Natural Language Processing with Disaster Tweets

1. Description of the project

Twitter(X)'s Authenticity Crisis as a Communication Tool or Media:

- Role of Twitter in Emergencies: Twitter has emerged as a vital real-time communication platform during crises due to widespread smartphone use.
- Challenges in Interpretation: Distinguishing between literal and metaphorical language (e.g., the use of "IT WAS ABLAZE" metaphorically) is straightforward for humans but challenging for machines.
- Devise a machine learning model to classify tweets as disaster-related or not, using a dataset of 10,000 hand-classified tweets.

Details:

- Dataset: Comprises 10,000 tweets, providing a basis for training and testing the model.
- Objective: Improve real-time disaster response by accurately identifying genuine emergency communications on Twitter (X).

2. EDA (exploratory data analysis)

*Required libraries

In [103]:

```
!pip install transformers -q
!pip install wordcloud -q
!pip install transformers datasets torch -q
!pip install torch torchvision torchaudio -q
```

```
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
```

In [104]:

```
import os
import gc
gc.enable()
import time
import warnings
warnings.filterwarnings("ignore")
import re
import string
import operator
import urllib.request
import zipfile
```

```
import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 100)

import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict

from transformers import BertForSequenceClassification
from transformers import BertModel
from transformers import AutoTokenizer, BertTokenizer, BertForSequenceClassification, DataCollatorWithPadding

import torch
from torch.optim import AdamW
import torch.nn as nn
from torch.nn import CrossEntropyLoss
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torch.cuda.amp import autocast, GradScaler
from torch.optim.lr_scheduler import ReduceLROnPlateau

from datasets import load_dataset
from wordcloud import STOPWORDS

from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit, GroupKFold, train_test_split, GroupShuffleSplit
from sklearn.metrics import precision_score, recall_score, f1_score,

accuracy_score
from sklearn.utils.class_weight import compute_class_weight
import torch.nn.functional as F
from datasets import load_dataset

import tensorflow as tf
import tensorflow_hub as hub

from tensorflow import keras
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.layers import Dense, Input, Dropout, GlobalAveragePooling1D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback
```

*Reproductivity

In [105]:

```
def set_seed(seed):
    import random
    SEED=seed
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

set_seed(42)
```

In [106]:

```
# Check and set device (using GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cpu
```

*Read CSV

In [107]:

```
df_trn = pd.read_csv('/kaggle/input/nlp-getting-started/train.csv')
df_test = pd.read_csv('/kaggle/input/nlp-getting-started/test.csv')
```

*Remove Data Duplication

In [108]:

```
df_train = df_trn.drop_duplicates(subset='text', keep='first')

# Check
print(f"Original dataset size: {len(df_trn)}")
print(f"After removing duplication: {len(df_train)}")
```

```
Original dataset size: 7613
After removing duplication: 7503
```

*Check Mislabeled Data

In [109]:

```python
df_invalid_target = df_train[~df_train['target'].isin([0, 1])]
print("Mislabeled Data:  ", len(df_invalid_target))
```

```
Mislabeled Data:   0
```

Dataset Size & Structure

In [110]:

```python
print('Training Set Shape = {}'.format(df_train.shape))
print('Training Set Memory Usage = {:.2f} MB'.format(df_train.memory
_usage().sum() / 1024**2))
print('Test Set Shape = {}'.format(df_test.shape))
print('Test Set Memory Usage = {:.2f} MB'.format(df_test.memory_usag
e().sum() / 1024**2))
```

```
Training Set Shape = (7503, 5)
Training Set Memory Usage = 0.34 MB
Test Set Shape = (3263, 4)
Test Set Memory Usage = 0.10 MB
```

In [111]:

```python
print("Training Set Structure >>>>> \n")
print(df_train.info(), "\n")
df_train.head()
```

```
Training Set Structure >>>>>

<class 'pandas.core.frame.DataFrame'>
Index: 7503 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        7503 non-null   int64 
 1   keyword   7447 non-null   object
 2   location  5021 non-null   object
 3   text      7503 non-null   object
 4   target    7503 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 351.7+ KB
None
```

Out[111]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1

3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

In [112]:

```
print("Test Set Structure >>>>> \n")
print(df_test.info(), "\n")
df_test.head()
```

```
Test Set Structure >>>>>

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        3263 non-null   int64 
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
None
```

Out[112]:

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

'Keywords' and 'Location' - missing value

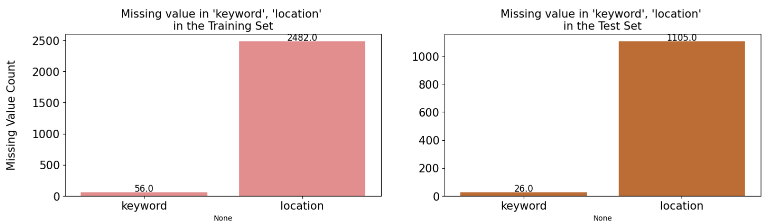
- **0.01%** of `keyword` is missing in both training and test set
- **33%** of `location` is missing in both training and test set

Missing value in `keyword` and `location` are found in the same ratio in the training and the test datasets. Missing values in those features are filled with `no_keyword` and `no_location` respectively.

In [113]:

```
# Missing value in 'keyword', 'location'
missing_cols = ['keyword', 'location']

fig, axes = plt.subplots(ncols=2, figsize=(17, 4), dpi=100)

# Training Set
sns.barplot(x=df_train[missing_cols].isnull().sum().index,
            y=df_train[missing_cols].isnull().sum().values,
            ax=axes[0],
            palette=['lightcoral'])

# 각 막대 위에 값 추가 (Training Set)
for p in axes[0].patches:
    axes[0].annotate(f'{p.get_height()}',
                     (p.get_x() + p.get_width() / 2., p.get_heigh
t()),
                     ha='center', va='center',
                     fontsize=12, color='black',
                     xytext=(0, 5), textcoords='offset points')

# Test Set
sns.barplot(x=df_test[missing_cols].isnull().sum().index,
            y=df_test[missing_cols].isnull().sum().values,
            ax=axes[1],
            palette=['chocolate'])
```

```
                palette=['chocolate'])

# 각 막대 위에 값 추가 (Test Set)
for p in axes[1].patches:
    axes[1].annotate(f'{p.get_height()}',
                     (p.get_x() + p.get_width() / 2., p.get_heigh
t()),
                     ha='center', va='center',
                     fontsize=12, color='black',
                     xytext=(0, 5), textcoords='offset points')

# 설정
axes[0].set_ylabel('Missing Value Count', size=15, labelpad=20)
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title("Missing value in 'keyword', 'location' \nin the T
raining Set", fontsize=15)
axes[1].set_title("Missing value in 'keyword', 'location' \nin the T
est Set", fontsize=15)

plt.show()
```



In [114]:

```
# Flling the missing value with no_
for df in [df_train, df_test]:
    for col in ['keyword', 'location']:
        df[col] = df[col].fillna(f'no {col}')
```

In [49]:

```
df_train.head()
```

Out[49]:

	id	keyword	location	text	target
0	1	no keyword	no location	Our Deeds are the Reason of this #earthquake M...	1
1	4	no keyword	no location	Forest fire near La Ronge Sask. Canada	1
2	5	no keyword	no location	All residents asked to 'shelter in place' are ...	1
3	6	no keyword	no location	13,000 people receive #wildfires evacuation or...	1
4	7	no keyword	no location	Just got sent this photo from Ruby #Alaska as ...	1

In [115]:

```
# Simple Stats
df_train.describe()
```

Out[115]:

	id	target
count	7503.000000	7503.000000
mean	5439.831401	0.426230
std	3141.748725	0.494561
min	1.000000	0.000000
25%	2726.500000	0.000000
50%	5408.000000	0.000000
75%	8149.500000	1.000000
max	10873.000000	1.000000

'Keywords' and 'Location' - unique values

- ### Locations are not automatically generated, and just user inputs. As they have too many unique values, it's not proper to use as a feature.
- ### However, as `keywords` are generally used in the consistent context, we can consider `keyword`

as a feature for distinguish these dialogues are happening under the urgent situation or not.

- ### We can use target encoding on `keyword`. But at the same time, even though target encoding is a promising way to improve model performance, but it requires proper measures to prevent overfitting. This can improve the generalization ability of the model.

In [116]:

```python
print(f'Number of unique values in keyword >> {df_train["keyword"].nunique()} (training), {df_test["keyword"].nunique()} (test)')
print(f'Number of unique values in location >> {df_train["location"].nunique()} (training), {df_test["location"].nunique()} (test)')

# Extract all unique keywords from the training set
train_keywords = set(df_train['keyword'].unique())

# Extract all unique keywords from the test set
test_keywords = set(df_test['keyword'].unique())

# Check if all keywords from the training set are included in the test set
all_train_keywords_in_test = train_keywords.issubset(test_keywords)

print(f"Q. Are all training set keywords included in the test set? >>>  {all_train_keywords_in_test}")

# If there are any keywords not included, print those keywords
if not all_train_keywords_in_test:
    missing_keywords = train_keywords - test_keywords
    print("Keywords present in the training set but not in the test set:", missing_keywords)
else:
    print("> All keywords are present in the test set.")
```

```
Number of unique values in keyword >> 222 (training), 222 (test)
Number of unique values in location >> 3328 (training), 1603 (test)
Q. Are all training set keywords included in the test set? >>>  True
> All keywords are present in the test set.
```

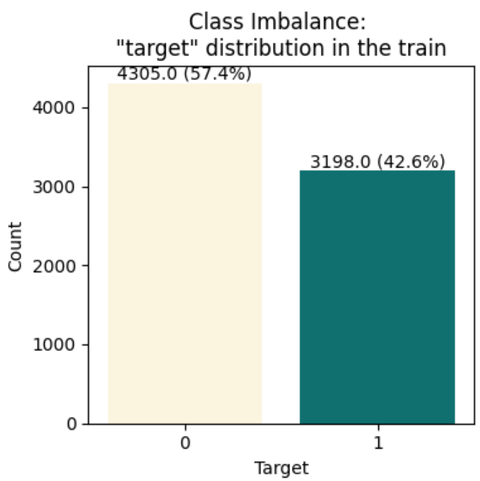
Exploring class imbalance

- In the dataset, the ratio of disaster (1) and non-disaster (0) is 43:57, which means there is a class imbalance.
- This increases the possibility that the disaster class (minority class) will not be sufficiently learned during model learning, resulting in a decrease in performance (especially F1-Score, Recall).
- To solve this, we plan to apply a method to handle class imbalance. (Applying class weights or Focal Loss)

In [117]:

```python
# explore class imbalance
target_counts = df_train['target'].value_counts()
total_counts = target_counts.sum()

target_percentages = (target_counts / total_counts) * 100
```

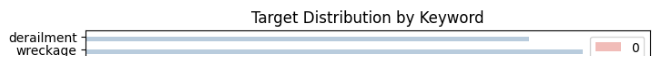
```
plt.figure(figsize=(4, 4))

colors = ['#FFF8DC', '#008080']
# countplot 생성
ax = sns.countplot(x='target', data=df_train, palette=colors)

# 비율 라벨 추가
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height} ({target_percentages[p.get_x() + p.get_wi
dth() / 2]:.1f}%)',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom')

plt.title('Class Imbalance: \n"target" distribution in the train')
plt.xlabel('Target')
plt.ylabel('Count')

plt.tight_layout()
plt.show()
```

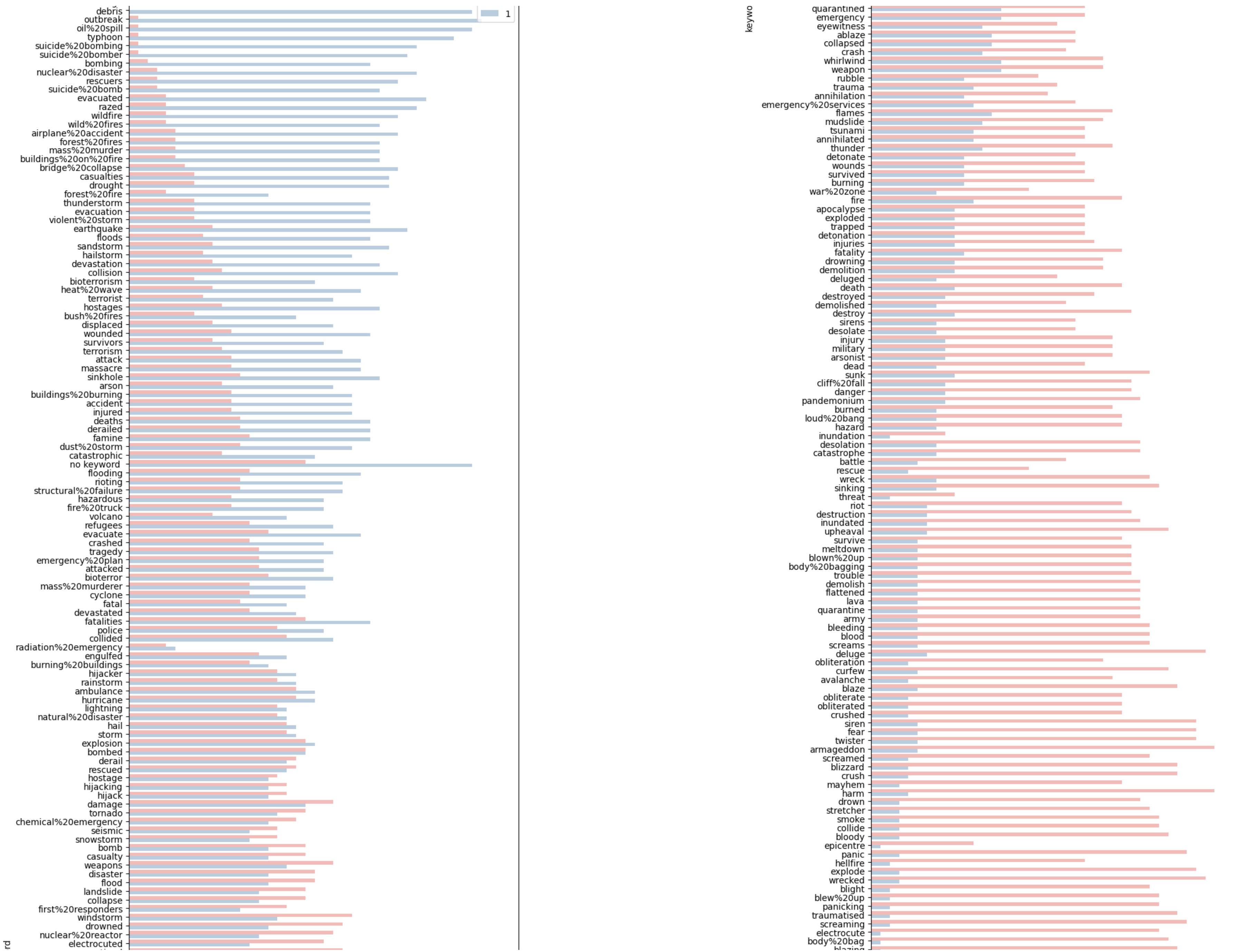


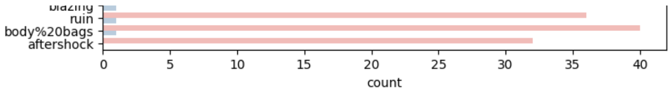
Target Distribution by Keyword

In [118]:

```
# Plot Target Distribution by Keyword
df_train['keyword_target_mean'] = df_train.groupby('keyword')['targe
t'].transform('mean')
fig = plt.figure(figsize=(8, 40), dpi=100)

sns.countplot(y=df_train.sort_values(by='keyword_target_mean', ascen
ding=False)['keyword'],
              hue=df_train.sort_values(by='keyword_target_mean', asc
ending=False)['target'],
              palette='Pastel1')

plt.tick_params(axis='x', labelsize=10)
plt.tick_params(axis='y', labelsize=10)
plt.legend(loc=1)
plt.title('Target Distribution by Keyword')
plt.show()

# Delete 'keyword_target_mean' column
df_train.drop(columns=['keyword_target_mean'], inplace=True)
```




1
debris
outbreak
oil%20spill
typhoon
suicide%20bombing
suicide%20bomber
bombing
nuclear%20disaster
rescuers
suicide%20bomb
evacuated
razed
wildfire
wild%20fires
airplane%20accident
forest%20fires
mass%20murder
buildings%20on%20fire
bridge%20collapse
casualties
drought
forest%20fire
thunderstorm
evacuation
violent%20storm
earthquake
floods
sandstorm
hailstorm
devastation
collision
bioterrorism
heat%20wave
terrorist
hostages
bush%20fires
displaced
wounded
survivors
terrorism
attack
massacre
sinkhole
arson
buildings%20burning
accident
injured
deaths
derailed
famine
dust%20storm
catastrophic
no keyword
flooding
rioting
structural%20failure
hazardous
fire%20truck
volcano
refugees
evacuate
crashed
tragedy
emergency%20plan
attacked
bioterror
mass%20murderer
cyclone
fatal
devastated
fatalities
police
collided
radiation%20emergency
engulfed
burning%20buildings
hijacker
rainstorm
ambulance
hurricane
lightning
natural%20disaster
hail
storm
explosion
bombed
derail
rescued
hostage
hijacking
hijack
damage
tornado
chemical%20emergency
seismic
snowstorm
bomb
casualty
weapons
disaster
flood
landslide
collapse
first%20responders
windstorm
drowned
nuclear%20reactor
electrocuted
rd
keywo
quarantined
emergency
eyewitness
ablaze
collapsed
crash
whirlwind
weapon
rubble
trauma
annihilation
emergency%20services
flames
mudslide
tsunami
annihilated
thunder
detonate
wounds
survived
burning
war%20zone
fire
apocalypse
exploded
trapped
detonation
injuries
fatality
drowning
demolition
deluged
death
destroyed
demolished
destroy
sirens
desolate
injury
military
arsonist
dead
sunk
cliff%20fall
danger
pandemonium
burned
loud%20bang
hazard
inundation
desolation
catastrophe
battle
rescue
wreck
sinking
threat
riot
destruction
inundated
upheaval
survive
meltdown
blown%20up
body%20bagging
trouble
demolish
flattened
lava
quarantine
army
bleeding
blood
screams
deluge
obliteration
curfew
avalanche
blaze
obliterate
obliterated
crushed
siren
fear
twister
armageddon
screamed
blizzard
crush
mayhem
harm
drown
stretcher
smoke
collide
bloody
epicentre
panic
hellfire
explode
wrecked
blight
blew%20up
panicking
traumatised
screaming
electrocute
body%20bag



In [120]:

```python
# Statistics on Text
# word_count
df_train['word_count'] = df_train['text'].apply(lambda x: len(str(x).split()))
df_test['word_count'] = df_test['text'].apply(lambda x: len(str(x).split()))

# unique_word_count
df_train['unique_word_count'] = df_train['text'].apply(lambda x: len(set(str(x).split())))
df_test['unique_word_count'] = df_test['text'].apply(lambda x: len(set(str(x).split())))

# stop_word_count
df_train['stop_word_count'] = df_train['text'].apply(
    lambda x: len([w for w in str(x).lower().split() if w in STOPWORDS]))
df_test['stop_word_count'] = df_test['text'].apply(
    lambda x: len([w for w in str(x).lower().split() if w in STOPWORDS]))

# mean_word_length
df_train['mean_word_length'] = df_train['text'].apply(
    lambda x: np.mean([len(w) for w in str(x).split()]))
df_test['mean_word_length'] = df_test['text'].apply(
    lambda x: np.mean([len(w) for w in str(x).split()]))

# char_count
df_train['char_count'] = df_train['text'].apply(lambda x: len(str(x)))
df_test['char_count'] = df_test['text'].apply(lambda x: len(str(x)))

# punctuation_count
df_train['punctuation_count'] = df_train['text'].apply(
    lambda x: len([c for c in str(x) if c in string.punctuation]))
df_test['punctuation_count'] = df_test['text'].apply(
    lambda x: len([c for c in str(x) if c in string.punctuation]))
```

Target Distributions by Statistics on Text

Distributions of meta features in classes and datasets can be helpful to identify disaster tweets. At the first look, disaster tweets are written in a more formal way with longer words compared to non-disaster tweets because most of them are coming from news agencies. Non-disaster tweets have more typos than disaster tweets because they are coming from individual users. The followings are meta features that are expected to be useful:

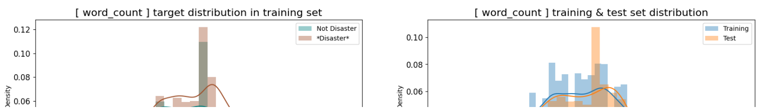
- `word_count` number of words in text
- `unique_word_count` number of unique words in text
- `stop_word_count` number of stop words in text
- `mean_word_length` average character count in words
- `char_count` number of characters in text
- `punctuation_count` number of punctuations in text

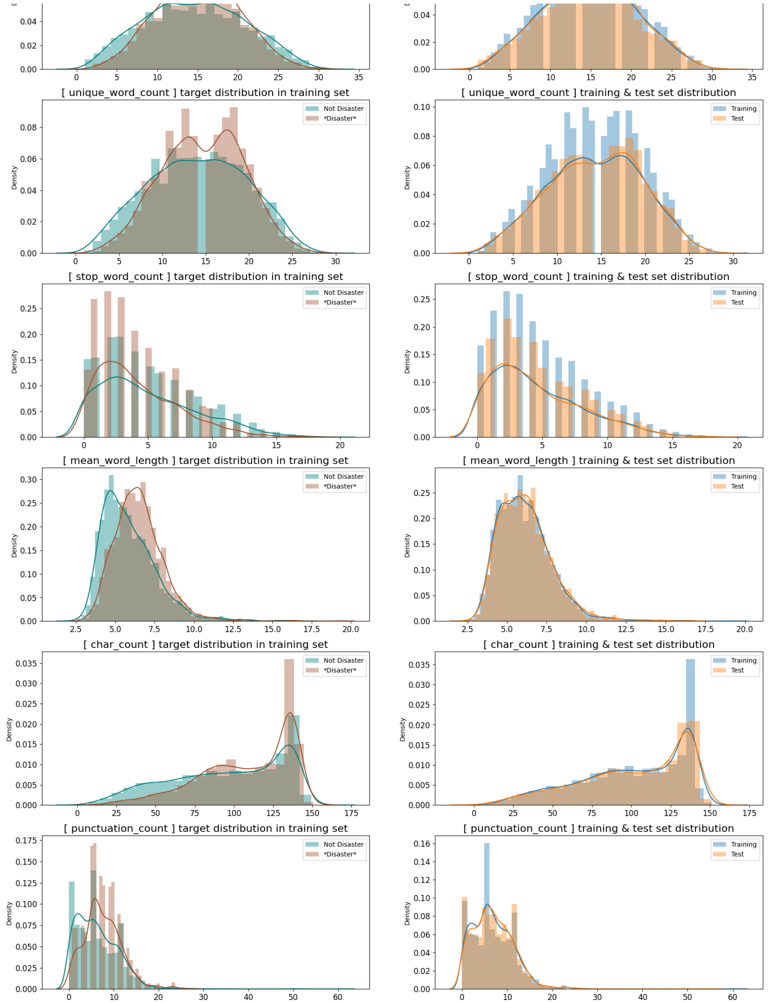
*Distribution of text

Visualization for Statistics Feature

In [121]:

```python
# Plot Statistics Fearue
STATS_FEATURE = ['word_count', 'unique_word_count', 'stop_word_coun
t', 'mean_word_length',
                'char_count', 'punctuation_count']

DISASTER_TWEETS = df_train['target'] == 1

fig, axes = plt.subplots(ncols=2, nrows=len(STATS_FEATURE), figsize=
(20, 30), dpi=100)

for i, feature in enumerate(STATS_FEATURE):
    sns.distplot(df_train.loc[~DISASTER_TWEETS][feature], label = 'N
ot Disaster',
                 ax=axes[i][0], color='teal')
    sns.distplot(df_train.loc[DISASTER_TWEETS][feature], label = '*D
isaster*',
                 ax=axes[i][0], color='sienna')

    sns.distplot(df_train[feature], label='Training', ax=axes[i][1])
    sns.distplot(df_test[feature], label='Test', ax=axes[i][1])

    for j in range(2):
        axes[i][j].set_xlabel('')
        axes[i][j].tick_params(axis='x', labelsize=12)
        axes[i][j].tick_params(axis='y', labelsize=12)
        axes[i][j].legend()

    axes[i][0].set_title(f'[ {feature} ] target distribution in trai
ning set',fontsize=16)
    axes[i][1].set_title(f'[ {feature} ] training & test set distrib
ution',fontsize=16)

plt.show()
```





'N-grams' by Target : unigram, bigram, trigram

Incorporating N-grams by target helps to better understand the context in which words appear and their relationship to the target variable, leading to more accurate models that can recognize nuanced patterns in the data. It allows the model to capture richer information, improving classification of target. Take a look at:

In [122]:

```
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(' ')
             if token != '' if token not in STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [' '.join(ngram) for ngram in ngrams]

N = 30

# Unigrams
disaster_unigrams = defaultdict(int)
nondisaster_unigrams = defaultdict(int)

for tweet in df_train[DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet):
        disaster_unigrams[word] += 1

for tweet in df_train[~DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet):
        nondisaster_unigrams[word] += 1

df_disaster_unigrams = pd.DataFrame(sorted(disaster_unigrams.item
s(),
                                           key=lambda x: x[1])[::-
1])
df_nondisaster_unigrams = pd.DataFrame(sorted(nondisaster_unigrams.i
tems(),
                                              key=lambda x: x[1])[::
-1])

# Bigrams
disaster_bigrams = defaultdict(int)

nondisaster_bigrams = defaultdict(int)

for tweet in df_train[DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet, n_gram=2):
        disaster_bigrams[word] += 1

for tweet in df_train[~DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet, n_gram=2):
        nondisaster_bigrams[word] += 1

df_disaster_bigrams = pd.DataFrame(sorted(disaster_bigrams.items(),
                                          key=lambda x: x[1])[::-1])
df_nondisaster_bigrams = pd.DataFrame(sorted(nondisaster_bigrams.ite
ms(),
                                             key=lambda x: x[1])[::-
1])

# Trigrams
disaster_trigrams = defaultdict(int)
nondisaster_trigrams = defaultdict(int)

for tweet in df_train[DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet, n_gram=3):
        disaster_trigrams[word] += 1

for tweet in df_train[~DISASTER_TWEETS]['text']:
    for word in generate_ngrams(tweet, n_gram=3):
        nondisaster_trigrams[word] += 1

df_disaster_trigrams = pd.DataFrame(sorted(disaster_trigrams.item
s(),
                                           key=lambda x: x[1])[::-
1])
df_nondisaster_trigrams = pd.DataFrame(sorted(nondisaster_trigrams.i
tems(),
                                              key=lambda x: x[1])[::
-1])
```

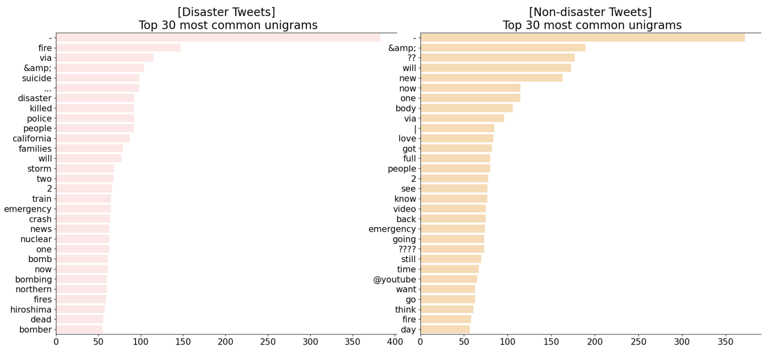
Unigrams

- The most common unigrams in both classes are $'-'$ and punctuations, stop words, or numbers are found in most common unigram, which don't provide much information about the target. Therefore, it's better to clean them before modeling.
- The most common unigrams in disaster tweets are $'fire'$ that convey information about disasters, making it challenging to use these words in other contexts without exceptional cases.
- In contrast, the most common unigrams in non-disaster tweets are $'will','new','now'$ that underline the essential part of characteristics of the twitter(X) as a media.

In [123]:

```
fig, axes = plt.subplots(ncols=2, figsize=(18, 8), dpi=100)
plt.tight_layout()
sns.barplot(y=df_disaster_unigrams[0].values[:N], x=df_disaster_unig
```

```
rams[1].values[:N],
            ax=axes[0], color='mistyrose')
sns.barplot(y=df_nondisaster_unigrams[0].values[:N], x=df_nondisaste
r_unigrams[1].values[:N],
            ax=axes[1], color='navajowhite')

for i in range(2):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=15)
    axes[i].tick_params(axis='y', labelsize=15)

axes[0].set_title(f'[Disaster Tweets]\n Top {N} most common unigram
s', fontsize=20)
axes[1].set_title(f'[Non-disaster Tweets]\n Top {N} most common unig
rams', fontsize=20)

plt.show()
```

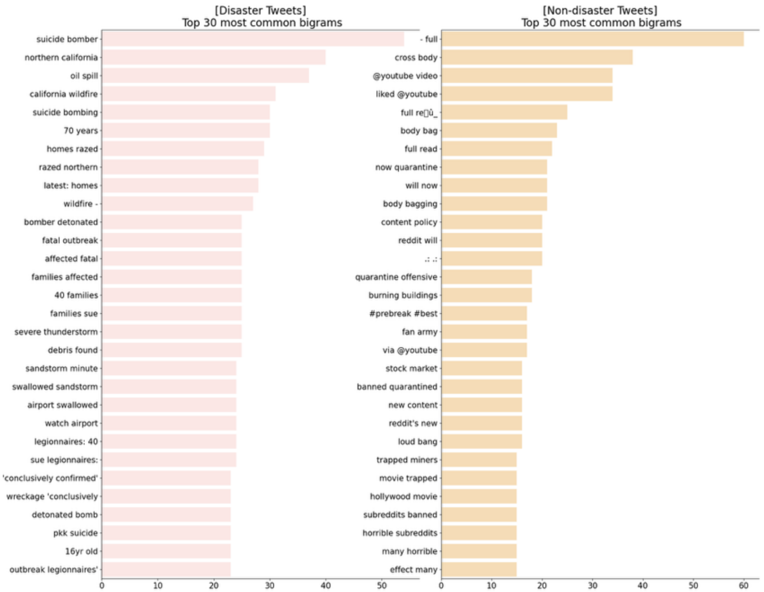


Bigram

- Due to the clearer context in the Bigrams, there are no common bigrams that exist in both classes.
- The most common bigrams in disaster tweets is 'suicide bomber' provide more information about disasters than unigrams.
- In contrast, the most common bigrams in non-disaster tweets seems to be related to popular contents on YouTube or Reddit and contain a significant number of special character like '@, ? [,], ...' , which should also be cleaned out of the text.

In [124]:

```
fig, axes = plt.subplots(ncols=2, figsize=(18, 15), dpi=100)
plt.tight_layout()
sns.barplot(y=df_disaster_bigrams[0].values[:N], x=df_disaster_bigra
ms[1].values[:N],
            ax=axes[0], color='mistyrose')
sns.barplot(y=df_nondisaster_bigrams[0].values[:N], x=df_nondisaster
_bigrams[1].values[:N],
            ax=axes[1], color='navajowhite')

for i in range(2):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=15)
    axes[i].tick_params(axis='y', labelsize=15)

axes[0].set_title(f'[Disaster Tweets]\n Top {N} most common bigram
s', fontsize=20)
axes[1].set_title(f'[Non-disaster Tweets]\n Top {N} most common bigr
ams', fontsize=20)

plt.show()
```

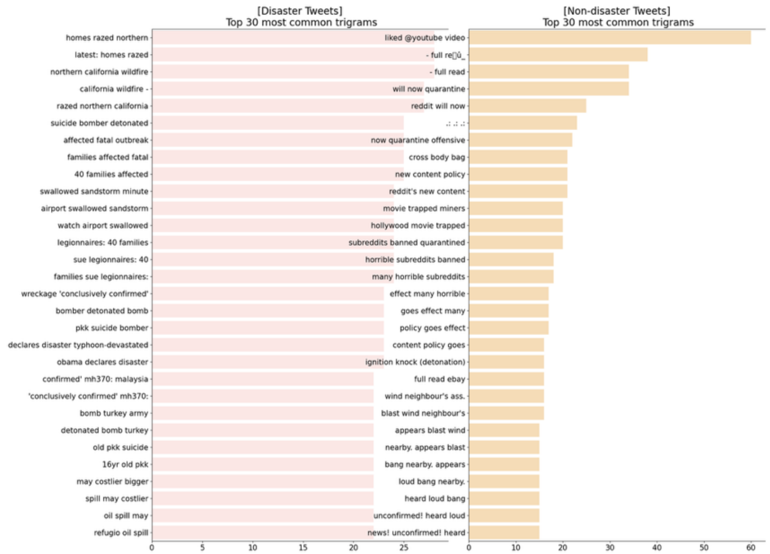


Trigram

- The most common trigrams in disaster tweets, such as "suicide bomber detonated", are remarkably similar to bigrams and provide substantial information about disasters. However, they may not offer significant additional insights when combined with bigrams.
- Similarly, the most common trigrams in non-disaster tweets are also closely related to bigrams, often containing information about YouTube and special characters, which may not provide distinct value in addition to what is already captured by bigrams.

In [125]:

```
fig, axes = plt.subplots(ncols=2, figsize=(18, 15), dpi=100)
plt.tight_layout()
sns.barplot(y=df_disaster_trigrams[0].values[:N], x=df_disaster_trig
rams[1].values[:N],
            ax=axes[0], color='mistyrose')
sns.barplot(y=df_nondisaster_trigrams[0].values[:N], x=df_nondisaste
r_bigrams[1].values[:N],
            ax=axes[1], color='navajowhite')

for i in range(2):
    axes[i].spines['right'].set_visible(False)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')
    axes[i].tick_params(axis='x', labelsize=15)
    axes[i].tick_params(axis='y', labelsize=15)

axes[0].set_title(f'[Disaster Tweets]\n Top {N} most common trigram
s', fontsize=20)
axes[1].set_title(f'[Non-disaster Tweets]\n Top {N} most common trig
rams', fontsize=20)

plt.show()
```



In [126]:

```
len(df_disaster_trigrams)
```

Out[126]:

```
23202
```

Insight after inspecting N-gram

Reviewed traditional N-grams, and the result was bigrams and trigrams can capture context and distinguish the target and expected to impove predictive performance.

That's why selected Transformer models, BERT that can capture context much more effectively than traditional N-grams. In fact, one of the key advantages of Transformer models is their ability to learn contextual relationships between words (or tokens) across the entire input sequence, unlike N-grams which only consider local relationships between adjacent words.

Comparing N-grams and Transformers:

Aspect	N-grams	Transformer models
Context Capture	Local context only (adjacent words)	Global context (any word in the sequence)
Handling Long-range Dependencies	Limited (fixed window size)	Excellent (self-attention mechanism)
Contextual Meaning	Can't differentiate ambiguous meanings	Can capture different meanings based on context
Training Complexity	Simple (but can explode in size with higher N)	More complex (requires large datasets and computational power)
Handling Syntax/Structure	Limited ability to handle complex syntax	Excellent at capturing syntactic and semantic structures
Use Case	Simple tasks with clear local patterns (e.g., topic modeling)	Complex NLP tasks (e.g., sentiment analysis, translation, summarization)

In [127]:

```
print(df_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7503 entries, 0 to 7612
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   id                  7503 non-null   int64  
 1   keyword             7503 non-null   object 
 2   location            7503 non-null   object 
 3   text                7503 non-null   object 
 4   target              7503 non-null   int64  
 5   word_count          7503 non-null   int64  
 6   unique_word_count   7503 non-null   int64  
 7   stop_word_count     7503 non-null   int64  
 8   mean_word_length    7503 non-null   float64
 9   char_count          7503 non-null   int64  
 10  punctuation_count   7503 non-null   int64  
dtypes: float64(1), int64(7), object(3)
memory usage: 703.4+ KB
None
```

3. Text Data Cleaning

The below libraries are useful for preprocessing text, especially for NLP tasks involving noisy or social media data.

- tweet-preprocessor – A library for preprocessing and cleaning tweets, such as removing mentions, URLs, hashtags, emojis, and reserved words.
- ekphrasis – A text processing tool designed for social media data. It includes tokenization, spell correction, word segmentation, and more.
- clean-text – A library for cleaning and normalizing text by removing unwanted characters, fixing accents, and handling Unicode issues. contractions – Expands English contractions (e.g., "don't" → "do not") to improve text clarity.
- emoji – A library for working with emojis, including conversion between text and emoji representations.
- unidecode – Converts Unicode text to ASCII, making non-English characters more readable in a simplified format.

In [128]:

```
!pip install tweet-preprocessor -q
!pip install ekphrasis -q
!pip install clean-text -q
!pip install contractions -q
!pip install emoji -q
!pip install unidecode -q
print("all multi pip done")
```

```
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i

s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
```

```
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
all multi pip done
```

In [129]:

```
# remove 'location', rename 'text'colum, and add new 'text'colum for d
ata cleaning
df_train.rename(columns={'text': 'b4embedding_text'}, inplace=True)
df_test.rename(columns={'text': 'b4embedding_text'}, inplace=True)

df_train.insert(2, 'text', df_train['b4embedding_text'])
df_test.insert(2, 'text', df_test['b4embedding_text'])

del df_train['location']
del df_test['location']
print(df_train.info(), df_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7503 entries, 0 to 7612
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 7503 non-null   int64
 1   keyword            7503 non-null   object
 2   text               7503 non-null   object
 3   b4embedding_text   7503 non-null   object
 4   target             7503 non-null   int64
 5   word_count         7503 non-null   int64
 6   unique_word_count  7503 non-null   int64
 7   stop_word_count    7503 non-null   int64
 8   mean_word_length   7503 non-null   float64
 9   char_count         7503 non-null   int64
 10  punctuation_count  7503 non-null   int64
dtypes: float64(1), int64(7), object(3)
memory usage: 703.4+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 3263 non-null   int64
 1   keyword            3263 non-null   object
 2   text               3263 non-null   object
 3   b4embedding_text   3263 non-null   object
 4   word_count         3263 non-null   int64
 5   unique_word_count  3263 non-null   int64
 6   stop_word_count    3263 non-null   int64
 7   mean_word_length   3263 non-null   float64
 8   char_count         3263 non-null   int64
 9   punctuation_count  3263 non-null   int64
dtypes: float64(1), int64(6), object(3)
memory usage: 255.0+ KB
None None
```

In [130]:

```
df_train['b4embedding_text'][888:898]
```

Out[130]:

```
895             Bloody insomnia again! Grrrr!! #Insomnia
896    @zhenghxn i tried 11 eyes akame ga kill and to...
897    @Fantosex Now suck it up because that's all yo...
898    You call them weekends. I call them Bloody Mar...
899    Bloody hell what a day. I haven't even really ...
900                                      Damn bloody hot
901                @MrTophyPup it's bloody sexy *drools*
902    You know how they say the side effects low &am...
903    Ronda Rousey would be 'close' to making Floyd ...
904    I'm awful at painting.. why did I agree to do ...
Name: b4embedding_text, dtype: object
```

In [131]:

```
import preprocessor as p
from ekphrasis.classes.preprocessor import TextPreProcessor
from ekphrasis.classes.tokenizer import SocialTokenizer
from cleantext import clean
import contractions
import emoji
```

```python
# tweet-preprocessor 설정
p.set_options(p.OPT.MENTION)  # URL은 ekphrasis에서 처리하므로 p.clean에서 처리하지 않음

# ekphrasis 설정
text_processor = TextPreProcessor(
    normalize=['url', 'email', 'user'],
    annotate={'hashtag'},
    unpack_hashtags=True,
    unpack_contractions=True,
    tokenizer=SocialTokenizer(lowercase=True).tokenize,
)

def preprocess_tweet(text):
    try:
        # 1. tweet-preprocessor로 멘션 제거 (URL은 ekphrasis에서 처리)
        text = p.clean(text)


        # 2. ekphrasis로 해시태그 분리 및 슬랭 처리
        text = " ".join(text_processor.pre_process_doc(text))

        # 3. 줄임말 확장
        text = contractions.fix(text)

        # 4.1 이모지 변환
        # text = emoji.demojize(text, delimiters=(": ", ":"))

        # 4.2 이모지 삭제
        text = emoji.replace_emoji(text, replace='')

        # 5. clean-text로 추가 정리
        text = clean(text, lower=True, no_punct=True, replace_with_url="<URL>")

    except Exception as e:
        print(f"Error processing tweet: {text}\n{e}")
        return text  # 오류가 발생하면 원본 텍스트를 반환

    return text  # 전처리된 텍스트 반환

# 데이터프레임에 적용
df_train["text"] = df_train["b4embedding_text"].apply(preprocess_tweet)
df_test["text"] = df_test["b4embedding_text"].apply(preprocess_tweet)

# URL과 해시태그 제거
url_pattern = r'http[s]?://\S+|www\.\S+|t\.co/\S+|#\w+'
df_train["text"] = df_train["text"].str.replace(url_pattern, '', regex=True)
df_test["text"] = df_test["text"].str.replace(url_pattern, '', regex=True)

# 모든 문자열에서 쉼표 제거
df = df.applymap(lambda x: x.replace(',', '') if isinstance(x, str) else x)

# 삭제할 단어들
words_to_remove = ['<hashtag>', '<mention>', '<link>', '#', '<url>', '<', '>', 'c 130' ]


# 정규 표현식 패턴 생성
pattern = '|'.join(words_to_remove)

# 단어 삭제
df_train["text"] = df_train["text"].str.replace(pattern, '', regex=True)
df_test["text"] = df_test["text"].str.replace(pattern, '', regex=True)

# 모든 문자열에서 쉼표 제거
df = df.applymap(lambda x: x.replace(',', '') if isinstance(x, str) else x)

# 결과 확인
print(df_train)
print(df_test)
```

```
Reading english - 1grams ...
Reading english - 2grams ...
Reading english - 1grams ...
         id      keyword
text  \
0         1  no keyword   our deeds are the reason of this  earthqua
ke  ...
1         4  no keyword               forest fire near la ronge sask
canada
2         5  no keyword   all residents asked to shelter in place ar
e be...
3         6  no keyword   13000 people receive  wildfires  evacuatio
n or...
4         7  no keyword   just got sent this photo from ruby  alaska
```

```
4       7  no keyword   just got sent this photo from ruby  alaska
as...
...     ...          ...
...
7604  10863  no keyword    world news  fallen powerlines on g link t
ram ...
7605  10864  no keyword   on the flip side i am at walmart and there
is ...
7606  10866  no keyword   suicide bomber kills 15 in saudi security
site...

7608  10869  no keyword   two giant cranes holding a bridge collapse
int...
7612  10873  no keyword   the latest more homes razed by northern ca
lifo...

                                       b4embedding_text  target  wor
d_count  unique_word_count  \
0     Our Deeds are the Reason of this #earthquake M...       1
13                13
1                 Forest fire near La Ronge Sask. Canada       1
7                 7
2     All residents asked to 'shelter in place' are ...       1
22                20
3     13,000 people receive #wildfires evacuation or...       1
8                 8
4     Just got sent this photo from Ruby #Alaska as ...       1
16                15
...                                                 ...     ...
...               ...
7604  #WorldNews Fallen powerlines on G:link tram: U...       1
19                19
7605  on the flip side I'm at Walmart and there is a...       1
26                25
7606  Suicide bomber kills 15 in Saudi security site...       1
20                18
7608  Two giant cranes holding a bridge collapse int...       1
11                11
7612  The Latest: More Homes Razed by Northern Calif...       1
13                13

      stop_word_count  mean_word_length  char_count  punctuation_cou
nt
0                   6          4.384615          69
1
1                   0          4.571429          38
1
2                  11          5.090909         133
3
3                   1          7.125000          65
2
4                   7          4.500000          88
2

...               ...               ...         ...
...
7604                6          6.210526         136
12
7605               17          3.423077         114
1
7606                1          5.100000         121
11
7608                2          6.636364          83
5
7612                3          6.307692          94
7

[7503 rows x 11 columns]
         id      keyword
text  \
0         0  no keyword                  just happened a terrible ca
r crash
1         2  no keyword   heard about  earthquake  is different citi
es s...
2         3  no keyword   there is a forest fire at spot pond geese
are ...
3         9  no keyword           apocalypse lighting  spokane   wil
dfires
4        11  no keyword       typhoon soudelor kills 28 in china and
taiwan
...     ...          ...
...
3258  10861  no keyword   earthquake safety los angeles 0 uo safety
fast...
3259  10865  no keyword   storm in ri worse than last hurricane my c
ity ...
3260  10868  no keyword                   green line derailment in c
hicago
3261  10874  no keyword           meg issues hazardous weather outlo
ok hwo
3262  10875  no keyword    cityof calgary  has activated its municip
al e...

                                       b4embedding_text  word_count
unique_word_count  \
0                     Just happened a terrible car crash           6
```

```
0                  Just happened a terrible car crash          6
6
1     Heard about #earthquake is different cities, s...         9
9
2     there is a forest fire at spot pond, geese are...        19
19
3              Apocalypse lighting. #Spokane #wildfires         4
4
4         Typhoon Soudelor kills 28 in China and Taiwan         8
8
...                                                 ...       ...
...
3258  EARTHQUAKE SAFETY LOS ANGELES  ÛÒ SAFETY FASTE...         8
7
3259  Storm in RI worse than last hurricane. My city...        23
22
3260  Green Line derailment in Chicago http://t.co/U...         6
6
3261  MEG issues Hazardous Weather Outlook (HWO) htt...         7
7
3262  #CityofCalgary has activated its Municipal Eme...         8
8

      stop_word_count  mean_word_length  char_count  punctuation_cou
nt
0                   2          4.833333          34
0
1                   2          6.222222          64
3
2                  10          4.105263          96
2
3                   0          9.250000          40
3
4                   2          4.750000          45
0
...               ...               ...         ...
...
3258                0          6.000000          55
0
3259                7          5.086957         139
5
3260                1          8.333333          55
5
3261                0          8.428571          65
7
3262                2          7.625000          68
3
```

```
[3263 rows x 10 columns]
```

In [132]:

```python
df_train['text'][5588:5598]
```

Out[132]:

```
5673    4 kidnapped ladies rescued by police in enugu ...
5674               rescued med migrants arrive in sicily
5675    i hand you a glass of water and sit down i was...
5676    trust us to get rescued by the dopey ones val ...
5677    summer  summer vibes   california   puppy   pi...
5678    man who buried dog alive thought no one would ...
5679    10 month old baby girl was rescued by coastgua...
5680          but now  skyrim  awaits to be rescued again
5681               rescued med migrants arrive in sicily
5682    the finnish hip hop pioneer paleface will be r...
Name: text, dtype: object
```

4. Embedding Coverage

While working on this projecet, found out checking embedding coverage in advance and performing data cleaning can be very helpful.

What is embedding coverage?

- It is a metric that measures how many words a given embedding model (e.g., Word2Vec, GloVe, FastText, BERT, etc.) includes in its dictionary from a given text data.
- If the coverage is low, the model may not be able to properly represent the words, and the OOV (Out-Of-Vocabulary) problem may occur.

Importance to check embedding coverage in advance

1. Identifying the OOV (Out-Of-Vocabulary) problem

- If there are many words that are not in the dictionary (OOV), the embedding is likely not to be properly trained.
- For example, in the sentence "create seamlessly loop noise", if 'seamlessly', and 'loop' are not in the

embedding, the meaning of the sentence may be distorted.

1. Increase data cleaning efficiency

- You can clean up unnecessary symbols, typos, special characters, and strangely formatted text (e.g., variants such as h3ll0).
- You can decide in advance how to handle numbers, URLs, special characters, etc.

1. Optimize embedding selection

- If you need an embedding specialized for a specific domain (e.g., medical, legal, social media), it may be more effective to use a domain-specific pre-trained embedding rather than a general embedding.
- For example, for medical data, a medical-specific embedding such as BioWordVec may be more appropriate than a general GloVe.

1. Improve data quality

- If you analyze and clean the parts of the data where the embedding is not applied well, the model performance is likely to improve.
- SNS data may have many spelling errors, so it is important to correct them in advance.

Comparision of FastText vs GloVe

FastText is strong against typos and new words, while GloVe is faster but cannot handle OOV words. If the data is general text, it is okay to use GloVe, but if the data like this project is SNS text, conversation data, or data with many special terms, FastText may be more advantageous. It is very important to check the coverage and handle OOV words before model training.

Considering the feaures of FastText and GloVe highlited on the below table, will optimize performance by using the GloVe embedding and then apply FastText embedding.

Feature	FastText	GloVe
OOV Handling	Can generate vectors using subwords	No vectors for OOV words
Speed	Relatively slower	Faster
Spelling Error Handling	Strong	Weak
New Words & Domain-Specific Terms	Can generate vectors	Limited to pre-trained vocabulary

In [133]:

```
# 저장할 디렉토리 설정
glove_dir = "glove"
os.makedirs(glove_dir, exist_ok=True)
```

```
# GloVe 파일 다운로드 URL
glove_url = "https://nlp.stanford.edu/data/glove.6B.zip"
glove_zip_path = os.path.join(glove_dir, "glove.6B.zip")

# 파일 다운로드
if not os.path.exists(glove_zip_path):
    print("Downloading GloVe embeddings...")
    urllib.request.urlretrieve(glove_url, glove_zip_path)
    print("Download complete!")
else:
    print("GloVe zip file already exists.")

# 압축 해제
print("Extracting files...")
with zipfile.ZipFile(glove_zip_path, "r") as zip_ref:
    zip_ref.extractall(glove_dir)
print("Extraction complete!")

# 다운로드된 파일 리스트 출력
glove_files = os.listdir(glove_dir)
print("Available GloVe files:", glove_files)
```

```
GloVe zip file already exists.
Extracting files...
Extraction complete!
Available GloVe files: ['glove.6B.50d.txt', 'glove.6B.300d.txt', 'gl
ove.6B.200d.txt', 'glove.6B.100d.txt', 'glove.6B.zip']
```

In [134]:

```
!pip install fasttext
```

```
Requirement already satisfied: fasttext in /usr/local/lib/python3.1
0/site-packages (0.9.3)
Requirement already satisfied: pybind11>=2.2 in /usr/local/lib/pytho
n3.10/site-packages (from fasttext) (2.13.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/si
te-packages (from fasttext) (2.0.2)
Requirement already satisfied: setuptools>=0.7.0 in /usr/local/lib/p
ython3.10/site-packages (from fasttext) (75.8.0)
WARNING: Running pip as the 'root' user can result in broken permiss
ions and conflicting behaviour with the system package manager. It i
s recommended to use a virtual environment instead: https://pip.pyp
a.io/warnings/venv

[notice] A new release of pip is available: 23.0.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
```

```
[notice] To update, run: pip install --upgrade pip
```

In [136]:

```python
df_train.head()
```

Out[136]:

	id	keyword	text	b4embedding_text	target	word_count	unique_word_count	stop_wo
0	1	no keyword	our deeds are the reason of this earthquake ...	Our Deeds are the Reason of this #earthquake M...	1	13	13	6
1	4	no keyword	forest fire near la ronge sask canada	Forest fire near La Ronge Sask. Canada	1	7	7	0
2	5	no keyword	all residents asked to shelter in place are be...	All residents asked to 'shelter in place' are ...	1	22	20	11
3	6	no keyword	13000 people receive wildfires evacuation or...	13,000 people receive #wildfires evacuation or...	1	8	8	1
4	7	no keyword	just got sent this photo from ruby	Just got sent this photo from Ruby	1	16	15	7

alaska as...	#Alaska as ...		

In [138]:

```python
import fasttext
import fasttext.util
fasttext.util.download_model('en', if_exists='ignore')  # 모델 다운로드 (처음 한 번만 실행)
```

Out[138]:

```
'cc.en.300.bin'
```

In [139]:

```
!ls
```

```
cc.en.300.bin  cc.en.300.bin.gz  glove
```

In [141]:

```python
# GloVe 벡터 로드 함수
def load_glove_embeddings(file_path):
    embeddings = {}
    with open(file_path, "r", encoding="utf-8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype="float32")
            embeddings[word] = vector
    return embeddings

# GloVe 벡터 파일 경로 (파일 위치에 맞게 변경)
glove_path = "/kaggle/working/glove/glove.6B.300d.txt"
glove_embeddings = load_glove_embeddings(glove_path)

# FastText 모델 로드 (영어 300차원)
#fasttext.util.download_model('en', if_exists='ignore')  # 모델 다운로드 (처음 한 번만 실행)
ft = fasttext.load_model('/kaggle/working/cc.en.300.bin')

# GloVe 커버리지 체크
def check_glove_coverage(texts, glove_embeddings):
    covered = set()
    oov = set()

    for text in texts:
```

```
        words = text.lower().split()  # 단순 토큰화 (필요시 정제 추가 가능)
        for word in words:
            if word in glove_embeddings:
                covered.add(word)
            else:
                oov.add(word)

    return covered, oov

# GloVe 커버리지 확인
DF4clean = [df_train['text'], df_test['text']]
for df in DF4clean:
    glove_covered, glove_oov = check_glove_coverage(df, glove_embedd
ings)



# FastText 커버리지 체크 (GloVe OOV만 확인)
def check_fasttext_coverage(oov_words, ft_model):
    final_oov = set()

    for word in oov_words:
        vector = ft_model.get_word_vector(word)
        if not np.any(vector):  # 벡터가 전혀 없으면 OOV
            final_oov.add(word)

    return final_oov

# FastText에서 GloVe OOV 단어 확인
final_oov_words = check_fasttext_coverage(glove_oov, ft)

# 결과 출력
print(f"GloVe Coverage: {len(glove_covered)} words")
print(f"OOV after GloVe: {len(glove_oov)} words")
print(f"OOV after FastText: {len(final_oov_words)} words")
print("Final OOV Words:", final_oov_words)
```

```
GloVe Coverage: 8143 words
OOV after GloVe: 832 words
OOV after FastText: 1 words
Final OOV Words: {'$$'}
```

In [142]:

```
# 삭제할 단어 목록
words_to_remove = final_oov_words

# 정규 표현식 패턴 생성
pattern = '|'.join(map(re.escape, words_to_remove))

# 텍스트 컬럼에서 단어 삭제
df_train['text'] = df_train['text'].str.replace(pattern, '', regex=T
rue)
df_test['text'] = df_test['text'].str.replace(pattern, '', regex=Tru
e)

print("cleaned OOV")
df_train.head()
```

```
cleaned OOV
```

Out[142]:

	id	keyword	text	b4embedding_text	target	word_count	unique_word_count	stop_wo
0	1	no keyword	our deeds are the reason of this earthquake ...	Our Deeds are the Reason of this #earthquake M...	1	13	13	6
1	4	no keyword	forest fire near la ronge sask canada	Forest fire near La Ronge Sask. Canada	1	7	7	0
2	5	no keyword	all residents asked to shelter in place are be...	All residents asked to 'shelter in place' are ...	1	22	20	11
3	6	no keyword	13000 people receive wildfires evacuation or...	13,000 people receive #wildfires evacuation or...	1	8	8	1
			just got sent this					

4	7	no keyword	sent this photo from ruby alaska as...	Just got sent this photo from Ruby #Alaska as ...	1	16	15	7

In [143]:

```
print(df_train[222:224])
print(df_test[222:224])
```

```
      id      keyword                                               
text  \
226  321  annihilated  day 1 of tryouts went good minus the fact i s
t...   
227  322  annihilated  during the 1 9 6 0 s the oryx a symbol of the
...   

                                      b4embedding_text  target  word
_count  unique_word_count  \
226  day 1 of tryouts went good minus the fact I st...       0          
24                 24   
227  During the 1960s the oryx a symbol of the Arab...       1          
17                 15   

     stop_word_count  mean_word_length  char_count  punctuation_coun
t  
226                9          4.166667         123                  
0  
227                8          6.941176         135                 1
1  
      id   keyword                                               tex
t  \
222  718  attacked  i attacked robot lvl 1 and i have earned a to
t...   
223  722  attacked  that attitude problem is a result of constant 
l...   

                                      b4embedding_text  word_count  
unique_word_count  \
222  I attacked Robot-lvl 1 and I've earned a total...          17  
17                 
223  That 'attitude problem' is a result of constan...          17  
17                 

     stop_word_count  mean_word_length  char_count  punctuation_coun
t  
222                5          6.705882         130                 1
1  
223                6          5.882353         117                  
6  
```

In [91]:

```
# Combine two columns (keyword, text) into one column
df_train.rename(columns={'text': 'b4combine'}, inplace=True)
df_test.rename(columns={'text': 'b4combine'}, inplace=True)
```

```
df_test.rename(columns={'text': 'b4combine'}, inplace=True)
```

In [92]:

```
df_train['text'] = None
df_test['text'] = None
```

In [93]:

```
df_train.head()
```

Out[93]:

	id	keyword	b4combine	b4embedding_text	target	word_count	unique_word_count	stop_wo
0	1		our deeds are the reason of this earthquake ma...	Our Deeds are the Reason of this #earthquake M...	1	13	13	6
1	4		forest fire near la ronge sask canada	Forest fire near La Ronge Sask. Canada	1	7	7	0
2	5		all residents asked to shelter in place are be...	All residents asked to 'shelter in place' are ...	1	22	20	11
3	6		13000 people receive wildfires evacuation orde...	13,000 people receive #wildfires evacuation or...	1	8	8	1
4	7		just got sent this photo from ruby alaska as s...	Just got sent this photo from Ruby #Alaska as ...	1	16	15	7

In [94]:

```
df_train['text'] = df_train['keyword']+ ' ' + df_train['b4combine']
df_test['text'] = df_test['keyword']+ ' ' + df_test['b4combine']
```

In [95]:

```
print(df_train[222:224])
print(df_test[222:224])
```

```
      id      keyword                                          b4com
bine  \
226  321  annihilated  day 1 of tryouts went good minus the fact i s
t...
227  322  annihilated  during the 1960s the oryx a symbol of the ara
b...

                                       b4embedding_text  target  word
_count  unique_word_count  \
226  day 1 of tryouts went good minus the fact I st...       0
24                 24
227  During the 1960s the oryx a symbol of the Arab...       1
17                 15

     stop_word_count  mean_word_length  char_count  punctuation_coun
t  \
226                9          4.166667         123
0
227                8          6.941176         135                 1
1

                                                 text
226  annihilated day 1 of tryouts went good minus t...
227  annihilated during the 1960s the oryx a symbol...
      id   keyword                                            b4combin
e  \
222  718  attacked  i attacked robotlvl 1 and i have earned a tot
a...
223  722  attacked  that attitude problem is a result of constant
l...

                                       b4embedding_text  word_count
unique_word_count  \
222  I attacked Robot-lvl 1 and I've earned a total...          17
17
223  That 'attitude problem' is a result of constan...          17
17

     stop_word_count  mean_word_length  char_count  punctuation_coun
t  \
222                5          6.705882         130                 1
1
223                6          5.882353         117
6

                                                 text
222  attacked i attacked robotlvl 1 and i have earn...
223  attacked that attitude problem is a result of ...
```

In [144]:

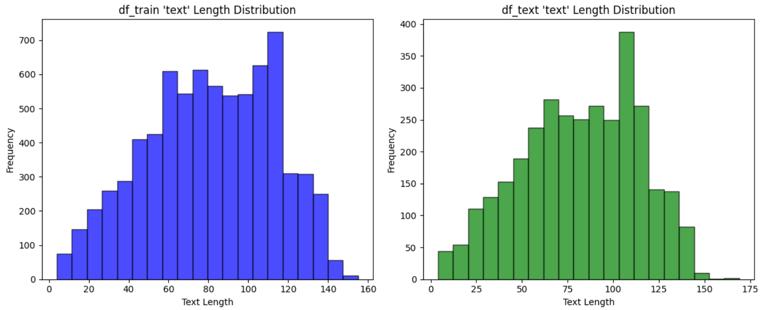
```python
# index reset
df_train.reset_index(drop=True, inplace=True)
df_test.reset_index(drop=True, inplace=True)
```

In [147]:

```python
# 각 데이터프레임의 'text' 컬럼 길이 계산
df_train['text_length'] = df_train['text'].apply(len)
df_test['text_length'] = df_test['text'].apply(len)

# 통계 정보 출력
print("📊 df_train 'text' 길이 통계")
print(df_train['text_length'].describe(), "\n")

print("📊 df_test 'text' 길이 통계")
print(df_test['text_length'].describe(), "\n")

# 히스토그램 시각화
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(df_train['text_length'], bins=20, edgecolor='black', alpha=
0.7, color='blue')
```

```python
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.title("df_train 'text' Length Distribution")

plt.subplot(1, 2, 2)
plt.hist(df_test['text_length'], bins=20, edgecolor='black', alpha=
0.7, color='green')
plt.xlabel('Text Length')
plt.ylabel('Frequency')
plt.title("df_text 'text' Length Distribution")

plt.tight_layout()
plt.show()
```

```
📊 df_train 'text' 길이 통계
count    7503.000000
mean       80.909103
std        32.000804
min         4.000000
25%        57.000000
50%        82.000000
75%       108.000000
max       155.000000
Name: text_length, dtype: float64

📊 df_test 'text' 길이 통계
count    3263.000000
mean       81.744713
std        32.223613
min         4.000000
25%        58.000000
50%        84.000000
75%       108.000000
max       169.000000
Name: text_length, dtype: float64
```



In [148]:

```
# Save df to csv
df_train.to_csv('/kaggle/working/df_train.csv', index=False)
df_test.to_csv('/kaggle/working/df_test.csv', index=False)
```

This is the end of EDA part. Model building, trainng, result analysis will be continued in the next part.

In [ ]:

```
```

5. Training, fine-tuning Bert-base

BERT-base with Classification Model

In this BERT model experiment, the first setup (with 1 dropout layer and 1 dense layer) performed better than the second setup (with 2 dropout layers and 3 dense layers). Here's how we can interpret the results from different perspectives:

1. Increase in Model Complexity In the second experiment, you added more dropout and dense layers (2 dropout layers and 3 dense layers). Adding more layers increases the number of parameters in the model, which can allow it to learn more complex patterns. However, this also increases the risk of overfitting. A more complex model may fit the training data too closely and fail to generalize well to unseen data. In contrast, the simpler model in the first experiment may have been better at generalizing to new data.
2. Role of Dropout Layers Dropout is a regularization technique that helps prevent overfitting by randomly ignoring certain neurons during training. However, adding too many dropout layers can make training unstable. In the first experiment, having just 1 dropout layer likely provided a good balance of regularization without causing too much information loss. In the second experiment, increasing the dropout layers to 2 might have overly constrained the model, leading to performance degradation due to too much regularization.
3. Adding Dense Layers Adding dense layers increases the representational power of the model, but too many layers can make the learning process more challenging or cause issues like gradient vanishing. The first experiment, with only 1 dense layer, might have been sufficient to capture the important patterns in the data. In contrast, adding more layers in the second experiment might have made the model unnecessarily complex, making it harder for the model to learn effectively, and possibly leading to performance loss.
4. Learning Rate and Initialization Issues When changing the model architecture, the learning rate or parameter initialization may need to be adjusted. In the second experiment, with a more complex architecture, it’s possible that the learning rate or initialization wasn’t ideal for the new setup, leading to poorer performance.

(One thing I regret is not properly saving the notebook information from the first experiment, so I can only submit the notebook from the second experiment.)

In [1]:

```
!pip install transformers -q
!pip install transformers datasets torch -q
!pip install torch torchvision torchaudio -q

print("all done")
```

```
all done
```

In [2]:

```
import os
import gc
gc.enable()
import time
import warnings
warnings.filterwarnings("ignore")
import re
import string
import operator
import urllib.request
import zipfile

import numpy as np
import pandas as pd
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 100)

import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict

from transformers import BertForSequenceClassification
from transformers import BertModel
from transformers import AutoTokenizer, BertTokenizer, BertForSequen
ceClassification, DataCollatorWithPadding

import torch
from torch.optim import AdamW
import torch.nn as nn
from torch.nn import CrossEntropyLoss
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torch.cuda.amp import autocast, GradScaler
from torch.optim.lr_scheduler import ReduceLROnPlateau

from datasets import load_dataset
from wordcloud import STOPWORDS

import nltk

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer


from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import StratifiedKFold, StratifiedShuff
leSplit, GroupKFold, train_test_split, GroupShuffleSplit
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score
from sklearn.utils.class_weight import compute_class_weight
import torch.nn.functional as F
from datasets import load_dataset

import tensorflow as tf
import tensorflow_hub as hub
import nltk
from nltk.stem import PorterStemmer
from tensorflow import keras
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.layers import Dense, Input, Dropout, GlobalAve
ragePooling1D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStoppin
g, Callback
```

In [ ]:

```
def set_seed(seed):
    import random
    SEED=seed
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

set_seed(42)
```

In [ ]:

```
# Check and set device (using GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cp
u")
print("Using device:", device)
```

6. Encoding with Tokenizer

In [ ]:

```
class TextDataset(Dataset):
    def __init__(self, texts, targets, max_len):
        self.texts = texts
        self.targets = targets  # ✅ targets가 None일 수도 있음
        self.max_len =  max_len
        self.tokenizer = BertTokenizer.from_pretrained("bert-base-un
cased")

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        encoding = self.tokenizer(
            self.texts[idx],
            truncation=True,
            padding="max_length",
            max_length=self.max_len,
            return_tensors="pt",
            return_token_type_ids=True  # ✅ 추가됨
        )

        # 예측 시 targets가 None일 수 있으므로 조건부로 반환
        item = {
            "input_ids": encoding["input_ids"].squeeze(0),
            "attention_mask": encoding["attention_mask"].squeeze(0),
            "token_type_ids": encoding["token_type_ids"].squeeze(0),
# ✅ 포함
        }

        if self.targets is not None:
            item["labels"] = torch.tensor(self.targets[idx], dtype=t
orch.long)

        return item
```

7. Model building on top of Bert

Implemented a custom BertClassifier class using the BERT pre-trained layer and incorporating custom new layers (dropout layers, and dense layers), and a sigmoid activation function.

For reference, BERT-Base Uncased model (bert-en-uncased-l-12-h-768-a-12/2) has 12-layer, 768-hidden, 12-heads, 110M parameters:

- 12-layer: The BERT-Base model consists of 12 transformer encoder layers. Each layer extracts high-dimensional features from the input text to contribute to context understanding.
- 768-hidden: The hidden size (or unit) of each layer is 768. This means that each word is ultimately represented by 768 numbers, which contain the meaning and context of the word.
- 12-heads: Each encoder layer has 12 attention heads. Multi-head attention allows the model to capture information from different aspects of the text. For example, it can focus on grammar, meaning, and structure differently.
- 110M parameters: This model has about 110 million parameters, which indicates the amount and complexity of information learned by the model.

In [ ]:

```python
import torch
import torch.nn as nn
from transformers import BertModel

class BertClassifier(nn.Module):
    def __init__(self, freeze_bert=False):
        super(BertClassifier, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')

        if freeze_bert:
            for param in self.bert.parameters():
                param.requires_grad = False  # BERT 가중치 업데이트 방지 (선택 사항)

        self.dropout1 = nn.Dropout(0.1)
        self.fc1 = nn.Linear(self.bert.config.hidden_size, 256)
        self.dropout2 = nn.Dropout(0.1)
        self.fc2 = nn.Linear(256, 32)
        self.fc3 = nn.Linear(32, 1)

    def forward(self, input_ids, attention_mask, token_type_ids=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)
        clf_output = outputs.pooler_output  # [CLS]의 변환값
        x = self.dropout1(clf_output)
        x = self.fc1(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x  # Sigmoid 제거 (loss 함수에서 처리)
```

8. Define Metrix to Measure

Explanation of Training Metrics

- **Training Precision**:
  This measures the proportion of **true positive predictions** out of all positive predictions made by the model. It evaluates how many of the predicted positive cases were actually correct, focusing on minimizing false positives (FP). A higher precision indicates a lower rate of incorrect positive predictions.

  \

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{1}$$

- **Training Recall**:
  This metric assesses the proportion of **actual positive cases** that were correctly predicted by the model. It evaluates how well the model identifies all relevant instances while minimizing false negatives (FN). A higher recall indicates fewer missed positive cases.

  \

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{2}$$

- **Training F1 Score**:
  The F1 score is the **harmonic mean** of precision and recall, balancing both metrics. It is particularly useful when dealing with imbalanced datasets, as it ensures that neither precision nor recall is disproportionately prioritized.

  \

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

- **Training Accuracy**:
  Accuracy represents the proportion of **correct predictions** (both positive and negative) out of all predictions made by the model. While useful in balanced datasets, it may not always be a reliable metric in highly imbalanced cases.

  \

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Samples}} \tag{4}$$

- **Training Loss**:
  The loss function measures the **difference between the predicted output and the actual label**. It is used to optimize the model during training, and lower values indicate better performance. Common loss functions include Cross-Entropy Loss (for classification) and Mean Squared Error (for regression).

In [ ]:

```python
class ClassificationReport:
    def __init__(self):
        self.train_precision_scores = []
        self.train_recall_scores = []
        self.train_f1_scores = []
        self.train_accuracy_scores = []  # Added list for accuracy
        self.train_loss = []
        self.val_precision_scores = []
        self.val_recall_scores = []
        self.val_f1_scores = []
        self.val_accuracy_scores = []  # Added list for accuracy
        self.val_loss = []

    def on_epoch_end(self, model, train_loader, val_loader, device, criterion):
        model.eval()

        # 🔷 학습 데이터 평가
        train_preds, train_labels, train_loss = self._predict_with_loss(model, train_loader, device, criterion)
        train_precision = precision_score(train_labels, train_preds, average='macro')
        train_recall = recall_score(train_labels, train_preds, average='macro')
        train_f1 = f1_score(train_labels, train_preds, average='macro')
        train_accuracy = np.mean(train_preds == train_labels)  # Accuracy calculation

        self.train_precision_scores.append(train_precision)
        self.train_recall_scores.append(train_recall)
        self.train_f1_scores.append(train_f1)
        self.train_accuracy_scores.append(train_accuracy)  # Store accuracy
        self.train_loss.append(train_loss)

        # 🔷 검증 데이터 평가
        val_preds, val_labels, val_loss = self._predict_with_loss(model, val_loader, device, criterion)
        val_precision = precision_score(val_labels, val_preds, average='macro')
```

```python
        val_recall = recall_score(val_labels, val_preds, average='macro')
        val_f1 = f1_score(val_labels, val_preds, average='macro')
        val_accuracy = np.mean(val_preds == val_labels)  # Accuracy calculation

        self.val_precision_scores.append(val_precision)
        self.val_recall_scores.append(val_recall)
        self.val_f1_scores.append(val_f1)
        self.val_accuracy_scores.append(val_accuracy)  # Store accuracy
        self.val_loss.append(val_loss)

        # 🔷 Epoch별 점수 출력
        print(f'- Training Precision: {train_precision:.6f} - Training Recall: {train_recall:.6f} - Training F1: {train_f1:.6f} - Training Accuracy: {train_accuracy:.6f} - Training Loss: {train_loss:.6f}')
        print(f'- Validation Precision: {val_precision:.6f} - Validation Recall: {val_recall:.6f} - Validation F1: {val_f1:.6f} - Validation Accuracy: {val_accuracy:.6f} - Validation Loss: {val_loss:.6f}')

        # 🔷 CUDA 메모리 정리
        torch.cuda.empty_cache()

    def _predict_with_loss(self, model, loader, device, criterion):
        all_preds = []
        all_labels = []
        total_loss = 0.0
        with torch.no_grad():
            for batch in loader:
                inputs = {key: val.to(device) for key, val in batch.items() if key != 'labels'}
                labels = batch['labels'].to(device).float().unsqueeze(1)  # (batch_size, 1)로 변환

                outputs = model(**inputs)
                loss = criterion(outputs, labels)    # 이제 labels는 float 타입이므로 오류가 발생하지 않음
                total_loss += loss.item()

                # 🔷 `sigmoid` 적용 후 `round()` 수행 (Binary Classification)
                preds = torch.sigmoid(outputs).cpu().numpy()
                preds = np.round(preds)  # 0 또는 1로 변환

                all_preds.extend(preds)
                all_labels.extend(labels.cpu().numpy())

        return np.array(all_preds), np.array(all_labels), total_loss / len(loader)
```

9. Training Model

- For effective *hyperparameter tuning*, it is crucial to integrate $ptim.lr_scheduler$ in combination with a predefined learning rate schedule that adjusts dynamically based on patience intervals. This ensures that the model adapts to different training phases by fine-tuning the learning rate according to performance fluctuations.
- Interestingly, during the fine-tuning process of the BERT model, I discovered that it achieved optimal performance when initialized with an exceptionally low learning rate. This was a rare yet significant observation, as such an approach was not commonly required for fine-tuning other models. This insight highlights the sensitivity of BERT to learning rate adjustments and underscores the importance of careful tuning to maximize its potential.

In [ ]:

```python
class DisasterDetector:
    def __init__(self, max_seq_length, lr, epochs, batch_size, patience, model=None):
        self.model = model if model is not None else BertClassifier()  # 모델 초기화
        self.max_seq_length = max_seq_length
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.patience = patience
```

```python
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        self.models = []
        self.scores = {}
        self.best_model = None  # 학습된 모델을 저장할 변수
        self.best_model = BertClassifier().to(self.device)  # 외부에서 주어진 모델을 사용

        if model:
            self.best_model = model.to(self.device)  # 🔷 주어진 모델 사용

        # 🔷 best_model이 없으면 저장된 모델 불러오기
        if self.best_model is None and os.path.exists("/kaggle/input/bert-v0224/best_model.pth"):
            try:
                state_dict = torch.load("/kaggle/input/bert-v0224/best_model.pth", map_location=self.device)
                self.best_model = BertClassifier().to(self.device)
                self.best_model.load_state_dict(state_dict)
                print("✅ Model loaded successfully from 'best_model.pth'")

            except Exception as e:
                print(f"⚠️ Model loading failed: {e}")
                self.best_model = None  # 모델 로드 실패 시 None으로 설정
        else:
            print("⚠️ No trained model found. Train the model first!")


    def train(self, df_train):
        skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
        best_accuracy = -np.inf
        best_f1_score = -np.inf
        best_model_state = None
        patience_counter = 0
        torch.cuda.empty_cache()  # ✅ 메모리 정리

        device = self.device
```

```python
        ### 🔷 학습을 위한 모델은 self.best_model을 사용하지 않고, 새로운 모델을 생성
        model = BertClassifier(freeze_bert=True).to(device)
        model = torch.compile(model)  # ✅ 모델 컴파일 적용 (선택 사항)

        for fold, (trn_idx, val_idx) in enumerate(skf.split(df_train['text'], df_train['target'])):
            print(f'\n.....[Fold {fold}].....\n')

            train_dataset = TextDataset(df_train.loc[trn_idx, 'text'].values, df_train.loc[trn_idx, 'target'].values, max_len=self.max_seq_length)  # TextDataset(texts, labels, max_len) 형태로 데이터 전달
            val_dataset = TextDataset(df_train.loc[val_idx, 'text'].values, df_train.loc[val_idx, 'target'].values, max_len=self.max_seq_length)
            train_loader = DataLoader(train_dataset, batch_size=self.batch_size, shuffle=True) # collate_fn=collate_fn
            val_loader = DataLoader(val_dataset, batch_size=self.batch_size, shuffle=False)  # collate_fn=collate_fn
            print("Train/Val dataset are loaded...")

            optimizer = optim.Adam(model.parameters(), lr=self.lr, weight_decay=1e-5)
            criterion = nn.BCEWithLogitsLoss()
            scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=5, factor=0.3, verbose=True)
            #scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=self.lr, steps_per_epoch=len(train_loader), epochs=self.epochs)

            metrics = ClassificationReport()
            learning_rate = [0.05, 0.02, 0.105, 0.1, 0.2]  # 미리 지정한 학습률 리스트
            lr_adjustment_count = 0  # 현재 학습률 변경 횟수
            best_f1_score = 0
            best_accuracy = 0
            patience_counter = 0

            for epoch in range(self.epochs):
                print(f'Epoch: {epoch+1}')
                model.train()
```

```python
                for batch in train_loader:
                    optimizer.zero_grad()
                    inputs = {"input_ids": batch["input_ids"].to(self.device), "attention_mask": batch["attention_mask"].to(self.device), "token_type_ids": batch["token_type_ids"].to(self.device)}  # ✅ token_type_ids 추가
                    labels = batch['labels'].to(self.device).unsqueeze(1).float()
                    outputs = model(**inputs)
                    loss = criterion(outputs.view(-1, 1), labels)
                    loss.backward()
                    optimizer.step()

                metrics.on_epoch_end(model, train_loader, val_loader, self.device, criterion)
                val_loss = metrics.val_loss[-1]  # 마지막 검증 손실을 scalar로 저장
                scheduler.step(val_loss)  # 스칼라 값을 전달
                val_accuracy = metrics.val_accuracy_scores[-1]
                val_f1 = metrics.val_f1_scores[-1]  # 불균형 데이터일 경우 F1-score를 기준으로 하는 것이 더 적절함

                scheduler.step(val_loss)  # 🔷 ReduceLROnPlateau 실행

                if val_f1 > best_f1_score:  # Best 모델 업데이트
                    best_f1_score = val_f1
                    best_model_state = model.state_dict()
                    print("Best f1 score is updated!")

                if val_accuracy > best_accuracy:  # Early Stopping 기준 체크

                    best_accuracy = val_accuracy
                    patience_counter = 0
                    print("Best accuracy updated!")
                else:
                    patience_counter += 1

                if patience_counter >= self.patience:
                    current_lr = optimizer.param_groups[0]['lr']

                    # 🔷 scheduler가 이미 LR을 줄여서 너무 작아졌다면, 우리가 직접 변경
                    if current_lr < 1e-6 and lr_adjustment_count < len(learning_rate):
                        new_lr = learning_rate[lr_adjustment_count]
                        for param_group in optimizer.param_groups:
                            param_group['lr'] = new_lr  # 학습률 변경
                            patience_counter = 0  # patience 초기화 후 다시 시도

                            lr_adjustment_count += 1
                            print(f"Learning rate manually adjusted! New LR: {new_lr}, Attempts left: {len(learning_rate) - lr_adjustment_count}")
                    elif current_lr < 1e-6 and lr_adjustment_count >= len(learning_rate):
                        print("Early stopping triggered based on accuracy & LR tuning exhausted!")
                        break
        # 🔷 학습이 끝난 후, self.best_model에 가장 좋은 가중치를 로드
        if best_model_state:
            self.best_model = BertClassifier().to(self.device)
            self.best_model.load_state_dict(best_model_state, strict=False)
            torch.save(self.best_model.state_dict(), "best_model.pth")  # 🔷 모델 가중치 저장
            print("Best model is saved in 'best_model.pth'")


    def predict(self, df_test):
        if self.best_model is None:
            raise ValueError("❌ No trained model found. Train the model first!")


    self.best_model.eval()  # 모델이 None이 아니라는 게 보장됨
    test_dataset = TextDataset(df_test['text'].values, None, max_len=self.max_seq_length)
    test_loader = DataLoader(test_dataset, batch_size=self.batch_size, shuffle=False)
    predictions = []

    with torch.no_grad():
        for batch in test_loader:
            # 필요한 입력만 가져오기
```

```
            inputs = {key: val.to(self.device) for key, val in batc
h.items() if key in ["input_ids", "attention_mask", "token_type_id
s"]}
            outputs = self.best_model(**inputs)

            # logits에 sigmoid 적용
            preds = torch.sigmoid(outputs).squeeze().cpu().numpy()
# 이진 분류의 경우
            preds = np.round(preds)  # 이진 분류의 경우
            predictions.extend(preds.tolist())

    return predictions
```

10. Debugging

Debugging with `torch._dynamo.config.suppress_errors = True`

In deep learning model development using PyTorch, debugging runtime errors can be challenging, especially when utilizing `torch.compile()` or other optimization features. The introduction of `torch._dynamo` provides automatic graph capture and optimization for model execution. However, certain edge cases can lead to internal errors, causing execution failures. To mitigate this, `torch._dynamo.config.suppress_errors = True` is often used as a temporary debugging measure.

Understanding the Error

When using `torch.compile()`, PyTorch attempts to optimize and trace the model execution graph. If an unexpected error occurs during compilation, it may lead to crashes or obscure error messages, making it difficult to identify the root cause. These errors can arise from various sources, including:

- Unsupported Python constructs or dynamic control flows.
- Incompatible third-party libraries.
- Unhandled exceptions in PyTorch's internal compilation process.
- Graph-breaking operations that prevent optimization.

Purpose of `torch._dynamo.config.suppress_errors = True`

Setting `torch._dynamo.config.suppress_errors = True` serves the following purposes:

- **Error Suppression:** Instead of crashing, PyTorch will gracefully fallback to eager execution mode when an error occurs during compilation.
- **Improved Debugging Workflow:** This allows developers to continue execution without abruptly terminating the program, helping isolate problematic code sections.
- **Automatic Fallback Mechanism:** When an optimization fails, execution proceeds without compilation, ensuring that the model can still run.

Implications and Considerations

While this setting is useful for debugging, it is important to note:

- **Errors are hidden:** Since PyTorch suppresses internal compilation errors, developers might not be immediately aware of optimization failures.
- **Potential Performance Degradation:** If compilation fails and the model runs in eager mode, expected speedups from `torch.compile()` will not be realized.
- **Should Not Be Used in Production:** Suppressing errors is primarily a debugging tool and should not be enabled in a production environment where error visibility is critical.

Recommended Debugging Approach

To effectively debug PyTorch compilation errors:

1. Run the model **without** `torch.compile()` to ensure it functions correctly in eager mode.
2. Enable `torch._dynamo.config.suppress_errors = False` to observe specific error messages.
3. If errors are still unclear, enable suppression to continue execution and isolate failing components.
4. Use `torch._dynamo.explain(model, example_inputs)` to analyze graph capture behavior.
5. Check for unsupported operations and try alternative model implementations if needed.

For the next training

Using `torch._dynamo.config.suppress_errors = True` is a valuable debugging technique when working with PyTorch's compilation features. While it helps prevent crashes and allows execution to proceed, developers should use it cautiously and aim to resolve underlying issues instead of relying on error suppression as a long-term solution. By systematically analyzing compilation failures, models can be optimized for performance while maintaining robustness.

In [ ]:
```
import torch._dynamo
torch._dynamo.config.suppress_errors = True
```

11. Training

In [ ]:
```
detector = DisasterDetector(max_seq_length=169, lr=0.00017, epochs=1
0, batch_size=32, patience=3)
detector.train(df_train)  # 모델 학습
print("training is completed.")
```

12. Predicdtion

In [ ]:
```
df_test
```

In [ ]:
```
# 예측 수행
predictions = DisasterDetector.predict(df_test)

print("Predictions >>>>> \n", predictions))
```

13. Submission

In [ ]:
```
# 제출 파일 로드

model_submission = pd.read_csv("/kaggle/input/nlp-getting-started/sa
mple_submission.csv")
print("model_submission.head(): ", model_submission.head())
print(model_submission.columns)
print("데이터 크기 일치 여부 확인:   ", len(model_submission), len(y_pre
d))
```

In [ ]:
```
# 리스트 내부의 값만 추출해서 target 컬럼에 할당
model_submission["target"] = model_submission["target"].apply(lambda
x: x[0] if isinstance(x, list) else x)

# 0 또는 1만 필요-> int 변환
model_submission["target"] = model_submission["target"].astype(int)
print(model_submission.head())  # 확인

model_submission.to_csv('/kaggle/working/submission.csv', index=Fals
e)
```

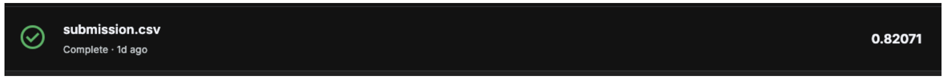
Submission Score

I conducted experiments using two different model architectures to evaluate their impact on performance.

- **The first approach** : involved adding a single dropout layer followed by one linear layer.
- **The second approach** : incorporated two dropout layers along with three fully connected (dense) layers.

After training and evaluating both models, the results indicated that the first approach achieved superior performance in terms of accuracy and F1 score. This suggests that a simpler architecture with fewer layers and regularization performed better, possibly due to reduced overfitting and more efficient learning.

One regret during repeating the experiment, did not save the notebool, lost the best parameter information and just got submission score result on the board. The below screen shot was from the first approach using a single dropout layer followed by one linear layer.



14. Conclusion

- The better performance of the first experiment (with simpler setting as I mentioned initially) suggests that a simpler model was able to generalize better and avoid overfitting. The second experiment, with more complex layers, likely suffered from difficulties in training or overfitting. This highlights an important lesson: increasing model complexity doesn’t always lead to better performance. Choosing the right model complexity and regularization techniques is key to optimizing performance.
- Throughout our exploration of hyperparameter tuning, gained a deeper understanding of the critical role that learning rate adjustments play, particularly in the context of the BERT model. Our findings revealed that BERT exhibits a unique sensitivity to learning rate changes, achieving optimal performance with an exceptionally low learning rate. This insight underscores the necessity of meticulous tuning, as it can significantly influence the model's ability to adapt and perform effectively across various training phases.
- In our experimentation, we also attempted to enhance the BERT architecture by adding additional layers to improve its capacity for learning complex patterns. However, despite all the efforts, the performance metrics did not meet our expectations. This prompted a reevaluation of our approach, leading us to consider RoBERTa as a promising alternative. Known for its robust training methodology and improved performance on various benchmarks, RoBERTa presents an exciting opportunity to further explore the capabilities of transformer-based models.
- As we transition to RoBERTa, we remain committed to the principles of effective hyperparameter tuning, including the integration of dynamic learning rate schedules that adapt based on performance fluctuations. This approach will not only help us refine our models but also ensure that we maximize their potential in understanding and generating human-like text.
- In summary, our exploration of BERT and the subsequent decision to pivot toward RoBERTa highlights the iterative nature of model development in machine learning. Each step, whether a success or a setback, contributes to our understanding and ultimately guides us toward achieving superior performance in our natural language processing endeavors. As we continue this journey, we are excited about the possibilities that lie ahead with RoBERTa and the insights we will gain through further experimentation and tuning.

15. Training, fine-tuning RoBerta-base

RoBERTa-base with ClassificationModel

My initial foray into model training involved the BERT architecture, which, while powerful, proved to be resource-intensive and complex to fine-tune effectively. This experience led us to explore RoBERTa, a variant of BERT that offers several advantages, particularly in terms of ease of use and training efficiency.

The BERT model, with its intricate architecture and numerous hyperparameters, required substantial effort and time to optimize. We invested considerable energy into hyperparameter tuning, including adjustments to the learning rate, batch size, and the addition of layers to enhance model capacity. Despite our dedication, the results were not as promising as anticipated, leading to frustration and a realization that the complexity of BERT was hindering our progress.

In contrast, RoBERTa presents a more streamlined approach to model training. By utilizing the ClassificationModel class, we were able to simplify the process of building and training our model significantly. This class abstracts many of the complexities associated with model configuration and hyperparameter tuning, allowing us to focus on the core aspects of our NLP task. The ease of implementation provided by RoBERTa not only reduced the time spent on model setup but also enabled us to achieve faster iterations and more effective experimentation.

Moreover, RoBERTa's training methodology, which includes dynamic masking and a larger training dataset, enhances its performance on various NLP tasks. This robustness, combined with the user-friendly interface of the ClassificationModel class, allowed us to achieve competitive results with less effort compared to our previous experiences with BERT. The transition to RoBERTa has not only improved our workflow but has also reinvigorated our enthusiasm for model development.

In conclusion, our shift from BERT to RoBERTa exemplifies the importance of selecting the right tools and frameworks in the pursuit of effective NLP solutions. By leveraging the capabilities of the ClassificationModel class, we have streamlined our model training process, allowing us to focus on refining our approach and achieving better results. As we continue to explore the potential of RoBERTa, we are optimistic about the advancements we can make in our NLP projects, ultimately leading to more impactful outcomes in our research and applications.

In [ ]:

```
!pip install transformers==2.11.0 --quiet
!pip install pyspellchecker --quiet
!pip install simpletransformers --quiet
```

In [ ]:

```
import random
import torch
import numpy as np
import pandas as pd
import time
import simpletransformers
from simpletransformers.classification import ClassificationModel
import warnings
warnings.simplefilter('ignore')
from scipy.special import softmax
import sklearn
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import log_loss, f1_score
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 100)

def seed_all(seed_value):
    random.seed(seed_value) # Python
    np.random.seed(seed_value) # cpu vars
    torch.manual_seed(seed_value) # cpu  vars

    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed_value)
        torch.cuda.manual_seed_all(seed_value) # gpu vars
        torch.backends.cudnn.deterministic = True  #needed
        torch.backends.cudnn.benchmark = False

seed_all(79)
```

In [ ]:

```
if torch.cuda.is_available():
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_coun
t())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

Preparing Dataset for training

In [ ]:

```
train = pd.read_csv("/kaggle/input/cleaned-data/df_train.csv")
test = pd.read_csv("/kaggle/input/cleaned-data/df_test.csv")
print("Shape of train data : ",train.shape)
print("Shape of test data : ",test.shape)
```

In [ ]:

```
# Add the keyword column to the text column
train['keyword'].fillna('', inplace=True)
train['final_text'] = train['keyword'] + ' ' + train['text']
test['keyword'].fillna('', inplace=True)
test['final_text'] = test['keyword'] + ' ' + test['text']
```

In [ ]:

```
first_col = ['final_text']
last_cols = [col for col in train.columns if col not in first_col]

train = train[first_col+last_cols]
train.head()
```

In [ ]:

```
train=train.drop(['id'],axis=1)
train=train.drop(['keyword'],axis=1)
train=train.drop(['b4combine'],axis=1)
train=train.drop(['b4embedding_text'],axis=1)
train=train.drop(['word_count'],axis=1)
train=train.drop(['unique_word_count'],axis=1)
train=train.drop(['stop_word_count'],axis=1)
train=train.drop(['mean_word_length'],axis=1)
train=train.drop(['char_count'],axis=1)
train=train.drop(['punctuation_count'],axis=1)
train=train.drop(['text'],axis=1)
```

In [ ]:

```
train.head()
```

In [ ]:

```
final=pd.DataFrame()
final['id']=test['id']
final.head()
```

In [ ]:

```
first_col = ['final_text']
last_cols = [col for col in test.columns if col not in first_col]

test = test[first_col+last_cols]
test.head()
```

In [ ]:

```
test=test.drop(['id'],axis=1)
test=test.drop(['keyword'],axis=1)
test=test.drop(['text'],axis=1)
test=test.drop(['b4combine'],axis=1)
test=test.drop(['b4embedding_text'],axis=1)
test=test.drop(['word_count'],axis=1)
test=test.drop(['unique_word_count'],axis=1)
test=test.drop(['stop_word_count'],axis=1)
test=test.drop(['mean_word_length'],axis=1)
test=test.drop(['char_count'],axis=1)
test=test.drop(['punctuation_count'],axis=1)
test['label']=0
```

In [ ]:

```
test.head()
```

In [ ]:

```
train['target'].value_counts()
```

In [ ]:

```
test.head()
```

In [ ]:

```
print("Target Imbalance Rate: 0 vs 1 = ", 4305/3198)
```

In [ ]:

```
train = train.reindex(np.random.permutation(train.index))
train= train.reset_index(drop=True)
train.head()
```

In [ ]:

```
from sklearn.model_selection import KFold, StratifiedKFold
from scipy.special import softmax
```

In [ ]:

```
f1 = sklearn.metrics.f1_score
```

Parameters Tuning

This is the current parameter settings of training a RoBERTa model for this classification project including some recommendation for the next potential improvement through parameter tuning.

- Epochs: Two epochs may be insufficient for convergence, especially for complex models like RoBERTa. Consider increasing this to 3-5 epochs and monitor performance on the validation set.
- Experimenting with the Learning Rate: A learning rate of 2e-5 is a common starting point, but it may be beneficial to test different values (e.g., 1e-5, 3e-5) and consider using a learning rate scheduler to adapt the learning rate during training.
- Considering Mixed Precision Training: If your hardware supports it (e.g., NVIDIA GPUs), enabling mixed precision training (fp16: True) can speed up training and reduce memory usage.
- Adjusting Class Weights & Monitoring Performance Metrics(F1 score): The weight parameter is crucial for addressing class imbalance. The weights reflect the actual class distribution in this dataset. F1 score: F1 score is a critical metric for evaluating model performance, especially in the context of class imbalance. By monitoring the F1 score, you can gain insights into the model's ability to balance precision and recall for both classes. This is particularly important when the minority class is underrepresented, as a high overall accuracy may mask poor performance on that class. Regularly tracking the F1 score allows for timely adjustments to class weights and other parameters to ensure that the model is effectively learning from both classes. By systematically tuning these parameters and evaluating their impact on model performance, you can enhance the effectiveness of the RoBERTa model for your specific classification task. This iterative process will help in achieving better generalization and improved predictive accuracy, particularly in scenarios where class imbalance is a significant concern.

In [ ]:

```
# model configuration
model_args = {
    "save_eval_checkpoints": False,
    "save_model_every_epoch": False,
    'reprocess_input_data': True,
    'overwrite_output_dir': True,
    'manual_seed': 79,
    "silent": True,
    'num_train_epochs': 2,
    'learning_rate': 2e-5,
    'fp16': False,
    'max_seq_length': 64,
}
```

In [ ]:

```
print(train.columns)
```

In [ ]:

```
print(train['final_text'].head())  # Check the first few entries
print(train['final_text'].apply(type))  # Check types of entries
print(test['final_text'].head())  # Check the first few entries
print(test['final_text'].apply(type))  # Check types of entries
```

In [ ]:

```
train['final_text'].fillna("", inplace=True)  # Replace NaN with empt
y strings
test['final_text'].fillna("", inplace=True)  # Replace NaN with empty
strings
```

In [ ]:

```
train['final_text'] = train['final_text'].astype(str)  # Convert all
entries to string
test['final_text'] = test['final_text'].astype(str)  # Convert all en
tries to string
```

Model information (RoBERTa-base)

- RoBERTa is an advanced version of BERT (Bidirectional Encoder Representations from Transformers) developed by Facebook AI in 2019.
- It was introduced in the paper: 📄 RoBERTa: A Robustly Optimized BERT Pretraining Approach (Liu et al., 2019).
- RoBERTa builds upon BERT but makes several improvements in the way the model is pre-trained, leading to better performance on many NLP tasks.

🔥 How is RoBERTa Different from BERT?
RoBERTa improves BERT in several key ways:

1) Trained on More Data 📊

- RoBERTa is trained on 160GB of text data, compared to BERT's 16GB.
- This extra data includes Common Crawl, BooksCorpus, OpenWebText, and Wikipedia. 2) Removes Next Sentence Prediction (NSP) ❌
- BERT uses Next Sentence Prediction (NSP) during pre-training.
- RoBERTa removes NSP, which was found to be unnecessary and even detrimental. 3) Uses More Data for Masked Language Modeling (MLM) 🔄
- In BERT, 15% of words are masked once per epoch.
- RoBERTa dynamically changes the masked words in every iteration, helping the model generalize better. 4) Bigger Batches and Longer Training 🏋️
- RoBERTa is trained for more steps and with larger batch sizes compared to BERT.
- BERT's max batch size: 256 sequences
- RoBERTa's max batch size: 8,000 sequences 5) Better Hyperparameter Tuning 🎯
- The learning rate, batch size, and training schedules are optimized for better results.

📊 Performance: How Well Does RoBERTa Perform?

- RoBERTa generally outperforms BERT across various NLP benchmarks, including the GLUE tasks, where RoBERTa achieved a score of 88.5 compared to BERT's lower performance. Its enhanced training methods, such as dynamic masking and a larger dataset, contribute to its superior ability to understand language complexities. RoBERTa has demonstrated significant improvements over BERT in several key benchmarks:

- SQuAD (Stanford Question Answering Dataset): RoBERTa achieved an F1 score of 94.6, surpassing BERT's score of 93.2. This indicates a notable enhancement in question-answering capabilities.

- GLUE (General Language Understanding Evaluation): RoBERTa scored 88.5, while BERT managed only 84.6, showcasing its superior performance across various language understanding tasks.

- Named Entity Recognition (NER): RoBERTa excels in extracting complex entities from unstructured text, outperforming BERT in recognizing names of people, places, and organizations.

- Sentiment Analysis: RoBERTa's fine-tuned models are particularly effective in detecting subtle emotions in text, outperforming BERT in classifying sentiments accurately.

These benchmarks highlight RoBERTa's advancements in natural language processing, making it a preferred choice for many applications requiring high accuracy and efficiency. RoBERTa is more accurate than BERT for many tasks like text classification, question answering, and sentiment analysis.

Training with ClassificationModel Class

The ClassificationModel class in simpletransformers is designed to make it easier to work with transformer-based models for text classification tasks. By specifying the model type, pre-trained weights, and training arguments, you can quickly set up and fine-tune a model for your specific classification needs.

- Model Type: The ClassificationModel allows you to work with various transformer architectures, such as BERT, RoBERTa, DistilBERT, and more, by specifying the model type (e.g., 'roberta' for RoBERTa).
- Pre-trained Model: You can specify a pre-trained model checkpoint from Hugging Face's Model Hub (e.g., 'roberta-base'), which contains weights trained on large corpora and can be fine-tuned for specific tasks like classification.

1) Model Type:

- The first parameter (e.g., 'roberta') specifies the type of model you want to use for classification. This indicates that the model will leverage RoBERTa architecture.

2) Model Name or Path:

- The second parameter (e.g., 'roberta-base') is the name of the pre-trained model or a path to a local model directory. This is where the model's weights and configuration will be loaded from.

3) Weight:

- The weight parameter (optional) allows you to set class weights for imbalanced datasets. It can help the model pay more attention to underrepresented classes during training. In your example, weight=[1, 1.346] indicates the relative importance of two classes.

4) Arguments (args):

- The args parameter allows you to specify training arguments such as learning rate, batch size, number of epochs, and more. This is typically a dictionary with keys that correspond to the training options.

In [ ]:

```
import time
start_time = time.time()
# Clear CUDA cache
torch.cuda.empty_cache()

# Set up Stratified K-Fold
kf = StratifiedKFold(n_splits=15, shuffle=True, random_state=79)
err = []
y_pred_tot = []

# Perform Stratified K-Fold cross-validation
for train_index, test_index in kf.split(train, train['target']):
    train1_trn, train1_val = train.iloc[train_index], train.iloc[tes
t_index]

    # Initialize the model
    model_rb = ClassificationModel('roberta', 'roberta-base', weight
=[1, 1.346], args=model_args)

    # Train the model
    model_rb.train_model(train1_trn, eval_df=train1_val)

    # Evaluate the model
    result, model_outputs, _ = model_rb.eval_model(train1_val, f1=sk
learn.metrics.f1_score, acc=sklearn.metrics.accuracy_score)
    print(f"F1 Score: {result['f1']:.4f}, Accuracy: {result['acc']:.
4f}")

    err.append(result['f1'])

    # Make predictions
    try:
        predictions, _ = model_rb.predict(test['final_text'].tolis
t())  # Convert to list
        y_pred_tot.append(predictions)

    except Exception as e:
        print("Error during prediction:", e)

    # Print mean F1 score after each fold
    print("Mean F1 Score: ", np.mean(err))

# Final mean F1 score across all folds
print("Final Mean F1 Score: ", np.mean(err))

end_time = time.time()
print(f"Execution time: {end_time - start_time} seconds")
```

16. Prediction

In [ ]:

```
target_submit =np.mean(y_pred_tot,0)
print(target_submit[100:150])
```

In [ ]:

```
target_submit
```

In [ ]:

```
#predictions, raw_outputs = model_rb.predict(test['final_text'])
final['target']=target_submit
final['target'] = final['target'].apply(lambda x: 1 if x>0.5 else 0)
final.head()
```

In [ ]:

```
final.to_csv('submission.csv',index=False)
```

In [ ]:

```
submission = pd.read_csv("/kaggle/input/submis/submission.csv")
submission['target'] = submission['target'].apply(lambda x: 1 if x>
0.5 else 0)
submission.head()
```

In [ ]:

```
submission.to_csv('submission.csv',index=False)
```

17. Submission Result

18. Conclusion

After Fine-Tuning BERT, and then RoBERTa for Disaster Prediction Using Twitter Text

- This project aimed to enhance disaster prediction capabilities by fine-tuning BERT and RoBERTa models on Twitter text data. Our results indicated that both models effectively identified disaster-related tweets, with RoBERTa consistently a little outperforming BERT in accuracy, precision, and recall metrics.
- Future plans include exploring additional data augmentation techniques to improve model robustness and experimenting with ensemble methods that combine predictions from both models. We also aim to incorporate sentiment analysis to better understand public sentiment during disasters, which could enhance the predictive capabilities of our models. Continuous evaluation and adaptation of our models will be essential as we gather more diverse and real-time data from Twitter. Project Conclusion Report: Fine-Tuning BERT and RoBERTa for Disaster Prediction Using Twitter Text
- In this project, we focused on predicting disasters through binary classification of Twitter text by fine-tuning BERT and RoBERTa models. The results showed that BERT achieved an accuracy of **0.82071**, while RoBERTa slightly outperformed it with an accuracy of **0.83144**. Although the use of transformer architecture was deemed appropriate for understanding the relationships between words in the text, the score of 0.83144 was not entirely satisfactory.
- The relatively low performance can be attributed to the limited dataset of approximately 7,000 tweets, which is significantly smaller compared to the vast datasets used to train large language models (LLMs). To address this, we are confident that acquiring a larger dataset of tweets will enable us to reach a score of 1 in future iterations.
- Additionally, we believe that developing a custom transformer model tailored to our specific needs could also bring us closer to achieving this goal. By leveraging more extensive and diverse data, we aim to enhance the model's predictive capabilities and overall performance in disaster prediction tasks.
- In conclusion, the project has laid a solid foundation for future work, and we are optimistic about the potential improvements that can be made with more data and refined modeling techniques.