```
# -*- coding: utf-8 -*-
"""Copy of bert-finetune.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/16i65Lv-JSkfI2ksjG4i7xMQuF-
R64-J7
"""

# Install necessary libraries
!pip install transformers datasets torch accelerate -U

# Import libraries
import pandas as pd
from datasets import load_dataset, ClassLabel
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, TrainingArguments, Trainer, pipeline
from sklearn.model_selection import train_test_split

# Load the dataset from Hugging Face
dataset = load_dataset("amirpoudel/bert-reviews-data")

# Explore the dataset
print(dataset)

# Define the classes and ensure they match with the dataset
class_names = ['neutral', 'positive', 'negative']
labels = ClassLabel(names=class_names)

# Encode Labels
def encode_labels(examples):
    examples['label'] = labels.str2int(examples['label'])
    return examples

# Apply label encoding
encoded_dataset = dataset.map(encode_labels)

# Tokenize the Text
tokenizer = AutoTokenizer.from_pretrained("bert-base-multilingual-cased")

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length",
truncation=True)

tokenized_dataset = encoded_dataset.map(tokenize_function, batched=True)

# Split the dataset into training and validation sets
# Use the DatasetDict.train_test_split method to preserve indices
train_val_dataset =
tokenized_dataset['train'].train_test_split(test_size=0.2)
train_dataset = train_val_dataset['train']
val_dataset = train_val_dataset['test']
```

```
print(train_dataset)
print(val_dataset)

# Load the Model and Define Training Arguments
model = AutoModelForSequenceClassification.from_pretrained("bert-base-
multilingual-cased", num_labels=3)

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    logging_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_dir="./logs",
)

# Define the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)

# Fine-Tune the Model
trainer.train()

# Save the Model
model.save_pretrained("./fine-tuned-model")
tokenizer.save_pretrained("./fine-tuned-model")

# Evaluate the Model
results = trainer.evaluate()
print(results)

# Use the Model for Prediction
fine_tuned_model =
AutoModelForSequenceClassification.from_pretrained("./fine-tuned-model")
fine_tuned_tokenizer = AutoTokenizer.from_pretrained("./fine-tuned-
model")

# Create a pipeline for sentiment analysis
sentiment_pipeline = pipeline("sentiment-analysis",
model=fine_tuned_model, tokenizer=fine_tuned_tokenizer)

# Define the mapping of label indices to class names
label_map = {0: "neutral", 1: "positive", 2: "negative"}

# Make predictions
new_text = ["Thakali chicken set khana mitho xa, environment ni ekdam
peace family friends sanga dinner aauna ekdam fit hunxa"]
```

```
predictions = sentiment_pipeline(new_text)

# Map label indices to class names
mapped_predictions = [{"label": label_map[int(pred['label'].split('_')[-
1])], "score": pred['score']} for pred in predictions]
print(mapped_predictions)

# Use the Model for Prediction
# Load the fine-tuned model and tokenizer
fine_tuned_model =
AutoModelForSequenceClassification.from_pretrained("./fine-tuned-model")
fine_tuned_tokenizer = AutoTokenizer.from_pretrained("./fine-tuned-
model")

# Create a pipeline for sentiment analysis
sentiment_pipeline = pipeline("sentiment-analysis",
model=fine_tuned_model, tokenizer=fine_tuned_tokenizer)

# Define the mapping of label indices to class names
label_map = {0: "neutral", 1: "positive", 2: "negative"}

# Make predictions
new_text = ["food is decent"]
predictions = sentiment_pipeline(new_text)
print(predictions)

# Map label indices to class names
mapped_predictions = [{"label": label_map[int(pred['label'].split('_')[-
1])], "score": pred['score']} for pred in predictions]
print(mapped_predictions)

#Save to drive
from google.colab import drive
drive.mount('/content/drive')
# Define the path in Google Drive where you want to save the model
model_save_path = '/content/drive/MyDrive/fine-tuned-model'

# Save the model and tokenizer to the specified path
model.save_pretrained(model_save_path)
tokenizer.save_pretrained(model_save_path)

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

from transformers import AutoModelForSequenceClassification,
AutoTokenizer, pipeline

# Define the path in Google Drive where the model is saved
model_load_path = '/content/drive/MyDrive/fine-tuned-model'

# Load the model and tokenizer from the specified path
model =
AutoModelForSequenceClassification.from_pretrained(model_load_path)
tokenizer = AutoTokenizer.from_pretrained(model_load_path)
```

```
# Create a sentiment analysis pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer)

# Use the pipeline for predictions
new_text = ["khana man parena"]
predictions= sentiment_pipeline(new_text)

# Define the mapping of label indices to class names
label_map = {0: "neutral", 1: "positive", 2: "negative"}

# Map label indices to class names
mapped_predictions = [{"label": label_map[int(pred['label'].split('_')[-
1])], "score": pred['score']} for pred in predictions]
print(mapped_predictions)
```