DisaserFINv4_1_Sentiment Analysis with Vader and Deep Learning using BERT-sentiment-analysis

---

Major References >> detailed references are attached in the bottom cell

- source : In depth series: Sentiment Analysis w Transformers (https://www.kaggle.com/code/emirkocak/in-depth-series-sentiment-analysis-w-transformers/notebook)
- source : Sentiment Analysis with Vader (https://github.com/cjhutto/vaderSentiment)
- Data_source : Disaster Tweet Sentiment Analysis (https://www.kaggle.com/code/stpeteishii/disaster-tweet-sentiment-analysis/input)

t

Vader와 딥러닝, BERT Sentiment 분석을 통한 재난 tweeter 연구분석결과

제출자 : 이지헌

제출일자 : 2024.01.31.

Colab Link : 노트북 WSL UBUNTU22.4에서 작업 후 코랩으로 올림(코랩에서 작업완료 후 본텍스트 수정예정)

- colab link (https://colab.research.google.com/drive/1Qrq6XPHI0UXBegjLlwOdqvfRlo4nWjrb#scrollTo=2bf8af15-5db8-4ba5-a202-eb0239b9237b)
- 구글 드라이브 링크 (https://drive.google.com/drive/folders/1gwjp2tFJcG5UUez44XO2gke-E5OrEtWf?usp=sharing)

- Ⅰ. Dataset Loading and Preprocessing
- Ⅱ. Tweet Vader Sentiment Analysis
- Ⅲ. EDA
- Ⅳ. Sentiment Analysis w Transformers Loading and Preprocessing
- Ⅴ. ERROR ANALYSIS
- Ⅵ. INFERENCE
- Ⅶ. References

---

1. 자료의 주요소스

- Kaggel 및 기타 온라인 자료로부터 제시된 기법들 및 데이터를 기반으로 하여 SNS tweeter 자료에 대한 감정분석(Sentiment Analysis)을 수행하고 몇가지 기법들의 결과에 대한 모델 성능 비교를 수행하였다. notebook의 모듈은 여러자료의 소스를 종합적으로 활용하였으며 데이터는 Kaggle 에서 제공되는 재난자료(https://www.kaggle.com/code/stpeteishii/disaster-tweet-sentiment-analysis/input)를 활용하여 새롭게 분석을 수행하였다. 참고한 Kaggle자료와 관련한 링크는 notebook에 reference로 제시하였다.

2. 연구목적

- 본 연구에서는 재난자료에 대한 감정분석의 두가지 접근방식을 비교하였다. 즉 Vader를 이용한 재난 Tweeter 텍스트의 감정분석을 수행하는 방식과 Transformers 모델을 기반으로 하는 언어분류기를 이용한 언어감정분석을 수행한 분류결과를 비교하였다. 시민과학이 일반화 되어가는 점을 고려할때 이와같은 인간의 행태 또는 감정 분석을 통한 현상파악은 유용한 툴이 될수 있다. 다만 사용하는 도구에 따라 텍스트자료로부터 유용한 자료의 추출이 가능하며 이를 위하여 어떤 모델을 사용할 것이고 이를 위하여 모델의 평가는 어떻게 해야하는지에 대한 충분한 숙고가 필요하다. 본 연구에서는 이에 대한 개괄적인 검토를 수행한다.

3. 감정분석 방법론에 따른 연구결과 비교

- 분류 문제의 경우 모델의 평가에는 다양한 측정지표들이 사용될수 있는데 본 연구에서는 재난관련한 tweeter text 에 대한 사전처리후 Vader를 이용한 분석결과와 자료특성에 대해 탐색적 도식화 과정을통해 살펴보았다. Wordcloud 작성시 stopword를 추가함

으로써 불용어를 조정하였다. 우선 Vader를 이용하여 예측된 Sentiment analysis 예측자료 에 대한 혼돈행렬평가를 수행행하고 그에따른 측정지표들을 평가하였다. 측정지표에는 정확도(Accuracy), 정밀도(Precision), 재현도(Recall), F-1지표를 제시하고 이들에 대한 결과를 살펴보았다. 우선 Disaster(Positive :1)에 대한 정확도, 정밀도, 재현도, F-1점수는 각각 39%, 13%, 45%, 21% 가 제시되었다. 전반적인 예측성능은 그다지 높지 않은 것으로 나타났다.

- BERT모델을 이용하기에 앞서 자료를 훈련용(training set), 테스트용(testing set), 검증용(Validation set)의 총 3가지로 구분하여 과적합 방지를 도모하였다. 이와같은 Transformer 모델을 이용할 경우 일반적으로 보다 세밀한 model tuining이 가능한 것으로 알려져 있다. 본 연구의 경우 모델구동시간상의 한계로 인하여 epoch 수를 25회로 한정하였으며 그 중 결과가 좋은 모델(F1 index가 가장 큰 것, F1 Score (Weighted): 0.918)을 이용한 테스팅을 진행하였다. 다만 테스팅 평가결과가 훈련결과에 비해 너무 낮게 나타나는 경우 과적합이 발생될수 있으므로 적합된 모델중 적절한 것을 골라서 검증세트에 대한 모델 평가를 진행하였으며 그 결과는 다음과 같다.
- 측정지표에는 정확도(Accuracy), 정밀도(Precision), 재현도(Recall), F-1지표를 제시하고 이들에 대한 결과를 살펴보았다. 우선 Disaster(Positive :1)에 대한 정확도, 정밀도, 재현도, F-1점수는 각각 92%, 73%, 78%, 75% 가 제시되었다. 전반적인 예측성능은 Vader를 이용한 분류결과에 비해 상당히 향상된 결과를 나타내고 있다.
- 이 자료의 경우에는 자료가 고르게 분포하고있지 않으므로 F-1 index를 이용하여 정밀도와 재현도가 고르게 고려된 성능을 평가하는 것으 적합해 보이며 그 결과 Bart 모델을 이용한 분석이 더 적합한 것으로 보여진다.

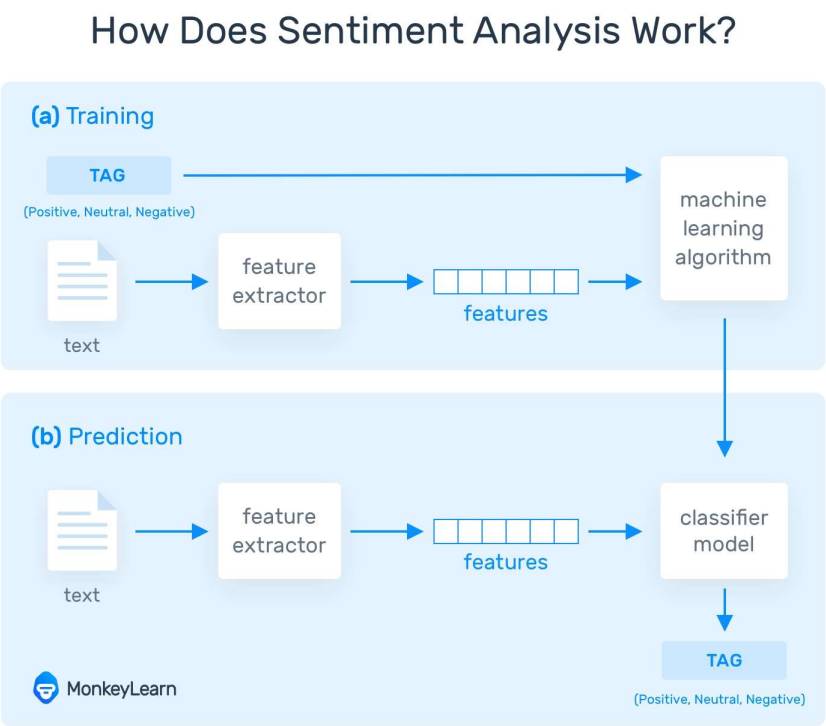
재난자료에 대한 감정분석의 두가지 접근방식을 비교하였다.

- 즉 Vader 텍스트의 감정분석을 수행하는 방식과
- Burt 모델을 기반으로 하는 언어분류기를 이용한 언어감정분석을 수행한 분류결과를 비교하였다.

In this notebook, you will:

- Load the Disaster Tweets
- Explore the dataset
- Preprocess the da
- Sentiment Analysis using Vader tLP
- Train your own model, fine-tuniEg BERT
- Generate the submission filehe submissionfile n file

How Does Sentiment Analysis Work?



source = https://monkeylearn.com/sentiment-analysis/

Sentiment analysis works to automatically determine emotional tone thanks to natural language processing (NLP), rule-based methods, and machine learning algorithms.

There are different ways we can do sentiment analysis, depending on how much data you need to analyze, how accurate your model needs to be, and how many resources you have.

We will talk about some of them below.

—

Data_source : https://www.kaggle.com/code/stpeteishii/disaster-tweet-sentiment-analysis/input

Figure

Ⅰ. Dataset Loading and Preprocessing

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
from distutils.dir_util import copy_tree
import os
print(os.getcwd())
copy_tree("/content/drive/MyDrive/KMOOC_TWEETER_Sentiment_Analysis", "/content/KMOOC_TWEETER_Sentiment_Analysis")
os.chdir('/content/drive/MyDrive/KMOOC_TWEETER_Sentiment_Analysis')
print(os.getcwd())
```

```
/content/drive/My Drive/KMOOC_TWEETER_Sentiment_Analysis
/content/drive/MyDrive/KMOOC_TWEETER_Sentiment_Analysis
```

```
import numpy as np
import pandas as pd
import os
from tqdm import tqdm

#For Preprocessing
import re    # RegEx for removing non-letter characters
import nltk  # natural language processing

nltk.download("stopwords")
nltk.download('vader_lexicon')

from nltk.corpus import stopwords
from nltk.stem.porter import *
from nltk.sentiment.vader import SentimentIntensityAnalyzer   #####

# For Building the model
from sklearn.model_selection import train_test_split
import tensorflow as tf
import seaborn as sns

#For data visualization
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

pd.options.plotting.backend = "plotly"
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
# Load Tweet dataset
df0 = pd.read_csv('./input/disaster_tweets.csv')
print(os.getcwd())
df0
```

```
/content/drive/MyDrive/KMOOC_TWEETER_Sentiment_Analysis
```

	id	keyword	location	text	target
0	0	ablaze	NaN	Communal violence in Bhainsa, Telangana. "Ston...	1
1	1	ablaze	NaN	Telangana: Section 144 has been imposed in Bha...	1
2	2	ablaze	New York City	Arsonist sets cars ablaze at dealership https:...	1
3	3	ablaze	Morgantown, WV	Arsonist sets cars ablaze at dealership https:...	1
4	4	ablaze	NaN	"Lord Jesus, your love brings freedom and pard...	0
...	...	...	...	...	...
11365	11365	wrecked	Blue State in a red sea	Media should have warned us well in advance. T...	0
11366	11366	wrecked	arohaonces	i feel directly attacked 💀 i consider moonbin ...	0

```
df0 = df0.drop(columns=['id', 'keyword', 'location'])
# df.columns = ['text', 'category']
# df.index.name = 'id'
df0['Category']='disaster'
df0.loc[df0['target']==0,'Category']='normal'
# df0 = df0.drop(columns=['target'])
df0.rename(columns ={'text':'Text', 'target':'Label'},inplace=True)

df0
```

	Text	Label	Category
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster
1	Telangana: Section 144 has been imposed in Bha...	1	disaster
2	Arsonist sets cars ablaze at dealership https:...	1	disaster
3	Arsonist sets cars ablaze at dealership https:...	1	disaster
4	"Lord Jesus, your love brings freedom and pard...	0	normal
...	...	...	...
11365	Media should have warned us well in advance. T...	0	normal
11366	i feel directly attacked 💀 i consider moonbin ...	0	normal
11367	i feel directly attacked 💀 i consider moonbin ...	0	normal
11368	ok who remember "outcast" nd the "dora" au?? T...	0	normal
11369	Jake Corway wrecked while running 14th at IRP.	1	disaster

11370 rows × 3 columns

```
Name0=df0['Category'].unique().tolist()
Name=sorted(Name0)
print(Name)
Name.reverse()
print(Name)
N=list(range(len(Name)))
normal_mapping=dict(zip(Name,N))
reverse_mapping=dict(zip(N,Name))
# df0['Label']=df0['Label'].map(normal_mapping)
```

```
['disaster', 'normal']
['normal', 'disaster']
```

```
print(df0['Label'].unique())
print(Name0)
print(N)
print(normal_mapping)
print(reverse_mapping)
```

```
[1 0]
['disaster', 'normal']
[0, 1]
{'normal': 0, 'disaster': 1}
{0: 'normal', 1: 'disaster'}
```

```
df=df0[['Text','Label','Category']][0:10000]
df
```

	Text	Label	Category
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster
1	Telangana: Section 144 has been imposed in Bha...	1	disaster
2	Arsonist sets cars ablaze at dealership https:...	1	disaster
3	Arsonist sets cars ablaze at dealership https:...	1	disaster
4	"Lord Jesus, your love brings freedom and pard...	0	normal
...	...	...	...
9995	3yrs after IPOB formed Biafra security Service...	0	normal
9996	France agrees to send more troops to West Afri...	1	disaster
9997	While the press feasts off a tiny "he-said, sh...	0	normal
9998	● NEWS ● #meduza #russia ☞ Man who made Russia...	0	normal
9999	Donald Trump posted a doctored islamophobia ph...	0	normal

10000 rows × 3 columns

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stopWords_nltk = set(stopwords.words("english"))    # stop words
stopWords_nltk.update(('https', 'https://', 'http', 'http://', '.co', 'co','1','2','3','4','5','6','7','8','9'))            # add st
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def tweet_to_words(tweet):
    ''' Convert tweet text into a sequence of words '''

    # convert to lowercase
    text = tweet.lower()
    # remove non letters
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    # tokenize
    words = text.split()
    # remove stopwords
    words = [w for w in words if w not in stopWords_nltk]
    # apply stemming
    words = [PorterStemmer().stem(w) for w in words]
    # return list
    return words
```

```
cleantext=[]
for item in tqdm(df['Text']):
    words=tweet_to_words(str(item))
    cleantext+=[words]
df['cleantext']=cleantext
df
```

```
100%|██████████| 10000/10000 [00:03<00:00, 2685.71it/s]
```

	Text	Label	Category	cleantext
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster	[commun, violenc, bhainsa, telangana, stone, p...
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0gl7nu...
4	"Lord Jesus, your love brings freedom and pard...	0	normal	[lord, jesu, love, bring, freedom, pardon, fil...
...	...	...	...	...
9995	3yrs after IPOB formed Biafra security Service...	0	normal	[3yr, ipob, form, biafra, secur, servic, yorub...
9996	France agrees to send more troops to West Afri...	1	disaster	[franc, agre, send, troop, west, africa, fight...

```
def unlist(list):
    words=''
    for item in list:
        words+=item+' '
    return words
```

```
1
```

Ⅱ. Tweet Vader Sentiment Analysis

SentimentIntensityAnalyzer()

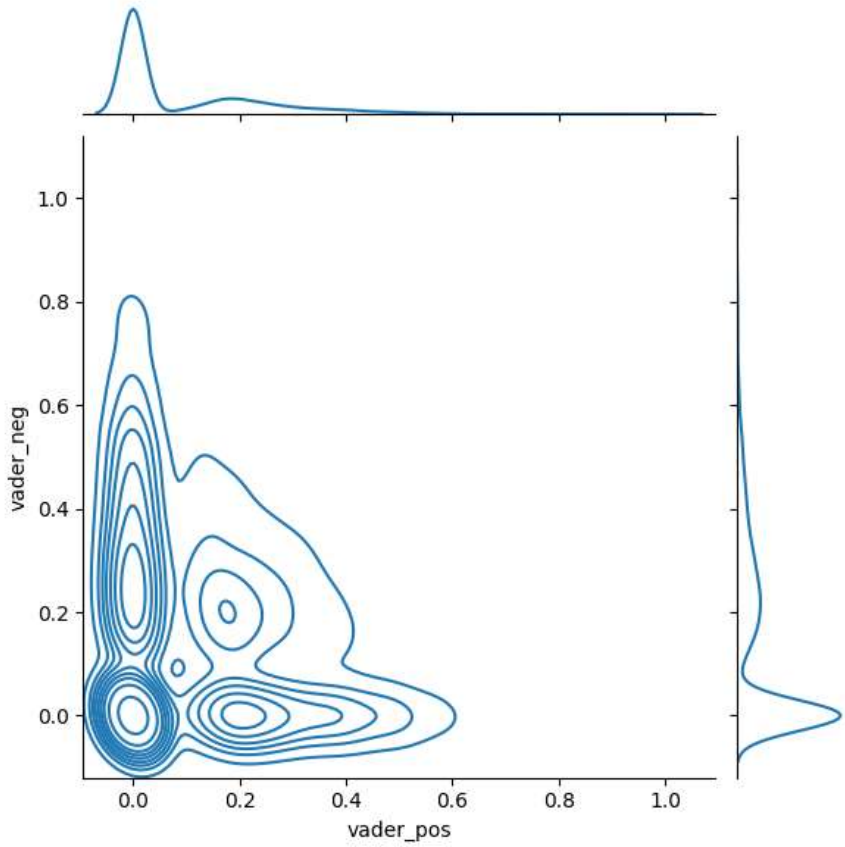
```
def compute_vader_scores(df, label):
    sid = SentimentIntensityAnalyzer()
    df["vader_neg"] = df[label].apply(lambda x: sid.polarity_scores(unlist(x))["neg"])
    df["vader_neu"] = df[label].apply(lambda x: sid.polarity_scores(unlist(x))["neu"])
    df["vader_pos"] = df[label].apply(lambda x: sid.polarity_scores(unlist(x))["pos"])
    df["vader_comp"] = df[label].apply(lambda x: sid.polarity_scores(unlist(x))["compound"])
    df['cleantext2'] = df[label].apply(lambda x: unlist(x))
    return df
```

```
df2 = compute_vader_scores(df,'cleantext')
df2
```

	Text	Label	Category	cleantext	vader_neg	vader_neu	vader_pos \
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster	[commun, violenc, bhainsa, telangana, stone, p...	0.000	1.000	0.000
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...	0.205	0.795	0.000
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...	0.000	1.000	0.000
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0gl7nu...	0.000	1.000	0.000
4	"Lord Jesus, your love brings freedom and pard...	0	normal	[lord, jesu, love, bring, freedom, pardon, fil...	0.000	0.421	0.579
...	...	...	...	...	...	...	...
9995	3yrs after IPOB formed Biafra security Service...	0	normal	[3yr, ipob, form, biafra, secur, servic, yorub...	0.000	1.000	0.000
9996	France agrees to send more troops to West Afri...	1	disaster	[franc, agre, send, troop, west, africa, fight...	0.462	0.538	0.000
9997	While the press feasts off a tiny	0	normal	[press, feast, tini, said, said, tiff,	0.139	0.861	0.000

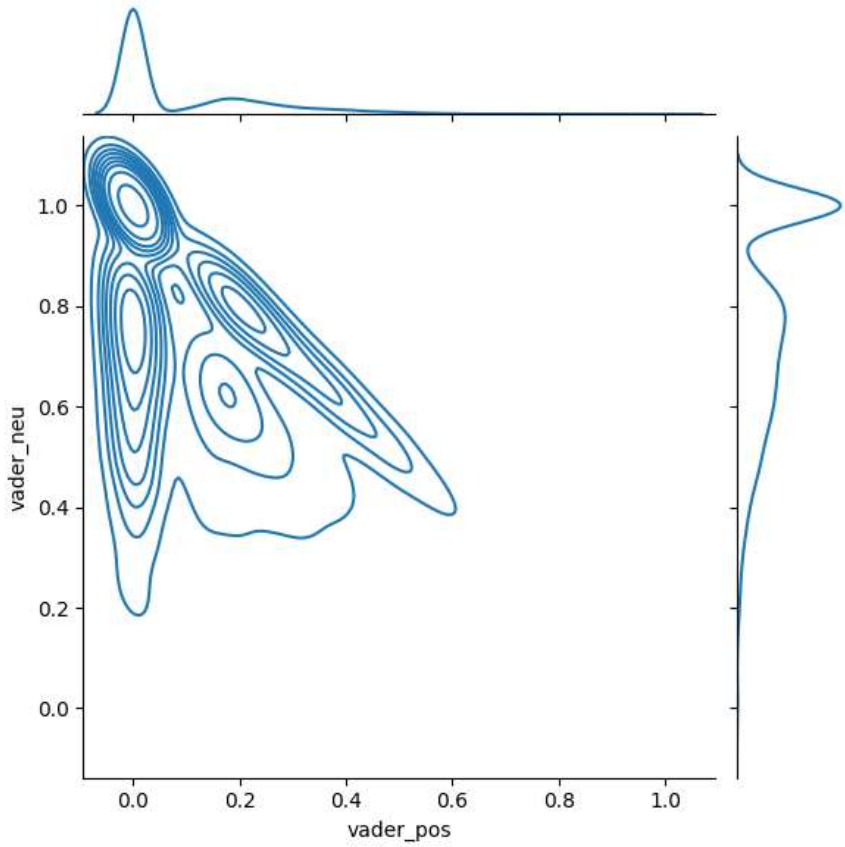
```
sns.jointplot(data=df2, x='vader_pos', y='vader_neg', kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7df889c25f60>
```



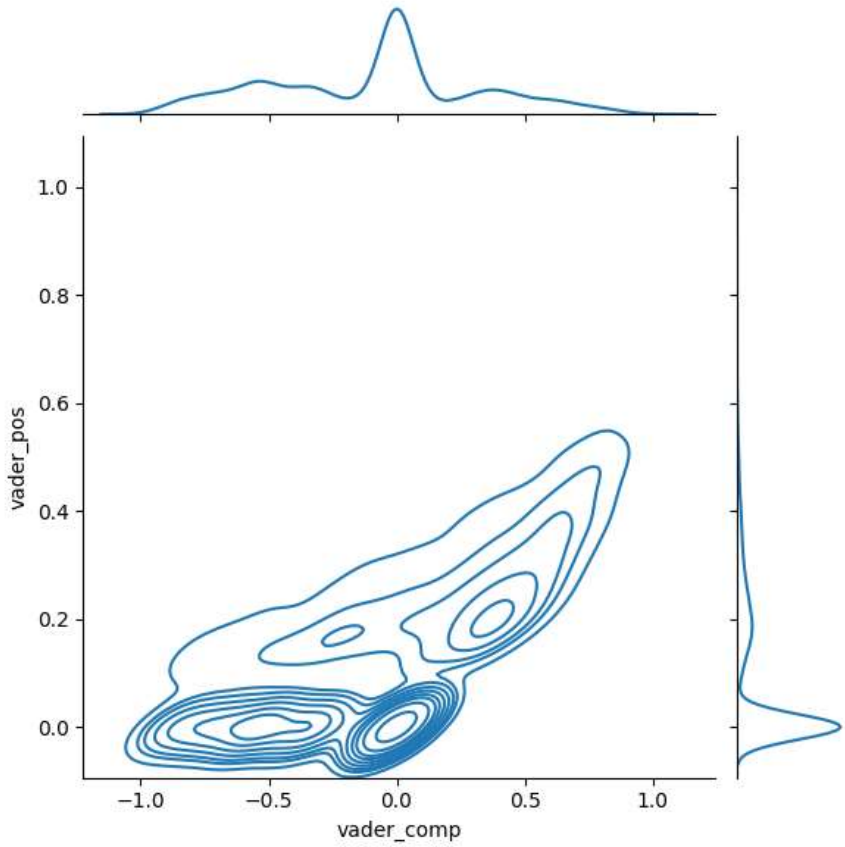

```
sns.jointplot(data=df2, x='vader_pos', y='vader_neu', kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7df8b7196950>
```




```
sns.jointplot(data=df2, x='vader_comp', y='vader_pos', kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7df8b6c61db0>
```



Class created

https://github.com/cjhutto/vaderSentiment

```
class0=[]
for i in range(len(df2)):
    if df2.loc[i,'vader_comp']<0.0:
        class0+=[0]
    else:
        class0+=[1]

df2['class']=class0
df2['class'].value_counts()
#['Vader_Nagative','Vader_Positive']
```

```
1    5914
0    4086
Name: class, dtype: int64
```

```
df2
```

	Stori...			stone, p...			
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...	0.205	0.795	0.00(
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...	0.000	1.000	0.00(
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0gl7nu...	0.000	1.000	0.00(
4	"Lord Jesus, your love brings freedom and pard...	0	normal	[lord, jesu, love, bring, freedom, pardon, fil...	0.000	0.421	0.57!
...	...	...	...	...	...	...	..
9995	3yrs after IPOB formed Biafra security Service...	0	normal	[3yr, ipob, form, biafra, secur, servic, yorub...	0.000	1.000	0.00(
9996	France agrees to send more troops to West Afri...	1	disaster	[franc, agre, send, troop, west, africa, fight...	0.462	0.538	0.00(
9997	While the press feasts off a tiny "he-said, sh...	0	normal	[press, feast, tini, said, said, tiff, warren,...	0.139	0.861	0.00(
9998	● NEWS ● #meduza #russia ☞ Man who made Russia...	0	normal	[news, meduza, russia, man, made, russia, firs...	0.333	0.667	0.00(
9999	Donald Trump posted a doctored islamophobia ph...	0	normal	[donald, trump, post, doctor, islamophobia, ph...	0.000	1.000	0.00(

10000 rows × 10 columns

```
df.groupby(['Label','class']).count()
```

		Text	Category	cleantext	vader_neg	vader_neu	vader_pos	vader_co
Label	class							
0	**0**	3113	3113	3113	3113	3113	3113	31
	1	5121	5121	5121	5121	5121	5121	51
1	**0**	973	973	973	973	973	973	9
	1	793	793	793	793	793	793	7

Vader-created classes and original labels

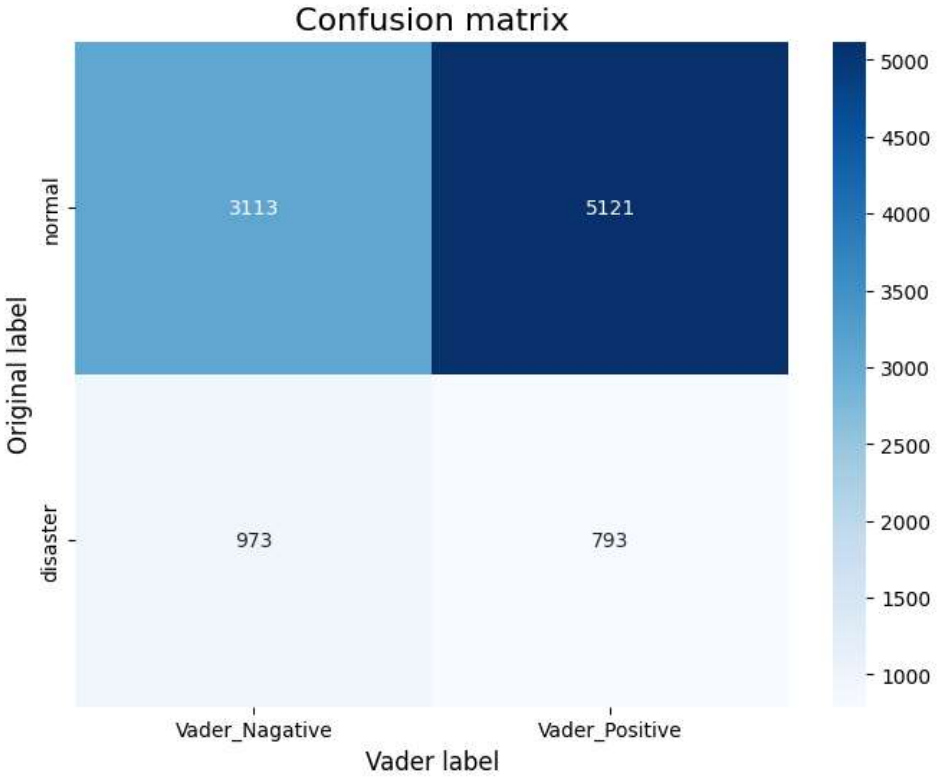
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(df2['Label'],df2['class'])
cm
```

```
array([[3113, 5121],
       [ 973,  793]])
```

```
Vader_class_name=['Vader_Nagative','Vader_Positive']
Label_name=Name
```

```
# plot confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap=plt.cm.Blues, annot=True, fmt='d',
            xticklabels=Vader_class_name,
            yticklabels=Label_name)
plt.title('Confusion matrix', fontsize=16)
plt.xlabel('Vader label', fontsize=12)
plt.ylabel('Original label', fontsize=12)
```

```
Text(70.72222222222221, 0.5, 'Original label')
```



Classification Reports for Vader Sentiment Analysis result

```
from sklearn.metrics import classification_report

print(classification_report(df2['Label'],df2['class']))
```

```
              precision    recall  f1-score   support

           0       0.76      0.38      0.51      8234
           1       0.13      0.45      0.21      1766

    accuracy                           0.39     10000
   macro avg       0.45      0.41      0.36     10000
weighted avg       0.65      0.39      0.45     10000
```

```
```

Ⅲ. EDA

Sentiment Analysis w Transformers

```
import pandas as pd
from wordcloud import WordCloud
import seaborn as sns
import re
import string
from collections import Counter, defaultdict

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from plotly.offline import plot

import matplotlib.gridspec as gridspec
from matplotlib.ticker import MaxNLocator
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
```

```
import warnings
warnings.filterwarnings('ignore')
```

Helper Functions

```
import re
from typing import Union, List

class CleanText():
    """ clearing text except digits () . , word character """

    def __init__(self, clean_pattern = r"[^A-ZĞÜŞİÖÇa-zğüıöçŞ0-9.₩"',()]"):
        self.clean_pattern =clean_pattern

    def __call__(self, text: Union[str, list]) -> List[List[str]]:

        if isinstance(text, str):
            docs = [[text]]

        if isinstance(text, list):
            docs = text

        text = [[re.sub(self.clean_pattern, " ", sent) for sent in sents] for sents in docs]

        return text

def remove_emoji(data):
    emoj = re.compile("["
        u"₩U0001F600-₩U0001F64F"  # emoticons
        u"₩U0001F300-₩U0001F5FF"  # symbols & pictographs
        u"₩U0001F680-₩U0001F6FF"  # transport & map symbols
        u"₩U0001F1E0-₩U0001F1FF"  # flags (iOS)
        u"₩U00002500-₩U00002BEF"
        u"₩U00002702-₩U000027B0"
        u"₩U00002702-₩U000027B0"
        u"₩U000024C2-₩U0001F251"
        u"₩U0001f926-₩U0001f937"
        u"₩U00010000-₩U0010ffff"
        u"₩u2640-₩u2642"
        u"₩u2600-₩u2B55"
        u"₩u200d"
        u"₩u23cf"
        u"₩u23e9"
        u"₩u231a"
        u"₩ufe0f"  # dingbats
        u"₩u3030"
                      "]+", re.UNICODE)
    return re.sub(emoj, '', data)

def tokenize(text):
    """ basic tokenize method with word character, non word character and digits """
    text = re.sub(r" +", " ", str(text))
    text = re.split(r"(₩d+|[a-zA-Zğüşıöçğüşİöç]+|₩W)", text)
    text = list(filter(lambda x: x != '' and x != ' ', text))
    sent_tokenized = ' '.join(text)
    return sent_tokenized

regex = re.compile('[%s]' % re.escape(string.punctuation))

def remove_punct(text):
    text = regex.sub(" ", text)
    return text

clean = CleanText()

```

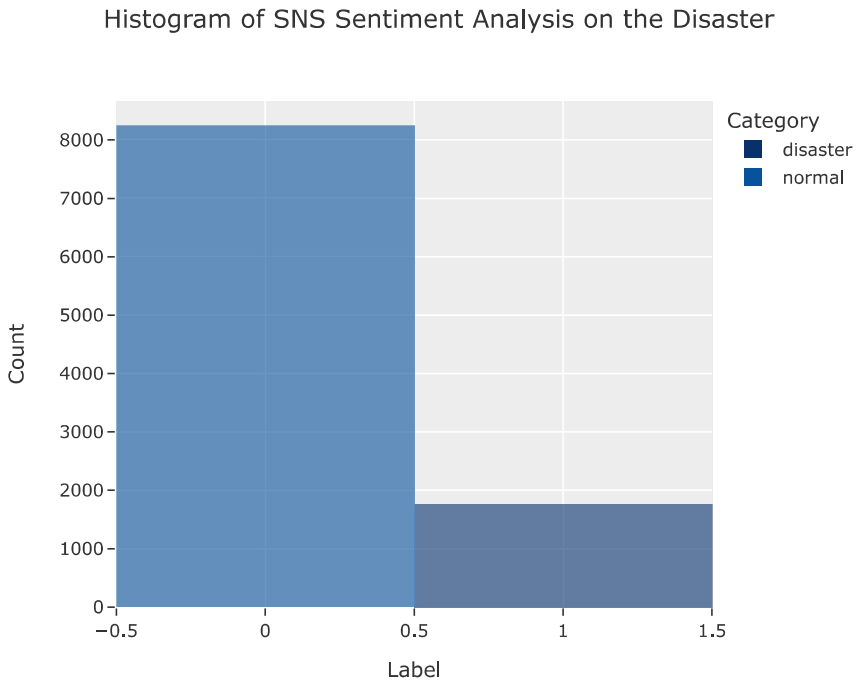
```
# label encode
def label_encode(x):
    if x == 0:
        return 0
    if x == 1:
        return 1

# label to name
def label2name(x):
    if x == 0:
        return "Negative"
    if x == 1:
        return "Positive"

```

Data Loading

```
df2.head()
```

	Text	Label	Category	cleantext	vader_neg	vader_neu	vader_pos	vader_cc
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster	[commun, violenc, bhainsa, telangana, stone, p...	0.000	1.000	0.000	0.0(
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...	0.205	0.795	0.000	-0.5!
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...	0.000	1.000	0.000	0.0(
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0ql7pu...	0.000	1.000	0.000	0.0(

```
# count of ratings
fig = px.histogram(df2,
             x = 'Label',
             title = 'Histogram of SNS Sentiment Analysis on the Disaster',
             template = 'ggplot2',
             color = 'Category',
             color_discrete_sequence= px.colors.sequential.Blues_r,
             opacity = 0.6,
             height = 525,
             width = 600,
            )

fig.update_yaxes(title='Count')
fig.show()
```




```
# basic info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Text        10000 non-null  object
 1   Label       10000 non-null  int64
 2   Category    10000 non-null  object
 3   cleantext   10000 non-null  object
 4   vader_neg   10000 non-null  float64
 5   vader_neu   10000 non-null  float64
 6   vader_pos   10000 non-null  float64
 7   vader_comp  10000 non-null  float64
 8   cleantext2  10000 non-null  object
 9   class       10000 non-null  int64
dtypes: float64(4), int64(2), object(4)
memory usage: 781.4+ KB
```

```
# encode label and mapping label name
df2["label"] = df2["Label"].apply(lambda x: label_encode(x))
df2["label_name"] = df2["label"].apply(lambda x: label2name(x))
```

```
# clean text, lowercase and remove punk
df2["cleantext2"] = df2["cleantext2"].apply(lambda x: remove_punct(clean(remove_emoji(x).lower())[0][0]))
```

```
df2.head(5)
```

	Text	Label	Category	cleantext	vader_neg	vader_neu	vader_pos	vader_cc
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster	[commun, violenc, bhainsa, telangana, stone, p...	0.000	1.000	0.000	0.0(
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...	0.205	0.795	0.000	-0.5
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...	0.000	1.000	0.000	0.0(
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0gl7nu...	0.000	1.000	0.000	0.0(
4	"Lord Jesus, your love brings freedom and pard...	0	normal	[lord, jesu, love, bring, freedom, pardon, fil...	0.000	0.421	0.579	0.9(

```
1
```

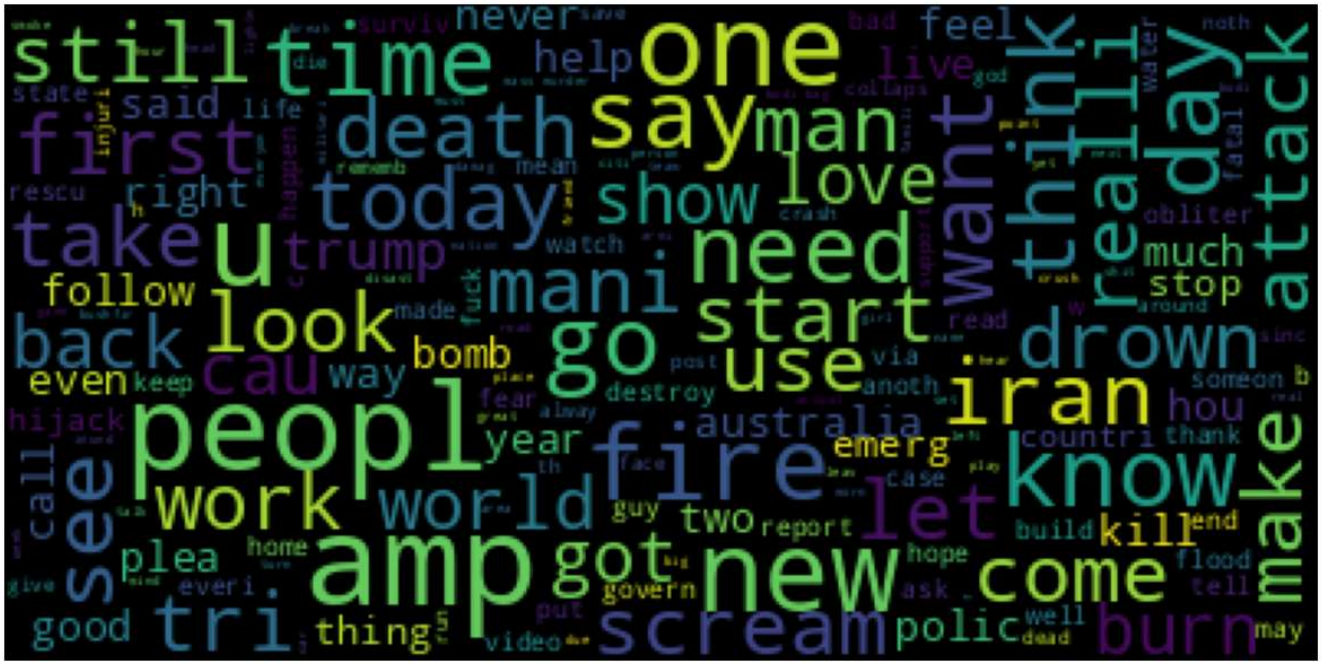
Visualizations

Word Cloud

- Word clouds generators work by breaking the text down into component words and counting how frequently they appear in the body of text. We can quickly obtain preliminary information about the data. We can understand what a dataset we don't know is talking about.

```
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='black',
        max_words=200,
        max_font_size=40,
        scale=1,
        random_state=1
).generate(" ".join(data))

    fig = plt.figure(1, figsize=(15, 15))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

```
show_wordcloud(df2["cleantext2"].values)
```



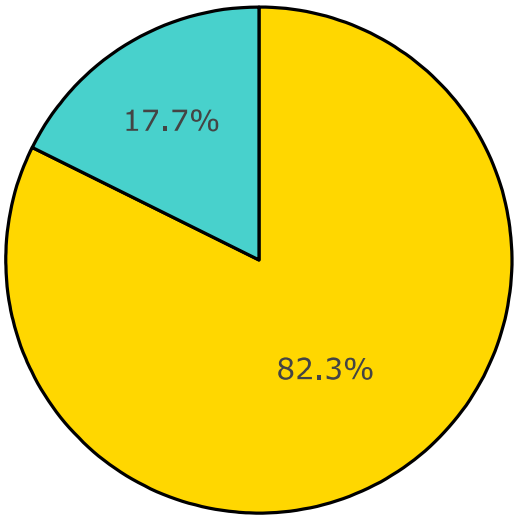

```
df.columns
```

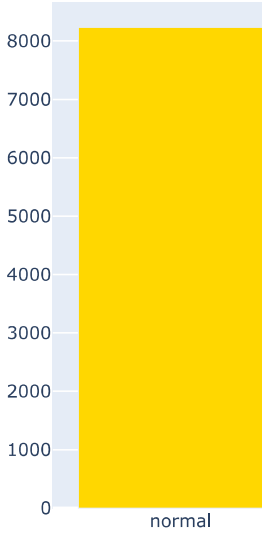
```
Index(['Text', 'Label', 'Category', 'cleantext', 'vader_neg', 'vader_neu',
       'vader_pos', 'vader_comp', 'cleantext2', 'class', 'label',
       'label_name'],
      dtype='object')
```

Target Count

How many targets do we have? Learning this information will give us an idea about the model we will build. It will also provide guidance on our methods of analyzing data.

```
fig = make_subplots(rows=1, cols=2, specs=[[{"type": "pie"}, {"type": "bar"}]])
colors = ['gold', 'mediumturquoise', 'lightgreen'] # darkorange
fig.add_trace(go.Pie(labels=df.Category.value_counts().index,
                             values=df.Label.value_counts().values), 1, 1)

fig.update_traces(hoverinfo='label+percent', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=2)))

fig.add_trace(go.Bar(x=df.Category.value_counts().index, y=df.Label.value_counts().values, marker_color = colors), 1,2)

fig.show()
```



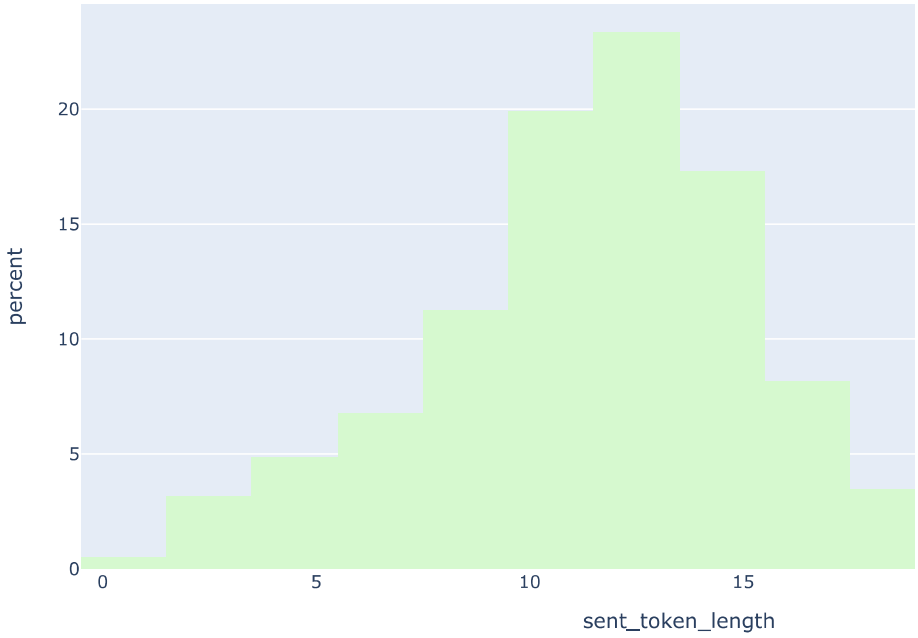


Token Counts with simple tokenizer

Finding out the number of tokens available for each sample will give us information about the length of our data. The classification algorithm we will use for a long text will not be the same as the algorithm used for a short text.

```
# tokenize data
df2["tokenized_Text"] = df.cleantext2.apply(lambda x: tokenize(x))
# calculate token count for any sent
df2["sent_token_length"] = df2["tokenized_Text"].apply(lambda x: len(x.split()))
```

```
fig = px.histogram(df, x="sent_token_length", nbins=20, color_discrete_sequence=px.colors.cmocean.algae, barmode='group', histnorm='
fig.show()
```



```
(df.sent_token_length < 25).mean()
```

```
0.9996
```

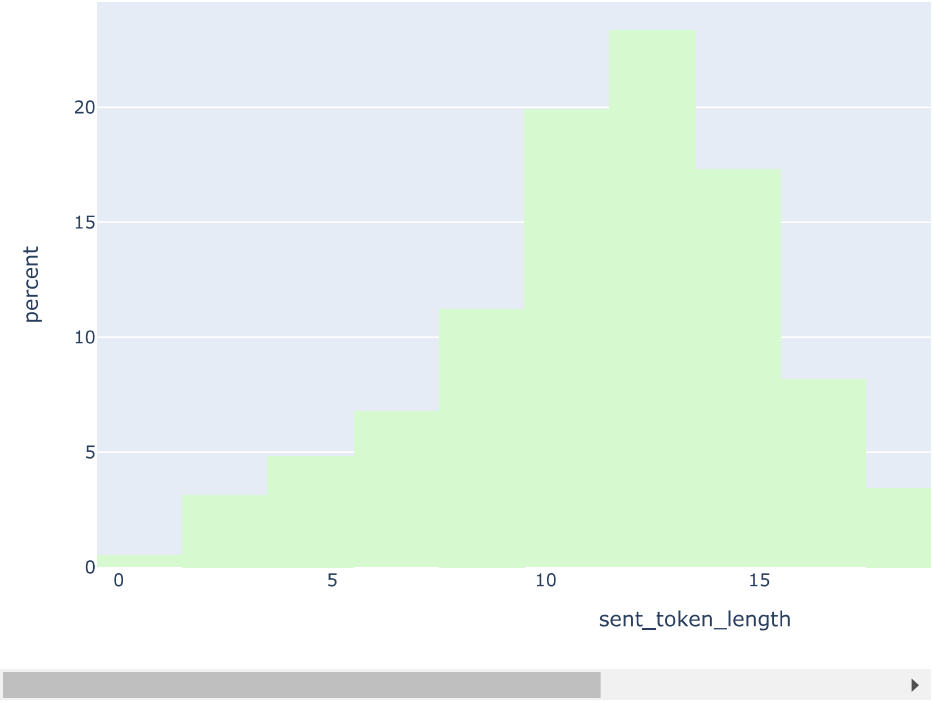
Token Counts with BERT tokenizer

Since we will create a Transformers-based model, the value that BERT tokinezer will give us is very important. With the information here, the value of the `seq_len` parameter that we will use while encoding the data will be decided.

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
                                          do_lower_case=True)
```

```
# data tokenize with bert tokenizer
df2["sent_bert_token_length"] = df2["cleantext2"].apply(lambda x: len(tokenizer(x, add_special_tokens=False)["input_ids"]))
```

```
fig = px.histogram(df, x="sent_token_length", nbins=20, color_discrete_sequence=px.colors.cmocean.algae, barmode='group', histnorm='
fig.show()
```




```
# Less than 35 covers how many of the data
(df.sent_bert_token_length < 25).mean()
```

```
0.8588
```

```

```

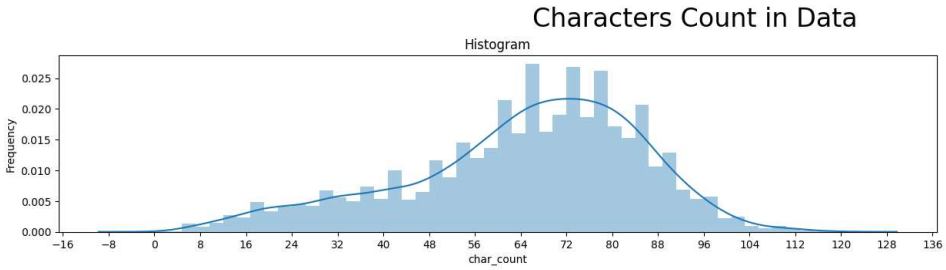
Characters Count in the Data

Let's look at the frequency of the number of characters. It will give us information about the overall size of our data

```
# valvulate char count for each Text
df2['char_count'] = df2['cleantext2'].apply(lambda x: len(str(x)))


def plot_dist3(df, feature, title):
    fig = plt.figure(constrained_layout=True, figsize=(18, 8))
    grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)
```

```
    grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)
    ax1 = fig.add_subplot(grid[0, :2])
    ax1.set_title('Histogram')
    sns.distplot(df.loc[:, feature],
                 hist=True,
                 kde=True,
                 ax=ax1,
                 )
    ax1.set(ylabel='Frequency')
    ax1.xaxis.set_major_locator(MaxNLocator(nbins=20))


    plt.suptitle(f'{title}', fontsize=24)
```

```
plot_dist3(df, 'char_count',
           'Characters Count in Data')
```



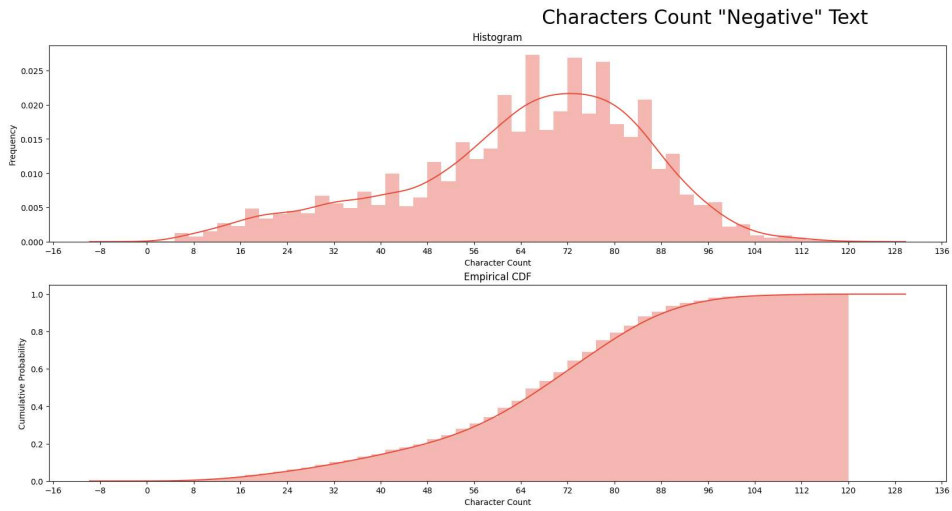
코딩을 시작하거나 AI로 코드를 생성하세요.

cleantext2 Lengths

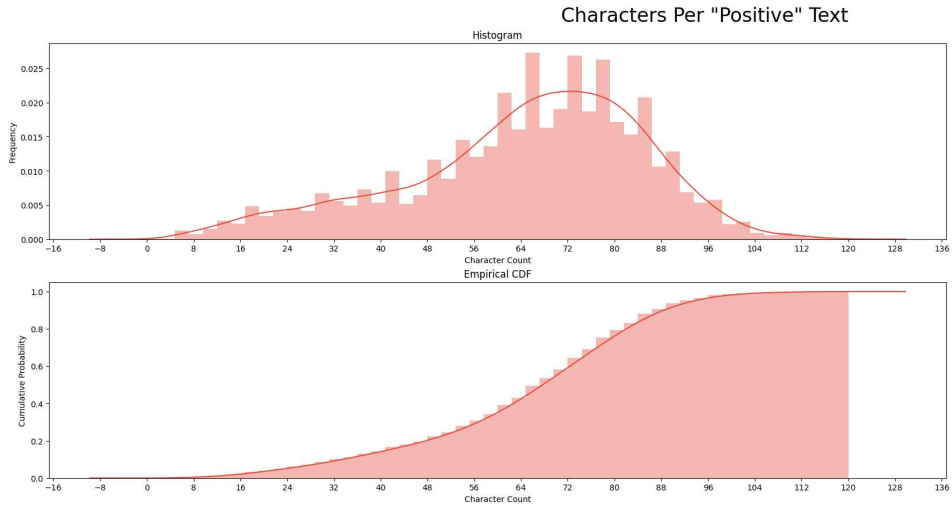
When we look at the number of characters per comment, it can give us very striking information about the data. Here, when we look at the length of the comments made by people according to their feelings, negative comments are shorter than neutral and positive comments. We can come to the notion that people simply express negative things :).

```
# Creating a new feature for the visualization.
df2['Character Count'] = df2['cleantext2'].apply(lambda x: len(str(x)))


def plot_dist3(df, feature, title):
    # Creating a customized chart. and giving in figsize and everything.
    fig = plt.figure(constrained_layout=True, figsize=(24, 12))
    # Creating a grid of 3 cols and 3 rows.
    grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)

    # Customizing the histogram grid.
    ax1 = fig.add_subplot(grid[0, :2])
    # Set the title.
    ax1.set_title('Histogram')
    # plot the histogram.
    sns.distplot(df2.loc[:, feature],
                 hist=True,
                 kde=True,
                 ax=ax1,
                 color='#e74c3c')
    ax1.set(ylabel='Frequency')
    ax1.xaxis.set_major_locator(MaxNLocator(nbins=20))

    # Customizing the ecdf_plot.
    ax2 = fig.add_subplot(grid[1, :2])
    # Set the title.
    ax2.set_title('Empirical CDF')
    # Plotting the ecdf_Plot.
    sns.distplot(df2.loc[:, feature],
                 ax=ax2,
                 kde_kws={'cumulative': True},
                 hist_kws={'cumulative': True},
                 color='#e74c3c')
    ax2.xaxis.set_major_locator(MaxNLocator(nbins=20))
    ax2.set(ylabel='Cumulative Probability')

    plt.suptitle(f'{title}', fontsize=24)
```

```
plot_dist3(df2[df2['Label'] == 0], 'Character Count',
           'Characters Count "Negative" Text')
```




```
plot_dist3(df2[df2['Label'] == 1], 'Character Count',
           'Characters Per "Positive" Text')
```



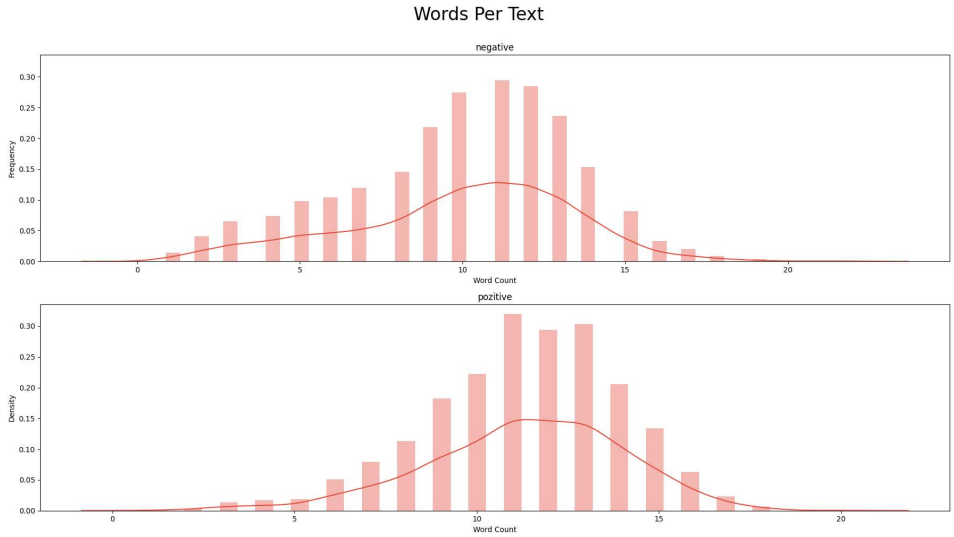

```
1
```

Word Counts

We see that the situation in the number of characters and the situation in the number of words are the same. We have seen that people use less word count when expressing negative things.

```
def plot_word_number_histogram(textno, textye):

    """A function for comparing word counts"""

    fig, axes = plt.subplots(ncols=1, nrows=2, figsize=(18, 10), sharey=True)
    sns.distplot(textno.str.split().map(lambda x: len(x)), ax=axes[0], color='#e74c3c')
    sns.distplot(textye.str.split().map(lambda x: len(x)), ax=axes[1], color='#e74c3c')

    axes[0].set_xlabel('Word Count')
    axes[0].set_ylabel('Frequency')
    axes[0].set_title('negative')
    axes[1].set_xlabel('Word Count')
    axes[1].set_title('pozitive')

    fig.suptitle('Words Per Text', fontsize=24, va='baseline')

    fig.tight_layout()
```

```
plot_word_number_histogram(df2[df2['Label'] == 0]['cleantext2'],
                           df2[df2['Label'] == 1]['cleantext2'])
```

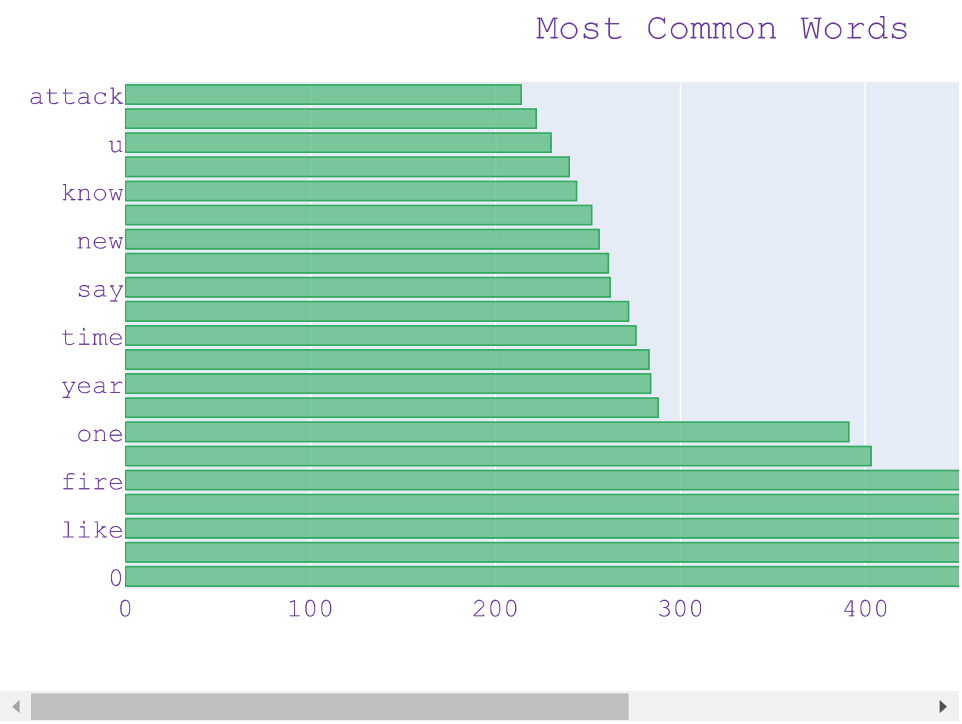



```
# remove punk
df2['tokenized_Text'] = df2['tokenized_Text'].apply(lambda x: remove_punct(x))
```

```
1
```

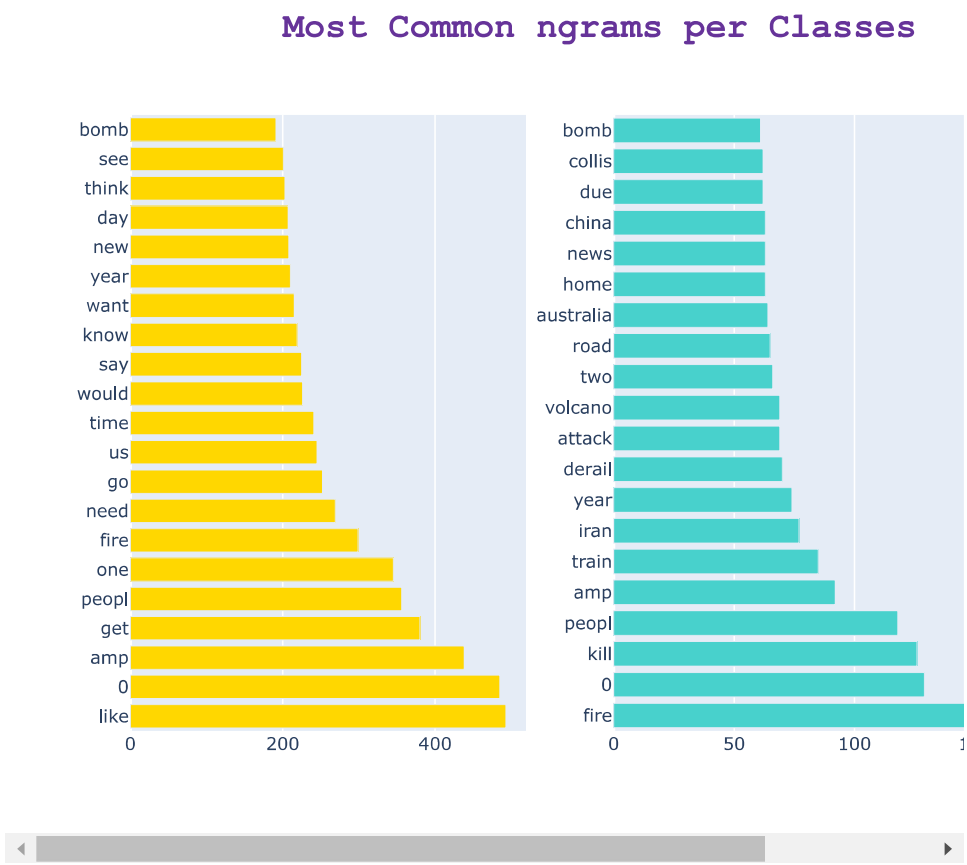
Most Common Words

```
texts = df2['tokenized_Text']
new = texts.str.split()
new = new.values.tolist()
corpus = [word for i in new for word in i]
counter = Counter(corpus)
most = counter.most_common()
x, y = [], []
for word, count in most[:30]:
    if word not in stopWords_nltk:
        x.append(word)
        y.append(count)

fig = go.Figure(go.Bar(
            x=y,
            y=x,
            orientation='h',  marker=dict(
        color='rgba(50, 171, 96, 0.6)',
        line=dict(
            color='rgba(50, 171, 96, 1.0)',
            width=1),
    ),
    name='Most common Word',))

fig.update_layout( title={
        'text': "Most Common Words",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'}, font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    ))

fig.show()
```



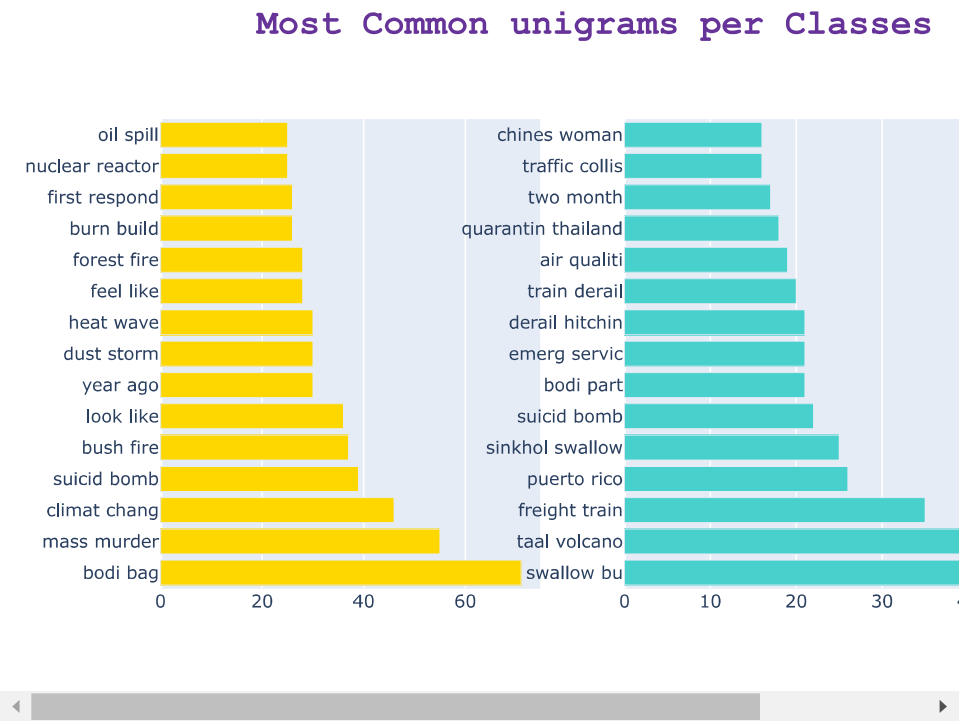
Most Common ngrams

```
fig = make_subplots(rows=1, cols=2)
title_ = ["negative", "positive"]

for i in range(2):
    texts = df2[df2["Label"] == i]['tokenized_Text']

    new = texts.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]
    counter = Counter(corpus)
    most = counter.most_common()
    x, y = [], []

    for word, count in most[:30]:
        if word not in stopWords_nltk:
            x.append(word)
            y.append(count)

    fig.add_trace(go.Bar(
                x=y,
                y=x,
                orientation='h', type="bar",
        name=title_[i], marker=dict(color=colors[i])), 1, i+1)

fig.update_layout(
    autosize=False,
    width=800,
    height=600,title=dict(
        text='<b>Most Common ngrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
        family="Courier New, monospace",
        size=24,
        color="RebeccaPurple"
        )
    ),)

fig.show()
```

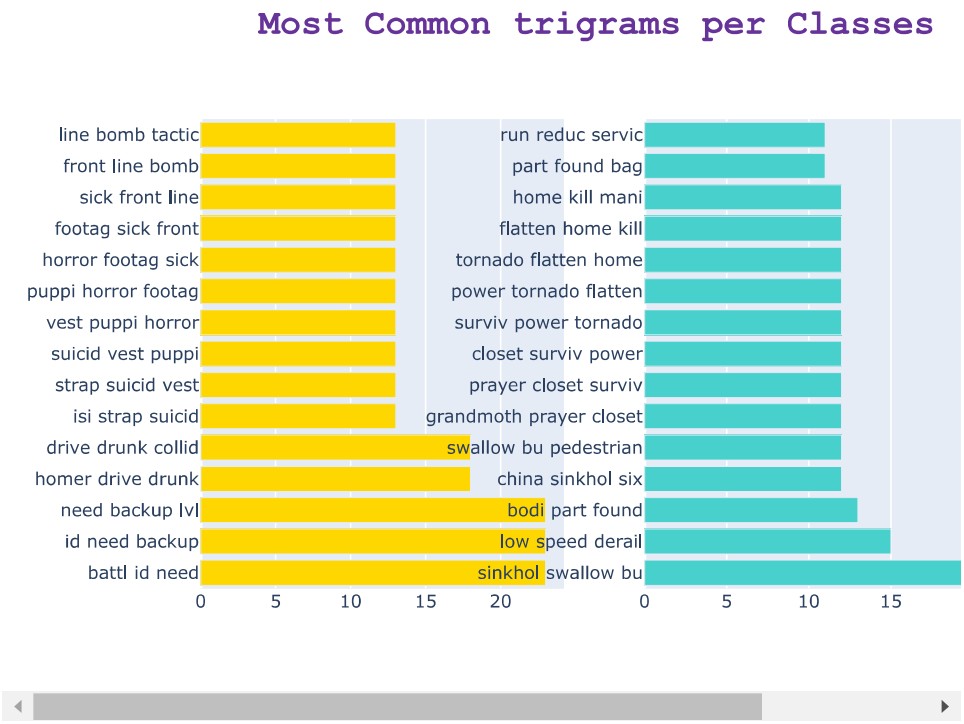


```
def _get_top_ngram(corpus, n=None):
    #getting top ngrams
    vec = CountVectorizer(ngram_range=(n, n),
                          max_df=0.9,
                          ).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
                  for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:15]
```

```
# unigram
fig = make_subplots(rows=1, cols=2)

title_ = ["negative", "positive"]

for i in range(2):
    texts = df2[df2["Label"] == i]['tokenized_Text']

    new = texts.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]
    top_n_bigrams = _get_top_ngram(texts, 2)[:15]
    x, y = map(list, zip(*top_n_bigrams))


    fig.add_trace(go.Bar(
                x=y,
                y=x,
                orientation='h', type="bar",
        name=title_[i], marker=dict(color=colors[i])), 1, i+1)


fig.update_layout(
    autosize=False,
    width=800,
    height=500,title=dict(
        text='<b>Most Common unigrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
        family="Courier New, monospace",
        size=24,
        color="RebeccaPurple"
        )
    ))
fig.show()
```




```
#trigram

fig = make_subplots(rows=1, cols=2)
title_ = ["negative", "positive"]

for i in range(2):
    texts = df2[df2["Label"] == i]['tokenized_Text']

    new = texts.str.split()
    new = new.values.tolist()
    corpus = [word for i in new for word in i]

    top_n_bigrams = _get_top_ngram(texts, 3)[:15]
    x, y = map(list, zip(*top_n_bigrams))

    fig.add_trace(go.Bar(
                x=y,
                y=x,
                orientation='h', type="bar",
        name=title_[i], marker=dict(color=colors[i])), 1, i+1),

fig.update_layout(
    autosize=False,
    width=800,
    height=500,title=dict(
        text='<b>Most Common trigrams per Classes</b>',
        x=0.5,
        y=0.95,
        font=dict(
        family="Courier New, monospace",
        size=24,
        color="RebeccaPurple"
        )
    ))

fig.show()
```

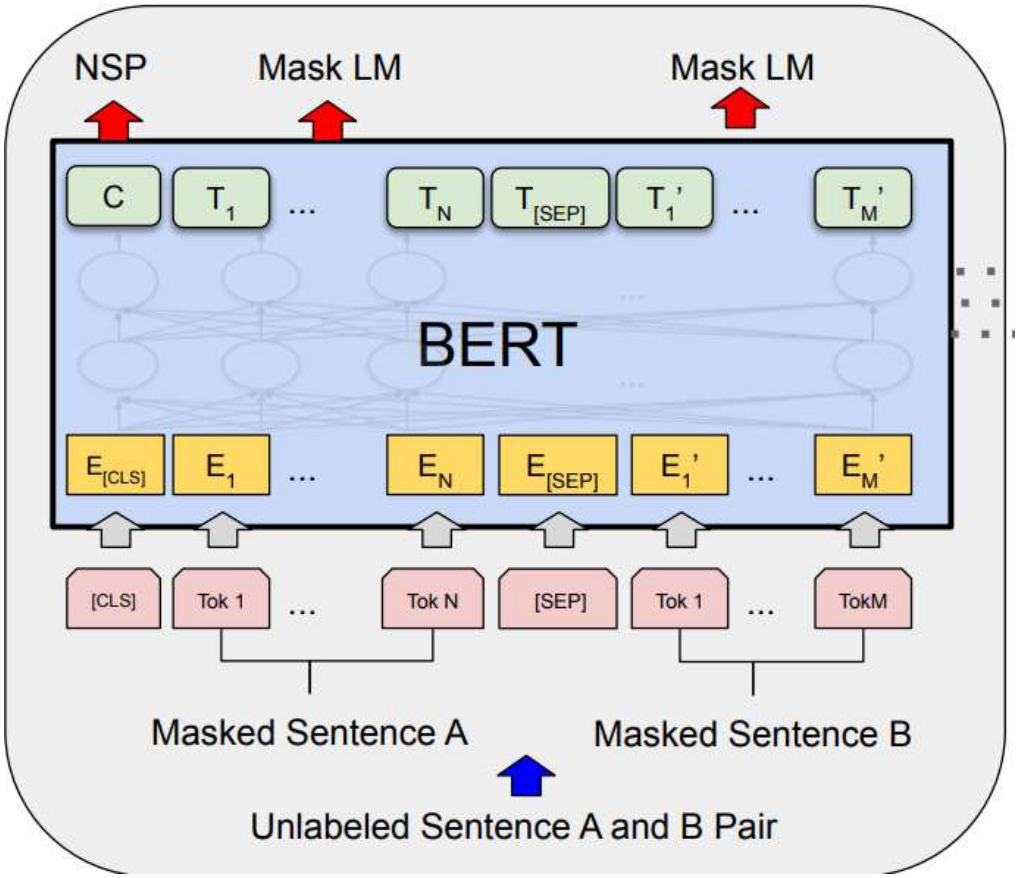


We examined and visualized the data, now we can move on to the model building part.

1

IV. Sentiment Analysis w Transformers Loading and Preprocessing

A brief information about BERT

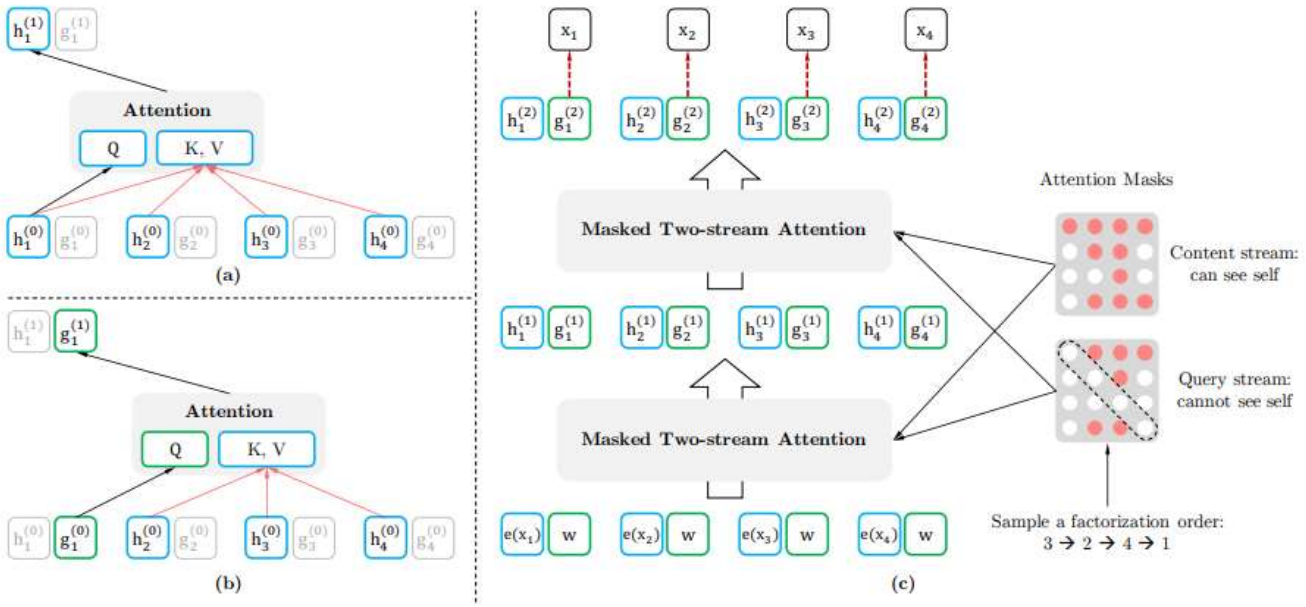


BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

BERT is a bi-directional transformer for pre-training over a lot of unlabeled textual data to learn a language representation that can be used to fine-tune for specific machine learning tasks. While BERT outperformed the NLP state-of-the-art on several challenging tasks, its performance improvement could be attributed to the bidirectional transformer, novel pre-training tasks of Masked Language Model and Next Structure Prediction along with a lot of data and Google's compute power.

The detailed workings of Transformer are described in a paper by Google.

A brief information about XLNET



XLNet is a large bidirectional transformer that uses improved training methodology, larger data and more computational power to achieve better than BERT prediction metrics on 20 language tasks.

To improve the training, XLNet introduces permutation language modeling, where all tokens are predicted but in random order. This is in contrast to BERT's masked language model where only the masked (15%) tokens are predicted. This is also in contrast to the traditional language models, where all tokens were predicted in sequential order instead of random order. This helps the model to learn bidirectional relationships and therefore better handles dependencies and relations between words. In addition, Transformer XL was used as the base architecture, which showed good performance even in the absence of permutation-based training.

XLNet was trained with over 130 GB of textual data and 512 TPU chips running for 2.5 days, both of which ar e much larger than BERT.

A brief information about RoBERTa

RoBERTa. Introduced at Facebook, Robustly optimized BERT approach RoBERTa, is a retraining of BERT with improved training methodology, 1000% more data and compute power.

To improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs. Larger batch-training sizes were also found to be more useful in the training procedure.

Importantly, RoBERTa uses 160 GB of text for pre-training, including 16GB of Books Corpus and English Wikipedia used in BERT. The additional data included CommonCrawl News dataset (63 million articles, 76 GB), Web text corpus (38 GB) and Stories from Common Crawl (31 GB). This coupled with whopping 1024 V100 Tesla GPU's running for a day, led to pre-training of RoBERTa.

Comparison of Transformer Models

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base**: 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

```
source = https://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8
```

In this table, the models are compared under 5 headings, let's take them all one by one.

1. When we look at the sizes of the models, BERT, RoBERTa and XLNet have the same values, while the size of the DistillBERT is smaller.
2. The biggest factor that determines Training Times is the size of the models and the data they have. As you can imagine, the time increases as the size increases :).
3. When we look at the performance, BERT considers the model as the base model. RoBERTa offers 2-20% better performance than BERT. A similar performance applies to XLNet. XLNet performs 2-15% better than BERT model. DisltiBERT, despite its small size, is not equally poor in performance. It performs only 3% worse.
4. When we look at its data, the model with the largest corpus is ROBERTa. It is followed by XLNET, then BERT and DistilBERT have the same data. One of the reasons for the higher performance of RoBERTa and XLNet is that the datasets are so high.
5. As it is known, there are MLM and NSP tasks in the BERT model. The RoBERTa model is the trained version of the BERT model without the NSP task. DiltilBERT is a reduced number of parameters of BERT, it maintains 97% performance, but uses only half the number of parameters (paper). To enhance the training, XLNet offers permutation language modeling where all tokens are predicted but in random order.

I recommend you to read the articles for more detailed information.

Preprocess for BERT Train

```
import pandas as pd
import numpy as np
import os
import random
from pathlib import Path
import json
```

```
import torch
from tqdm.notebook import tqdm

from transformers import BertTokenizer
from torch.utils.data import TensorDataset

from transformers import BertForSequenceClassification
```

```
class Config():
    seed_val = 17
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
    epochs = 1
    batch_size = 6
    seq_length = 25
    lr = 2e-5
    eps = 1e-8
    pretrained_model = 'bert-base-uncased'
    test_size=0.15
    random_state=42
    add_special_tokens=True
    return_attention_mask=True
    pad_to_max_length=True
    do_lower_case=False
    return_tensors='pt'

config = Config()
```

```
# params will be saved after training
params = {"seed_val": config.seed_val,
    "device":str(config.device),
    "epochs":config.epochs,
    "batch_size":config.batch_size,
    "seq_length":config.seq_length,
    "lr":config.lr,
    "eps":config.eps,
    "pretrained_model": config.pretrained_model,
    "test_size":config.test_size,
    "random_state":config.random_state,
    "add_special_tokens":config.add_special_tokens,
    "return_attention_mask":config.return_attention_mask,
    "pad_to_max_length":config.pad_to_max_length,
    "do_lower_case":config.do_lower_case,
    "return_tensors":config.return_tensors,
         }
```

```
# set random seed and device
import random

device = config.device

random.seed(config.seed_val)
np.random.seed(config.seed_val)
torch.manual_seed(config.seed_val)
torch.cuda.manual_seed_all(config.seed_val)
```

```
df2.head()
```

	Text	Label	Category	cleantext	vader_neg	vader_neu	vader_pos	vader_cc
0	Communal violence in Bhainsa, Telangana. "Ston...	1	disaster	[commun, violenc, bhainsa, telangana, stone, p...	0.000	1.000	0.000	0.0(
1	Telangana: Section 144 has been imposed in Bha...	1	disaster	[telangana, section, 144, impos, bhainsa, janu...	0.205	0.795	0.000	-0.5
2	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, goqvyj...	0.000	1.000	0.000	0.0(
3	Arsonist sets cars ablaze at dealership https:...	1	disaster	[arsonist, set, car, ablaz, dealership, 0gl7nu...	0.000	1.000	0.000	0.0(
4	"Lord Jesus, your love brings freedom and pard...	0	normal	[lord, jesu, love, bring, freedom, pardon, fil...	0.000	0.421	0.579	0.9(

Train and Validation Split

```
#split train test
from sklearn.model_selection import train_test_split

train_df_, val_df = train_test_split(df2,
                                     test_size=0.10,
                                     random_state=config.random_state,
                              stratify=df2.Label.values)
```

```
train_df_.head()
```

	Text	Label	Category	cleantext	vader_neg	vader_neu	vader_pos	va
1824	Porhub's #4 most watched video this week with ...	0	normal	[porhub, watch, video, week, million, view, te...	0.000	0.775	0.225	
672	Five soldiers were trapped under the #avalanch...	1	disaster	[five, soldier, trap, avalanch, effort, trace,...	0.302	0.698	0.000	
1977	This story of Jimmy is utterly heartbreaking. ...	0	normal	[stori, jimmi, utterli, heartbreak, peopl, tal...	0.527	0.473	0.000	
7487	One thing that warlord actual at DARC and I de...	0	normal	[one, thing, warlord, actual, darc, definit, a...	0.194	0.806	0.000	
1144	the only things that aren't under that handle ...	0	normal	[thing, handl, prob, like, snapchat, blizzard,...	0.000	0.583	0.417	

```
train_df, test_df = train_test_split(train_df_,
                                     test_size=0.10,
                                     random_state=42,
                                stratify=train_df_.Label.values)
```

```
# count of unique label  control
print(len(train_df['Label'].unique()))
print(train_df.shape)
```

```
2
(8100, 17)
```

```
# count of unique label  control
print(len(val_df['label'].unique()))
print(val_df.shape)
```

```
2
(1000, 17)
```

```
print(len(test_df['label'].unique()))
print(test_df.shape)
```

```
2
(900, 17)
```

BertTokenizer and Encoding the Data

```
# create tokenizer
tokenizer = BertTokenizer.from_pretrained(config.pretrained_model,
                                          do_lower_case=config.do_lower_case)
```

```
encoded_data_train = tokenizer.batch_encode_plus(
    train_df.cleantext2.values,
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
)
encoded_data_val = tokenizer.batch_encode_plus(
    val_df.cleantext2.values,
    add_special_tokens=config.add_special_tokens,
    return_attention_mask=config.return_attention_mask,
    pad_to_max_length=config.pad_to_max_length,
    max_length=config.seq_length,
    return_tensors=config.return_tensors
)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly
```

```
input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(train_df.Label.values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(val_df.Label.values)
```

```
dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)
```

Creating the Model

- `bert-base-uncased` is a smaller pre-trained model.
- Using `num_labels` to indicate the number of output labels.

```
model = BertForSequenceClassification.from_pretrained(config.pretrained_model,
                                                      num_labels=3,
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly in
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Data Loaders

- `DataLoader` combines a dataset and a sampler, and provides an iterable over the given dataset.
- We use `RandomSampler` for training and `SequentialSampler` for validation.
- Given the limited memory in my environment, I set `batch_size=64`.

```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=config.batch_size)

dataloader_validation = DataLoader(dataset_val,
                                   sampler=SequentialSampler(dataset_val),
                                   batch_size=config.batch_size)
```

Optimizer & Scheduler

```
from transformers import AdamW, get_linear_schedule_with_warmup

optimizer = AdamW(model.parameters(),
                  lr=config.lr,
                  eps=config.eps)


scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=len(dataloader_train)*config.epochs)
```

Performance Metrics

We will use f1 score as performance metrics.

```
from sklearn.metrics import f1_score

def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')

def accuracy_per_class(preds, labels, label_dict):
    label_dict_inverse = {v: k for k, v in label_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'Class: {label_dict_inverse[label]}')
        print(f'Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}₩n')
```

Training Loop

```
def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
```

```
config.device
```

```
device(type='cuda', index=0)
```

```
model.to(config.device)

for epoch in tqdm(range(1, config.epochs+1)):

    model.train()

    loss_train_total = 0
    # allows you to see the progress of the training
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)

    for batch in progress_bar:

        model.zero_grad()

        batch = tuple(b.to(config.device) for b in batch)
```