Emotional Analysis using Hugging Face Ecosystem

Set Environment

In this notebook, we have to install following additional libraries (compared to previous notebooks) from Huggingface to enhance our workflow: **transformers**, **datasets**, **evaluate**, and **accelearte**. In addition, we are also installing **wandb**.

- The transformers library provides **Trainer** class that we will use to manage Training process.
- The **datasets** library simplifies the process of accessing and manipulating a wide array of datasets.
- The **evaluate** library offers a suite of standardized metrics and methods for robust and consistent model evaluation.
- We will not use **accelerate** library directly. However , we need to install it as transformer librray usses it in the background.
- Finally **wandb** library provide tools for efficient experiment tracking.

In [1]:
```
import sys
# If in Colab, then import the drive module from google.colab
if 'google.colab' in str(get_ipython()):
  from google.colab import drive
  # Mount the Google Drive to access files stored there
  drive.mount('/content/drive')

  # !pip install torchtext -qq
  # # Install the torchinfo library quietly
  !pip install torchinfo -qq
  # # !pip install torchtext --upgrade -qq
  !pip install torchmetrics -qq
  # !pip install torchinfo -qq
  !pip install fast_ml -qq
  !pip install joblib -qq
  # !pip install sklearn -qq
  # !pip install pandas -qq
  # !pip install numpy -qq
  !pip install scikit-multilearn -qq
  !pip install transformers evaluate wandb accelerate -U -qq
  !pip install pytorch-ignite -qq -U

  basepath = '/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied
_Natural_Language_Processing'
  sys.path.append('/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Ap
plied_Natural_Language_Processing/0_Custom_files')
else:
  basepath = '/Users/harikrishnadev/Library/CloudStorage/GoogleDrive-h
arikrish0607@gmail.com/My Drive/Colab_Notebooks/BUAN_6342_Applied_Natu
ral_Language_Processing/'
  sys.path.append('/Users/harikrishnadev/Library/CloudStorage/GoogleDr
ive-harikrish0607@gmail.com/My Drive/Colab_Notebooks/BUAN_6342_Applied
_Natural_Language_Processing/0_Custom_files')
```

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

In [2]:
```
# Importing PyTorch library for tensor computations and neural network
modules
import torch
import torch.nn as nn

# For working with textual data vocabularies and for displaying model
summaries
from torchtext.vocab import vocab

# General-purpose Python libraries for random number generation and nu
merical operations
import random
import numpy as np

# Utilities for efficient serialization/deserialization of Python obje
cts and for element tallying
import joblib
from collections import Counter

# For creating lightweight attribute classes and for partial function
application
from functools import partial

# For filesystem path handling, generating and displaying confusion ma
trices, and date-time manipulations
from pathlib import Path
from sklearn.metrics import confusion_matrix
from datetime import datetime

# For plotting and visualization
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline

### NEW #########################
# imports from Huggingface ecosystem
from transformers.modeling_outputs import SequenceClassifierOutput
from transformers import PreTrainedModel, PretrainedConfig
from transformers import TrainingArguments, Trainer
from datasets import Dataset, load_dataset
import evaluate

# wandb library
import wandb

import os
```

Specify Project Folders

```
In [3]: # base_folder = Path(basepath)
        # data_folder = base_folder/'datasets/aclImdb'
        # model_folder = base_folder/'models/nlp_spring_2024/imdb/nn'
        # custom_functions = base_folder/'custom-functions'

        # Set the base folder path using the Path class for better path handli
        ng
        base_folder = Path(basepath)

        # Define the data folder path by appending the relative path to the ba
        se folder
        # This is where the data files will be stored
        data_folder = base_folder / '0_Data_Folder'

        # Define the model folder path for saving trained models
        # This path points to a specific folder designated for NLP models rela
        ted to the IMDb dataset
        model_folder = data_folder

        custom_functions = base_folder / '0_Custom_files'
```

```
In [4]: model_folder.mkdir(exist_ok=True, parents = True)
```

```
In [5]: model_folder
```

```
Out[5]: PosixPath('/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Na
        tural_Language_Processing/0_Data_Folder')
```

Loading data

```
In [6]: if 'google.colab' in str(get_ipython()):
            !chmod 600 /content/drive/MyDrive/Colab_Notebooks/BUAN_6382_Applie
        d_DeepLearning/Data/.kaggle/kaggle.json
            !ls -la /content/drive/MyDrive/Colab_Notebooks/BUAN_6382_Applied_D
        eepLearning/Data/.kaggle
        else:
            !chmod 600 '/Users/harikrishnadev/Library/CloudStorage/GoogleDrive
        -harikrish0607@gmail.com/My Drive/Colab_Notebooks/BUAN_6382_Applied_De
        epLearning/Data/.kaggle/kaggle.json'
            ! ls -la '/Users/harikrishnadev/Library/CloudStorage/GoogleDrive-h
        arikrish0607@gmail.com/My Drive/Colab_Notebooks/BUAN_6382_Applied_Deep
        Learning/Data/.kaggle'
```

```
total 1
-rw------- 1 root root 70 Nov 27 02:27 kaggle.json
```

In [7]:
```
if 'google.colab' in str(get_ipython()):
    os.environ['KAGGLE_CONFIG_DIR']='/content/drive/MyDrive/Colab_Note
books/BUAN_6382_Applied_DeepLearning/Data/.kaggle'
else:
    os.environ['KAGGLE_CONFIG_DIR']='/Users/harikrishnadev/Library/Clo
udStorage/GoogleDrive-harikrish0607@gmail.com/My Drive/Colab_Notebook
s/BUAN_6382_Applied_DeepLearning/Data/.kaggle'
```

In [8]:
```
! kaggle competitions download -c emotion-detection-spring2014
```

```
Downloading emotion-detection-spring2014.zip to /content
  0% 0.00/609k [00:00<?, ?B/s]
100% 609k/609k [00:00<00:00, 36.4MB/s]
```

In [9]:
```
! unzip emotion-detection-spring2014.zip
```

```
Archive:  emotion-detection-spring2014.zip
  inflating: sample_submission.csv
  inflating: test.csv
  inflating: train.csv
```

In [10]:
```
import pandas as pd
train_dataset = pd.read_csv('train.csv', usecols=lambda column: column
!= 'ID')
```

In [11]:
```
type(train_dataset)
```

Out[11]:
```
 pandas.core.frame.DataFrame
 def __init__(data=None, index: Axes | None=None, columns: Axes | No
 ne=None, dtype: Dtype | None=None, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
```

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

Out[12]:
```
Index(['Tweet', 'anger', 'anticipation', 'disgust', 'fear', 'joy', 'lo
ve',
       'optimism', 'pessimism', 'sadness', 'surprise', 'trust'],
      dtype='object')
```

In [13]:
```
label_columns = ['anger', 'anticipation', 'disgust', 'fear', 'joy', 'l
ove', 'optimism', 'pessimism', 'sadness', 'surprise', 'trust']
```

In [14]:
```
len(label_columns)
```

Out[14]:
```
11
```

In [15]:
```
trainset = Dataset.from_dict({
    'texts': train_dataset['Tweet'],
    'labels': train_dataset[label_columns].values.tolist(),  # Exclude 
'Tweet' column
})
```

In [16]:
```
trainset.features
```

Out[16]:
```
{'texts': Value(dtype='string', id=None),
 'labels': Sequence(feature=Value(dtype='int64', id=None), length=-1, 
id=None)}
```

In [17]:
```
trainset.features['labels']
```

Out[17]:
```
Sequence(feature=Value(dtype='int64', id=None), length=-1, id=None)
```

In [18]:
```
trainset[1]
```

Out[18]:
```
{'texts': 'Whatever you decide to do make sure it makes you #happy.',
 'labels': [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]}
```

In [19]:
```
import pandas as pd
pd.DataFrame(train_dataset['Tweet']).head()
```

Out[19]:

	Tweet
0	“Worry is a down payment on a problem you may ...
1	Whatever you decide to do make sure it makes y...
2	@Max_Kellerman it also helps that the majorit...
3	Accept the challenges so that you can literall...
4	My roommate: it's okay that we can't spell bec...

Create Custom Model and Model Config Class

In [20]:
```
class CustomConfig(PretrainedConfig):
    def __init__(self, vocab_size=0, embedding_dim=256, hidden_dim1=51
2, hidden_dim2=256, num_labels=11, **kwargs):
        super().__init__()
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.hidden_dim1 = hidden_dim1
        self.hidden_dim2 = hidden_dim2
        self.num_labels = num_labels
```

In [21]:
```
class CustomLSTM(PreTrainedModel):
    config_class = CustomConfig

    def __init__(self, config):
        super().__init__(config)

        self.embedding_bag = nn.EmbeddingBag(config.vocab_size, confi
g.embedding_dim)
        self.lstm = nn.LSTM(config.embedding_dim, config.hidden_dim1,
batch_first=True)
        self.layers = nn.Sequential(
            nn.Linear(config.hidden_dim1, config.hidden_dim2),
            nn.BatchNorm1d(num_features=config.hidden_dim2),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(config.hidden_dim2, config.num_labels)  # 11 out
put labels
        )

    def forward(self, input_ids, offsets, labels=None):
        embed_out = self.embedding_bag(input_ids, offsets)
        # print('embed shape', embed_out.shape)
        lstm_out, _ = self.lstm(embed_out.unsqueeze(0))
        lstm_out = lstm_out.squeeze(0)
        logits = self.layers(lstm_out)
        # print('logit shape', logits.shape)
        # print('labels', labels.shape)
        # print('labels', type(labels))

        loss = None
        if labels is not None:
            loss_fct = nn.BCEWithLogitsLoss()
            loss = loss_fct(logits, labels)

        return SequenceClassifierOutput(
            loss=loss,
            logits=logits
        )
```

Train Model

Collate Function

In [22]:
```
def get_vocab(dataset, min_freq=1):
    """
    Generate a vocabulary from a dataset.

    Args:
        dataset (Dataset): A Hugging Face Dataset object. The dataset
should
                                  have a key 'texts' that contains the text d
ata.
```

```
        min_freq (int): The minimum frequency for a token to be includ
ed in
                        the vocabulary.

    Returns:
        torchtext.vocab.Vocab: Vocabulary object containing tokens fro
m the
                               dataset that meet or exceed the specifi
ed
                               minimum frequency. It also includes a s
pecial
                               '<unk>' token for unknown words.
    """
    # Initialize a counter object to hold token frequencies
    counter = Counter()

    # Update the counter with tokens from each text in the dataset
    # Iterating through texts in the dataset
    for text in dataset['Tweet']:  ###### Change from previous functio
n ####
        counter.update(str(text).split())

    # Create a vocabulary using the counter object
    # Tokens that appear fewer times than `min_freq` are excluded
    my_vocab = vocab(counter, min_freq=min_freq)

    # Insert a '<unk>' token at index 0 to represent unknown words
    my_vocab.insert_token('<unk>', 0)

    # Set the default index to 0
    # This ensures that any unknown word will be mapped to '<unk>'
    my_vocab.set_default_index(0)

    return my_vocab
```

In [23]:
```
tweet_vocab = get_vocab(train_dataset, min_freq=2)
tweet_vocab
```

Out[23]:
```
Vocab()
```

In [24]:
```
tweet_vocab['Accept']
```

Out[24]:
```
45
```

In [25]:
```
# Creating a function that will be used to get the indices of words fr
om vocab
def tokenizer(text, vocab):
    """Converts text to a list of indices using a vocabulary dictionar
y"""
    return [vocab[token] for token in str(text).split()]
```

In [26]:
```
def collate_batch(batch, my_vocab):
    """
    Prepares a batch of data by transforming texts into indices based
on a vocabulary and
    converting labels into a tensor.

    Args:
        batch (list of dict): A batch of data where each element is a
dictionary with keys
                              'labels' and 'texts'. 'labels' are the s
entiment labels, and
                              'texts' are the corresponding texts.
        my_vocab (torchtext.vocab.Vocab): A vocabulary object that map
s tokens to indices.

    Returns:
        dict: A dictionary with three keys:
              - 'input_ids': a tensor containing concatenated indices
of the texts.
              - 'offsets': a tensor representing the starting index of
each text in 'input_ids'.
              - 'labels': a tensor of the labels for each text in the
batch.

    The function transforms each text into a list of indices based on
the provided vocabulary.
    It also converts the labels into a tensor. The 'offsets' are compu
ted to keep track of the
    start of each text within the 'input_ids' tensor, which is a flatt
ened representation of all text indices.
    """

    # Get labels and texts from batch dict samples
    labels = [sample['labels'] for sample in batch]
    # print(labels)
    texts = [sample['texts'] for sample in batch]

    # Convert the list of labels into a tensor of dtype int32
    labels = torch.tensor(labels, dtype=torch.float32)
    # print(labels)
    # print(labels.shape)

    # Convert the list of texts into a list of lists; each inner list
contains the vocabulary indices for a text
    list_of_list_of_indices = [tokenizer(text, my_vocab) for text in t
exts]

    # Concatenate all text indices into a single tensor
    input_ids = torch.cat([torch.tensor(i, dtype=torch.int64) for i in
list_of_list_of_indices])

    # Compute the offsets for each text in the concatenated tensor
    offsets = [0] + [len(i) for i in list_of_list_of_indices]
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)

    return {
        'input_ids': input_ids,
            'offsets': offsets,
            'labels': labels
        }
```

In [27]:
```
tweet_vocab = get_vocab(train_dataset, min_freq=2)
collate_fn = partial(collate_batch, my_vocab=tweet_vocab)
```

Instantiate Model

We will now specify the model using (1) model config class - `CustomConfig` and (2) model class - `CustomLSTM` created earlier.

In [28]:
```
my_config = CustomConfig(vocab_size=len(tweet_vocab))
```

In [29]:
```
my_config
```

Out[29]:
```
CustomConfig {
  "embedding_dim": 256,
  "hidden_dim1": 512,
  "hidden_dim2": 256,
  "id2label": {
    "0": "LABEL_0",
    "1": "LABEL_1",
    "2": "LABEL_2",
    "3": "LABEL_3",
    "4": "LABEL_4",
    "5": "LABEL_5",
    "6": "LABEL_6",
    "7": "LABEL_7",
    "8": "LABEL_8",
    "9": "LABEL_9",
    "10": "LABEL_10"
  },
  "label2id": {
    "LABEL_0": 0,
    "LABEL_1": 1,
    "LABEL_10": 10,
    "LABEL_2": 2,
    "LABEL_3": 3,
    "LABEL_4": 4,
    "LABEL_5": 5,
    "LABEL_6": 6,
    "LABEL_7": 7,
    "LABEL_8": 8,
    "LABEL_9": 9
  },
  "transformers_version": "4.39.3",
  "vocab_size": 10344
}
```

In [30]:
```
my_config.id2label = {
    0: 'anger',
    1: 'anticipation',
    2: 'disgust',
    3: 'fear',
    4: 'joy',
    5: 'love',
    6: 'optimism',
    7: 'pessimism',
    8: 'sadness',
    9: 'surprise',
    10: 'trust'
}
```

In [31]:
```
# Generating id_to_label by reversing the key-value pairs in label_to_
id
my_config.label2id = {v: k for k, v in my_config.id2label .items()}
```

In [32]:
```
my_config
```

Out[32]:
```
CustomConfig {
  "embedding_dim": 256,
  "hidden_dim1": 512,
  "hidden_dim2": 256,
  "id2label": {
    "0": "anger",
    "1": "anticipation",
    "2": "disgust",
    "3": "fear",
    "4": "joy",
    "5": "love",
    "6": "optimism",
    "7": "pessimism",
    "8": "sadness",
    "9": "surprise",
    "10": "trust"
  },
  "label2id": {
    "anger": 0,
    "anticipation": 1,
    "disgust": 2,
    "fear": 3,
    "joy": 4,
    "love": 5,
    "optimism": 6,
    "pessimism": 7,
    "sadness": 8,
    "surprise": 9,
    "trust": 10
  },
  "transformers_version": "4.39.3",
  "vocab_size": 10344
}
```

In [34]:
```
model = CustomLSTM(config=my_config)
```

```
In [35]: model
```

```
Out[35]: CustomLSTM(
           (embedding_bag): EmbeddingBag(10344, 256, mode='mean')
           (lstm): LSTM(256, 512, batch_first=True)
           (layers): Sequential(
             (0): Linear(in_features=512, out_features=256, bias=True)
             (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
             (2): ReLU()
             (3): Dropout(p=0.5, inplace=False)
             (4): Linear(in_features=256, out_features=11, bias=True)
           )
         )
```

Compute_metrics function

```
In [36]: from datasets import load_metric
         def compute_metrics(eval_pred):
           combined_metrics = evaluate.combine([evaluate.load("accuracy"), eval
         uate.load("f1", average = "macro")])
           logits, labels = eval_pred
           predictions = (logits>0.5).astype(int).reshape(-1)
           evaluations = combined_metrics.compute(predictions=predictions, refe
         rences=labels.astype(int).reshape(-1))
           return evaluations
```

Training Arguments

In [37]:
```
# Configure training parameters
training_args = TrainingArguments(

    # Training-specific configurations
    num_train_epochs=20,
    per_device_train_batch_size=128, # Number of samples per training
batch
    per_device_eval_batch_size=128, # Number of samples per validation
batch
    weight_decay=0.1, # weight decay (L2 regularization)
    learning_rate=0.001, # learning arte
    optim='adamw_torch', # optimizer
    remove_unused_columns=False, # flag to retain unused columns

    # Checkpoint saving and model evaluation settings
    output_dir=str(model_folder),  # Directory to save model checkpoin
ts
    evaluation_strategy='steps',  # Evaluate model at specified step i
ntervals
    eval_steps=50,  # Perform evaluation every 50 training steps
    save_strategy="steps",  # Save model checkpoint at specified step
intervals
    save_steps=50,  # Save a model checkpoint every 50 training steps
    load_best_model_at_end=True,  # Reload the best model at the end o
f training
    save_total_limit=2,  # Retain only the best and the most recent mo
del checkpoints
    # Use 'accuracy' as the metric to determine the best model
    metric_for_best_model="accuracy",
    greater_is_better=True,  # A model is 'better' if its accuracy is
higher


    # Experiment logging configurations
    logging_strategy='steps',
    logging_steps=50,
    report_to='wandb',  # Log metrics and results to Weights & Biases
platform
    run_name='tweet_hf_trainer',  # Experiment name for Weights & Bias
es
)
```

Initialize Trainer

In [64]:
```
trainset[0]
```

Out[64]:
```
{'texts': "“Worry is a down payment on a problem you may never have'.
\xa0Joyce Meyer.  #motivation #leadership #worry",
 'labels': [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1]}
```

In [39]:
```
# Split train dataset into train and validation sets
# train_size = int(0.8 * len(train_dataset))
# valid_size = len(train_dataset) - train_size
train_set = trainset.train_test_split(test_size=0.2)
```

In [63]:
```
train_set
```

Out[63]:
```
DatasetDict({
    train: Dataset({
        features: ['texts', 'labels'],
        num_rows: 6179
    })
    test: Dataset({
        features: ['texts', 'labels'],
        num_rows: 1545
    })
})
```

In [41]:
```
[sample['labels'] for sample in train_set['train']][:5]
```

Out[41]:
```
[[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]]
```

In [42]:
```
[sample['texts'] for sample in train_set['train']][:5]
```

Out[42]:
```
['petrify  me in the fossil type of way',
 'I WANNA GET UP AND DANCE!!!! (but everyone is in bed) this sucks! Ev
eryone wake up!! 😯  #hyper #happy #letsdance #dirtydancinginthemoonlig
ht👌 ',
 "@PatBlanchfield so you mean “like Uber but for despair for someone o
ther than the driver'",
 '@ImJim_YoureNot  cyber bully',
 "@DailyMailCeleb @KTHopkins Katie I love how you describe yourself as
'normal'. Really you're anything but!  #neverchange"]
```

In [43]:
```
label_mt = train_set['train']['labels']
# label_mt
```

In [44]:
```
type(label_mt)
```

Out[44]:
```
list
```

In [45]:
```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_set['train'],
    eval_dataset = train_set['test'],
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
)
```

```
/usr/local/lib/python3.10/dist-packages/accelerate/accelerator.py:436:
FutureWarning: Passing the following arguments to `Accelerator` is dep
recated and will be removed in version 1.0 of Accelerate: dict_keys
(['dispatch_batches', 'split_batches', 'even_batches', 'use_seedable_s
ampler']). Please pass an `accelerate.DataLoaderConfiguration` instea
d:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, spl
it_batches=False, even_batches=True, use_seedable_sampler=True)
  warnings.warn(
```

Setup WandB

In [46]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import userdata
    wandb.login(key=userdata.get('wandb'))
else:
    !wandb login
```

```
wandb: W&B API key is configured. Use `wandb login --relogin` to force
relogin
wandb: WARNING If you're specifying your api key in code, ensure this
code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variabl
e, or running `wandb login` from the command line.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

In [47]:
```
# specify the project name where the experiment will be logged
%env WANDB_PROJECT = nlp_course_spring_2024-emotion-analysis-hf-traine
r-lstm
```

```
env: WANDB_PROJECT=nlp_course_spring_2024-emotion-analysis-hf-trainer-
lstm
```

Training and Validation

In [48]:
```
trainer.train()
```

Changes to your `wandb` environment variables will be ignored because your `wandb` session has already started. For more information on how to modify your settings with `wandb.init()` arguments, please refer to the W&B docs (https://wandb.me/wandb-init).

```
wandb: Currently logged in as: harikrish0607 (harikrishnad). Use `wand
b login --relogin` to force relogin
```

Tracking run with wandb version 0.16.6

Run data is saved locally in `/content/wandb/run-20240405_231255-z5y5es7b`

Syncing run **tweet_hf_trainer (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/z5y5es7b)** to Weights & Biases (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm) (docs (https://wandb.me/run))

View project at https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm)

View run at https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/z5y5es7b (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/z5y5es7b)

[980/980 02:20, Epoch 20/20]

Step	Training Loss	Validation Loss	Accuracy	F1
50	0.519900	0.492763	0.784819	0.007598
100	0.456100	0.460963	0.788879	0.083759
150	0.425300	0.446586	0.798235	0.195637
200	0.397300	0.434848	0.804648	0.267108
250	0.369600	0.429968	0.811297	0.324415
300	0.346300	0.437914	0.815063	0.387926
350	0.328700	0.435834	0.818241	0.403783
400	0.310600	0.437066	0.820124	0.427421
450	0.291400	0.443022	0.819006	0.452084
500	0.280600	0.448930	0.822830	0.454034
550	0.266700	0.446900	0.823831	0.467639
600	0.259400	0.465008	0.822713	0.466631
650	0.246000	0.458587	0.825007	0.476408
700	0.241200	0.462138	0.823360	0.479542
750	0.232900	0.472338	0.823183	0.481628
800	0.224600	0.474630	0.824007	0.481539
850	0.215800	0.475016	0.823595	0.484703
900	0.215800	0.481325	0.823654	0.488305
950	0.212600	0.480865	0.823477	0.486301

```
Removed shared tensor {'lstm.bias_ih_l0', 'lstm.bias_hh_l0', 'lstm.wei
ght_hh_l0'} while saving. This should be OK, but check by verifying th
at you don't receive any warning while reloading
There were missing keys in the checkpoint model loaded: ['lstm.weight_
hh_l0', 'lstm.bias_ih_l0', 'lstm.bias_hh_l0'].
```

```
Out[48]: TrainOutput(global_step=980, training_loss=0.3044367989715265, metrics
={'train_runtime': 142.9704, 'train_samples_per_second': 864.374, 'tra
in_steps_per_second': 6.855, 'total_flos': 51918688180800.0, 'train_lo
ss': 0.3044367989715265, 'epoch': 20.0})
```

In [49]:
```
trainer.evaluate()
```

[13/13 00:03]

Out[49]:
```
{'eval_loss': 0.45691511034965515,
 'eval_accuracy': 0.8245954692556634,
 'eval_f1': 0.4736005650715169,
 'eval_runtime': 6.4902,
 'eval_samples_per_second': 238.051,
 'eval_steps_per_second': 2.003,
 'epoch': 20.0}
```

In [50]:
```
test_dataset = pd.read_csv('test.csv', usecols=lambda column: column !
= 'ID')
```

In [67]:
```
testset = Dataset.from_dict({
    'texts': test_dataset['Tweet'].to_list(),
    'labels': [[0] * 11] * len(test_dataset),  # Exclude 'Tweet' colum
n
})
```

In [68]:
```
testset[0]
```

Out[68]:
```
{'texts': '@Adnan__786__ @AsYouNotWish Dont worry Indian army is on it
s ways to dispatch all Terrorists to Hell',
 'labels': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]}
```

In [69]:
```
valid_output = trainer.predict(testset)
```

In [70]:
```
valid_output._fields
```

Out[70]:
```
('predictions', 'label_ids', 'metrics')
```

In [71]:
```
valid_preds = np.argmax(valid_output.predictions, axis=-1)
valid_labels = np.array(valid_output.label_ids)
```

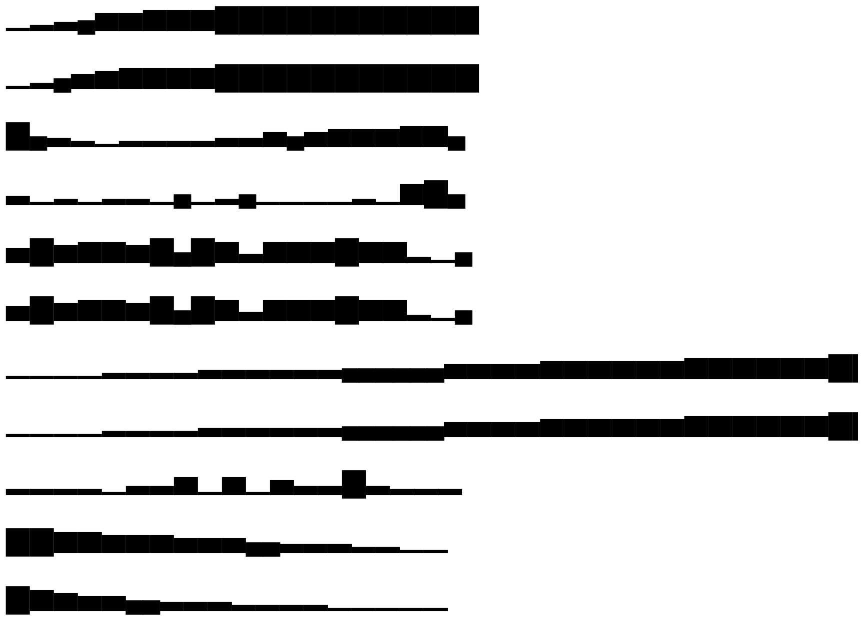
Get best checkpoint

In [75]:
```
# After training, let us check the best checkpoint
# We need this for Inference
best_model_checkpoint_step = trainer.state.best_model_checkpoint.split
('-')[-1]
print(f"The best model was saved at step {best_model_checkpoint_ste
p}.")
```

```
The best model was saved at step 650.
```

In [76]:
```
wandb.finish()
```

Run history:



```
eval/accuracy
eval/f1
eval/loss
eval/runtime
eval/samples_per_second
eval/steps_per_second
train/epoch
train/global_step
train/grad_norm
train/learning_rate
train/loss
```

Run summary:

```
eval/accuracy              0.8246
eval/f1                    0.4736
eval/loss                  0.45692
eval/runtime               6.4902
eval/samples_per_second    238.051
eval/steps_per_second      2.003
total_flos                 51918688180800.0
train/epoch                20.0
train/global_step          980
train/grad_norm            0.22986
train/learning_rate        3e-05
train/loss                 0.2126
train_loss                 0.30444
train_runtime              142.9704
train_samples_per_second   864.374
train_steps_per_second     6.855
```

View run **tweet_hf_trainer** at: https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/z5y5es7b (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/z5y5es7b)
View project at: https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm)
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `./wandb/run-20240405_231255-z5y5es7b/logs`

Performance on Test Set

In [77]:
```
# Define the path to the best model checkpoint
# 'model_checkpoint' variable is constructed using the model folder pa
th and the checkpoint step
# This step is identified as having the best model performance during
training
model_checkpoint = model_folder/f'checkpoint-{best_model_checkpoint_st
ep}'
```

In [78]:
```
# Instantiate the CustomMLP model with predefined configurations
# 'my_config' is an instance of the CustomConfig class, containing spe
cific model settings like
# vocabulary size, embedding dimensions, etc.
model = CustomLSTM(my_config)
```

In [79]:
```
model
```

Out[79]:
```
CustomLSTM(
  (embedding_bag): EmbeddingBag(10344, 256, mode='mean')
  (lstm): LSTM(256, 512, batch_first=True)
  (layers): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

In [80]:
```
# Load the pre-trained weights into the CustomMLP model from the speci
fied checkpoint
# 'model_checkpoint' refers to the path where the model's best-perform
ing state is saved
# This step ensures the model is initialized with weights from its mos
t effective training state
model = model.from_pretrained(model_checkpoint, config = my_config)
```

```
Some weights of CustomLSTM were not initialized from the model checkpo
int at /content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Natura
l_Language_Processing/0_Data_Folder/checkpoint-650 and are newly initi
alized: ['lstm.bias_hh_l0', 'lstm.bias_ih_l0', 'lstm.weight_hh_l0']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

In [81]:
```
model
```

Out[81]:
```
CustomLSTM(
  (embedding_bag): EmbeddingBag(10344, 256, mode='mean')
  (lstm): LSTM(256, 512, batch_first=True)
  (layers): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

In [82]:
```
# Create a partial function 'collate_fn' using 'collate_batch' with 'm
y_vocab' set to 'imdb_vocab'
# This function will be used by the Trainer to process batches of data
during evaluation
collate_fn = partial(collate_batch, my_vocab=tweet_vocab)

# Configure training arguments for model evaluation
# 'output_dir' specifies where to save the results
# 'per_device_eval_batch_size' sets the batch size for evaluation, adj
usted based on available GPU memory
# 'do_train = False' and 'do_eval=True' indicate that training is not
performed, but evaluation is
# 'remove_unused_columns=False' ensures that all columns in the datase
t are retained during evaluation
# 'report_to=[]' disables logging to external services like Weights &
Biases

training_args = TrainingArguments(
    output_dir="./results",
    per_device_eval_batch_size=16,
    do_train=False,
    do_eval=True,
    remove_unused_columns=False,
    report_to=[]
)
```

In [83]:
```
# Initialize the Trainer with the specified model and training argumen
ts
# 'model' is the CustomMLP model loaded with pre-trained weights
# 'training_args' contains the configurations for evaluation, includin
g batch sizes and output directory
# 'eval_dataset' is set to 'testset', which is the dataset used for ev
aluating the model
# 'data_collator' is assigned 'collate_fn', the function for processin
g batches of data
# 'compute_metrics' is a function that calculates evaluation metrics l
ike accuracy and F1 score

trainer = Trainer(
    model=model,
    args=training_args,
    eval_dataset=testset,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
)
```

```
/usr/local/lib/python3.10/dist-packages/accelerate/accelerator.py:436:
FutureWarning: Passing the following arguments to `Accelerator` is dep
recated and will be removed in version 1.0 of Accelerate: dict_keys
(['dispatch_batches', 'split_batches', 'even_batches', 'use_seedable_s
ampler']). Please pass an `accelerate.DataLoaderConfiguration` instea
d:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, spl
it_batches=False, even_batches=True, use_seedable_sampler=True)
  warnings.warn(
```

In [84]:
```
trainer.evaluate()
```

[204/204 00:58]

Out[84]:
```
{'eval_loss': 1.4013041257858276,
 'eval_accuracy': 0.8668582108287539,
 'eval_f1': 0.0,
 'eval_runtime': 60.4449,
 'eval_samples_per_second': 53.917,
 'eval_steps_per_second': 3.375}
```

In [85]:
```
test_predictions = trainer.predict(testset)
```

In [86]:
```
test_predictions._fields
```

Out[86]:
```
('predictions', 'label_ids', 'metrics')
```

```
In [87]: test_predictions.metrics
```

```
Out[87]: {'test_loss': 1.4013041257858276,
          'test_accuracy': 0.8668582108287539,
          'test_f1': 0.0,
          'test_runtime': 63.6613,
          'test_samples_per_second': 51.193,
          'test_steps_per_second': 3.204}
```

```
In [92]: test_predictions.label_ids
```

```
Out[92]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
In [91]: test_preds = np.argmax(test_predictions.predictions, axis=1)
         test_labels = np.array(test_predictions.label_ids)
```

Model Inference

Model inference is the stage in the machine learning process where a trained model is used to make predictions on new, unseen data. Unlike the training or evaluation phases, labels are not required at this stage, as the primary goal is to apply the model's learned patterns and knowledge to generate predictions.

```
In [93]: testset
```

```
Out[93]: Dataset({
             features: ['texts', 'labels'],
             num_rows: 3259
         })
```

```
In [94]: sample_X = testset['texts']
```

Step 1. Preprocessing

```
In [95]: device = 'cpu'
         # Convert the list of texts into a list of lists; each inner list cont
         ains the vocabulary indices for a text
         list_of_list_of_indices = [tokenizer(text, tweet_vocab) for text in sa
         mple_X]

         # Compute the offsets for each text in the concatenated tensor
         offsets = [0] + [len(i) for i in list_of_list_of_indices]
         offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)

         # Concatenate all text indices into a single tensor
         indices = torch.cat([torch.tensor(i, dtype=torch.int64) for i in list_
         of_list_of_indices])
```

Step 2: Get Predictions

```
In [96]: # move model to appropriate device
         model.to(device)

         # put model in evaluation mode
         model.eval()

         # get outputs (logits) from model
         outputs = model(indices, offsets)
         outputs
```

```
Out[96]: SequenceClassifierOutput(loss=None, logits=tensor([[ -0.1290, -12.750
         4, -10.1923,  ..., -33.7357, -22.6788, -27.0854],
                 [  2.9891, -19.6176,  -9.1088,  ..., -37.8043, -30.5580, -36.7
         081],
                 [  1.8495, -20.7513, -12.8665,  ..., -43.2483, -32.6759, -38.2
         814],
                 ...,
                 [ -2.6846, -24.7797, -18.1971,  ..., -45.8147, -39.0044, -44.8
         310],
                 [ -3.8930, -25.0800, -18.7555,  ..., -43.6822, -37.4418, -42.2
         962],
                 [ -2.5615, -25.2132, -17.8407,  ..., -45.1339, -37.5399, -44.5
         249]],
                grad_fn=<AddmmBackward0>), hidden_states=None, attentions=None)
```

```
In [97]: outputs.logits
```

```
Out[97]: tensor([[ -0.1290, -12.7504, -10.1923,  ..., -33.7357, -22.6788, -27.0
854],
        [  2.9891, -19.6176,  -9.1088,  ..., -37.8043, -30.5580, -36.7
081],
        [  1.8495, -20.7513, -12.8665,  ..., -43.2483, -32.6759, -38.2
814],
        ...,
        [ -2.6846, -24.7797, -18.1971,  ..., -45.8147, -39.0044, -44.8
310],
        [ -3.8930, -25.0800, -18.7555,  ..., -43.6822, -37.4418, -42.2
962],
        [ -2.5615, -25.2132, -17.8407,  ..., -45.1339, -37.5399, -44.5
249]],
       grad_fn=<AddmmBackward0>)
```

Step 3: Post Processing

```
In [106]: predictions = outputs.logits >= 0
          predictions = predictions.numpy()
          original_labels = {column for id, column in enumerate(label_columns)}
```

```
In [107]: original_labels
```

```
Out[107]: {'anger',
           'anticipation',
           'disgust',
           'fear',
           'joy',
           'love',
           'optimism',
           'pessimism',
           'sadness',
           'surprise',
           'trust'}
```

```
In [111]: predictions
```

```
Out[111]: array([[False, False, False, ..., False, False, False],
                 [ True, False, False, ..., False, False, False],
                 [ True, False, False, ..., False, False, False],
                 ...,
                 [False, False, False, ..., False, False, False],
                 [False, False, False, ..., False, False, False],
                 [False, False, False, ..., False, False, False]])
```

```
In [109]: mapped_labels = []
          for prediction_row in predictions:
            mapped_row = [label for label, pred in zip(original_labels, predicti
          on_row) if pred]
            mapped_labels.append(mapped_row)
```

```
In [115]: mapped_labels[:5]
```

```
Out[115]: [['anger'],
           ['anticipation', 'anger'],
           ['anticipation', 'anger'],
           ['anticipation', 'anger'],
           ['anticipation', 'anger']]
```

```
In [113]: submission = pd.read_csv('/content/sample_submission.csv')
```

```
In [116]: submission.head()
```

Out[116]:

	ID	anger	anticipation	disgust	fear	joy	love	optimism	pessimism	sadness	surprise
0	2018-01559	0	0	0	0	0	0	0	0	0	0
1	2018-03739	0	0	0	0	0	0	0	0	0	0
2	2018-00385	0	0	0	0	0	0	0	0	0	0
3	2018-03001	0	0	0	0	0	0	0	0	0	0
4	2018-01988	0	0	0	0	0	0	0	0	0	0

```
In [121]: submission.columns
```

```
Out[121]: Index(['ID', 'anger', 'anticipation', 'disgust', 'fear', 'joy', 'lov
          e',
                 'optimism', 'pessimism', 'sadness', 'surprise', 'trust'],
                dtype='object')
```

```
In [118]: predictions_num = predictions.astype(int)
```

```
In [119]: predictions_num[:5]
```

```
Out[119]: array([[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]])
```

In [120]:
```python
pd.DataFrame(predictions_num,columns=label_columns)
```

Out[120]:

	anger	anticipation	disgust	fear	joy	love	optimism	pessimism	sadness	surprise	trus
0	0	0	0	0	1	0	0	0	0	0	(
1	1	0	0	0	1	0	0	0	0	0	(
2	1	0	0	0	1	0	0	0	0	0	(
3	1	0	0	0	1	0	0	0	0	0	(
4	1	0	0	0	1	0	0	0	0	0	(
...	...	...	...	...	...	...	...	...	...	...	..
3254	0	0	0	0	1	0	0	0	0	0	(
3255	0	0	0	0	1	0	0	0	0	0	(
3256	0	0	0	0	1	0	0	0	0	0	(
3257	0	0	0	0	1	0	0	0	0	0	(
3258	0	0	0	0	1	0	0	0	0	0	(

3259 rows × 11 columns

In [122]:
```python
submission[['anger', 'anticipation', 'disgust', 'fear', 'joy', 'love',
            'optimism', 'pessimism', 'sadness', 'surprise', 'trust']]
= predictions_num
```

In [123]:
```python
submission.head()
```

Out[123]:

	ID	anger	anticipation	disgust	fear	joy	love	optimism	pessimism	sadness	surprise
0	2018-01559	0	0	0	0	1	0	0	0	0	0
1	2018-03739	1	0	0	0	1	0	0	0	0	0
2	2018-00385	1	0	0	0	1	0	0	0	0	0
3	2018-03001	1	0	0	0	1	0	0	0	0	0
4	2018-01988	1	0	0	0	1	0	0	0	0	0

In [127]:
```python
submission.to_csv(model_folder/'lstm.csv', index = False)
```

In [128]:
```python
from kaggle import api
comp = 'emotion-detection-spring2014'
api.competition_submit(model_folder/'lstm.csv', 'lstm apr6', comp)
```

```
Warning: Looks like you're using an outdated API Version, please consi
der updating (server 1.6.7 / client 1.5.16)

100%|██████████| 105k/105k [00:00<00:00, 124kB/s]
```

Out[128]:
```
Successfully submitted to Emotion Detection Spring2024
```