import torch

import os

import sys

from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline, Trainer, TrainingArguments

from torch.utils.data import Dataset, DataLoader

import pandas as pd

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

# Download necessary NLTK resources

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

# Initialize lemmatizer and stopwords

lemmatizer = WordNetLemmatizer()

stop\_words = set(stopwords.words('english'))

# Function to process text using basic NLP with nltk

def process\_text(text):

tokens = nltk.word\_tokenize(text)

filtered\_tokens = [lemmatizer.lemmatize(word.lower()) for word in tokens if word.isalpha() and word.lower() not in stop\_words]

return ' '.join(filtered\_tokens)

class EmotionDataset(Dataset):

def \_init\_(self, file\_path='emotion\_dataset.csv', tokenizer=None, max\_length=128):

if not os.path.exists(file\_path):

raise FileNotFoundError(f"Dataset file {file\_path} not found.")

self.data = pd.read\_csv(file\_path)

self.tokenizer = tokenizer

self.max\_length = max\_length

def \_len\_(self):

return len(self.data)

def \_getitem\_(self, idx):

text = self.data.iloc[idx]['text']

label = self.data.iloc[idx]['label']

processed\_text = process\_text(text) # Use NLP process without spaCy

inputs = self.tokenizer(processed\_text, truncation=True, padding='max\_length', max\_length=self.max\_length)

inputs['label'] = int(label)

return {key: torch.tensor(val) for key, val in inputs.items()}

class EmotionAwareChatbot:

def \_init\_(self, model\_name='j-hartmann/emotion-english-distilroberta-base', dataset\_file='emotion\_dataset.csv'):

self.tokenizer = AutoTokenizer.from\_pretrained(model\_name)

self.model = AutoModelForSequenceClassification.from\_pretrained(model\_name)

if dataset\_file and os.path.exists(dataset\_file):

self.finetune\_model(dataset\_file)

self.emotion\_classifier = pipeline('sentiment-analysis', model=self.model, tokenizer=self.tokenizer)

self.last\_emotion = None # Track the last emotion detected

def finetune\_model(self, dataset\_file):

print("Starting fine-tuning with dataset...")

dataset = EmotionDataset(dataset\_file, self.tokenizer)

training\_args = TrainingArguments(output\_dir='./results', num\_train\_epochs=3, per\_device\_train\_batch\_size=8)

trainer = Trainer(model=self.model, args=training\_args, train\_dataset=dataset)

trainer.train()

def classify\_emotion(self, text):

processed\_text = process\_text(text) # Use NLP process without spaCy

result = self.emotion\_classifier(processed\_text)

# For simplicity, returning sentiment analysis result as emotion

return result[0]['label'], result[0]['score']

def respond(self, text):

# Handle greetings explicitly

greetings = ['hi', 'hello', 'hey', 'good morning', 'good afternoon', 'good evening']

if any(greeting in text.lower() for greeting in greetings):

return "Hello! How can I assist you today?"

# Handle specific complaints like work-related frustration

work\_related\_phrases = ['work was so hard', 'work is exhausting', 'I’m stressed at work', 'I had a tough day at work']

if any(phrase in text.lower() for phrase in work\_related\_phrases):

self.last\_emotion = 'work-related stress' # Track that the last emotion was work-related stress

return "It sounds like you had a challenging day at work. I'm really sorry to hear that. How can I help you feel better?"

# If the user previously mentioned work-related stress, continue the conversation meaningfully

if self.last\_emotion == 'work-related stress':

if 'yes' in text.lower():

return "I'm glad you're open to discussing it. How about telling me what specifically made your day hard? I'm here to help."

# Handle emotions based on text input

emotion, score = self.classify\_emotion(text)

if emotion == 'anger' or emotion == 'frustration':

response = "I'm really sorry you're feeling this way. Let's work together to resolve this issue."

elif emotion == 'joy':

response = "That's awesome! I'm glad you're having a great time. How can I make your day even better?"

elif emotion == 'sadness':

response = "I can sense you're feeling down. I'm here to listen and help. Please let me know what you need."

elif emotion == 'fear':

response = "I'm sorry you're feeling this way. Is there something specific that's worrying you?"

elif emotion == 'surprise':

response = "Wow, it sounds like you're surprised! How can I assist you in this moment?"

else: # Neutral or other emotions

response = "Thank you for reaching out. How can I assist you today?"

return response

# Example usage with continuous interaction

chatbot = EmotionAwareChatbot(dataset\_file='emotion\_dataset.csv')

print("Chatbot is ready to chat! Type 'exit' to end the conversation.")

while True:

user\_input = input("You: ")

if user\_input.lower() in ['exit', 'quit', 'bye']:

print("Chatbot: Goodbye! Have a great day!")

break

response = chatbot.respond(user\_input)

print("Chatbot: ", response)