ag\_news\_csv = "https://s3.amazonaws.com/fast-ai-nlp/ag\_news\_csv.tgz"

amazon\_review\_full\_csv = "https://s3.amazonaws.com/fast-ai-nlp/amazon\_review\_full\_csv.tgz"

amazon\_review\_polarity\_csv = "https://s3.amazonaws.com/fast-ai-nlp/amazon\_review\_polarity\_csv.tgz"

dbpedia\_csv = "https://s3.amazonaws.com/fast-ai-nlp/dbpedia\_csv.tgz"

yahoo\_answers\_csv = "https://s3.amazonaws.com/fast-ai-nlp/yahoo\_answers\_csv.tgz"

sogou\_news\_csv = "https://s3.amazonaws.com/fast-ai-nlp/sogou\_news\_csv.tgz"

yelp\_review\_full\_csv = "https://s3.amazonaws.com/fast-ai-nlp/yelp\_review\_full\_csv.tgz"

yelp\_review\_polarity\_csv = "https://s3.amazonaws.com/fast-ai-nlp/yelp\_review\_polarity\_csv.tgz"

pip install torch==2.3.1 transformers==4.42.3 accelerate==0.32.1

python -m venv .venu

source .venu/bin/activate # For Linux/Mac

.\.venu\Scripts\activate # For Windows

A blue line on a black background

描述已自动生成

import pandas as pd  
import tarfile  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments  
import torch  
  
# Data Preparation  
# Extract data from the tar.gz file  
data\_tg = tarfile.open('data/yelp\_review\_polarity\_csv.tgz')  
data\_tg.extractall('data')  
data\_tg.close()  
  
# Load the dataset  
train\_df = pd.read\_csv('data/yelp\_review\_polarity\_csv/train.csv', header=None)  
test\_df = pd.read\_csv('data/yelp\_review\_polarity\_csv/test.csv', header=None)  
  
# Convert labels to binary (1 for positive, 0 for negative)  
train\_df[0] = (train\_df[0] == 2).astype(int)  
test\_df[0] = (test\_df[0] == 2).astype(int)  
  
# Creating training dataframe according to BERT by adding the required columns  
df\_bert = pd.DataFrame({  
 'id': range(len(train\_df)),  
 'label': train\_df[0],  
 'alpha': ['a']\*train\_df.shape[0],  
 'text': train\_df[1].replace(r'\n', ' ', regex=True)  
})  
  
# Splitting training data file into \*train\* and \*dev\*  
df\_bert\_train, df\_bert\_dev = train\_test\_split(df\_bert, test\_size=0.01)  
  
# Reset indices  
df\_bert\_train = df\_bert\_train.reset\_index(drop=True)  
df\_bert\_dev = df\_bert\_dev.reset\_index(drop=True)  
  
# Creating test dataframe according to BERT  
df\_bert\_test = pd.DataFrame({  
 'id': range(len(test\_df)),  
 'text': test\_df[1].replace(r'\n', ' ', regex=True)  
})  
  
# Reset indices  
df\_bert\_test = df\_bert\_test.reset\_index(drop=True)  
  
# Adding labels to the test dataframe  
df\_bert\_test['label'] = test\_df[0]  
  
# Saving dataframes to .tsv format as required by BERT  
df\_bert\_train.to\_csv('data/train.tsv', sep='\t', index=False, header=False)  
df\_bert\_dev.to\_csv('data/dev.tsv', sep='\t', index=False, header=False)  
df\_bert\_test.to\_csv('data/test.tsv', sep='\t', index=False, header=False)  
  
# Model Training  
tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')  
model = BertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=2)  
  
class YelpDataset(torch.utils.data.Dataset):  
 def \_\_init\_\_(self, dataframe, tokenizer, max\_len):  
 self.tokenizer = tokenizer  
 self.data = dataframe  
 self.text = dataframe.text  
 self.targets = dataframe.label  
 self.max\_len = max\_len  
  
 def \_\_len\_\_(self):  
 return len(self.text)  
  
 def \_\_getitem\_\_(self, index):  
 text = str(self.text[index])  
 inputs = self.tokenizer.encode\_plus(  
 text,  
 None,  
 add\_special\_tokens=True,  
 max\_length=self.max\_len,  
 padding='max\_length',  
 return\_token\_type\_ids=True,  
 truncation=True  
 )  
 return {  
 'input\_ids': torch.tensor(inputs['input\_ids'], dtype=torch.long),  
 'attention\_mask': torch.tensor(inputs['attention\_mask'], dtype=torch.long),  
 'token\_type\_ids': torch.tensor(inputs['token\_type\_ids'], dtype=torch.long),  
 'labels': torch.tensor(self.targets[index], dtype=torch.long)  
 }  
  
# Sample a smaller portion of the dataset to achieve around 100 steps  
# Target number of steps = 100, batch\_size = 16, gradient\_accumulation\_steps = 2  
# num\_samples\_needed = 100 / 2 \* 16 = 800  
df\_bert\_train = df\_bert\_train.sample(n=800).reset\_index(drop=True) # Sample 800 samples  
df\_bert\_dev = df\_bert\_dev.sample(frac=0.05).reset\_index(drop=True)  
  
print(f'df\_bert\_train len: {len(df\_bert\_train)}')  
  
train\_dataset = YelpDataset(df\_bert\_train, tokenizer, max\_len=128)  
dev\_dataset = YelpDataset(df\_bert\_dev, tokenizer, max\_len=128)  
  
training\_args = TrainingArguments(  
 output\_dir='./results',  
 num\_train\_epochs=1, # Reduce the number of epochs  
 per\_device\_train\_batch\_size=16, # Increase the batch size  
 per\_device\_eval\_batch\_size=16,  
 warmup\_steps=500,  
 weight\_decay=0.01,  
 logging\_dir='./logs',  
 gradient\_accumulation\_steps=2, # Accumulate gradients over 2 steps  
)  
  
trainer = Trainer(  
 model=model,  
 args=training\_args,  
 train\_dataset=train\_dataset,  
 eval\_dataset=dev\_dataset  
)  
  
trainer.train()  
  
# Evaluation of Accuracy  
# Function to evaluate accuracy on a dataset  
def evaluate\_accuracy(dataframe, dataset):  
 trainer.model.eval()  
 outputs = trainer.predict(dataset)  
 predictions = torch.argmax(torch.tensor(outputs.predictions), axis=1)  
 accuracy = accuracy\_score(dataframe.label, predictions.numpy())  
 return accuracy  
  
train\_accuracy = evaluate\_accuracy(df\_bert\_train, train\_dataset)  
dev\_accuracy = evaluate\_accuracy(df\_bert\_dev, dev\_dataset)  
  
print(f'Train Accuracy: {train\_accuracy}')  
print(f'Dev Accuracy: {dev\_accuracy}')  
  
# Evaluating on the test set  
test\_dataset = YelpDataset(df\_bert\_test, tokenizer, max\_len=128)  
  
test\_accuracy = evaluate\_accuracy(df\_bert\_test, test\_dataset)  
print(f'Test Accuracy: {test\_accuracy}')