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 numerical operations
import random
import numpy as np

# Utilities for efficient serialization/deserialization of Python objects 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 matrices, 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_Notebooks/BUAN_6382_Applied_DeepLearning/Data/.kaggle'
else:
    os.environ['KAGGLE_CONFIG_DIR']='/Users/harikrishnadev/Library/CloudStorage/GoogleDrive-harikrish0607@gmail.com/My Drive/Colab_Notebooks/BUAN_6382_Applied_DeepLearning/Data/.kaggle'
```

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

```
emotion-detection-spring2014.zip: Skipping, found more recently modified local copy (use --force to force download)
```

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

```
Archive:  emotion-detection-spring2014.zip
replace sample_submission.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

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 | None=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', 'love',
       'optimism', 'pessimism', 'sadness', 'surprise', 'trust'],
      dtype='object')
```

In [13]:
```
label_columns = ['anger', 'anticipation', 'disgust', 'fear', 'joy', 'love', '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]:
```python
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 [33]:
```
model = CustomLSTM(config=my_config)
```

In [34]:
```
model
```

Out[34]:
```
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 [35]:
```
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 [36]:
```
# 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 checkpoints
    evaluation_strategy='steps',  # Evaluate model at specified step intervals
    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 of training
    save_total_limit=2,  # Retain only the best and the most recent model 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 & Biases
)
```

Initialize Trainer

In [37]:
```
trainset[0]
```

Out[37]:
```
{'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 [38]:
```
# 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 [39]:
```
train_set
```

Out[39]:
```
DatasetDict({
    train: Dataset({
        features: ['texts', 'labels'],
        num_rows: 6179
    })
    test: Dataset({
        features: ['texts', 'labels'],
        num_rows: 1545
    })
})
```

In [40]:
```
[sample['labels'] for sample in train_set['train']][:5]
```

Out[40]:
```
[[0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]]
```

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

Out[41]:
```
['@RossKemp Omg that is just horrific. Something needs to be done.
😢 ',
 "@lemonlover666 'shit' doesn't even begin to describe these fiery lit
tle demons straight from hell 🌝🌚  ;)",
 "So is texting a guy 'I'm ready for sex now' considered flirting?' ",
 '@ErinAndrews I ❤you on DWTS You make my night every show! 😘  #hilari
ous',
 '@jonnyp_43 @MedicNow like going to a so called cardiac arrest that t
urned out to be a cut finger!  #medchat']
```

In [42]:
```
label_mt = train_set['train']['labels']
# label_mt
```

In [43]:
```
type(label_mt)
```
Out[43]:
```
list
```

In [44]:
```
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 [45]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import userdata
    wandb.login(key=userdata.get('wandb'))
else:
    !wandb login
```

```
wandb: Currently logged in as: harikrish0607 (harikrishnad). Use `wand
b 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 [46]:
```
# 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 [47]:
```
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).

Tracking run with wandb version 0.16.6

Run data is saved locally in `/content/wandb/run-20240406_031838-9gnpp0dj`

Syncing run **tweet_hf_trainer (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/9gnpp0dj)** 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/9gnpp0dj (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/9gnpp0dj)

[980/980 02:07, Epoch 20/20]

Step	Training Loss	Validation Loss	Accuracy	F1
50	0.525800	0.476494	0.785702	0.011937
100	0.453900	0.452734	0.792468	0.156422
150	0.424200	0.442214	0.799588	0.222374
200	0.393800	0.447006	0.804472	0.320867
250	0.367800	0.434952	0.811062	0.334784
300	0.345000	0.430266	0.814357	0.365829
350	0.325500	0.435075	0.815769	0.396143
400	0.307200	0.438178	0.818594	0.419288
450	0.294700	0.442478	0.819947	0.421987
500	0.281000	0.451399	0.819653	0.427638
550	0.268700	0.456868	0.817240	0.442970
600	0.257200	0.466666	0.816652	0.439770
650	0.249000	0.460222	0.818770	0.451176
700	0.239900	0.472729	0.818182	0.459223
750	0.233600	0.475820	0.817652	0.457173
800	0.226000	0.482135	0.817240	0.463001
850	0.220800	0.486564	0.818241	0.464180
900	0.217000	0.488756	0.817123	0.461165
950	0.213200	0.490178	0.817240	0.462071

```
Removed shared tensor {'lstm.weight_hh_l0', 'lstm.bias_hh_l0', 'lstm.b
ias_ih_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[47]:
```
TrainOutput(global_step=980, training_loss=0.3045059525236791, metrics
={'train_runtime': 130.0129, 'train_samples_per_second': 950.521, 'tra
in_steps_per_second': 7.538, 'total_flos': 51986699360400.0, 'train_lo
ss': 0.3045059525236791, 'epoch': 20.0})
```

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

[13/13 00:03]

Out[48]:
```
{'eval_loss': 0.4371524751186371,
 'eval_accuracy': 0.8216534274786702,
 'eval_f1': 0.42101241642788917,
 'eval_runtime': 5.3227,
 'eval_samples_per_second': 290.267,
 'eval_steps_per_second': 2.442,
 'epoch': 20.0}
```

In [49]:
```
test_dataset = pd.read_csv('test.csv', usecols=lambda column: column !
= 'ID')
```

In [50]:
```
testset = Dataset.from_dict({
    'texts': test_dataset['Tweet'].to_list(),
    'labels': [[0] * 11] * len(test_dataset),  # Exclude 'Tweet' colum
n
})
```

In [51]:
```
testset[0]
```

Out[51]:
```
{'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 [58]:
```
valid_output = trainer.predict(trainset)
```

In [59]:
```
valid_output._fields
```

Out[59]:
```
('predictions', 'label_ids', 'metrics')
```

In [60]:
```
valid_output
```

Out[60]:
```
PredictionOutput(predictions=array([[-5.3966823 , -2.149156  , -4.4205
43  , ..., -2.407467  ,
        -3.5893316 , -2.7319257 ],
       [-3.5293622 , -2.0207589 , -3.6790767 , ..., -3.1571753 ,
        -4.085616  , -1.7995695 ],
       [ 1.3206013 , -1.4652557 ,  0.58503056, ..., -1.6515752 ,
        -4.043876  , -3.6812434 ],
       ...,
       [ 0.26989844, -2.2456157 ,  0.48509836, ..., -0.82595587,
        -2.494619  , -3.714057  ],
       [-2.3613973 , -1.2833263 , -1.7518766 , ..., -2.0807812 ,
        -2.474759  , -3.2791905 ],
       [-3.5882635 , -1.2984607 , -3.2742472 , ..., -2.6869671 ,
        -3.6157537 , -2.6514888 ]], dtype=float32), label_ids=array
([[0., 1., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.],
       ...,
       [1., 0., 1., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32), metrics={'test_
loss': 0.3018510639667511, 'test_accuracy': 0.8626594793088838, 'test_
f1': 0.570186747209842, 'test_runtime': 20.4263, 'test_samples_per_sec
ond': 378.139, 'test_steps_per_second': 2.986})
```

In [61]:
```
valid_output.metrics
```

Out[61]:
```
{'test_loss': 0.3018510639667511,
 'test_accuracy': 0.8626594793088838,
 'test_f1': 0.570186747209842,
 'test_runtime': 20.4263,
 'test_samples_per_second': 378.139,
 'test_steps_per_second': 2.986}
```

In [62]:
```
valid_preds = np.argmax(valid_output.predictions, axis=-1)
valid_labels = np.array(valid_output.label_ids)
```

Get best checkpoint

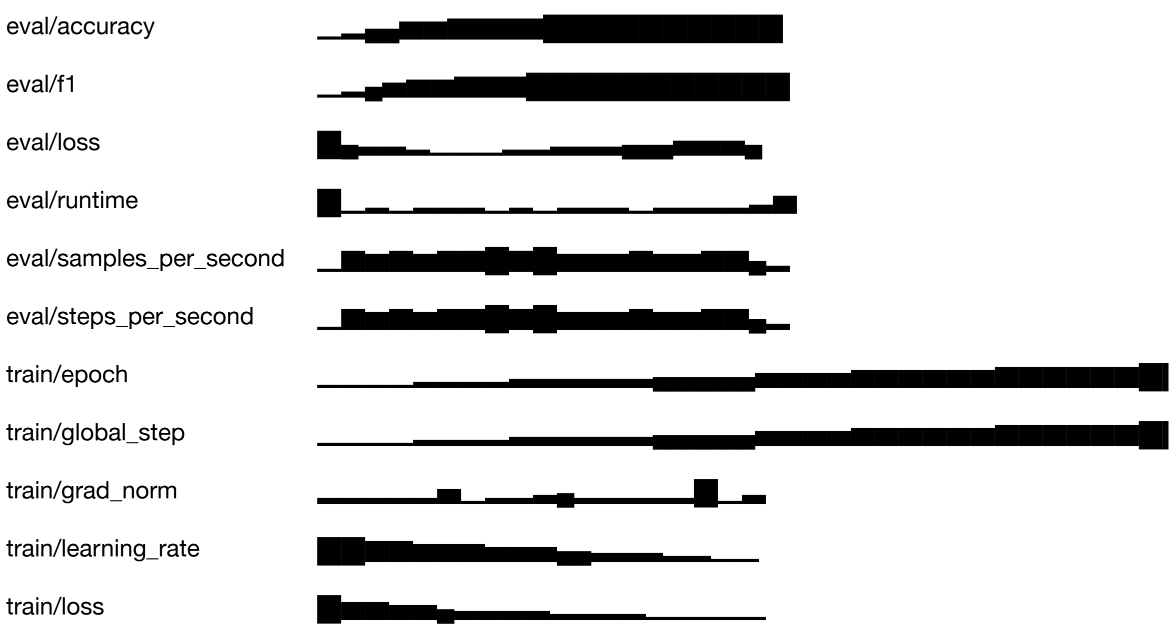
In [63]:
```
# After training, let us check the best checkpoint
# We need this for Inference
best_model_checkpoint_step = trainer.state.best_model_checkpoint.split
('-')[-1]
print(f"The best model was saved at step {best_model_checkpoint_ste
p}.")
```

```
The best model was saved at step 450.
```

In [ ]:

```
wandb.finish()
```

Run history:



Run summary:

eval/accuracy	0.82883
eval/f1	0.49152
eval/loss	0.44813
eval/runtime	8.6445
eval/samples_per_second	178.727
eval/steps_per_second	1.504
total_flos	51678033237600.0
train/epoch	20.0
train/global_step	980
train/grad_norm	0.25436
train/learning_rate	3e-05
train/loss	0.2117
train_loss	0.30485
train_runtime	140.1115
train_samples_per_second	882.012
train_steps_per_second	6.994

View run **tweet_hf_trainer** at: https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/lpcmu83b (https://wandb.ai/harikrishnad/nlp_course_spring_2024-emotion-analysis-hf-trainer-lstm/runs/lpcmu83b)
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-20240406_024840-lpcmu83b/logs`

Performance on Test Set

In [64]:
```
# 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 [ ]:
```
# 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 [ ]:
```
model
```

Out[ ]:
```
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 [ ]:
```
# 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-800 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 [ ]:
```
model
```

Out[ ]:
```
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 [65]:
```
# 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 [67]:
```
# 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,
)
```

In [68]:
```
trainer.evaluate()
```

[204/204 00:51]

Out[68]:
```
{'eval_loss': 0.32463908195495605,
 'eval_accuracy': 0.9155346034756897,
 'eval_f1': 0.0,
 'eval_runtime': 53.9586,
 'eval_samples_per_second': 60.398,
 'eval_steps_per_second': 3.781}
```

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 [73]:
```
testset
```

Out[73]:
```
Dataset({
    features: ['texts', 'labels'],
    num_rows: 3259
})
```

In [74]:
```
sample_X = testset['texts']
```

Step 1. Preprocessing

In [75]:
```
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 [76]:
```
offsets
```

Out[76]:
```
tensor([    0,    16,    37,  ..., 51755, 51764, 51780])
```

In [77]:
```
# 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[77]:
```
SequenceClassifierOutput(loss=None, logits=tensor([[-0.4919, -0.6693,
-0.5796,  ..., -2.7760, -3.1979, -3.8497],
        [-1.4947, -1.1219, -1.0892,  ..., -2.2226, -3.2427, -3.2990],
        [-1.8687, -2.3732, -2.0220,  ..., -1.1419, -3.6685, -4.0010],
        ...,
        [ 2.8092, -2.5832,  2.1831,  ..., -2.8296, -4.5090, -6.6450],
        [-2.4587, -1.3115, -2.5440,  ..., -3.5842, -3.3005, -2.5384],
        [ 1.0494, -2.7709,  0.9989,  ..., -2.7948, -4.4848, -5.7047]],
       grad_fn=<AddmmBackward0>), hidden_states=None, attentions=None)
```

In [78]:
```
outputs.logits
```

Out[78]:
```
tensor([[-0.4919, -0.6693, -0.5796,  ..., -2.7760, -3.1979, -3.8497],
        [-1.4947, -1.1219, -1.0892,  ..., -2.2226, -3.2427, -3.2990],
        [-1.8687, -2.3732, -2.0220,  ..., -1.1419, -3.6685, -4.0010],
        ...,
        [ 2.8092, -2.5832,  2.1831,  ..., -2.8296, -4.5090, -6.6450],
        [-2.4587, -1.3115, -2.5440,  ..., -3.5842, -3.3005, -2.5384],
        [ 1.0494, -2.7709,  0.9989,  ..., -2.7948, -4.4848, -5.7047]],
       grad_fn=<AddmmBackward0>)
```

Step 3: Post Processing

In [80]:
```python
predictions = torch.abs(outputs.logits)
predictions = predictions.detach().numpy()
original_labels = {column for id, column in enumerate(label_columns)}
```

In [98]:
```python
sigmoid_result = 1 / (1 + np.exp(-predictions))
```

In [101]:
```python
sigmoid_result[:5]
```

Out[101]:
```
array([[0.6205552 , 0.66134536, 0.64097303, 0.8530372 , 0.82767195,
        0.99310243, 0.8130186 , 0.9767854 , 0.9413638 , 0.9607566 ,
        0.97915685],
       [0.8167762 , 0.7543421 , 0.74822634, 0.87020504, 0.6384884 ,
        0.96375287, 0.8003727 , 0.9666098 , 0.90225774, 0.96240944,
        0.9643957 ],
       [0.8663077 , 0.9147643 , 0.8830894 , 0.82890666, 0.7103004 ,
        0.95478606, 0.60899615, 0.9758605 , 0.7580216 , 0.9751212 ,
        0.98203063],
       [0.7951113 , 0.8347093 , 0.67801625, 0.5929609 , 0.7056416 ,
        0.95062083, 0.6542946 , 0.84180367, 0.63851005, 0.915472  ,
        0.9544312 ],
       [0.6956899 , 0.9716575 , 0.57856745, 0.59916157, 0.9867883 ,
        0.99950933, 0.9195833 , 0.86709803, 0.74857056, 0.99371237,
        0.9983854 ]], dtype=float32)
```

In [129]:
```python
threshold = 0.6
predictions_labels = (sigmoid_result > threshold)
```

In [130]:
```python
predictions_labels.astype(int)[:5]
```

Out[130]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1]])
```

In [131]:
```python
submission = pd.read_csv('/content/sample_submission.csv')
```

In [132]:
```python
submission.head()
```

Out[132]:

	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 [133]:
```python
submission.columns
```

Out[133]:
```
Index(['ID', 'anger', 'anticipation', 'disgust', 'fear', 'joy', 'lov
e',
       'optimism', 'pessimism', 'sadness', 'surprise', 'trust'],
      dtype='object')
```

In [134]:
```python
predictions_num = predictions_labels.astype(int)
```

In [135]:
```python
predictions_num[:5]
```

Out[135]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1]])
```

In [136]:
```python
submission[['anger', 'anticipation', 'disgust', 'fear', 'joy', 'love',
            'optimism', 'pessimism', 'sadness', 'surprise', 'trust']]
= predictions_num
```

In [137]:
```python
submission.head()
```

Out[137]:

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

In [138]:
```python
submission.to_csv(model_folder/'lstm.csv', index = False)
```

In [139]:
```python
from kaggle import api
comp = 'emotion-detection-spring2014'
api.competition_submit(model_folder/'lstm.csv', 'lstm apr6 fixed', com
p)
```

```
100%|██████████| 105k/105k [00:01<00:00, 55.3kB/s]
```

Out[139]:
```
Successfully submitted to Emotion Detection Spring2024
```