1. Set Environment

```
# 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')

  # Install the latest version of torchtext library quietly without
showing output
  !pip install torchtext -qq
  !pip install transformers evaluate wandb datasets accelerate -U -qq
## NEW LINES ##
  basepath = '/content/drive/MyDrive/data/'
else:
  basepath = '/home/harpreet/Insync/google_drive_shaannorr/data'


Mounted at /content/drive
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 56.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 65.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 89.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 2.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.6/121.6 MB 8.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 MB 14.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.2/124.2 MB 8.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.0/196.0 MB 6.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.0/166.0 MB 7.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 14.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 54.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.8/8.8 MB 31.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 12.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 54.9 MB/s eta
0:00:00
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 44.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.3/297.3 kB 32.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 15.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 20.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 18.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 24.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 266.1/266.1 kB 2.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 8.3 MB/s eta
0:00:00
```

```python
# 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
```

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

# wandb library
import wandb
```

Specify Project Folders

```
base_folder = Path(basepath)
data_folder = base_folder / 'datasets/my_project'
model_folder = base_folder / 'models/nlp_spring_2024/my_project'

model_folder.mkdir(exist_ok=True, parents=True)
data_folder.mkdir(exist_ok=True, parents=True)
```

2. Load Data

```
import pandas as pd

# Load train and test data
train_data =
pd.read_csv("/content/drive/MyDrive/data/datasets/train.csv")
test_data =
pd.read_csv("/content/drive/MyDrive/data/datasets/test.csv")
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7724 entries, 0 to 7723
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ID            7724 non-null   object
 1   Tweet         7724 non-null   object
 2   anger         7724 non-null   int64
 3   anticipation  7724 non-null   int64
 4   disgust       7724 non-null   int64
 5   fear          7724 non-null   int64
 6   joy           7724 non-null   int64
 7   love          7724 non-null   int64
 8   optimism      7724 non-null   int64
 9   pessimism     7724 non-null   int64
 10  sadness       7724 non-null   int64
 11  surprise      7724 non-null   int64
 12  trust         7724 non-null   int64
```

```
dtypes: int64(11), object(2)
memory usage: 784.6+ KB
```

```
test_data.head()
```


```
{"summary":"{\n  \"name\": \"test_data\",\n  \"rows\": 3259,\n  \"fields\": [\n    {\n      \"column\": \"ID\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 3259,\n        \"samples\": [\n          \"2018-00951\",\n          \"2018-04124\",\n          \"2018-04333\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Tweet\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 3259,\n        \"samples\": [\n          \"When you're on a diet and your whole family orders pizza. \",\n          \"@lukeWHC09 Hahaaa! Was fuming with that \\ud83d\\ude1e\\ud83d\\ude02\",\n          \"@CaxtonSupport I'm sorry but I don't know what DM stands for. I know a PM \\ud83d\\ude02. But sadly not a DM. It won't create an account to find.\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"anger\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"anticipation\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"disgust\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"fear\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"joy\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"love\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"optimism\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\": [\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"pessimism\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
```

```
[\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sadness\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"surprise\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"NONE\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trust\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 1,\n        \"samples\": [\n
\"NONE\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"test_data"}
```


3. Create Huggingface Dataset

```
y_train = train_data[['anger', 'anticipation', 'disgust', 'fear',
'joy', 'love','optimism', 'pessimism', 'sadness', 'surprise',
'trust']].to_numpy()
X_train = train_data.drop(columns = ['anger', 'anticipation',
'disgust', 'fear', 'joy', 'love','optimism', 'pessimism', 'sadness',
'surprise', 'trust'])
print(X_train.head())
print(y_train)
```

```
           ID                                              Tweet
0  2017-21441  “Worry is a down payment on a problem you may ...
1  2017-31535  Whatever you decide to do make sure it makes y...
2  2017-21068  @Max_Kellerman  it also helps that the majorit...
3  2017-31436  Accept the challenges so that you can literall...
4  2017-22195  My roommate: it's okay that we can't spell bec...
[[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]]
```

```
X_train[:5]
```


```
{"summary":"{\n  \"name\": \"X_train[:5]\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"ID\",\n      \"properties\":
{\n        \"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"2017-31535\",\n          \"2017-22195\",\n
\"2017-21068\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tweet\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
```

```
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Whatever you decide to do make sure it makes you #happy.\",\n
\"My roommate: it's okay that we can't spell because we have
autocorrect. #terrible #firstworldprobs\",\n          \"@Max_Kellerman
it also helps that the majority of NFL coaching is inept. Some of Bill
O'Brien's play calling was wow, ! #GOPATS\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
X_train_list=list(X_train['Tweet'])
X_train_list[:5]
```

```
["“Worry is a down payment on a problem you may never have'. \xa0Joyce
Meyer.  #motivation #leadership #worry",
 'Whatever you decide to do make sure it makes you #happy.',
 "@Max_Kellerman  it also helps that the majority of NFL coaching is
inept. Some of Bill O'Brien's play calling was wow, ! #GOPATS",
 "Accept the challenges so that you can literally even feel the
exhilaration of victory.' -- George S. Patton 🐶",
 "My roommate: it's okay that we can't spell because we have
autocorrect. #terrible #firstworldprobs"]
```

```
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid = train_test_split(X_train_list,
y_train, test_size = 0.3, random_state = 1)
```

```
trainset = Dataset.from_dict({
    'texts': X_train,
    'labels': y_train
})
```

```
validset = Dataset.from_dict({
    'texts': X_valid,
    'labels': y_valid
})
print(y_train.shape), print(y_valid.shape)
```

```
(5406, 11)
(2318, 11)
```

```
(None, None)
```

```
trainset[:5]
```

```
{'texts': ['@JuliaHB1 Bloody right #fume',
  'You boys dint know the game am I the game... life after death...
better chose and know who side you on before my wrath does come upon
us😤😤😤',
  'Peter is aesthetically pleasing to look at',
  'The weather changed from sunny and bright to gloomy just in time to
match my afternoon mood. 😒',
```

```
  "#TerrorStatePak we r confirm that #navazsharif is post graduate
distinction student of university of #terrorism. He can't spare
himself."],
 'labels': [[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
  [1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0],
  [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
  [0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
  [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]]}
```

4. Create Custom Model and Model Config Class

```
from transformers import PretrainedConfig, PreTrainedModel
import torch
import torch.nn as nn
from transformers.modeling_outputs import SequenceClassifierOutput

class CustomConfig(PretrainedConfig):
    def __init__(self, vocab_size=0, embedding_dim=0, hidden_dim1=0,
hidden_dim2=0, 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

class CustomMLP(PreTrainedModel):
    config_class = CustomConfig

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

        self.embedding_bag = nn.EmbeddingBag(config.vocab_size,
config.embedding_dim)
        self.layers = nn.Sequential(
            nn.Linear(config.embedding_dim, config.hidden_dim1),
            nn.BatchNorm1d(num_features=config.hidden_dim1),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            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)
        )

    def forward(self, input_ids, offsets, labels=None):
        embed_out = self.embedding_bag(input_ids, offsets)
        logits = self.layers(embed_out)
```

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

        return SequenceClassifierOutput(
            loss=loss,
            logits=logits
        )

from torchtext.vocab import vocab
from collections import Counter
from transformers import PretrainedConfig, PreTrainedModel
import torch
import torch.nn as nn
from functools import partial
```

5. Train Model

5.1 Collate Function

```
from torchtext.vocab import vocab
from collections import Counter
from transformers import PretrainedConfig, PreTrainedModel
import torch
import torch.nn as nn
from functools import partial

def get_vocab(dataset, min_freq=1):

    # 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['texts']:
        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
```

```
def tokenizer(text, vocab):
    """Converts text to a list of indices using a vocabulary
dictionary"""
    return [vocab[token] for token in str(text).split()]

def collate_batch(batch, my_vocab):

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

    # Convert the list of labels into a tensor of dtype int32
    labels = torch.tensor(labels, dtype=torch.float64)

    # 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
texts]

    # Concatenate all text indices into a single tensor
    input_ids = torch.cat([torch.tensor(i, dtype=torch.int32) 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)
    # print('Labels:',labels.shape)
    # print('Inputs:',input_ids.shape)
    return {
        'input_ids': input_ids,
        'offsets': offsets,
        'labels': labels
    }

my_vocab = get_vocab(trainset, min_freq=2)
collate_fn = partial(collate_batch, my_vocab=my_vocab)
print(len(my_vocab))
```

```
7036
```

5.2. Instantiate Model

```
my_config = CustomConfig(vocab_size=len(my_vocab),
                         embedding_dim=300,
                         hidden_dim1=200,
                         hidden_dim2=100,
                         num_labels=11)

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' }# 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()}
my_config
```

```
CustomConfig {
  "embedding_dim": 300,
  "hidden_dim1": 200,
  "hidden_dim2": 100,
  "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": 7036
}
```

```
model = CustomMLP(config=my_config)
model
```

```
CustomMLP(
  (embedding_bag): EmbeddingBag(7036, 300, mode='mean')
  (layers): Sequential(
    (0): Linear(in_features=300, out_features=200, bias=True)
    (1): BatchNorm1d(200, 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=200, out_features=100, bias=True)
```

```
    (5): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=100, out_features=11, bias=True)
  )
)
```

5.3. compute_metrics function

```
def compute_metrics(eval_pred):
    combined_metrics = evaluate.combine([evaluate.load("accuracy"),
                                         evaluate.load("f1",
average="macro")])

    logits, labels = eval_pred
    predictions = (logits >= 0).astype(float)
    predictions = predictions.reshape(-1)
    labels = labels.reshape(-1)
    evaluations = combined_metrics.compute(
        predictions=predictions, references=labels,)
    return evaluations
```

5.4. Training Arguments

```
# Configure training parameters
training_args = TrainingArguments(

    # Training-specific configurations
    num_train_epochs=5,
    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='tweets_hf_trainer',  # Experiment name for Weights &
Biases
)
```

5.5. Initialize Trainer

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=trainset,
    eval_dataset = validset,
    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
deprecated and will be removed in version 1.0 of Accelerate:
dict_keys(['dispatch_batches', 'split_batches', 'even_batches',
'use_seedable_sampler']). Please pass an
`accelerate.DataLoaderConfiguration` instead:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None,
split_batches=False, even_batches=True, use_seedable_sampler=True)
  warnings.warn(
```

5.5.Setup WandB

```
!wandb login
```

```
wandb: Currently logged in as: likith-gv. Use `wandb login --relogin`
to force relogin
```

```
# specify the project name where the experiment will be logged
%env WANDB_PROJECT = nlp_course_spring_2024-tweers-hf-trainer
```

```
env: WANDB_PROJECT=nlp_course_spring_2024-tweers-hf-trainer
```

5.6. Training and Validation

```
trainer.train()
```

```
<IPython.core.display.HTML object>

TrainOutput(global_step=215, training_loss=0.4507412932639898,
metrics={'train_runtime': 32.3881, 'train_samples_per_second':
834.566, 'train_steps_per_second': 6.638, 'total_flos':
5691331309950.0, 'train_loss': 0.4507412932639898, 'epoch': 5.0})
```

Evaluate model on Validation Set

```
trainer.evaluate()
```

```
<IPython.core.display.HTML object>

{'eval_loss': 0.4557138681411743,
 'eval_accuracy': 0.7990430622009569,
 'eval_f1': 0.29304635761589404,
 'eval_runtime': 7.1941,
 'eval_samples_per_second': 322.208,
 'eval_steps_per_second': 2.641,
 'epoch': 5.0}
```

```
valid_output = trainer.predict(validset)
```

```
<IPython.core.display.HTML object>
```

```
valid_output._fields
```

```
('predictions', 'label_ids', 'metrics')
```

```
valid_preds = np.argmax(valid_output.predictions, axis=-1)
valid_labels = np.array(valid_output.label_ids)
```

```
valid_output.predictions
```

```
array([[-0.8721291 , -0.63750863, -0.79234296, ..., -0.7699966 ,
        -1.7431048 , -1.8171076 ],
       [ 0.21804972, -1.8704255 ,  0.07443783, ..., -0.66786796,
        -2.7027721 , -2.314043  ],
       [-0.98549616, -1.5033733 , -1.0546671 , ..., -1.4770135 ,
        -2.4267561 , -2.042008  ],
       ...,
       [-0.79744464, -1.1138262 , -0.7581991 , ..., -0.21841574,
        -2.2992592 , -2.5593033 ],
       [-0.12223329, -1.5724753 ,  0.05216947, ...,  0.25240228,
        -2.175886  , -2.4378479 ],
       [ 0.31055814, -1.5928577 ,  0.28096542, ..., -0.5192161 ,
        -2.3861136 , -2.1210666 ]], dtype=float32)
```

```
valid_preds = (valid_output.predictions >= 0).astype(float)
print(valid_preds)
valid_labels = np.array(valid_output.label_ids)
print(valid_labels)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 1. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]]
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 1. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 1. ... 1. 0. 0.]
 [1. 0. 1. ... 0. 0. 0.]]
```

Get best checkpoint

```
# 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_step}.")
```

```
The best model was saved at step 200.
```

```
wandb.finish()
```

```
{"model_id":"a9c2b3a2acfe4043a2230dcfe251e834","version_major":2,"vers
ion_minor":0}
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

6. Performance on Test Set

Load Model from checkpoint

```
# Define the path to the best model checkpoint
# 'model_checkpoint' variable is constructed using the model folder
path 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_step}'

# Instantiate the CustomMLP model with predefined configurations
# 'my_config' is an instance of the CustomConfig class, containing
specific model settings like
# vocabulary size, embedding dimensions, etc.
model = CustomMLP(my_config)

model

CustomMLP(
  (embedding_bag): EmbeddingBag(7036, 300, mode='mean')
  (layers): Sequential(
    (0): Linear(in_features=300, out_features=200, bias=True)
    (1): BatchNorm1d(200, 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=200, out_features=100, bias=True)
    (5): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=100, out_features=11, bias=True)
  )
)

# Load the pre-trained weights into the CustomMLP model from the
specified checkpoint
# 'model_checkpoint' refers to the path where the model's best-
performing state is saved
# This step ensures the model is initialized with weights from its
most effective training state
model = model.from_pretrained(model_checkpoint, config = my_config)

model

CustomMLP(
  (embedding_bag): EmbeddingBag(7036, 300, mode='mean')
  (layers): Sequential(
    (0): Linear(in_features=300, out_features=200, bias=True)
    (1): BatchNorm1d(200, 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=200, out_features=100, bias=True)
    (5): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
```

```
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=100, out_features=11, bias=True)
  )
)
```

Load Test Data

```
import pandas as pd
from datasets import Dataset


# Load test data
test_data =
pd.read_csv("/content/drive/MyDrive/data/datasets/test.csv")

# Preprocess test data
X_test_cleaned = test_data['Tweet'].tolist()

# Replace 'NONE' with 0 in the label columns
label_columns = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
'love', 'optimism', 'pessimism', 'sadness', 'surprise', 'trust']
test_data[label_columns] = test_data[label_columns].replace('NONE',
0).to_numpy(dtype=np.int64)

# Convert label columns to int type
test_data[label_columns] = test_data[label_columns].astype(int)

# Create Dataset object for test set
testset = Dataset.from_dict({
    'texts': X_test_cleaned,
    'labels': test_data[label_columns].values.tolist()
})
```

Define Collate function and compute_metrics for Test Data

```
def collate_batch_TEST(batch, my_vocab):

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

    # Convert the list of labels into a tensor of dtype int32
    #labels = torch.tensor(labels, dtype=torch.float64)

    # 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
texts]

    # Concatenate all text indices into a single tensor
```

```
    input_ids = torch.cat([torch.tensor(i, dtype=torch.int32) 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)
    # print('Labels:',labels.shape)
    # print('Inputs:',input_ids.shape)
    return {
        'input_ids': input_ids,
        'offsets': offsets
    }

import numpy as np
from sklearn.metrics import accuracy_score, f1_score

def calculate_metrics_TEST(eval_pred):
    # Unpack logits and labels
    logits, labels = eval_pred

    # Calculate probabilities using sigmoid function
    probabilities = 1 / (1 + np.exp(-logits))

    # Convert probabilities to binary predictions using a threshold of
0.25
    predictions = (probabilities >= 0.25).astype(int)

    # Print average probability
    print(np.average(probabilities))

    # Calculate accuracy and F1 score
    accuracy = accuracy_score(labels.flatten(), predictions.flatten())
    f1 = f1_score(labels, predictions, average='macro',
zero_division=0)

    # Return metrics
    return {'accuracy': accuracy, 'f1': f1, 'predictions':
predictions}
```

Instantiate Trainer for evaluation

```
# Configure training arguments for model evaluation
collate_fn = partial(collate_batch, my_vocab=my_vocab)

training_args = TrainingArguments(
    output_dir="./results",
    per_device_eval_batch_size=16,
    do_train=False,
    do_eval=False,
    remove_unused_columns=False,
```

```
    report_to=[]
)

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

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

Evaluate model on Test Set

```
# Perform evaluation
trainer.evaluate()
```

```
<IPython.core.display.HTML object>
```

```
0.26346296
```

```
{'eval_loss': 0.32896798849105835,
 'eval_accuracy': 0.5294987307874697,
 'eval_f1': 0.0,
 'eval_predictions': array([[1, 1, 1, ..., 1, 0, 0],
        [1, 0, 1, ..., 1, 0, 0],
        [1, 1, 1, ..., 1, 0, 0],
        ...,
        [1, 1, 1, ..., 1, 0, 0],
        [0, 0, 1, ..., 0, 0, 0],
        [1, 0, 1, ..., 1, 0, 0]]),
 'eval_runtime': 50.0219,
 'eval_samples_per_second': 65.151,
 'eval_steps_per_second': 4.078}
```

```
# Perform prediction
test_predictions = trainer.predict(testset)
test_predictions
```

```
<IPython.core.display.HTML object>
```

```
0.26346296

PredictionOutput(predictions=array([[-0.46897268, -0.8100302 , -
0.46633953, ..., -0.97437865,
        -1.911838  , -2.0565841 ],
       [-0.40363795, -1.592375  , -0.5889262 , ..., -0.9800425 ,
        -2.6056476 , -2.6518428 ],
       [-0.9022042 , -1.0245067 , -0.72619665, ..., -0.6349895 ,
        -2.146284  , -2.4238257 ],
       ...,
       [-0.7538263 , -0.82958984, -0.5013249 , ..., -0.2357219 ,
        -1.8263197 , -1.1388258 ],
       [-1.106128  , -1.2753541 , -1.0308765 , ..., -1.4351181 ,
        -2.4279962 , -2.1087582 ],
       [-0.2910459 , -1.7398988 , -0.188061  , ..., -0.8629301 ,
        -2.0721703 , -1.99468   ]], dtype=float32),
label_ids=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.]]), metrics={'test_loss':
0.32896798849105835, 'test_accuracy': 0.5294987307874697, 'test_f1':
0.0, 'test_predictions': array([[1, 1, 1, ..., 1, 0, 0],
       [1, 0, 1, ..., 1, 0, 0],
       [1, 1, 1, ..., 1, 0, 0],
       ...,
       [1, 1, 1, ..., 1, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [1, 0, 1, ..., 1, 0, 0]]), 'test_runtime': 50.0568,
'test_samples_per_second': 65.106, 'test_steps_per_second': 4.075})

submission_df=trainer.evaluate()

<IPython.core.display.HTML object>

0.26346296

submission_df['eval_predictions']

array([[1, 1, 1, ..., 1, 0, 0],
       [1, 0, 1, ..., 1, 0, 0],
       [1, 1, 1, ..., 1, 0, 0],
       ...,
       [1, 1, 1, ..., 1, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [1, 0, 1, ..., 1, 0, 0]])
```

7. Export submission.csv for evaluation

```
import pandas as pd

# Load test.csv to get the first column
test_df = pd.read_csv("/content/drive/MyDrive/data/datasets/test.csv")

first_column = test_df.iloc[:, 0]

# Convert eval_predictions to a DataFrame
predictions_df = pd.DataFrame(submission_df['eval_predictions'])

# Concatenate the first column with predictions_df
result_df = pd.concat([first_column, predictions_df], axis=1)
print(result_df)
```

```
               ID  0  1  2  3  4  5  6  7  8  9  10
0     2018-01559  1  1  1  0  1  0  1  0  1  0   0
1     2018-03739  1  0  1  0  1  0  1  0  1  0   0
2     2018-00385  1  1  1  0  1  0  1  0  1  0   0
3     2018-03001  1  0  1  1  1  0  1  1  1  0   0
4     2018-01988  1  0  1  1  1  0  1  0  1  0   0
...          ... .. .. .. .. .. .. .. .. .. ..  ..
3254  2018-03848  1  1  1  0  1  0  1  0  0  0   0
3255  2018-00416  1  0  1  0  1  0  1  0  1  0   0
3256  2018-03717  1  1  1  1  1  1  1  1  1  0   0
3257  2018-03504  0  0  1  0  1  1  1  0  0  0   0
3258  2018-00115  1  0  1  0  1  0  1  0  1  0   0

[3259 rows x 12 columns]
```

```
result_df.to_csv('/content/drive/MyDrive/data/datasets/
submission.csv', index=False)
```