```
import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import torch
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import numpy as np


import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
print("Loading dataset...")
df = pd.read_parquet("hf://datasets/google-research-datasets/go_emotions/raw/train-00000-of-00001.parquet")

# Define the target emotions
target_emotions = ["joy", "sadness", "neutral"]

# Filter rows where at least one of the target emotions is present
print("Filtering relevant emotions...")
filtered_df = df[(df[target_emotions].sum(axis=1) > 0)]

# Relabel target emotions to 0, 1, 2
print("Relabeling emotions...")
def relabel(row):
    if row["joy"] == 1:
        return 0
    elif row["sadness"] == 1:
        return 1
    elif row["neutral"] == 1:
        return 2
    return -1  # Shouldn't happen if filtering is correct

# Select only the relevant columns
print("Dropping unnecessary columns...")
filtered_df = filtered_df[["text", "joy", "sadness", "neutral"]]


filtered_df["label"] = filtered_df.apply(relabel, axis=1)

# Check the distribution of labels
print(filtered_df["label"].value_counts())

# Split the dataset into training and testing sets
print("Splitting dataset into train and test sets...")
train_texts, test_texts, train_labels, test_labels = train_test_split(
    filtered_df["text"], filtered_df["label"], test_size=0.2, random_state=42
)

# Display a sample
print("Sample data:")
print(filtered_df.head())
```

```
Loading dataset...
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Filtering relevant emotions...
Relabeling emotions...
Dropping unnecessary columns...
label
2    55298
0     7983
1     6682
Name: count, dtype: int64
Splitting dataset into train and test sets...
Sample data:
                                                text  joy  sadness  neutral  \
0                                   That game hurt.    0        1        0
2      You do right, if you don't care then fuck 'em!    0        0        1
4   [NAME] was nowhere near them, he was by the Fa...    0        0        1
10  I have, and now that you mention it, I think t...    0        0        1
12                             BUT IT'S HER TURN! /s    0        0        1
```

```
    label
0       1
2       2
4       2
10      2
12      2
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Load tokenizer
print("Loading BERT tokenizer...")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Tokenize function
def tokenize_function(texts):
    return tokenizer(list(texts), padding=True, truncation=True, return_tensors="pt")

print("Tokenizing data...")
train_encodings = tokenize_function(train_texts)
test_encodings = tokenize_function(test_texts)

# Convert to PyTorch Dataset
class EmotionDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

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

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

train_dataset = EmotionDataset(train_encodings, train_labels.tolist())
test_dataset = EmotionDataset(test_encodings, test_labels.tolist())
```

```
Loading BERT tokenizer...
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 2.06kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 3.77MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 7.45MB/s]
config.json: 100%  570/570 [00:00<00:00, 16.7kB/s]
Tokenizing data...
```

```
# Load model
print("Loading BERT model...")
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)

# Compute class weights
print("Computing class weights...")
class_weights = compute_class_weight("balanced", classes=np.unique(train_labels), y=train_labels)
class_weights_tensor = torch.tensor(class_weights).to("cuda" if torch.cuda.is_available() else "cpu")

# Define custom loss function with class weights
from torch.nn import CrossEntropyLoss

def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="weighted")
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "f1": f1, "precision": precision, "recall": recall}
```

```
Loading BERT model...
model.safetensors: 100%  440M/440M [00:04<00:00, 63.4MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initiali
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Computing class weights...
```

```
# Training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

# Train the model
print("Starting training...")
trainer.train()

# Evaluate the model
print("Evaluating model...")
results = trainer.evaluate()
print("Results:", results)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will
  warnings.warn(
<ipython-input-8-c9130939fef2>:17: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`.
  trainer = Trainer(
Starting training...
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please spe
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.19.1
Run data is saved locally in /content/wandb/run-20241227_024523-erda5wu8
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/241547662-fccu-university/huggingface
View run at https://wandb.ai/241547662-fccu-university/huggingface/runs/erda5wu8
[10497/10497 2:33:16, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.310000	0.381239	0.868720	0.854785	0.863268	0.868720
2	0.327200	0.381292	0.867577	0.866498	0.865952	0.867577
3	0.220500	0.422805	0.864146	0.864239	0.864385	0.864146

```
Evaluating model...
[875/875 03:49]
Results: {'eval_loss': 0.38123923540115356, 'eval_accuracy': 0.8687200743228757, 'eval_f1': 0.8547854137691079, 'eval_precision': 0.8632
```

```
import os
# Define a directory to save the model in Google Drive
drive_save_path = "/content/drive/My Drive/bert-emotions"
os.makedirs(drive_save_path, exist_ok=True)
```

```
print("Saving the fine-tuned model to Google Drive...")
trainer.save_model(drive_save_path)

# Example inference
print("Example inference...")
from transformers import pipeline

emotion_classifier = pipeline("text-classification", model=drive_save_path, tokenizer=tokenizer)
example_texts = ["I am so happy!", "I feel really sad.", "It's a normal day."]
for text in example_texts:
    print(f"Input: {text} | Prediction: {emotion_classifier(text)}")
```

```
Saving the fine-tuned model to Google Drive...
Device set to use cuda:0
Example inference...
Input: I am so happy! | Prediction: [{'label': 'LABEL_0', 'score': 0.8858675956726074}]
Input: I feel really sad. | Prediction: [{'label': 'LABEL_1', 'score': 0.9675180912017822}]
Input: It's a normal day. | Prediction: [{'label': 'LABEL_2', 'score': 0.9872069954872131}]
```

```
from transformers import TFBertForSequenceClassification

# Convert the Hugging Face PyTorch model to TensorFlow
print("Converting model to TensorFlow/Keras format...")
tf_model = TFBertForSequenceClassification.from_pretrained("/content/drive/My Drive/bert-emotions", from_pt=True)

# Save the TensorFlow/Keras model
keras_drive_save_path = "/content/drive/My Drive/bert-emotions-tf"
tf_model.save_pretrained(keras_drive_save_path)

print(f"Model successfully saved in TensorFlow/Keras format at: {keras_drive_save_path}")
```

```
Converting model to TensorFlow/Keras format...
All PyTorch model weights were used when initializing TFBertForSequenceClassification.

All the weights of TFBertForSequenceClassification were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertForSequenceClassification for
Model successfully saved in TensorFlow/Keras format at: /content/drive/My Drive/bert-emotions-tf
```

```
from transformers import TFBertForSequenceClassification, BertTokenizer
import tensorflow as tf

# Load the saved TensorFlow model
print("Loading the TensorFlow/Keras model...")
model0 = TFBertForSequenceClassification.from_pretrained("/content/drive/My Drive/bert-emotions-tf")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

# Example inference
example_texts = ["I am feeling great!", "I am so sad right now.", "Just a regular day."]
print("DRIVE\n")
for text in example_texts:
    inputs = tokenizer(text, return_tensors="tf", truncation=True, padding=True)
    outputs = model0(inputs)
    probabilities = tf.nn.softmax(outputs.logits, axis=-1)
    predicted_label = tf.argmax(probabilities, axis=1).numpy()[0]
    label_map = {0: "joy", 1: "sadness", 2: "neutral"}
    print(f"Input: {text} | Prediction: {label_map[predicted_label]}")
```

```
Loading the TensorFlow/Keras model...
Some layers from the model checkpoint at /content/drive/My Drive/bert-emotions-tf were not used when initializing TFBertForSequenceClass
- This IS expected if you are initializing TFBertForSequenceClassification from the checkpoint of a model trained on another task or wit
- This IS NOT expected if you are initializing TFBertForSequenceClassification from the checkpoint of a model that you expect to be exac
All the layers of TFBertForSequenceClassification were initialized from the model checkpoint at /content/drive/My Drive/bert-emotions-tf
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertForSequenceClassification for
DRIVE

Input: I am feeling great! | Prediction: joy
Input: I am so sad right now. | Prediction: sadness
Input: Just a regular day. | Prediction: neutral
```