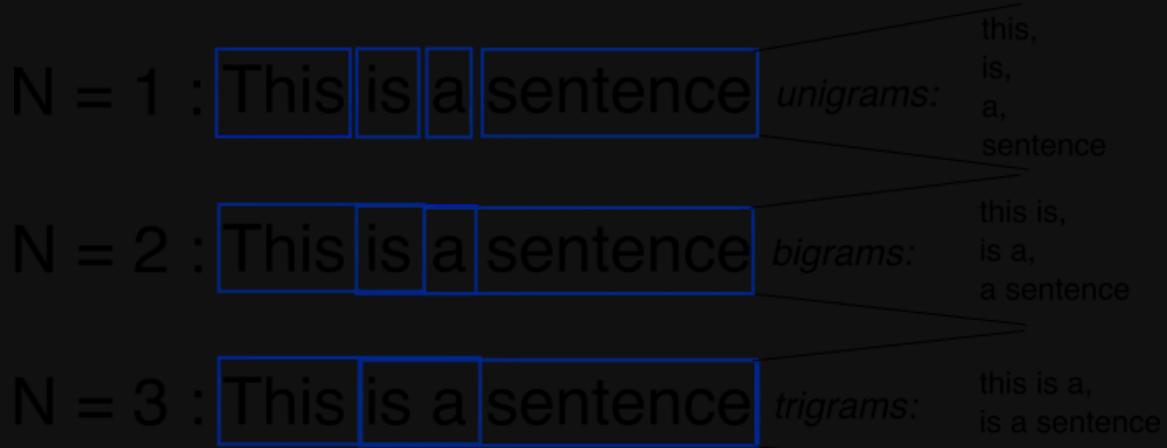
```
{1: '<OOV>',
 2: 'fire',
 3: 'like',
 4: 'amp',
 5: 'thousand',
 6: 'emergency',
 7: 'burn',
 8: 'body',
 9: 'bomb',
10: 'disaster',
11: 'crash',
12: 'storm',
13: 'flood',
14: 'new',
15: 'attack',
16: 'collapse',
17: 'building',
18: 'scream',
19: 'suicide',
20: 'drown'}
```

Saving the preprocessors with CloudPickle for the baseline model

```
In [ ]: with open('tokenizer.bin', 'wb') as f_out:
    cloudpickle.dump((tokenizer, dict_words), f_out)
```

N-grams

N-grams are contiguous sequences of 'n' items, typically words in the context of NLP.



Unigrams

```
In [ ]: def get_top_n_words(data, ngram_range=(1, 1)):
    vectorizer = CountVectorizer(ngram_range=ngram_range)
    count_matrix = vectorizer.fit_transform(data)
    count_array = count_matrix.toarray()
```

```

bag_of_words = pd.DataFrame(data=count_array, columns = vectorizer.get_feature_names()
                             )
return bag_of_words

def plot_top_words(data, bag_of_words, top=20):
    top_word_disaster = bag_of_words[data["target"] == 1].sum().sort_values(ascending=False)
    top_word_disaster = top_word_disaster[:top]

    top_word_notdisaster = bag_of_words[data["target"] == 0].sum().sort_values(ascending=False)
    top_word_notdisaster = top_word_notdisaster[:top]

    plt.subplots(1, 2, figsize=(18, 10), dpi=200)

    plt.subplot(1, 2, 1)
    sns.barplot(y=top_word_notdisaster.index, x=top_word_notdisaster, color="#1f77b4")
    plt.title("Non-disaster Tweets")

    plt.subplot(1, 2, 2)
    sns.barplot(y=top_word_disaster.index, x=top_word_disaster, color="#FE5340")
    plt.title("Disaster Tweets")
    plt.suptitle(f"Top {top} words in tweets after preprocessing")
    plt.tight_layout()
    return

bag_of_words = get_top_n_words(data=df_cl['clean_text'], ngram_range=(1, 1))
plot_top_words(data=df_cl, bag_of_words=bag_of_words, top=20)

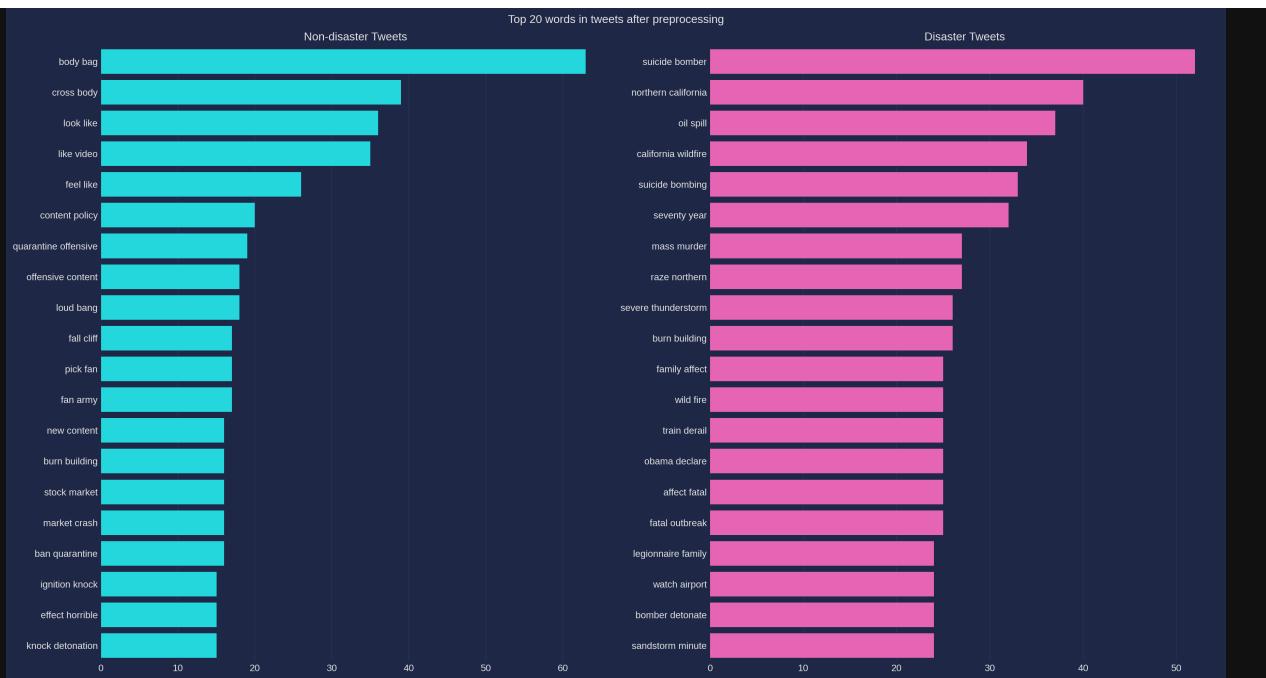
```



We can see the difference between the unigrams of disaster tweets and non-disaster tweets.
The disaster tweets contain unigrams with negative connotations.

Bigrams

```
In [ ]: bag_of_words = get_top_n_words(data=df_cl['clean_text'], ngram_range=(2, 2))
plot_top_words(data=df_cl, bag_of_words=bag_of_words, top=20)
```



We can see the difference between the bigrams of disaster tweets and non-disaster tweets.
The disaster tweets contain bigrams with negative connotations.

Trigrams

```
In [ ]: bag_of_words = get_top_n_words(data=df_cl['clean_text'], ngram_range=(3, 3))
plot_top_words(data=df_cl, bag_of_words=bag_of_words, top=20)
```



We can see the difference between the trigrams of disaster tweets and non-disaster tweets.
The disaster tweets contain trigrams with negative connotations.

Distribution of the context feature by tweet type

```
In [ ]: pd.set_option("mode.copy_on_write", False)
target = df_cl["target"].map({0: "Non-disaster Tweets", 1: "Disaster Tweets"})

plt.subplots(3, 2, figsize=(20, 15), dpi=300)
for i, col in enumerate(["n_words", "n_characters", "n_hashtags", "n_mentions",
                        "n_urls", "n_punctuations"]):
    plt.subplot(3, 2, i+1)
    sns.histplot(x=df_cl[col], hue=target, hue_order=["Non-disaster Tweets",
```



Statistical analysis of context features in non-disaster tweets

```
In [ ]: df_cl.loc[df_cl["target"] == 0, ["n_words", "n_characters", "n_hashtags", "n
```

```
Out[ ]:
```

	count	mean	std	min	25%	50%	75%	max
n_words	4286.0	14.655623	6.563463	1.0	10.0	14.0	19.0	37.0
n_characters	4286.0	82.882408	34.296479	6.0	56.0	84.0	110.0	219.0
n_hashtags	4286.0	0.383341	0.990170	0.0	0.0	0.0	0.0	11.0
n_mentions	4286.0	0.425338	0.771670	0.0	0.0	0.0	1.0	8.0
n_urls	4286.0	0.511433	0.667103	0.0	0.0	0.0	1.0	4.0
n_punctuations	4286.0	2.993234	3.279255	0.0	1.0	2.0	4.0	58.0

Statistical analysis of context features in disaster tweets

```
In [ ]: df_cl.loc[df_cl["target"] == 1, ["n_words", "n_characters", "n_hashtags", "n
```

```
Out[ ]:
```

	count	mean	std	min	25%	50%	75%	max
n_words	3161.0	14.937678	5.742379	1.0	10.0	15.0	19.0	36.0
n_characters	3161.0	90.240430	30.637671	9.0	67.0	91.0	112.0	202.0
n_hashtags	3161.0	0.506485	1.198643	0.0	0.0	0.0	1.0	13.0
n_mentions	3161.0	0.278709	0.630315	0.0	0.0	0.0	0.0	5.0
n_urls	3161.0	0.782347	0.625982	0.0	0.0	1.0	1.0	3.0
n_punctuations	3161.0	2.933249	2.801385	0.0	1.0	2.0	4.0	21.0

In general terms, the context features have very similar statistical characteristics between the different tweet types.

Splitting the dataset

Separating the target

```
In [ ]: x = df_cl.drop(["target"], axis=1)  
y = df_cl["target"]
```

Creating the training and validation set

```
In [ ]: x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.20, stratify=y)  
x_val, x_test, y_val, y_test = train_test_split(x_val, y_val, test_size=0.50)
```

```
x_train.head()
```

Out []:	keyword	text	clean_text	n_words	n_characters	n_hasht
id						
3015	death	I feel like death	feel like death	4	17	
276	ambulance	#reuters Twelve feared killed in Pakistani air...	reuters fear kill pakistan air ambulance heli...	10	74	
3075	deaths	real magic in real life:\n\nwomen went missing...	real magic real life woman miss ohio town focu...	20	106	
2209	chemical%20emergency	Emergency Response and Hazardous Chemical Mana...	emergency response hazardous chemical manageme...	9	78	
996	blazing	SHOUOUT TO @kasad1lla CAUSE HER VOCALS ARE BLA...	cause vocals blaze hot like weather	14	78	

Dealing with an imbalance target

```
In [ ]: class_weight = compute_class_weight(class_weight="balanced", classes=np.unique(y_train))
class_weight = dict(zip(np.unique(y_train), class_weight))
class_weight
```

```
Out [ ]: {0: 0.8688739789964994, 1: 1.1777382364570976}
```

Training a baseline model for Tweet classification

Creating the dataset for my baseline model

Selecting the useful columns

```
In [ ]: X_train_tokenized = X_train.loc[:, ["tokenized_text", "tokenized_keyword", "n_..."]]
```

```
X_test_tokenized = X_test.loc[:, ["tokenized_text", "tokenized_keyword", "n_"]

X_train_tokenized.head()
```

Creating a Tensorflow Dataset

```
In [ ]: train_text, train_keyword, train_context = X_train_tokenized["tokenized_text"]
train_data_text = Dataset.from_tensor_slices(train_text)
train_data_keyword = Dataset.from_tensor_slices(train_keyword)
train_data_context = Dataset.from_tensor_slices(train_context)
train_labels = Dataset.from_tensor_slices(np.expand_dims(y_train.values, axis=1))

val_text, val_keyword, val_context = X_val_tokenized["tokenized_text"].to_list()
val_data_text = Dataset.from_tensor_slices(val_text)
val_data_keyword = Dataset.from_tensor_slices(val_keyword)
val_data_context = Dataset.from_tensor_slices(val_context)
val_labels = Dataset.from_tensor_slices(np.expand_dims(y_val.values, axis=-1))

test_text, test_keyword, test_context = X_test_tokenized["tokenized_text"].to_list()
test_data_text = Dataset.from_tensor_slices(test_text)
test_data_keyword = Dataset.from_tensor_slices(test_keyword)
test_data_context = Dataset.from_tensor_slices(test_context)
test_labels = Dataset.from_tensor_slices(np.expand_dims(y_test.values, axis=-1))

train_data_tokenized_ = Dataset.zip(((train_data_text, train_data_keyword, t
val_data_tokenized_ = Dataset.zip(((val_data_text, val_data_keyword, val_dat
test_data_tokenized_ = Dataset.zip(((test_data_text, test_data_keyword, test

AUTOTUNE = tf.data.AUTOTUNE
```

Baseline model architecture

```
In [ ]: def get_baseline_model():
    tf.keras.backend.clear_session()

    inp1 = Input(shape=(max_length_tweet,), name="text")
    x = Embedding(input_dim=vocab_size, output_dim=16, input_length=max_length_tweet)(inp1)
    x = Bidirectional(LSTM(4, dropout=0.4, recurrent_dropout=0.0))(x)
    out1 = Dense(1, activation="sigmoid")(x)

    inp2 = Input(shape=(max_length_keyword,), name="keyword")
    t = Embedding(input_dim=vocab_size, output_dim=16, input_length=max_length_keyword)(inp2)
    t = Bidirectional(LSTM(4, dropout=0.4, recurrent_dropout=0.0))(t)
    out2 = Dense(1, activation="sigmoid")(t)

    inp3 = Input(shape=(6,), name="context")
    z = Dense(32, activation="relu")(inp3)
    z = Dropout(0.4)(z)
    z = Dense(16, activation="relu")(z)
    out3 = Dense(1, activation="sigmoid")(z)

    k = Concatenate()([out1, out2, out3])
    out = Dense(1, activation="sigmoid")(k)

    model = Model(inputs=[inp1, inp2, inp3], outputs=out)
    return model

model_base = get_baseline_model()
model_base.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:9
 0: UserWarning: Argument `input_length` is deprecated. Just remove it.
    warnings.warn(
Model: "functional"
```

Layer (type)	Output Shape	Params
context (InputLayer)	(None, 6)	0
text (InputLayer)	(None, 105)	0
keyword (InputLayer)	(None, 6)	0
dense_2 (Dense)	(None, 32)	224
embedding (Embedding)	(None, 105, 16)	141,216
embedding_1 (Embedding)	(None, 6, 16)	141,216
dropout (Dropout)	(None, 32)	0
bidirectional (Bidirectional)	(None, 8)	672
bidirectional_1 (Bidirectional)	(None, 8)	672
dense_3 (Dense)	(None, 16)	528
dense (Dense)	(None, 1)	9
dense_1 (Dense)	(None, 1)	9
dense_4 (Dense)	(None, 1)	17
concatenate (Concatenate)	(None, 3)	0
dense_5 (Dense)	(None, 1)	4

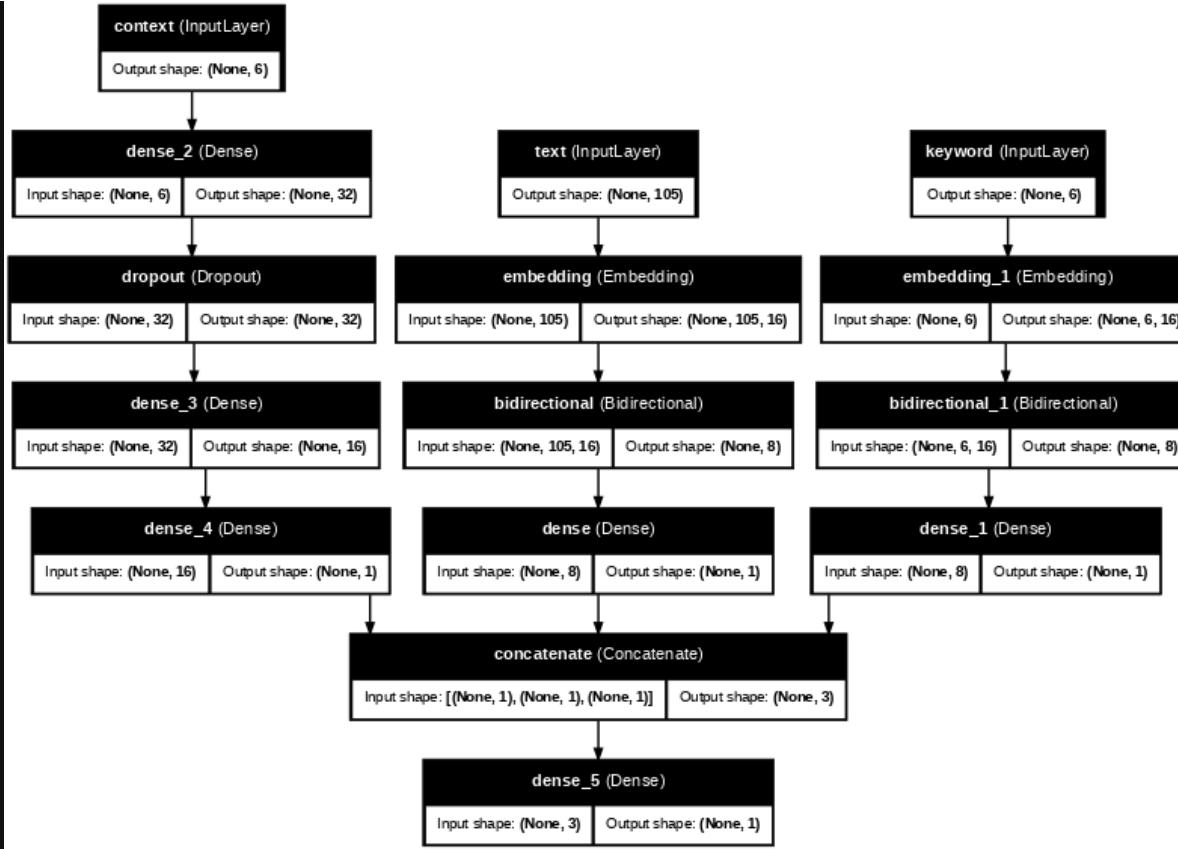
Total params: 284,567 (1.09 MB)

Trainable params: 284,567 (1.09 MB)

Non-trainable params: 0 (0.00 B)

```
In [ ]: plot_model(model_base, show_layer_names=True, show_shapes=True, dpi=50)
```

Out []:



Let's train

Effect of varying the batch size Parameter

```
In [ ]: def train_model(model, train_data, val_data, batch_size=32, epochs=100, version=1):
    csvlogger = CSVLogger(dst_path+f"histories/history_model_{version}.csv",
                          separator=',')
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy')
    history = model.fit(train_data,
                         validation_data=val_data,
                         batch_size=batch_size,
                         validation_batch_size=batch_size,
                         epochs=epochs,
                         callbacks=[csvlogger],
                         class_weight=class_weight)
    return history

for bs in [4, 16, 32, 64]:
    train_data_tokenized = train_data_tokenized_.batch(bs).prefetch(buffer_size=AUTO)
    val_data_tokenized = val_data_tokenized_.batch(bs).prefetch(buffer_size=AUTO)
    test_data_tokenized = test_data_tokenized_.batch(bs).prefetch(buffer_size=AUTO)
    model_base = get_baseline_model()
    history_base = train_model(model=model_base,
                               train_data=train_data_tokenized,
                               val_data=val_data_tokenized,
                               batch_size=bs,
                               epochs=50,
```

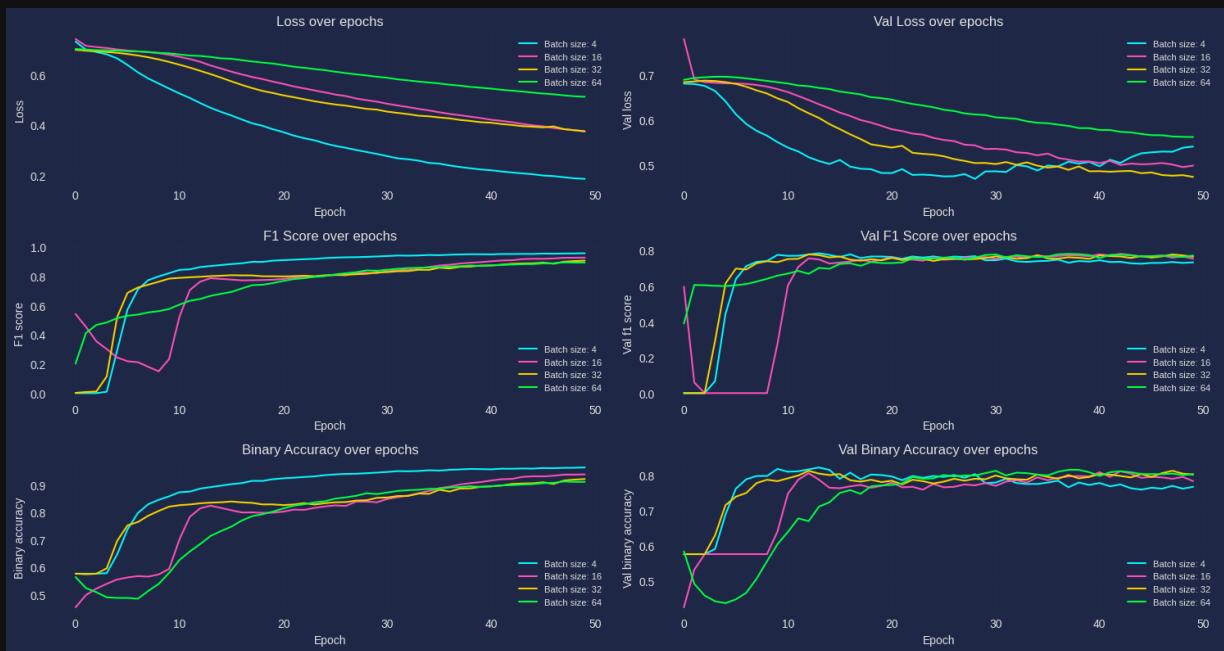
```

        version=f"base_{bs}"
    )

In [ ]: parameters = ["loss", "val_loss", "f1_score", "val_f1_score", "binary_accuracy"]
fig, axs = plt.subplots(3, 2, figsize=(15, 8))
for bs in [4, 16, 32, 64]:
    version=f"base_{bs}"
    df_hist = pd.read_csv(dst_path+f"histories/history_model_{version}.csv")

    for i, col in enumerate(parameters):
        plt.subplot(3, 2, i+1)
        param = col.replace('_', ' ').title()
        plt.plot(df_hist.index, df_hist[col], label=f"Batch size: {bs}")
        plt.ylabel(col.capitalize().replace("_", " "))
        plt.xlabel("Epoch")
        plt.legend(fontsize=8)
        plt.grid(True, alpha=0.3, which="both")
        plt.title(f"{param} over epochs")
plt.tight_layout()

```



If we focus on the validation loss, we can see that a batch size of 4 shows a slight increase in the last epochs, while batch sizes of 16 and 32 show a decrease in the last epoch.

```

In [ ]: def train_model(model, train_data, val_data, epochs=100, version="base"):
    checkpoint_filepath = dst_path+f'model_{version}.keras'
    checkpoint = ModelCheckpoint(checkpoint_filepath, monitor='val_loss', save_best_only=True)
    reduce = ReduceLROnPlateau(monitor='val_loss', patience=6, verbose=1)
    early = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)
    csvlogger = CSVLogger(dst_path+f"histories/history_model_{version}.csv", append=True)

    model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy')
    history = model.fit(train_data,
                        validation_data=val_data,
                        epochs=epochs,
                        callbacks=[reduce, early, checkpoint, csvlogger],

```

```

        class_weight=class_weight
    )
    return history

batch_size = 32
train_data_tokenized = train_data_tokenized_.batch(batch_size).prefetch(buffer_size)
val_data_tokenized = val_data_tokenized_.batch(batch_size).prefetch(buffer_size)
test_data_tokenized = test_data_tokenized_.batch(batch_size).prefetch(buffer_size)

history_base = train_model(model=model_base,
                           train_data=train_data_tokenized,
                           val_data=val_data_tokenized,
                           epochs=200,
                           version="base"
)

```

Visualizing the metrics over training epochs of the baseline model

```

In [ ]: def delete_empty_subplots(fig, axs):
    for ax_row in axs:
        for ax in ax_row:
            if ax.title.get_text() == "":
                fig.delaxes(ax)
    return fig, axs

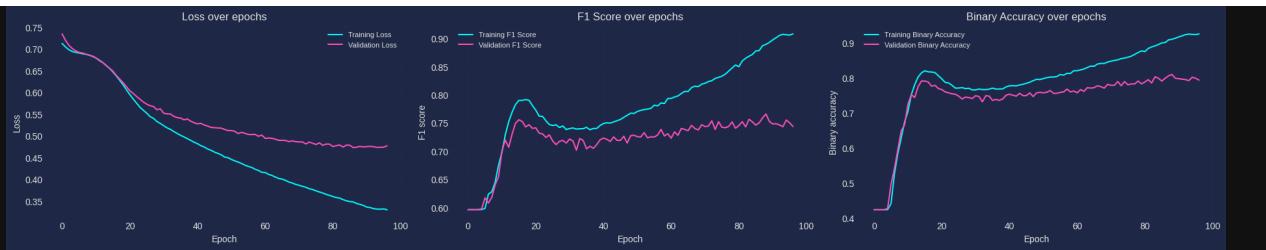
def show_history(version, parameters=None):
    df_hist = pd.read_csv(dst_path+f"histories/history_model_{version}.csv")
    fig, axs = plt.subplots(2, 4, figsize=(25, 7))

    for i, col in enumerate(parameters):
        plt.subplot(2, 4, i+1)
        param = col[0].replace('_', ' ').title()
        plt.plot(df_hist.index, df_hist[col[0]], label=f"Training {param}")
        plt.plot(df_hist.index, df_hist[col[1]], label=f"Validation {param}")
        plt.ylabel(col[0].capitalize().replace("_", " "))
        plt.xlabel("Epoch")
        plt.legend(fontsize=8)
        plt.grid(True, alpha=0.3, which="both")
        plt.title(f"{param} over epochs")

    fig, axs = delete_empty_subplots(fig, axs)
    plt.tight_layout()
    plt.show()
    return

show_history(version="base", parameters=[["loss", "val_loss"], ["f1_score", ""]]

```



Evaluating the baseline model

```
In [ ]: model_base.evaluate(test_data_tokenized)
```

24/24 **0s** 7ms/step - binary_accuracy: 0.803
1 - f1_score: 0.7502 - loss: 0.4989

```
Out [ ]: [0.49932780861854553, 0.7437184453010559, 0.7946308851242065]
```

Fine-Tuning with the BERT architecture (Bidirectional Encoder Representations from Transformers)

No description has been provided for this image

Creating the dataset for the BERT model

Selecting the useful columns

```
In [ ]: X_train_clean = X_train.loc[:, "text"]  
X_val_clean = X_val.loc[:, "text"]  
X_test_clean = X_test.loc[:, "text"]  
  
x_train_clean
```

```
Out [ ]: id  
3015 I feel like death  
276 #reuters Twelve feared killed in Pakistani air...  
3075 real magic in real life:\n\nwomen went missing...  
2209 Emergency Response and Hazardous Chemical Mana...  
996 SHOUOUT TO @kasadilla CAUSE HER VOCALS ARE BLA...  
...  
10137 Ancient Mayan Tablet with Hieroglyphics Honors...  
1061 @KatRamsland Yes I'm a bleeding heart liberal.  
3509 MP trains derailment: »It's the freakiest ...  
4022 DISASTER AVERTED: Police kill gunman with 'noa...  
1501 US »Institute Of Peace» Chairman Wants Rus...  
Name: text, Length: 5957, dtype: object
```

Creating a Tensorflow Dataset

```
In [ ]: train_data_bert = Dataset.from_tensor_slices(X_train_clean.values)
train_labels = Dataset.from_tensor_slices(np.expand_dims(y_train.values, axis=1))

val_data_bert = Dataset.from_tensor_slices(X_val_clean.values)
val_labels = Dataset.from_tensor_slices(np.expand_dims(y_val.values, axis=1))

test_data_bert = Dataset.from_tensor_slices(X_test_clean.values)
test_labels = Dataset.from_tensor_slices(np.expand_dims(y_test.values, axis=1))

train_data_bert = Dataset.zip((train_data_bert, train_labels))
val_data_bert = Dataset.zip((val_data_bert, val_labels))
test_data_bert = Dataset.zip((test_data_bert, test_labels))

AUTOTUNE = tf.data.AUTOTUNE
batch_size = 32

preset = "bert_small_en_uncased"
preprocessor = BertPreprocessor.from_preset(preset, trainable=True)

train_data_bert = train_data_bert.map(preprocessor).batch(batch_size).prefetch(AUTOTUNE)
val_data_bert = val_data_bert.map(preprocessor).batch(batch_size).prefetch(AUTOTUNE)
test_data_bert = test_data_bert.map(preprocessor).batch(batch_size).prefetch(AUTOTUNE)
```

BERT model architecture

```
In [ ]: def get_bert_model(preset):
    tf.keras.backend.clear_session()
    tokens_ids = Input(shape=(512,), name='token_ids')
    padding_mask = Input(shape=(512,), name='padding_mask')
    segment_ids = Input(shape=(512, ), name='segment_ids')
    encoder = BertBackbone.from_preset(preset)
    for i, layer in enumerate(encoder.layers):
        encoder.layers[i].trainable = True
    outputs = encoder({'token_ids':tokens_ids, 'padding_mask':padding_mask,
                      'segment_ids':segment_ids})
    my_net = outputs["pooled_output"]
    my_net = Dense(64, activation="relu")(my_net)
    my_net = Dropout(0.4)(my_net)
    my_net = Dense(1, activation="sigmoid")(my_net)
    return Model(inputs={'token_ids':tokens_ids, 'padding_mask':padding_mask,
                        'segment_ids':segment_ids}, outputs=my_net)

model_bert = get_bert_model(preset)
model_bert.summary()
```

Model: "functional"

Layer (type)	Output Shape	Params
padding_mask (InputLayer)	(None, 512)	0
segment_ids (InputLayer)	(None, 512)	0
token_ids (InputLayer)	(None, 512)	0
bert_backbone (BertBackbone)	[None, 512), (None, 512, 512)]	28,763,648
dense (Dense)	(None, 64)	32,832
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

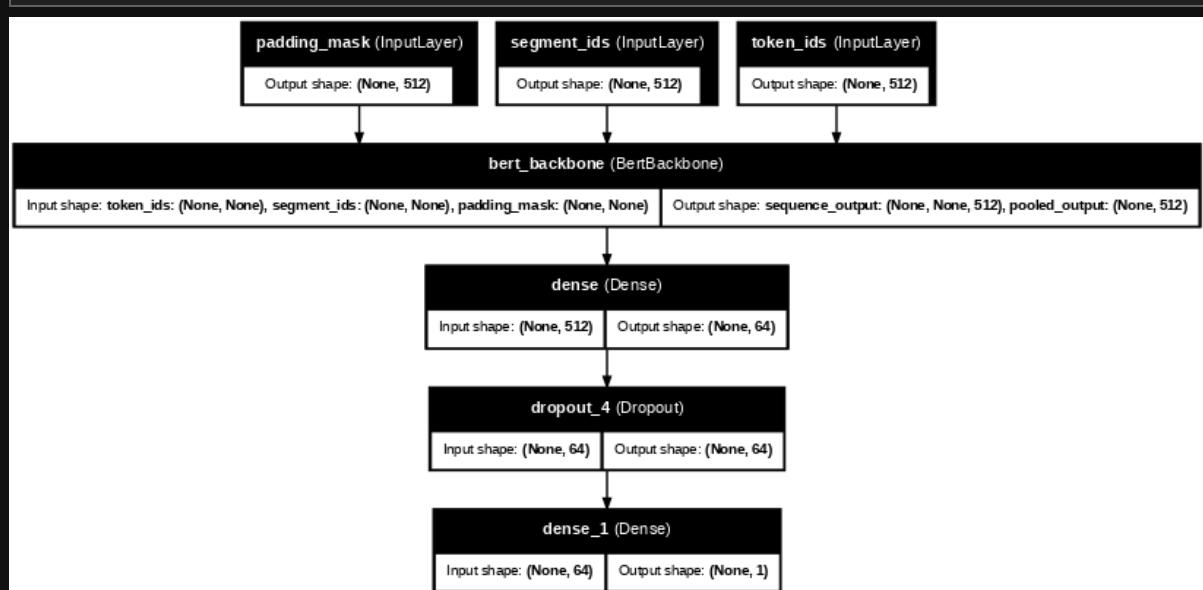
Total params: 28,796,545 (109.85 MB)

Trainable params: 28,796,545 (109.85 MB)

Non-trainable params: 0 (0.00 B)

```
In [ ]: plot_model(model_bert, show_layer_names=True, show_shapes=True, dpi=50)
```

Out []:



Let's train the BERT model

```
In [ ]: def train_model(model, train_data, val_data, epochs=100, version="base"):
    checkpoint_filepath = dst_path+f'model_{version}.keras'
    checkpoint = ModelCheckpoint(checkpoint_filepath, monitor='val_loss', save_best_only=True)
    reduce = ReduceLROnPlateau(monitor='val_loss', patience=3, verbose=1)
    early = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)
    csvlogger = CSVLogger(dst_path+f"histories/history_model_{version}.csv",
    model.compile(optimizer=Adam(learning_rate=1e-6), loss = 'binary_crossentropy')
    history = model.fit(train_data,
```

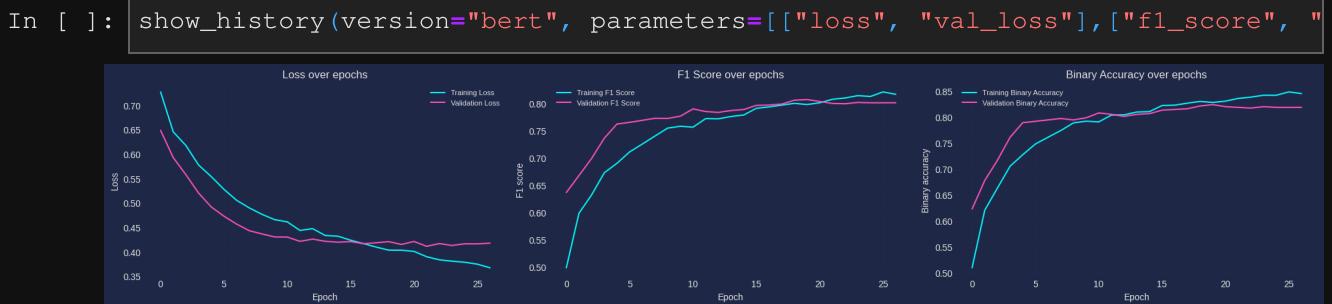
```

        validation_data=val_data,
        epochs=epochs,
        callbacks=[reduce, early, checkpoint, csvlogger],
        class_weight=class_weight
    )
    return history

history_bert = train_model(model_bert, train_data_bert, val_data_bert, epoch

```

Visualizing the metrics over training epochs of the BERT model



Evaluating the BERT model

In []: `model_bert.evaluate(test_data_bert)`

```

24/24 ━━━━━━━━━━━━━━━━ 2s 82ms/step - binary_accuracy: 0.80
      77 - f1_score: 0.7672 - loss: 0.4223
Out[ ]: [0.4325513541698456, 0.7717041373252869, 0.809395968914032]

```

Saving the preprocessor with CloudPickle

In []: `with open('preprocessor.bin', 'wb') as f_out:
 cloudpickle.dump(preprocessor, f_out)`

Evaluation metrics

Comparison table

In []: `def get_scores(y_true, y_pred, y_pred_proba):
 return {'AUC_ROC':roc_auc_score(y_true, y_pred_proba),
 'F1_Score':f1_score(y_true, y_pred, average='weighted'),
 'Accuracy':accuracy_score(y_true, y_pred),
 'Precision':precision_score(y_true, y_pred, average='weighted'),
 'Recall':recall_score(y_true, y_pred, average='weighted')}}

scores = []`

```

y_test_pred_proba_base = model_base.predict(test_data_tokenized, verbose=0)
y_test_pred_base = tf.math.round(y_test_pred_proba_base)
scores.append(get_scores(y_test, y_test_pred_base, y_test_pred_proba_base))

y_test_pred_proba_bert = model_bert.predict(test_data_bert, verbose=0)
y_test_pred_bert = tf.math.round(y_test_pred_proba_bert)
scores.append(get_scores(y_test, y_test_pred_bert, y_test_pred_proba_bert))

comparison = pd.DataFrame(data=scores, index=["Baseline Model", "BERT model"])
comparison.style.highlight_max(color = 'green', axis = 0).highlight_min(colo

```

Out []:

	AUC_ROC	F1_Score	Accuracy	Precision	Recall
Baseline Model	0.826001	0.780994	0.783893	0.784091	0.783893
BERT model	0.875719	0.808962	0.809396	0.808825	0.809396

Confusion matrix

In []:

```

def my_cm(y_true, y_pred, title):
    cm_val = confusion_matrix(y_true, y_pred)
    cm_pgs = np.round(confusion_matrix(y_true, y_pred, normalize='true')*100

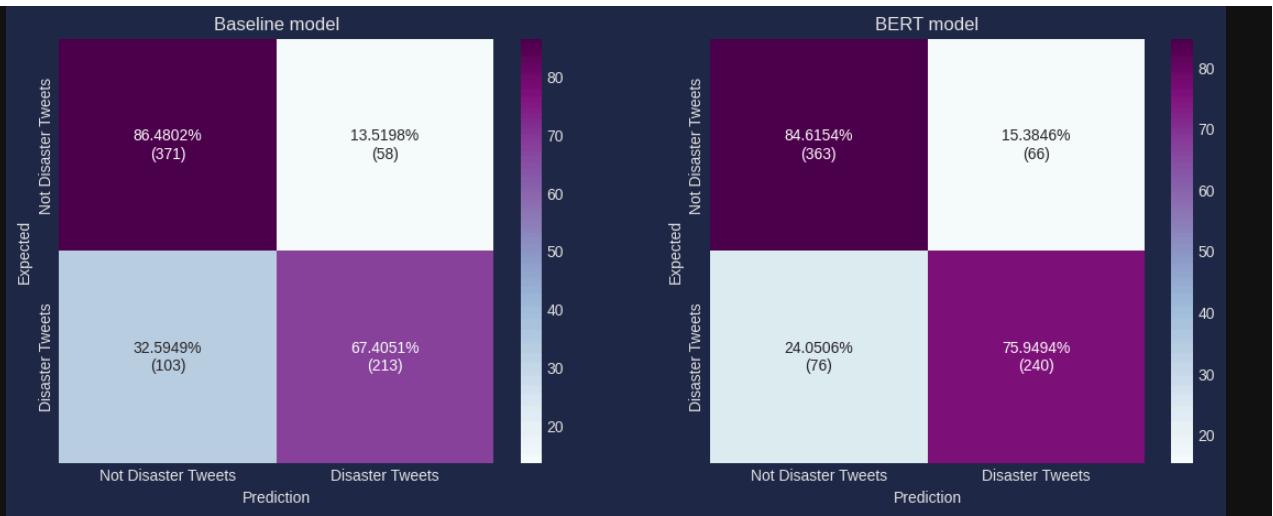
    formatted_text = (np.asarray([f"{pgs}%\n({val})" for val, pgs in zip(cm_
        sns.heatmap(cm_pgs, annot=formatted_text, fmt=' ', cmap='BuPu', yticklabe
        plt.title(title)
        plt.xlabel("Prediction")
        plt.ylabel("Expected")
        return

classes = ["Non-disaster Tweets", "Disaster Tweets"]
plt.subplots(1, 2, figsize=(14, 5))

plt.subplot(1, 2, 1)
my_cm(y_test, y_test_pred_base, title="Baseline model")

plt.subplot(1, 2, 2)
my_cm(y_test, y_test_pred_bert, title="BERT model")

```



Generating the submission file

Loading a sample submission

```
In [ ]: pd.read_csv(src_path+"nlp-getting-started/sample_submission.csv").head()
```

Out []:

	id	target
0	0	0
1	2	0
2	3	0
3	9	0
4	11	0

Loading the test file

```
In [ ]: df_submission = pd.read_csv(src_path+"nlp-getting-started/test.csv", usecols=[0,1])
df_submission.head()
```

Out []:

	id	text
0	0	Just happened a terrible car crash
1	2	Heard about #earthquake is different cities, s...
2	3	there is a forest fire at spot pond, geese are...
3	9	Apocalypse lighting. #Spokane #wildfires
4	11	Typhoon Soudelor kills 28 in China and Taiwan

Classifying the tweets in the test file

```
In [ ]: preprocessed_text = preprocessor(df_submission.text)
prediction = model_bert.predict(preprocessed_text)
prediction = tf.cast(tf.math.round(prediction), tf.uint8)
prediction
```

102/102 ━━━━━━━━━━ 13s 103ms/step

```
Out [ ]: <tf.Tensor: shape=(3263, 1), dtype=uint8, numpy=
array([[1],
       [1],
       [1],
       ...,
       [1],
       [1],
       [1]], dtype=uint8)>
```

Creating the submission file

```
In [ ]: df_submission["target"] = prediction
submission = df_submission.loc[:, ["id", "target"]]
submission.head()
```

```
Out [ ]:   id  target
```

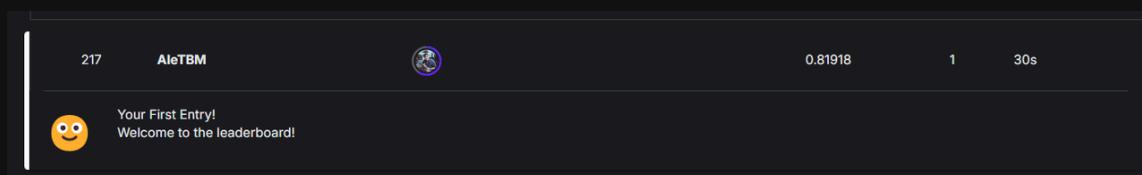
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1

Save the submission file

```
In [ ]: submission.to_csv("submission.csv", index=False)
```

My leaderboard position in the competition

Currently, my position is #217



Thank you for your time! 😊

Converting Notebook to PDF

```
In [8]: !apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic  
!pip install pypandoc nbconvert[webpdf]  
!playwright install  
!playwright install-deps  
  
from google.colab import drive  
from IPython.display import clear_output  
drive.mount('/content/drive')  
clear_output(wait=False)
```

```
In [11]: %%capture  
!jupyter nbconvert --to webpdf /content/drive/MyDrive/ML_Zoomcamps_2024/Caps
```