DM2024 Lab 2 Homework: Report on Kaggle competition

Name: Fong Kai Jun (馮凱駿)

Student ID: X1130020

A report of my work developing the model for the competition (Exported form Jupyter Notebook code to pdf with comments). This report includes my preprocessing steps, the feature engineering steps and an explanation of my model. I have also mentioned different things I tried and the insights gained.

Table of Contents

---

1) Data exploration and cleaning

1.1 Importing libraries and dataset

In [3]:
```
# import Library
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
import umap
import gensim
import tensorflow
import keras
import ollama
import langchain
import langchain_community
import langchain_core
import bs4
import chromadb
import gradio

%matplotlib inline

print("gensim: " + gensim.__version__)
print ("tensorflow:" + tensorflow.__version__)
print ("keras:" + keras.__version__)

# Standard libraries and general utilities
import os
import json
import re

# NLP and vectorisers
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Machine Learning and Deep Learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import precision_score, recall_score
from sklearn.naive_bayes import MultinomialNB # Naive Bayes classifier
from sklearn.model_selection import cross_val_score
from sklearn.utils import resample

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
from torch.optim import AdamW, Adam
from torch.optim.lr_scheduler import StepLR
from tqdm import tqdm

# BERT model
from transformers import BertTokenizer, BertForMaskedLM, BertForSequenceC

# Text Preprocessing and Tokenization
from tensorflow.keras.preprocessing.text import Tokenizer  # Tokenizer fo
from tensorflow.keras.preprocessing.sequence import pad_sequences  # Padd
```

```python
# TensorFlow and Keras (Deep Learning Framework)
from tensorflow.keras.utils import to_categorical # OHE
```

```
gensim: 4.3.3
tensorflow:2.18.0
keras:3.6.0
```

I noticed that the tweets are in .json format, and first use pandas to convert the json data into a dataframe.


```
{} tweets_DM.json
    1   {"_score": 391, "_index": "hashtag_tweets", "_source": {"tweet": {"hashtags": ["Sna
    2   {"_score": 433, "_index": "hashtag_tweets", "_source": {"tweet": {"hashtags": ["fre
    3   {"_score": 232, "_index": "hashtag_tweets", "_source": {"tweet": {"hashtags": ["bib
```


In [1]:

```python
import pandas as pd

df = pd.read_json('tweets_DM.json', lines=True) # lines for line-delimite

print(df.head())
```

```
   _score          _index                                              _sour
ce  \
0     391  hashtag_tweets  {'tweet': {'hashtags': ['Snapchat'], 'tweet_i
d...
1     433  hashtag_tweets  {'tweet': {'hashtags': ['freepress', 'TrumpLe
g...
2     232  hashtag_tweets  {'tweet': {'hashtags': ['bibleverse'], 'tweet
_...
3     376  hashtag_tweets  {'tweet': {'hashtags': [], 'tweet_id': '0x1cd
5...
4     989  hashtag_tweets  {'tweet': {'hashtags': [], 'tweet_id': '0x2de
2...

            _crawldate   _type
0  2015-05-23 11:42:47  tweets
1  2016-01-28 04:52:09  tweets
2  2017-12-25 04:39:20  tweets
3  2016-01-24 23:53:05  tweets
4  2016-01-08 17:18:59  tweets
```

In [3]:

```python
print(f'The length of the dataframe is {len(df)}')

print("Dataframe Shape:", df.shape)
```

```
The length of the dataframe is 1867535
Dataframe Shape: (1867535, 5)
```

We notice that the _source column is nested by looking at the first row

In [4]:

```python
df['_source'][0]
```

Out[4]:

```
{'tweet': {'hashtags': ['Snapchat'],
  'tweet_id': '0x376b20',
  'text': 'People who post "add me on #Snapchat" must be dehydrated. Cuz
man.... that\'s <LH>'}}
```

We should split this into several colums instead

In [5]:
```
df_flattened = pd.concat(
    [df.drop(columns=['_source']),
     pd.json_normalize(df['_source'])],  # Flatten the '_source' column
    axis=1
)

# Flatten the 'tweet' column as it is nested twice
if 'tweet' in df_flattened:
    df_flattened = pd.concat(
        [df_flattened.drop(columns=['tweet']),
         pd.json_normalize(df_flattened['tweet'])],  # Flatten the 'tweet
        axis=1
    )

print(df_flattened.head())
```

```
   _score          _index           _crawldate   _type  \
0     391  hashtag_tweets  2015-05-23 11:42:47  tweets
1     433  hashtag_tweets  2016-01-28 04:52:09  tweets
2     232  hashtag_tweets  2017-12-25 04:39:20  tweets
3     376  hashtag_tweets  2016-01-24 23:53:05  tweets
4     989  hashtag_tweets  2016-01-08 17:18:59  tweets

                     tweet.hashtags tweet.tweet_id  \
0                        [Snapchat]       0x376b20
1  [freepress, TrumpLegacy, CNN]       0x2d5350
2                     [bibleverse]       0x28b412
3                                []       0x1cd5b0
4                                []       0x2de201

                                          tweet.text
0  People who post "add me on #Snapchat" must be ...
1  @brianklaas As we see, Trump is dangerous to #...
2  Confident of your obedience, I write to you, k...
3                Now ISSA is stalking Tasha 😂😂😂 <LH>
4  "Trust is not the same as faith. A friend is s...
```

In [6]:
```
print("Dataframe Shape:", df_flattened.shape)

df_flattened.columns
```

```
Dataframe Shape: (1867535, 7)
```

Out[6]:
```
Index(['_score', '_index', '_crawldate', '_type', 'tweet.hashtags',
       'tweet.tweet_id', 'tweet.text'],
      dtype='object')
```

We now split this main tweet dataframe into the train and test sets. This is done by merging the emotions from the emotion.csv file

In [7]:
```
df_emotion = pd.read_csv("emotion.csv")
```

In [8]:
```
print(f'The length of df_emotion dataframe is {len(df_emotion)}')

print("Dataframe Shape:", df_emotion.shape)

df_emotion.columns
```

```
The length of df_emotion dataframe is 1455563
Dataframe Shape: (1455563, 2)
```

Out[8]: Index(['tweet_id', 'emotion'], dtype='object')

In [9]:
```
# We merge based on the tweet ids, and hence it is good pratice to ensure
df_flattened['tweet.tweet_id'] = df_flattened['tweet.tweet_id'].astype(st
df_emotion['tweet_id'] = df_emotion['tweet_id'].astype(str)

merged_df = pd.merge(
    df_flattened,
    df_emotion,
    left_on='tweet.tweet_id',
    right_on='tweet_id',
    how='outer' # outer join is done as we wish to filter the test set ba
)

print(merged_df.head())
```

```
   _score          _index            _crawldate   _type  \
0      62  hashtag_tweets  2017-05-14 11:39:43  tweets
1     242  hashtag_tweets  2015-05-16 10:36:47  tweets
2     915  hashtag_tweets  2016-10-15 20:46:37  tweets
3     756  hashtag_tweets  2016-02-14 15:55:45  tweets
4     213  hashtag_tweets  2016-07-25 17:05:35  tweets

                                     tweet.hashtags tweet.tweet_id  \
0                                                []       0x1c7f0f
1                                     [BlackMirror]       0x1c7f10
2  [twitch, Destinybeta, Destiny, Destiny2, Desti...       0x1c7f11
3                                                []       0x1c7f12
4                                    [auspol, fizza]       0x1c7f13

                                         tweet.text  tweet_id       emoti
on
0  @JZED74 While inappropriate AF, he likely wasn...       NaN           N
aN
1   o m g Shut Up And Dance though #BlackMirror <LH>  0x1c7f10           j
oy
2  On #twitch <LH> on the #Destinybeta #Destiny #...  0x1c7f11  anticipati
on
3  I tried to figure out why you mean so much to ...       NaN           N
aN
4  The only “big plan” you ever had in your life,...       NaN           N
aN
```

In [10]:
```
merged_df['tweet_id'].isna().sum()
```

Out[10]: 411972

This number should correspond to our test set emotions

In [11]:
```
# split the dataset based on whether 'emotion' is NaN or not
df_train = merged_df[~merged_df['emotion'].isna()].copy() # Non-NA rows f
df_test = merged_df[merged_df['emotion'].isna()].copy() # NA rows for emo

# drop tweet_id column which has NaN values
df_train.drop(columns=['tweet_id'], inplace=True)
df_test.drop(columns=['tweet_id'], inplace=True)

# Rename tweet.tweet_id to id to standarise with sampleSubmission
df_train.rename(columns={'tweet.tweet_id': 'id'}, inplace=True)
```

```
df_test.rename(columns={'tweet.tweet_id': 'id'}, inplace=True)

print(f"df_train shape: {df_train.shape}")
print(f"df_test shape: {df_test.shape}")
print("df_train preview:")
print(df_train.head())
print("df_test preview:")
print(df_test.head())
```

```
df_train shape: (1455563, 8)
df_test shape: (411972, 8)
df_train preview:
   _score          _index           _crawldate   _type  \
1     242  hashtag_tweets  2015-05-16 10:36:47  tweets
2     915  hashtag_tweets  2016-10-15 20:46:37  tweets
5     939  hashtag_tweets  2016-07-04 07:22:56  tweets
6     181  hashtag_tweets  2016-04-16 12:53:40  tweets
7     970  hashtag_tweets  2017-04-22 17:50:28  tweets

                                      tweet.hashtags        id  \
1                                     [BlackMirror]  0x1c7f10
2  [twitch, Destinybeta, Destiny, Destiny2, Desti...  0x1c7f11
5                                                []  0x1c7f14
6  [Confession, NationalCandyCornDay, CouldEatThe...  0x1c7f15
7                                                []  0x1c7f16

                                          tweet.text       emotion
1   o m g Shut Up And Dance though #BlackMirror <LH>           joy
2  On #twitch <LH> on the #Destinybeta #Destiny #...  anticipation
5  A nice sunny wak this morning not many <LH> ar...           joy
6  I'm one of those people who love candy corn......           joy
7  @metmuseum What are these? They look like some...       disgust
df_test preview:
   _score          _index           _crawldate   _type   tweet.hashtags  \
0      62  hashtag_tweets  2017-05-14 11:39:43  tweets               []
3     756  hashtag_tweets  2016-02-14 15:55:45  tweets               []
4     213  hashtag_tweets  2016-07-25 17:05:35  tweets  [auspol, fizza]
8     603  hashtag_tweets  2017-01-21 19:25:33  tweets               []
9     609  hashtag_tweets  2017-04-25 16:36:47  tweets               []

         id                                         tweet.text emotion
0  0x1c7f0f  @JZED74 While inappropriate AF, he likely wasn...     NaN
3  0x1c7f12  I tried to figure out why you mean so much to ...     NaN
4  0x1c7f13  The only “big plan” you ever had in your life,...     NaN
8  0x1c7f17  Looking back on situations old & new, recent o...     NaN
9  0x1c7f18  @jasoninthehouse Why do you insist on talking ...     NaN
```

In [12]:
```
df_train.columns
```

Out[12]:
```
Index(['_score', '_index', '_crawldate', '_type', 'tweet.hashtags', 'i
d',
       'tweet.text', 'emotion'],
      dtype='object')
```

We now process all columns and ensure that they are of the correct data types

In [13]:
```
def convert_columns(df):
    column_types = {
        '_score': 'float',
        '_index': 'string',
```

```
        '_crawldate': 'datetime64[ns]',
        '_type': 'string',
        'tweet.hashtags': 'object', # convert to list (keep as object in
        'id': 'string',
        'tweet.text': 'string',
        'emotion': 'string'
    }

    for column, dtype in column_types.items():
        if column in df.columns:
            try:
                if dtype == 'object':
                    df[column] = df[column].apply(lambda x: x if isinstan
                else:
                    df[column] = df[column].astype(dtype)
            except Exception as e:
                print(f"Failed to convert column '{column}' to {dtype}: {

    return df

df_train = convert_columns(df_train)
df_test = convert_columns(df_test)
```

1.2 Cleaning dataset

In [25]:

```
columns_to_check = ['tweet.text']

duplicated_rows_test = df_train[df_train.duplicated(subset=columns_to_che

if not duplicated_rows_test.empty:
    print(f"Actual duplicated rows based on columns {columns_to_check}:")
    print(f'Total: {len(duplicated_rows_test)}')
    print(duplicated_rows_test)
else:
    print(f"No duplicates with more than one entry found based on columns
```

```
Actual duplicated rows based on columns ['tweet.text']:
Total: 6381
         _score          _index          _crawldate   _type tweet.hashtags  \
207       885.0  hashtag_tweets 2016-07-16 10:42:02  tweets             []
271       510.0  hashtag_tweets 2015-04-28 12:19:09  tweets             []
324       488.0  hashtag_tweets 2015-05-17 02:28:11  tweets             []
363       284.0  hashtag_tweets 2016-09-02 12:54:06  tweets             []
380        10.0  hashtag_tweets 2015-10-05 21:37:29  tweets             []
...         ...             ...                 ...     ...            ...
1865061   124.0  hashtag_tweets 2017-05-11 09:41:03  tweets             []
1865151   900.0  hashtag_tweets 2015-04-08 07:22:02  tweets             []
1865744     9.0  hashtag_tweets 2017-06-20 19:04:36  tweets             []
1866844   343.0  hashtag_tweets 2016-08-24 20:47:04  tweets             []
1867048   793.0  hashtag_tweets 2017-09-04 20:20:18  tweets             []

               id                                         tweet.text   emotion
207      0x1c7fde  Thank you God for blessing me with another bra...     trust
271      0x1c801e                Scarce CALLS OUT Zaptie |  <LH>  😯   surprise
324      0x1c8053                                   GoodMorning <LH>       joy
363      0x1c807a                        Happy Thanksgiving 🦃🍽️🍁  <LH>       joy
380      0x1c808b                              Thank God for life <LH>     trust
...           ...                                                ...       ...
1865061  0x38f474                           Today was a great day <LH>       joy
1865151  0x38f4ce                                     Good night <LH>   sadness
1865744  0x38f71f                                     goodnight. <LH>       joy
1866844  0x38fb6b                          Such a beautiful day <LH>       joy
1867048  0x38fc37                                      No words. <LH>   disgust

[6381 rows x 8 columns]
```

First we check for duplicates. I understand that there are over 6381 with the duplicated tweet.text; however, I decided to add an additional column to check, '_score' to validate that those are completely seperate rows (assumption here is that _score might be a column that can be used for the prediction as well.)

In [29]:
```
columns_to_check = ['tweet.text', '_score']

duplicated_rows = df_train[df_train.duplicated(subset=columns_to_check, k

if not duplicated_rows.empty:
    print(f"Actual duplicated rows based on columns {columns_to_check}:")
    print(f'Total: {len(duplicated_rows)}')
    print(duplicated_rows)
else:
    print(f"No duplicates with more than one entry found based on columns
```

```
Actual duplicated rows based on columns ['tweet.text', '_score']:
Total: 22
         _score         _index          _crawldate   _type tweet.hashtags  \
24048     600.0  hashtag_tweets 2016-10-05 12:14:07  tweets         [Teen]
368379    253.0  hashtag_tweets 2017-10-03 19:59:26  tweets             []
409108    253.0  hashtag_tweets 2015-04-01 01:52:48  tweets             []
464909     58.0  hashtag_tweets 2015-03-12 04:23:43  tweets         [Teen]
609661    577.0  hashtag_tweets 2017-12-26 20:49:25  tweets             []
693518    330.0  hashtag_tweets 2017-04-11 22:04:25  tweets             []
734786    107.0  hashtag_tweets 2015-09-15 10:07:34  tweets             []
788672    853.0  hashtag_tweets 2015-10-07 04:24:34  tweets             []
837098    600.0  hashtag_tweets 2017-06-13 18:56:52  tweets         [Teen]
909491    577.0  hashtag_tweets 2016-07-21 16:34:17  tweets             []
924861    853.0  hashtag_tweets 2016-05-04 10:24:53  tweets             []
1075782    58.0  hashtag_tweets 2015-11-07 14:30:44  tweets         [Teen]
1083812   659.0  hashtag_tweets 2016-12-24 08:34:44  tweets             []
1229207   127.0  hashtag_tweets 2016-08-22 09:36:31  tweets             []
1245277   127.0  hashtag_tweets 2017-07-01 13:51:26  tweets             []
1388766   107.0  hashtag_tweets 2017-01-03 07:49:10  tweets             []
1397565   330.0  hashtag_tweets 2015-10-05 22:49:16  tweets             []
1507970   723.0  hashtag_tweets 2015-06-24 18:38:54  tweets             []
1521077   723.0  hashtag_tweets 2016-03-13 07:41:21  tweets             []
1522156   659.0  hashtag_tweets 2017-05-06 12:43:44  tweets             []
1527043   552.0  hashtag_tweets 2016-09-25 05:21:06  tweets    [BraShakes]
1788221   552.0  hashtag_tweets 2015-03-02 23:48:31  tweets    [BraShakes]

               id                                         tweet.text  \
24048    0x1cdcff  Particularly BLOOD that was laced heavily with...
368379   0x221e0a                            Today was a good day <LH>
409108   0x22bd23                            Today was a good day <LH>
464909   0x23971c  I didnt want to leave her body to be picked at...
609661   0x25cc8c        Neopone EXPOSED by HUGE YouTuber |  <LH>  😰
693518   0x27141d                                        feeling <LH>
734786   0x27b551  I can tell I'm going to love spending the next...
788672   0x2887cf                            Today was a good day <LH>
837098   0x2944f9  Particularly BLOOD that was laced heavily with...
909491   0x2a5fc2        Neopone EXPOSED by HUGE YouTuber |  <LH>  😰
924861   0x2a9bcc                            Today was a good day <LH>
1075782  0x2ce955  I didnt want to leave her body to be picked at...
1083812  0x2d08b3                       Do more of what makes you <LH>
1229207  0x2f40a6                     please god protect our vets <LH>
1245277  0x2f7f6c                     please god protect our vets <LH>
1388766  0x31afed  I can tell I'm going to love spending the next...
1397565  0x31d24c                                        feeling <LH>
1507970  0x338191  JoeySalads CALLS OUT Book Is Knowledge Mankind...
1521077  0x33b4c4  JoeySalads CALLS OUT Book Is Knowledge Mankind...
1522156  0x33b8fb                       Do more of what makes you <LH>
1527043  0x33cc12  @Vodacom I chose #BraShakes as my shaker <LH> ...
1788221  0x37c84c  @Vodacom I chose #BraShakes as my shaker <LH> ...

              emotion
24048             joy
368379          trust
409108            joy
464909            joy
609661       surprise
693518        sadness
734786            joy
788672            joy
```

```
837098             joy
909491        surprise
924861           trust
1075782            joy
1083812            joy
1229207   anticipation
1245277   anticipation
1388766            joy
1397565          trust
1507970       surprise
1521077       surprise
1522156            joy
1527043          trust
1788221          trust
```

I noticed that there are some rows with the same text and scores but classified with different emotions. This is a **Data Quality** issue. Let's investigate it further

In [30]:
```
columns_to_check = ['tweet.text', '_score', 'emotion']

duplicate_groups_w_emotion = df_train[df_train.duplicated(subset=columns_

if not duplicate_groups_w_emotion.empty:
    print(f"Actual duplicated rows based on columns {columns_to_check}:")
    print(f'Total: {len(duplicate_groups_w_emotion)}')
    print(duplicate_groups_w_emotion)
else:
    print(f"No duplicates with more than one entry found based on columns
```

```
Actual duplicated rows based on columns ['tweet.text', '_score', 'emotio
n']:
Total: 16
         _score          _index          _crawldate   _type tweet.hashtags
\
24048     600.0  hashtag_tweets 2016-10-05 12:14:07  tweets         [Teen]
464909     58.0  hashtag_tweets 2015-03-12 04:23:43  tweets         [Teen]
609661    577.0  hashtag_tweets 2017-12-26 20:49:25  tweets             []
734786    107.0  hashtag_tweets 2015-09-15 10:07:34  tweets             []
837098    600.0  hashtag_tweets 2017-06-13 18:56:52  tweets         [Teen]
909491    577.0  hashtag_tweets 2016-07-21 16:34:17  tweets             []
1075782    58.0  hashtag_tweets 2015-11-07 14:30:44  tweets         [Teen]
1083812   659.0  hashtag_tweets 2016-12-24 08:34:44  tweets             []
1229207   127.0  hashtag_tweets 2016-08-22 09:36:31  tweets             []
1245277   127.0  hashtag_tweets 2017-07-01 13:51:26  tweets             []
1388766   107.0  hashtag_tweets 2017-01-03 07:49:10  tweets             []
1507970   723.0  hashtag_tweets 2015-06-24 18:38:54  tweets             []
1521077   723.0  hashtag_tweets 2016-03-13 07:41:21  tweets             []
1522156   659.0  hashtag_tweets 2017-05-06 12:43:44  tweets             []
1527043   552.0  hashtag_tweets 2016-09-25 05:21:06  tweets     [BraShakes]
1788221   552.0  hashtag_tweets 2015-03-02 23:48:31  tweets     [BraShakes]

               id                                         tweet.text  \
24048    0x1cdcff  Particularly BLOOD that was laced heavily with...
464909   0x23971c  I didnt want to leave her body to be picked at...
609661   0x25cc8c        Neopone EXPOSED by HUGE YouTuber |  <LH>  😰
734786   0x27b551  I can tell I'm going to love spending the next...
837098   0x2944f9  Particularly BLOOD that was laced heavily with...
909491   0x2a5fc2        Neopone EXPOSED by HUGE YouTuber |  <LH>  😰
1075782  0x2ce955  I didnt want to leave her body to be picked at...
1083812  0x2d08b3                        Do more of what makes you <LH>
1229207  0x2f40a6                      please god protect our vets <LH>
1245277  0x2f7f6c                      please god protect our vets <LH>
1388766  0x31afed  I can tell I'm going to love spending the next...
1507970  0x338191  JoeySalads CALLS OUT Book Is Knowledge Mankind...
1521077  0x33b4c4  JoeySalads CALLS OUT Book Is Knowledge Mankind...
1522156  0x33b8fb                        Do more of what makes you <LH>
1527043  0x33cc12  @Vodacom I chose #BraShakes as my shaker <LH> ...
1788221  0x37c84c  @Vodacom I chose #BraShakes as my shaker <LH> ...

              emotion
24048             joy
464909            joy
609661       surprise
734786            joy
837098            joy
909491       surprise
1075782           joy
1083812           joy
1229207  anticipation
1245277  anticipation
1388766           joy
1507970      surprise
1521077      surprise
1522156           joy
1527043         trust
1788221         trust
```

Let's remove the duplicates that have an earlier crawl date. We will store the ids in a list first as I wish to concat it with the remianing ids that have different emotions for the same text.

In [23]:
```
sorted_duplicates = duplicate_groups_w_emotion.sort_values(by=['tweet.tex

earlier_ids = sorted_duplicates.groupby('tweet.text').apply(
    lambda group: group[group['_crawldate'] != group['_crawldate'].min()]
).reset_index(drop=True)

print(f"IDs with the same 'tweet.text' but earlier '_crawldate':")
print(list(earlier_ids))
```

```
IDs with the same 'tweet.text' but earlier '_crawldate':
['0x33cc12', '0x33b8fb', '0x31afed', '0x2ce955', '0x33b4c4', '0x25cc8c',
'0x2944f9', '0x2f7f6c']
```

```
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/218265398
9.py:3: DeprecationWarning: DataFrameGroupBy.apply operated on the groupin
g columns. This behavior is deprecated, and in a future version of pandas
the grouping columns will be excluded from the operation. Either pass `inc
lude_groups=False` to exclude the groupings or explicitly select the group
ing columns after groupby to silence this warning.
  earlier_ids = sorted_duplicates.groupby('tweet.text').apply(
```

In [47]:
```
merged_result = duplicated_rows[['id']].merge(
    duplicate_groups_w_emotion[['id']],
    how='outer',
    on='id',
    indicator=True
)
# print(merged_result)

same_text_diff_emotion = list(merged_result[merged_result['_merge'] == 'l
print("id with same text but different emotions :")
print(same_text_diff_emotion)
```

```
id with same text but different emotions :
['0x221e0a', '0x22bd23', '0x27141d', '0x2887cf', '0x2a9bcc', '0x31d24c']
```

In [50]:
```
tweet_id_to_drop = list(earlier_ids) + same_text_diff_emotion
tweet_id_to_drop
```

Out[50]:
```
['0x33cc12',
 '0x33b8fb',
 '0x31afed',
 '0x2ce955',
 '0x33b4c4',
 '0x25cc8c',
 '0x2944f9',
 '0x2f7f6c',
 '0x221e0a',
 '0x22bd23',
 '0x27141d',
 '0x2887cf',
 '0x2a9bcc',
 '0x31d24c']
```

In [55]:
```
df_train_filtered = df_train[~df_train['id'].isin(tweet_id_to_drop)]

print(f"Original number of rows: {len(df_train)}")
print(f"Number of rows after filtering: {len(df_train_filtered)}")
print(f'This line ensures that the filtering is done correctly: {(len(df_
```

```
Original number of rows: 1455563
Number of rows after filtering: 1455549
This line ensures that the filtering is done correctly: True
```

In [56]:
```
print(f'There are {sum(df_train_filtered["tweet.text"].isna())} null comm
```

```
There are 0 null comments in the dataset
```

1.2.1 Retrospective review of data cleaning

Originally, I felt that having a different score but same text would classify them as different entries. However, I did not test this logic and try to filter just purely on ['tweet.text', 'emotion']. This is a critical step I missed that would largely affect the data quality. This is because there are over (6381 - 4878) 1503 text that have different emotions. These should hav ebeen removed but I only managed to remove 6 (0.4%) of them.

In [57]:
```
columns_to_check = ['tweet.text', 'emotion']

duplicate_groups_w_emotion_no_score = df_train[df_train.duplicated(subset

if not duplicate_groups_w_emotion_no_score.empty:
    print(f"Actual duplicated rows based on columns {columns_to_check}:")
    print(f'Total: {len(duplicate_groups_w_emotion_no_score)}')
    print(duplicate_groups_w_emotion_no_score)
else:
    print(f"No duplicates with more than one entry found based on columns
```

```
Actual duplicated rows based on columns ['tweet.text', 'emotion']:
Total: 4878
         _score          _index          _crawldate   _type tweet.hashtags
\
207       885.0  hashtag_tweets 2016-07-16 10:42:02  tweets             []
271       510.0  hashtag_tweets 2015-04-28 12:19:09  tweets             []
324       488.0  hashtag_tweets 2015-05-17 02:28:11  tweets             []
380        10.0  hashtag_tweets 2015-10-05 21:37:29  tweets             []
503       712.0  hashtag_tweets 2017-12-19 22:17:53  tweets             []
...         ...             ...                 ...     ...            ...
1864566   770.0  hashtag_tweets 2015-10-23 09:04:26  tweets             []
1865061   124.0  hashtag_tweets 2017-05-11 09:41:03  tweets             []
1865151   900.0  hashtag_tweets 2015-04-08 07:22:02  tweets             []
1866844   343.0  hashtag_tweets 2016-08-24 20:47:04  tweets             []
1867048   793.0  hashtag_tweets 2017-09-04 20:20:18  tweets             []

               id                                         tweet.text   emo
tion
207      0x1c7fde  Thank you God for blessing me with another bra...     t
rust
271      0x1c801e                 Scarce CALLS OUT Zaptie |  <LH>  😯   sur
prise
324      0x1c8053                                   GoodMorning <LH>
joy
380      0x1c808b                             Thank God for life <LH>     t
rust
503      0x1c8106    Millions of purple freshwater balderdash ! <LH>     a
nger
...           ...                                                ...
...
1864566  0x38f285  Billions of pickled dunder-headed zapotecs ! <LH>     a
nger
1865061  0x38f474                          Today was a great day <LH>
joy
1865151  0x38f4ce                                    Good night <LH>   sad
ness
1866844  0x38fb6b                           Such a beautiful day <LH>
joy
1867048  0x38fc37                                      No words. <LH>   dis
gust

[4878 rows x 8 columns]
```

1.3 Feature Engineering

Let's analyse tweet length, and study the descriptive statistics.

In [59]:
```python
df_train_filtered['tweet_length'] = df_train_filtered['tweet.text'].apply
print(df_train_filtered['tweet_length'].describe())
```

```
count    1.455549e+06
mean     1.515639e+01
std      6.448589e+00
min      1.000000e+00
25%      1.000000e+01
50%      1.500000e+01
75%      2.000000e+01
max      1.050000e+02
Name: tweet_length, dtype: float64
```

```
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/124521534
5.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train_filtered['tweet_length'] = df_train_filtered['tweet.text'].appl
y(lambda x: len(x.split(" ")))
```

All tweets are less than 105 words, which indicates no obvious outliers. This is due to the maximum character count for tweets https://developer.x.com/en/docs/counting-characters#:~:text=In%20most%20cases%2C%20the%20text,280%20characters%20o

We have a more trougher analysis of the tweet.text

In [78]:

```python
for i in range(10, 20):
    print(df_train_filtered.iloc[2*i + 10]['tweet.text'])
for i in range(10, 20):
    print(df_train_filtered.iloc[2*i + 10]['emotion'])
```

```
She is <LH>  God is looking out for those close ro me.
"Faith is a Choice to trust God even when the road seems uncertain" <LH> <
LH> #GodsCertainty ~D.W.
Conquer fear, there's nothing else!  @PastorCleetus  <LH> <LH>
Put my clothes in the dryer last night and never turned it on. <LH>
Have you guys seen the moon? 🌕 #Love
31 Push the <LH> #everyday. Be some #special. <LH> December 08, 2017 at 1
2:15AM
I can't believe the parole board released OJ.....#stupid
@sarahloganwwe i just watch your match on the wwe.  <LH> work. I hope to s
ee more
compostible bin bags keep splitting <LH> and I can’t have a garden waste b
in where I live #solutionneeded from @maidstonebc
@HSBCUKBusiness still trying to open a business account. Do u actually und
erstand the meaning of good customer service? #appalling <LH>
joy
trust
anticipation
joy
joy
joy
disgust
joy
anger
disgust
```

We notice 3 things.

1. The presence of hashtags (#) can either be part of the sentence of to mention a speific topic
2. Usernames are tagged using the '@', and they can be part of the sentece (i.e. @PastorCleetus) or replying to a person @HSBCUKBusiness. These usernames might not be very useful for models to process as they are usually specialised words
3. Most importantly, there are several in many tweets

1.3.1 Understanding LH

Lets see an example of the in a sentence

"@sarahloganwwe i just watch your match on the wwe. work. I hope to see more"

The above corresponds to a joy emotion. It seems that could be some form of MASKING, and the most likely word is "Good".

The reminds me of the masking that is used for BERT models. In fact, BERT has a BertForMaskedLM that can be used as data imputation for these words. The only pre-processing is for us to convert to [MASK]. This can be done via regex

In [82]:
```
# 1. Preprocessing Functions

# Remove @ mentions based on position
def remove_mentions(text):
    if text.startswith('@'):
        return re.sub(r'@\w+\s*', '', text)  # Remove entire mention at t
    return text.replace('@', '')  # Remove only the @ symbol elsewhere

# Convert hashtags to proper words
def convert_hashtags(text):
    def split_camel_case(hashtag):
        words = re.sub(r'([a-z])([A-Z])', r'\1 \2', hashtag)  # Split cam
        words = re.sub(r'([A-Z]+)([A-Z][a-z])', r'\1 \2', words)  # Handl
        return words

    hashtags = re.findall(r'#\w+', text)
    for hashtag in hashtags:
        words = split_camel_case(hashtag[1:])  # Remove '#' and convert
        text = text.replace(hashtag, words)
    return text

def preprocess_texts(df):
    processed_texts = []
    for i, row in tqdm(df.iterrows(), total=len(df), desc="Processing tex
        text = row['tweet.text']
        text = remove_mentions(text)
        text = convert_hashtags(text)
        text = re.sub(r'<LH>', '[MASK]', text) # Convert <LH> to [MASK] f
        processed_texts.append(text)
    return processed_texts

df_train_filtered['processed_text'] = preprocess_texts(df_train_filtered)
df_test['processed_text'] = preprocess_texts(df_test)
```

```
Processing texts: 100%|██████████| 1455549/1455549 [00:36<00:00, 40077.41i
t/s]
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/335029272
5.py:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train_filtered['processed_text'] = preprocess_texts(df_train_filtere
d)
Processing texts: 100%|██████████| 411972/411972 [00:08<00:00, 49853.05it/
s]
```

In [83]:
```python
df_train_filtered['processed_text']
```

Out[83]:
```
1          o m g Shut Up And Dance though Black Mirror [M...
2          On twitch [MASK] on the Destinybeta Destiny De...
5          A nice sunny wak this morning not many [MASK] ...
6          I'm one of those people who love candy corn......
7          What are these? They look like something toddl...
                                 ...
1867529    Um  My vote For [MASK] For Song Of The Summer ...
1867530     Where is Wes Hoolahan?!  WA Lv IRL COYBIG [MASK]
1867531    Fake news! [MASK] propagated by Tumpkins. [MAS...
1867533                         ..today was brutal  ..Hungover
1867534    Love it when I sun burn my forehead!! NOT!! 😫😱...
Name: processed_text, Length: 1455549, dtype: object
```

In [88]:
```python
for i in range(15,70, 5):
    print(df_test.iloc[i]['processed_text'])
```

```
Woke up to news that Bruce Springsteen is playing Broadway on my birthday.
 [MASK]
I love talking to u kb released stressed I know u will sit down and actual
ly listen thanks [MASK] u
Omg!! May God keep her strong. [MASK]
I done came to far to quit now .. [MASK]
This twitter client, always calling me when his lady needs cab service [MA
SK]
There's no chance they're going to regress to mediocrity! [MASK]
Is it weird that I eat my hot Cheetos with a honey bun? [MASK]
Nana keeps randomly shouting "I love my girls" and we are [MASK]
The bike! Best response of day! Love it. hilarious and [MASK] genius
Everyone's understandably excited about AFI debuting 37mm tonight but I'm
just crying over Wester & 6 to 8 in the same setlist [MASK]
Welp Jessica won HOH. Bye Paul 😭 bb19 BB19paul [MASK]
```

1.3.2 Feature Engineering: Processed text with no stopwords

Create a new column without stopwords to be used for baseline model

In [95]:
```python
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

df_train_filtered['processed_text_no_stopwords'] = df_train_filtered['pro
    lambda x: ' '.join(word for word in x.split() if word.lower() not in
)
```

```
print(f"Sample Text Without Stopwords:\n{df_train_filtered['processed_tex
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/kaijunfong/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Sample Text Without Stopwords:
1               g Shut Dance though Black Mirror [MASK]
2    twitch [MASK] Destinybeta Destiny Destiny2 Des...
5    nice sunny wak morning many [MASK] aroud, whit...
6    I'm one people love candy corn... lot. 😁😂 Conf...
7    these? look like something toddlers make summe...
Name: processed_text_no_stopwords, dtype: object
```

```
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/394515750
9.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train_filtered['processed_text_no_stopwords'] = df_train_filtered['pr
ocessed_text'].apply(
```

In [97]:

```
df_test['processed_text_no_stopwords'] = df_test['processed_text'].apply(
    lambda x: ' '.join(word for word in x.split() if word.lower() not in 
)

print(f"Sample Text Without Stopwords:\n{df_test['processed_text_no_stopw
```

```
Sample Text Without Stopwords:
0              inappropriate AF, likely kidding. [MASK]
3    tried figure mean much me. think single reason...
4    “big plan” ever life, promote TurnbullMalcolm ...
8    Looking back situations old & new, recent what...
9    insist talking Clintons? white house. Quit try...
Name: processed_text_no_stopwords, dtype: object
```

1.3.3 Feature Engineering: Sentiment Analysis using Vader Scores

We can make use of NLTK's vader to conduct sentiment analysis
https://medium.com/@skillcate/sentiment-analysis-using-nltk-vader-98f67f2e6130

In [113…

```
from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

df_train_filtered['vader_scores'] = df_train_filtered['processed_text_no_
# print(df_train_filtered['vader_scores'].head(3))
df_train_filtered['vader_compound'] = df_train_filtered['vader_scores'].a

print(f"Sample VADER Sentiment Scores:\n{df_train_filtered[['processed_te
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/kaijunfong/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/93724503
6.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train_filtered['vader_scores'] = df_train_filtered['processed_text_no
_stopwords'].apply(SentimentIntensityAnalyzer().polarity_scores)
Sample VADER Sentiment Scores:
                         processed_text_no_stopwords  vader_compound
1           g Shut Dance though Black Mirror [MASK]          0.0000
2  twitch [MASK] Destinybeta Destiny Destiny2 Des...          0.0000
5  nice sunny wak morning many [MASK] aroud, whit...          0.8344
6  I'm one people love candy corn... lot. 😁😂 Conf...          0.2732
7  these? look like something toddlers make summe...          0.3612
/var/folders/xv/b11rtkss1v3_w8__j41y1m_00000gn/T/ipykernel_23866/93724503
6.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_train_filtered['vader_compound'] = df_train_filtered['vader_scores'].
apply(lambda x: x['compound'])
```

In [114…

```
df_test['vader_scores'] = df_test['processed_text_no_stopwords'].apply(Se
df_test['vader_compound'] = df_test['vader_scores'].apply(lambda x: x['co

print(f"Sample VADER Sentiment Scores:\n{df_test[['processed_text_no_stop
```

```
Sample VADER Sentiment Scores:
                         processed_text_no_stopwords  vader_compound
0          inappropriate AF, likely kidding. [MASK]          0.1027
3  tried figure mean much me. think single reason...          0.0000
4  “big plan” ever life, promote TurnbullMalcolm ...          0.3818
8  Looking back situations old & new, recent what...          0.2732
9  insist talking Clintons? white house. Quit try...          0.0000
```

1.3.4 Data Distribution (Over and undersampling)

We should also see the distributions of the emotions to check for balance between classes

In [103…

```
print("Original Class Distribution:")
print(df_train_filtered['emotion'].value_counts())
```

```
Original Class Distribution:
emotion
joy             516011
anticipation    248934
trust           205474
sadness         193436
disgust         139101
fear             63999
surprise         48727
anger            39867
Name: count, dtype: Int64
```

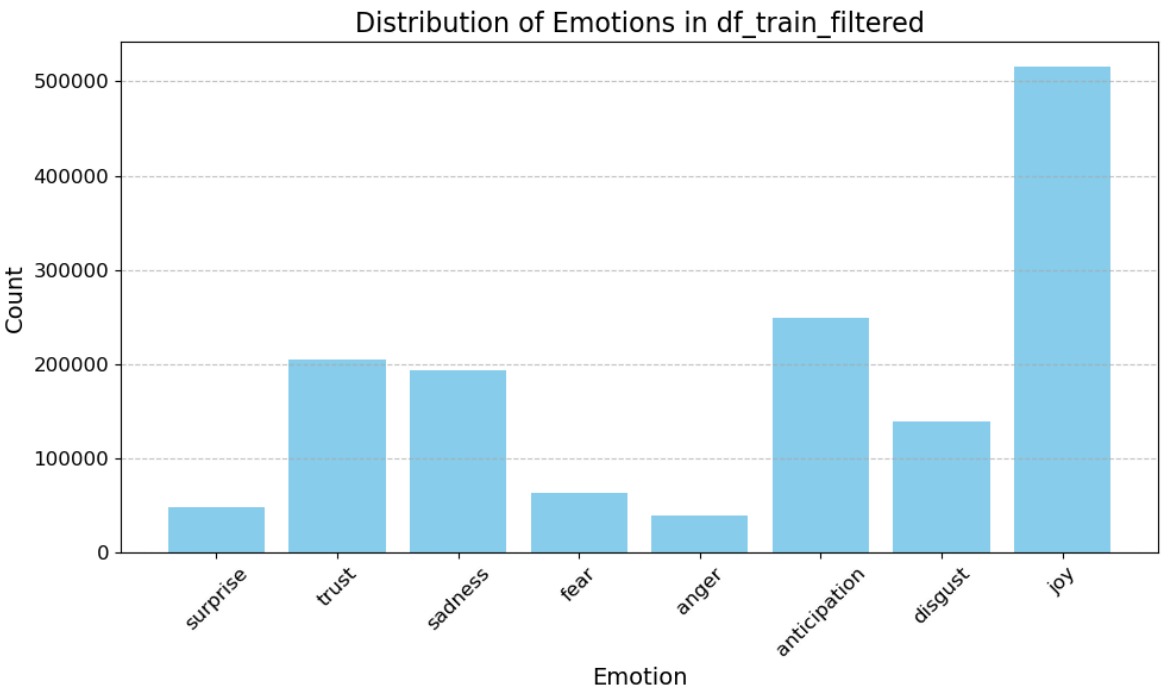
In [104…

```
unique_emotions = set(df_train_filtered['emotion'])

emotion_counts = {emotion: (df_train_filtered['emotion'] == emotion).sum(

plt.figure(figsize=(10, 6))
plt.bar(emotion_counts.keys(), emotion_counts.values(), color='skyblue')
plt.title('Distribution of Emotions in df_train_filtered', fontsize=16)
plt.xlabel('Emotion', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

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



As seen, the original dataset is quite umbalanced with more than 1/3 of the counts being joy. Surpirse, Fear and Anger are the least represented classes.

There are 3 total datasets that will be used for training:

1. Original dataset as is (post cleaning)
2. Oversample the data to be used with traditional Machine learning methods
3. Undersample the data to be used with BERT models (due to long training process)

In [105…

```
# For over-sampling
max_samples = df_train_filtered['emotion'].value_counts().max() # find ma

oversampled_df = pd.concat([
    resample(
        df_train_filtered[df_train_filtered['emotion'] == emotion],
        replace=True,  # Allow duplication
        n_samples=max_samples,
        random_state=42
    )
    for emotion in df_train_filtered['emotion'].unique()
])

print("\nOversampled Class Distribution:")
print(oversampled_df['emotion'].value_counts())
```

```
Oversampled Class Distribution:
emotion
joy             516011
anticipation    516011
disgust         516011
trust           516011
sadness         516011
fear            516011
anger           516011
surprise        516011
Name: count, dtype: Int64
```

Undersample for BERT

In [106…

```
# For under-sampling
min_samples = df_train_filtered['emotion'].value_counts().min() # find mi

undersampled_df = pd.concat([
    resample(
        df_train_filtered[df_train_filtered['emotion'] == emotion],
        replace=False,  # No duplication
        n_samples=min_samples,
        random_state=42
    )
    for emotion in df_train_filtered['emotion'].unique()
])

print("\nUndersampled Class Distribution:")
print(undersampled_df['emotion'].value_counts())
```

```
Undersampled Class Distribution:
emotion
joy             39867
anticipation    39867
disgust         39867
trust           39867
sadness         39867
fear            39867
anger           39867
surprise        39867
Name: count, dtype: Int64
```

1.3.5 Using BERT to predict Masked Words

This undersampled data will go through further processing with BERT to predict the masked words

Reference: https://huggingface.co/docs/transformers/v4.46.3/en/model_doc/bert#transformers.Bert

(This was done via Kaggle notebook with GPU P100)

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}") # use P100 GPU for faster processing spe

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
mask_model = BertForMaskedLM.from_pretrained("bert-base-uncased").to(devi
classification_model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=undersampled_df['emotion'].nunique()
).to(device)

def predict_masked_words(text, emotion):
    # concat emotion for context
    contextual_text = f"[CLS] {emotion} [SEP] {text}" # recall [CLS] mark

    # tokenise the text into embeddings
    tokens = tokenizer(
        contextual_text, return_tensors="pt", max_length=128, truncation=
    ).to(device)

    with torch.no_grad():
        outputs = mask_model(**tokens)
    logits = outputs.logits # logits is the prediction

    # retrieve index of [MASK]
    mask_token_index = torch.where(tokens["input_ids"] == tokenizer.mask_
    predicted_tokens = []
    for idx in mask_token_index:
        predicted_index = logits[0, idx].argmax(dim=-1).item() # predict
        predicted_token = tokenizer.decode([predicted_index]) # convert t
        predicted_tokens.append(predicted_token)

    # Replace [MASK] tokens with predictions
    for mask_token, predicted_token in zip(mask_token_index.tolist(), pre
        tokens["input_ids"][0, mask_token] = tokenizer.convert_tokens_to_

    processed_text = tokenizer.decode(tokens["input_ids"][0], skip_specia

    return re.sub(r'^\w+\s*', '', processed_text) # Remove the emotion (f

# Preprocess training data to predict [MASK]
def prediction_BERT_train_data(df):
    processed_texts = []
    for i, row in tqdm(df.iterrows(), total=len(df), desc="Processing Tra
        text = row['processed_text']
        emotion = row['emotion']
        if '[MASK]' in text:
            text = predict_masked_words(text, emotion)
        processed_texts.append(text)
    df['processed_text'] = processed_texts
    return df
```

```
undersampled_df['processed_text'] = prediction_BERT_train_data(undersampl
```

In [ ]:
```
def predict_masked_words_no_emotion(text):
    contextual_text = f"[CLS] {text}"  # recall [CLS] marks the start

    # tokenise the text into embeddings
    tokens = tokenizer(
        contextual_text, return_tensors="pt", max_length=128, truncation=
    ).to(device)

    with torch.no_grad():
        outputs = mask_model(**tokens)
    logits = outputs.logits # logits is the prediction

    # retrieve index of [MASK]
    mask_token_index = torch.where(tokens["input_ids"] == tokenizer.mask_
    predicted_tokens = []
    for idx in mask_token_index:
        predicted_index = logits[0, idx].argmax(dim=-1).item() # predict
        predicted_token = tokenizer.decode([predicted_index]) # convert t
        predicted_tokens.append(predicted_token)

    # Replace [MASK] tokens with predictions
    for mask_token, predicted_token in zip(mask_token_index.tolist(), pre
        tokens["input_ids"][0, mask_token] = tokenizer.convert_tokens_to_

    processed_text = tokenizer.decode(tokens["input_ids"][0], skip_specia

    return processed_text

# Preprocess test data to predict [MASK]
def prediction_BERT_test_data(df):
    processed_texts = []
    for i, row in tqdm(df.iterrows(), total=len(df), desc="Processing tes
        text = row['processed_text']
        if '[MASK]' in text:
            text = predict_masked_words_no_emotion(text)
        processed_texts.append(text)
    df['processed_text'] = processed_texts
    return df

df_test['processed_text'] = prediction_BERT_test_data(df_test)
```

Successfully ran in 5051.9s

Accelerator	Environment	Output
GPU P100	Latest Container Image	245.97 MB

1.4 Save Data (reference Lab 2)

We will save our data in Pickle format. The pickle module implements binary protocols for serializing and de-serializing a Python object structure.

Some advantages for using pickle structure:

- Because it stores the attribute type, it's more convenient for cross-platform use.
- When your data is huge, it could use less space to store also consume less loading time.

In [ ]:
```
## save to pickle file
df_train_filtered.to_pickle("train_df_original_dist.pkl")
oversampled_df.to_pickle("train_df_undersample.pkl")
undersampled_df.to_pickle('train_df_undersample_BERT.pkl')
df_test.to_pickle('test_BERT.pkl')
```

2) Machine Learning Models

2.1 Naive Bayes

Using scikit-learn `Naive Bayes` performs word frequency and uses these as features to train a model.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

Naive Bayes was chosen as it had the best results for lab 2 master, even better than deep learning models

We will use Naive Bayes with 5 different tokenisers and compare their peformance (Please see appendix for code)

1. Bag of Words (CountVectorizer default tokenizer=None)
2. CountVectorizer(tokenizer=word_tokenize)
3. TfidfVectorizer
4. Word2Vec Tokeniser (pre-trained)
5. Word2Vec Tokeniser (custom)

In [ ]:
```
# 5 different tokenisers as dictionaries to iternate them
features = {
    'CountVectorizer_1': count_vector_1,
    'CountVectorizer_2 (with tokenizer)': count_vector_2,
    'TF-IDF Vectorizer': tfidf_vector,
    'Custom Word2Vec': custom_embeddings,
    'Pretrained Word2Vec': pretrained_embeddings
}

from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(oversampled_df['emotion'])

# Initialize MinMaxScaler for dense embeddings
scaler = MinMaxScaler()

# Evaluate each vectorizer using Naive Bayes and 5-fold cross-validation
for name, feature in features.items():
    print(f"Evaluating: {name}")

    # Check if feature is dense or sparse
    if isinstance(feature, np.ndarray):  # Dense (e.g., Word2Vec)
        # Scale the dense embeddings to non-negative range (0 to 1)
        feature = scaler.fit_transform(feature)

    # Initialize Naive Bayes classifier
    clf = MultinomialNB()
```

```
    try:
        # Perform cross-validation
        scores = cross_val_score(clf, feature, y, cv=5, scoring='accuracy
        print(f"{name} - Mean Accuracy: {np.mean(scores):.4f} ± {np.std(s
    except ValueError as e:
        print(f"Error with {name}: {e}")
```

```
Evaluating: CountVectorizer_1
CountVectorizer_1 - Mean Accuracy: 0.9625 ± 0.0005
Evaluating: CountVectorizer_2 (with tokenizer)
CountVectorizer_2 (with tokenizer) - Mean Accuracy: 0.9468 ± 0.0006
Evaluating: TF-IDF Vectorizer
TF-IDF Vectorizer - Mean Accuracy: 0.9416 ± 0.0007
Evaluating: Custom Word2Vec
Custom Word2Vec - Mean Accuracy: 0.6177 ± 0.0013
Evaluating: Pretrained Word2Vec
Pretrained Word2Vec - Mean Accuracy: 0.8691 ± 0.0015
```

In [ ]:

```
scaler = MinMaxScaler() # need to scale the dense vectors

label_encoder = LabelEncoder() # encode emotions: not necessary for non-n
y_encoded = label_encoder.fit_transform(oversampled_df['emotion'])

# Map feature names to their corresponding training data for refitting
train_features = {
    'NB_CountVectorizer_1': count_vector_1,
    'NB_CountVectorizer_2': count_vector_2,
    'NB_TFIDF_Vectorizer': tfidf_vector,
    'NB_Pretrained_Word2Vec': scaler.fit_transform(pretrained_embeddings)
}

test_features = {
    'NB_CountVectorizer_1': vectorizer_1.transform(df_test['processed_tex
    'NB_CountVectorizer_2': vectorizer_2.transform(df_test['processed_tex
    'NB_TFIDF_Vectorizer': vectorizer_3.transform(df_test['processed_text
    'NB_Pretrained_Word2Vec': scaler.transform(pretrained_embeddings_test
}

for name, feature in test_features.items():
    print(f"Predicting with: {name}")

    ## build Naive Bayes model
    nb_classifier = MultinomialNB()

    ## training!
    clf.fit(train_features[name], y_encoded)

    ## predict!
    predictions_encoded = clf.predict(feature)

    # Decode predictions back to original labels
    predictions = label_encoder.inverse_transform(predictions_encoded)

    submission_df = df_test[['id']].copy()
    submission_df['emotion'] = predictions

    filename = f'BERT_Processed_Undersampled_{name}.csv'
    submission_df.to_csv(filename, index=False)
    print(f"Predictions saved to {filename}")
```

```
Predicting with: NB_CountVectorizer_1
Predictions saved to BERT_Processed_Undersampled_NB_CountVectorizer_1.csv
Predicting with: NB_CountVectorizer_2
Predictions saved to BERT_Processed_Undersampled_NB_CountVectorizer_2.csv
Predicting with: NB_TFIDF_Vectorizer
Predictions saved to BERT_Processed_Undersampled_NB_TFIDF_Vectorizer.csv
Predicting with: NB_Pretrained_Word2Vec
Predictions saved to BERT_Processed_Undersampled_NB_Pretrained_Word2Vec.cs
v
```

2.2 Neural Network Model

The markdown below shows the structure of the Neural Network model. This was done by tokenising the processed_text column for the oversampled_df, which contains the most amount of data (souce of Neural Network).

I included several dropout layers to prevent overfitting.

The accuracy for the validation set was 43.32%, which is much poorer than the 67% in section 6.5 of lab 2. This could be due to the dropout layers I added, which backfired and made the neural network model underfit.

The full code can be found in the appexdix below

Layer (type)	**Output Shape**
text_input (InputLayer)	(None, 100)
embedding (Embedding)	(None, 100, 128)
flatten (Flatten)	(None, 12800)
dense (Dense)	(None, 128)
dropout (Dropout)	(None, 128)
dense_1 (Dense)	(None, 64)
dropout_1 (Dropout)	(None, 64)
output (Dense)	(None, 8)

Total params: 25,621,960 (97.74 MB)

Trainable params: 25,621,960 (97.74 MB)

Non-trainable params: 0 (0.00 B)

Validation Accuracy: 0.4332

2.3 BERT Model

My best performing model involves a combination of BERT to first predict the MASK words and then using BERT to do prediction. This was run on kaggle using P100 GPU and took around 5h to complete.

Source:
https://huggingface.co/docs/transformers/v4.46.3/en/model_doc/bert#transformers.Bert

In [ ]:
```
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, Ad
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

undersampled_df['emotion_encoded'] = undersampled_df['emotion'].astype('c
num_classes = undersampled_df['emotion_encoded'].nunique()

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=len(undersampled_df['emotion'].unique
).to(device)

# Create a class that processes the input text data and returns a diction
class EmotionDataset(Dataset):
    def __init__(self, texts, labels=None):
        self.texts = texts
        self.labels = labels

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

    def __getitem__(self, idx):
        encoding = tokenizer(
            self.texts[idx],
            return_tensors="pt",
            max_length=128,
            padding="max_length",
            truncation=True
        )
        item = {key: val.squeeze(0) for key, val in encoding.items()}
        if self.labels is not None: # for test set there is no labels
            item['labels'] = torch.tensor(self.labels[idx])
        return item

X_train, X_val, y_train, y_val = train_test_split(
    undersampled_df['processed_text'].tolist(),
    undersampled_df['emotion_encoded'].tolist(),
    test_size=0.2, # 80-20 split
    random_state=42
)

train_dataset = EmotionDataset(X_train, y_train)
val_dataset = EmotionDataset(X_val, y_val)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

```
def train_model(model, train_loader, val_loader, epochs=5):
    optimizer = AdamW(model.parameters(), lr=5e-5)

    for epoch in range(epochs):
        model.train()
        total_loss = 0

        for batch in tqdm(train_loader, desc = f"Training Epoch {epoch +
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            optimizer.zero_grad()
            outputs = model(input_ids, attention_mask=attention_mask, lab
            loss = outputs.loss
            total_loss += loss.item()
            loss.backward()
            optimizer.step()

        print(f"Epoch {epoch + 1} Training Loss: {total_loss / len(train_

        # vaalidation
        model.eval()
        val_preds, val_labels = [], []

        with torch.no_grad():
            for batch in val_loader:
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['labels'].to(device)

                outputs = model(input_ids, attention_mask=attention_mask)
                preds = torch.argmax(outputs.logits, dim=1)

                val_preds.extend(preds.cpu().numpy())
                val_labels.extend(labels.cpu().numpy())

        val_accuracy = accuracy_score(val_labels, val_preds)
        print(f"Epoch {epoch + 1} Validation Accuracy: {val_accuracy:.4f}
        print(classification_report(val_labels, val_preds, target_names =

train_model(model, train_loader, val_loader, epochs=5)
```

```
Epoch 5 Training Loss: 0.6489
Epoch 5 Validation Accuracy: 0.5459
              precision    recall  f1-score   support

    surprise       0.66      0.46      0.55      7847
         joy       0.71      0.63      0.67      7996
        fear       0.50      0.47      0.48      8034
anticipation       0.60      0.68      0.63      7990
       trust       0.52      0.54      0.53      8035
     disgust       0.44      0.52      0.47      8046
     sadness       0.44      0.56      0.49      7904
       anger       0.62      0.51      0.56      7936

    accuracy                           0.55     63788
   macro avg       0.56      0.55      0.55     63788
weighted avg       0.56      0.55      0.55     63788
```

In [ ]:

```python
# process test dataset
test_dataset = EmotionDataset(df_test['processed_text'].tolist())
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# prediction function
def predict_test(model, test_loader):
    model.eval()
    predictions = []

    with torch.no_grad():
        for batch in tqdm(test_loader, desc="Generating Test Predictions"
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1)

            predictions.extend(preds.cpu().numpy())

    return predictions

## predict
test_predictions = predict_test(model, test_loader)

# decode back to emotion labels
df_test['predicted_emotion'] = test_predictions
df_test['predicted_emotion'] = df_test['predicted_emotion'].map(
    dict(enumerate(df_test['emotion'].astype('category').cat.categories))
)

filename = 'BERT_Processed.csv'
df_test[['id', 'predicted_emotion']].to_csv(filename, index=False)
print(f"Predictions saved to {filename}")
```

3) Reflections (Insights)

The table below summarises the different methods I tried and the test results on kaggle (approximately 30% of the test data)

Dataset	Model	Mean Accuracy/ Validation accuracy	Kaggle Results (30%)
Unprocessed tweets but oversampled	Naive Bayes with CountVectorizer_1	0.6390 ± 0.0005	0.38051
	Naive Bayes with CountVectorizer_2 (with tokenizer)	0.6516 ± 0.0007	0.38521
	Naive Bayes with TF-IDF Vectorizer	0.3875 ± 0.0004	-
	Naive Bayes with Custom Word2Vec	0.2849 ± 0.0005	-
BERT Predicted without stopwords	Naive Bayes with CountVectorizer_1	**0.9625 ± 0.0005**	0.34657
	Naive Bayes with CountVectorizer_2 (with tokenizer)	0.9468 ± 0.0006	-
	Naive Bayes with TF-IDF Vectorizer	0.9416 ± 0.0007	-
	Naive Bayes with Custom Word2Vec	0.6177 ± 0.0013	-
	Naive Bayes with Pretrained Word2Vec	0.8691 ± 0.0015	0.16906
BERT predicted with neural networks	Neural Network with dropouts	0.4332	0.33328
BERT predicted with BERT model	**BERT model**	0.5459	**0.42903**

Insights gained:

Here are the main takeaways from my experience attempting the hackathon:

1. **Discovery of as a Mask**: Realising that the token could represent a mask was a significant breakthrough. It likely contains crucial keywords directly related to emotion classification, making its prediction and handling pivotal. The inclusion of the emotions for the BertForMaskedLM model should help to provide even

more context to the prediction, and it can not be done without first understanding the structure of BERT model in class.

2. **BERT Superior Performance**: The BERT model performed the best, achieving similar test and validation accuracies. This underscores BERT's ability to consider both left-to-right and right-to-left contexts, and remains a very strong benchmark for NLP standards even till today.

To further improve this model, I would:

- Clean the dataset by removing instances where identical text corresponds to different emotions (as discussed in Section 1.2.1).
- Use batching and oversampling of the orginal distribution of the data to provide a balanced, high-quality dataset, enhancing the model's ability to learn effectively.

3. **Overfitting in Naive Bayes**: The Naive Bayes model trained with BERT predictions and without stopwords overfitted the most. While it achieved a high validation accuracy, its performance on the test set was worse compared to the unprocessed tweets. This may indicate a need for regularisation strategies.

4. **Performance of Unprocessed Tweets**: Unprocessed tweets (without stopwords and removed) outperformed those with BERT-predicted tokens for the test set for naive bayes model. This could be due to:

- Random chance (though unlikely given consistent results).
- The oversampling technique reducing overfitting, leading to a model with lower validation accuracy but better generalisation to unseen test data.

5. **Neural Network Model Challenges**: The neural network model with BERT predictions performed the worst, which was unexpected. One possible improvement would be increasing the training data. However, longer training durations, like me trying to make use of the orginal distribution of text for mask prediction led to a Kaggle timeout error, limiting exploration in this area.

Version 1 was canceled after 43200.1s (timeout exceeded)

Accelerator	Environment	Output
GPU P100	Latest Container Image	0 B

```
Your notebook was stopped because it exceeded the max allowed execution duration. Exit code: 137
```

6. **Handling Large Datasets**: Dealing with large datasets highlighted the importance of using more batching to manage memory and prevent kernel crashes. Merging results post-processing can also help manage resource limitations and avoid timeout issues.

7. **Better than Random**: All my models achieved accuracies greater than 12.5%, outperforming random selection, which is an encouraging baseline comparison for evaluating model performance.

4) Appendix

Code for Bag of Words (CountVectorizer default tokenizer=None)

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import vstack

# build analyzers (bag-of-words)
vectorizer_1 = CountVectorizer()

# 1. Learn a vocabulary dictionary of all tokens in the raw documents.
vectorizer_1.fit(oversampled_df['processed_text_no_stopwords'])

batch_size = 10000 # we will need to do a batching else I will run out of

sparse_matrix_list = []

# 2. Batching Transform documents to document-term matrix.
for i in range(0, len(oversampled_df), batch_size):
    print(f'Batch {round(i/batch_size) + 1} of {round(len(oversampled_df)
    batch_texts = oversampled_df['processed_text_no_stopwords'][i:i + bat
    batch_vector = vectorizer_1.transform(batch_texts)  # Use transform,
    sparse_matrix_list.append(batch_vector)

count_vector_1 = vstack(sparse_matrix_list) # merge the batching

# observe some feature names
feature_names_1 = vectorizer_1.get_feature_names_out()

print(f"CountVectorizer Feature Names:\n{feature_names_1[:10]}")
```

```
Batch 1 of 32
Batch 2 of 32
Batch 3 of 32
Batch 4 of 32
Batch 5 of 32
Batch 6 of 32
Batch 7 of 32
Batch 8 of 32
Batch 9 of 32
Batch 10 of 32
Batch 11 of 32
Batch 12 of 32
Batch 13 of 32
Batch 14 of 32
Batch 15 of 32
Batch 16 of 32
Batch 17 of 32
Batch 18 of 32
Batch 19 of 32
Batch 20 of 32
Batch 21 of 32
Batch 22 of 32
Batch 23 of 32
Batch 24 of 32
Batch 25 of 32
Batch 26 of 32
Batch 27 of 32
Batch 28 of 32
Batch 29 of 32
Batch 30 of 32
Batch 31 of 32
Batch 32 of 32
CountVectorizer Feature Names:
['00' '000' '0000' '0000000000000001' '000001' '000005' '00009jordan'
 '0001' '0004btc' '000am']
```

Code for CountVectorizer(tokenizer=word_tokenize)

In [121…

```
# build analyzers (bag-of-words)
vectorizer_2 = CountVectorizer(tokenizer=word_tokenize)

# 1. Learn a vocabulary dictionary of all tokens in the raw documents.
vectorizer_2.fit(oversampled_df['processed_text_no_stopwords'])

batch_size = 10000 # we will need to do a batching else I will run out of

sparse_matrix_list = []

# 2. Batching Transform documents to document-term matrix.
for i in range(0, len(oversampled_df), batch_size):
    print(f'Batch {round(i/batch_size) + 1} of {round(len(oversampled_df)
    batch_texts = oversampled_df['processed_text_no_stopwords'][i:i + bat
    batch_vector = vectorizer_2.transform(batch_texts)  # Use transform,
    sparse_matrix_list.append(batch_vector)

count_vector_2 = vstack(sparse_matrix_list) # merge the batching

feature_names_2 = vectorizer_2.get_feature_names_out()
```

```
# observe some feature names
print(f"CountVectorizer (tokenizer=word_tokenize) Feature Names:\n{featur
```

```
Batch 1 of 32
Batch 2 of 32
Batch 3 of 32
Batch 4 of 32
Batch 5 of 32
Batch 6 of 32
Batch 7 of 32
Batch 8 of 32
Batch 9 of 32
Batch 10 of 32
Batch 11 of 32
Batch 12 of 32
Batch 13 of 32
Batch 14 of 32
Batch 15 of 32
Batch 16 of 32
Batch 17 of 32
Batch 18 of 32
Batch 19 of 32
Batch 20 of 32
Batch 21 of 32
Batch 22 of 32
Batch 23 of 32
Batch 24 of 32
Batch 25 of 32
Batch 26 of 32
Batch 27 of 32
Batch 28 of 32
Batch 29 of 32
Batch 30 of 32
Batch 31 of 32
Batch 32 of 32
CountVectorizer (tokenizer=word_tokenize) Feature Names:
['!' '#' '$' '%' '&' "'" "''" "'-" "'." "'07"]
```

Code for TfidfVectorizer

In [122…

```
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extra
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import vstack

vectorizer_3 = TfidfVectorizer(max_features=1000)

# 1. Learn a vocabulary dictionary of all tokens in the raw documents.
vectorizer_3.fit(oversampled_df['processed_text_no_stopwords'])

batch_size = 10000 # we will need to do a batching else I will run out of

sparse_matrix_list = []

# 2. Batching Transform documents to document-term matrix.
for i in range(0, len(oversampled_df), batch_size):
    print(f'Batch {round(i/batch_size) + 1} of {round(len(oversampled_df)
    batch_texts = oversampled_df['processed_text_no_stopwords'][i:i + bat
    batch_vector = vectorizer_3.transform(batch_texts)  # Use transform,
    sparse_matrix_list.append(batch_vector)
```

```
tfidf_vector = vstack(sparse_matrix_list)  # merge the batching

feature_names_3 = vectorizer_3.get_feature_names_out()

# observe some feature names
print(f"Top 10 TF-IDF Features:\n{feature_names_3[:10]}")
```

```
Batch 1 of 32
Batch 2 of 32
Batch 3 of 32
Batch 4 of 32
Batch 5 of 32
Batch 6 of 32
Batch 7 of 32
Batch 8 of 32
Batch 9 of 32
Batch 10 of 32
Batch 11 of 32
Batch 12 of 32
Batch 13 of 32
Batch 14 of 32
Batch 15 of 32
Batch 16 of 32
Batch 17 of 32
Batch 18 of 32
Batch 19 of 32
Batch 20 of 32
Batch 21 of 32
Batch 22 of 32
Batch 23 of 32
Batch 24 of 32
Batch 25 of 32
Batch 26 of 32
Batch 27 of 32
Batch 28 of 32
Batch 29 of 32
Batch 30 of 32
Batch 31 of 32
Batch 32 of 32
Top 10 TF-IDF Features:
['00' '00am' '00pm' '01' '02' '03' '04' '05' '06' '07']
```

Code for Word2Vec Tokeniser (pre-trained)

In [128…

```
from gensim.models import Word2Vec
import numpy as np

# Prepare training corpus, similar to .apply(lambda x: nltk.word_tokenize
training_corpus = [text.split() for text in oversampled_df['processed_tex

## setting
vector_dim = 100
window_size = 5
min_count = 1
training_epochs = 20

## model
custom_word2vec = Word2Vec(sentences=training_corpus,
                           vector_size=vector_dim, window=window_size,
                           min_count=min_count, epochs=training_epochs, se
```

```python
import numpy as np

# Function to get sentence embeddings by averaging word vectors
def get_sentence_embedding(model, sentence):
    vectors = [model.wv[word] for word in sentence.split() if word in mod
    if vectors:
        return np.mean(vectors, axis=0) # we will use the mean as propose
    else:
        return np.zeros(model.vector_size)

batch_size = 10000 # we will need to do a batching else I will run out of

custom_embeddings = []

# 2. Batching Transform documents to sentence vectors.
for i in range(0, len(oversampled_df), batch_size):
    print(f'Batch {round(i/batch_size) + 1} of {round(len(oversampled_df)
    batch_texts = oversampled_df['processed_text_no_stopwords'][i:i + bat
    batch_embeddings = [get_sentence_embedding(custom_word2vec, text) for
    custom_embeddings.extend(batch_embeddings)

custom_embeddings = np.array(custom_embeddings) # list -> array

print(f"Custom Word2Vec Embeddings Shape: {custom_embeddings.shape}")
# Custom Word2Vec Embeddings Shape: (4128088, 100) for oversampled data
```

```
Batch 1 of 32
Batch 2 of 32
Batch 3 of 32
Batch 4 of 32
Batch 5 of 32
Batch 6 of 32
Batch 7 of 32
Batch 8 of 32
Batch 9 of 32
Batch 10 of 32
Batch 11 of 32
Batch 12 of 32
Batch 13 of 32
Batch 14 of 32
Batch 15 of 32
Batch 16 of 32
Batch 17 of 32
Batch 18 of 32
Batch 19 of 32
Batch 20 of 32
Batch 21 of 32
Batch 22 of 32
Batch 23 of 32
Batch 24 of 32
Batch 25 of 32
Batch 26 of 32
Batch 27 of 32
Batch 28 of 32
Batch 29 of 32
Batch 30 of 32
Batch 31 of 32
Batch 32 of 32
Custom Word2Vec Embeddings Shape: (318936, 100)
```

Code for Word2Vec Tokeniser (custom)

In [143…

```
from gensim.models import KeyedVectors

## Note: this model is very huge, this will take some time ...
model_path = "GoogleNews-vectors-negative300.bin.gz"
w2v_google_model = KeyedVectors.load_word2vec_format(model_path, binary=T
print('load ok')

# Function to get sentence embeddings using pre-trained Word2Vec
def get_pretrained_embedding(model, sentence):
    vectors = [model[word] for word in sentence.split() if word in model]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(model.vector_size)

batch_size = 10000 # we will need to do a batching else I will run out of

pretrained_embeddings = []

# 2. Batching Transform documents to sentence vectors.
for i in range(0, len(oversampled_df), batch_size):
    batch_texts = oversampled_df['processed_text_no_stopwords'][i:i + bat
    batch_embeddings = [get_pretrained_embedding(w2v_google_model, text)
    pretrained_embeddings.extend(batch_embeddings)

pretrained_embeddings = np.array(pretrained_embeddings) # list -> array

print(f"Pre-trained Word2Vec Embeddings Shape: {pretrained_embeddings.sha

pretrained_embeddings_test = []

# 2. Batching Transform documents to sentence vectors.
for i in range(0, len(test_df), batch_size):
    batch_texts = test_df['processed_text_no_stopwords'][i:i + batch_size
    batch_embeddings = [get_pretrained_embedding(w2v_google_model, text)
    pretrained_embeddings_test.extend(batch_embeddings)

pretrained_embeddings_test = np.array(pretrained_embeddings_test) # list

print(f"Pre-trained Word2Vec Embeddings Shape for test: {pretrained_embed
```

```
load ok
Pre-trained Word2Vec Embeddings Shape: (318936, 300)
Pre-trained Word2Vec Embeddings Shape for test: (411972, 300)
```

Code for Neural Network Model

In [ ]:

```
from keras.models import Model
from keras.layers import Input, Dense
from keras.layers import ReLU, Softmax
from keras.callbacks import CSVLogger

## deal with label (string -> one-hot)
label_encoder = LabelEncoder()
label_encoder.fit(oversampled_df['emotion'])
print('check label: ', label_encoder.classes_)
num_classes = len(label_encoder.classes_)
```

```
# Tokenize text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(oversampled_df['processed_text'])
max_length = 100 # max length for padding
vocab_size = len(tokenizer.word_index) + 1

oversampled_df['processed_text_encoded'] = tokenizer.texts_to_sequences(o
X_text = pad_sequences(oversampled_df['processed_text_encoded'], maxlen=m

# One-hot encode labels
y = to_categorical(oversampled_df['emotion_encoded'], num_classes=num_cla

X_train, X_val, y_train, y_val = train_test_split(X_text, y, test_size=0.

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
    return le.inverse_transform(dec)

y_train = label_encode(label_encoder, y_train)
y_val = label_encode(label_encoder, y_val)

input_shape = X_train.shape[1]
output_shape = len(label_encoder.classes_)

# input layer
model_input = Input(shape=(input_shape, ))  # 500
X = Embedding(vocab_size, 128, input_length=max_length)(model_input)  # E
X = Flatten()(X)  # Flatten embeddings

# 1st hidden layer
X_W1 = Dense(units=128)(X)
H1 = ReLU()(X_W1)

# 2nd hidden layer
H1_W2 = Dense(units=64)(H1)
H2 = ReLU()(H1_W2)
H2 = Dropout(0.3)(H2) # prevent overfitting

# Output layer
H2_W3 = Dense(units=output_shape)(H2) #8
H3 = Softmax()(H2_W3)

model_output = H3

# create model
model = Model(inputs=[model_input], outputs=[model_output])

# loss function & optimizer
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# show model construction
model.summary()
```

```
csv_logger = CSVLogger('logs/training_log.csv')

# training setting
epochs = 30
batch_size = 32

# training!
history = model.fit(X_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    callbacks=[csv_logger],
                    validation_data = (X_val, y_val))
print('training finish')

# 6. Evaluate the Model
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy:.4f}")
```

In [ ]:

```
# Tokenize the test text data
test_sequences = tokenizer.texts_to_sequences(test_df['processed_text'])
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='p

## predict
pred_result = model.predict(test_padded, batch_size=128)
pred_result = label_decode(label_encoder, pred_result)

test_df['emotion'] = pred_result
output_df = test_df[['id', 'emotion']]

filename = 'NN_BERT.csv'
output_df.to_csv(filename, index=False)
print(f"Predictions saved to {filename}")
```