



**Emotion-Aware Chatbot for Customer Support**

|  |  |  |
| --- | --- | --- |
| **College Code & Name** | **3135 - Panimalar Engineering College Chennai City Campus** | |
| **Subject Code & Name** | **NM1090 - Natural Language Processing (NLP) Techniques** | |
| **Year and Semester** | **III Year - VI Semester** | |
| **Project Created by** | **1.** | **THINESH KUMAAR K (313522104166)** |
|  | **2.** | **THIRUPPATHI BALAJI E (313522104167)** |
|  | **3.** | **CHETAN VM (313522104168)** |
|  | **4.** | **SUDARSAN S (313522104164)** |

## ABSTRACT

Emotion recognition is a crucial aspect of enhancing user experiences in digital communication, particularly in customer support interactions. This project focuses on the development of an Emotion-Aware Chatbot utilizing Natural Language Processing (NLP) and Deep Learning techniques. Specifically, the chatbot leverages a pre-trained transformer model (DistilRoBERTa)s fine-tuned for emotion classification, enabling it to detect user emotions such as joy, sadness, anger, and fear directly from text input. The chatbot is designed to provide empathetic, context-aware responses, making it more engaging and supportive for users. Text preprocessing techniques, including tokenization, stop word removal, and lemmatization, are applied to improve the quality of emotion detection. The model is fine-tuned using a custom emotion dataset, and its performance is evaluated based on classification accuracy. The results demonstrate the chatbot’s ability to effectively understand and respond to user emotions, providing a more human-like interaction experience. Additionally, the report discusses the challenges associated with emotion classification, such as handling ambiguous text and maintaining context in conversations. Potential improvements, including enhanced context tracking and the use of reinforcement learning, are also explored to further optimize the chatbot’s performance.

**1.Introduction**

In the era of digital communication, understanding user emotions has become a critical aspect of enhancing user experiences in customer support. The Emotion-Aware Chatbot aims to leverage Natural Language Processing (NLP) and Deep Learning to recognize user emotions and provide empathetic, context-aware responses.

**2. TECHNOLOGIES USED**

* **Programming Languages:** Python for development.
* **Libraries:** PyTorch for deep learning, Hugging Face Transformers for model handling, NLTK for text processing, and Pandas for data manipulation.
* **Pre-trained Model:** DistilRoBERTa (Hugging Face) for emotion classification.

**3. PROJECT IMPLEMENTATION**

* **Data Collection:** A CSV dataset of text samples labeled with emotions (joy, sadness, anger, fear) is used.
* **Text Processing:** Text is preprocessed using tokenization, stopword removal, and lemmatization for better emotion detection.
* **Model Fine-Tuning:** A pre-trained DistilRoBERTa model is fine-tuned on the emotion dataset using the Hugging Face Trainer API (3 epochs, batch size 8).
* **Chatbot Logic:** The chatbot detects user emotions, recognizes greetings, handles work-related stress, and generates empathetic, context-aware responses.

**4.Coding**

**#**import necessary libraries.

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']:

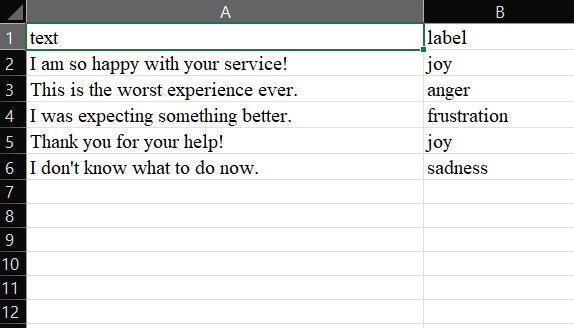
        print("Chatbot: Goodbye! Have a great day!")

        break

    response = chatbot.respond(user\_input)

    print("Chatbot: ", response)

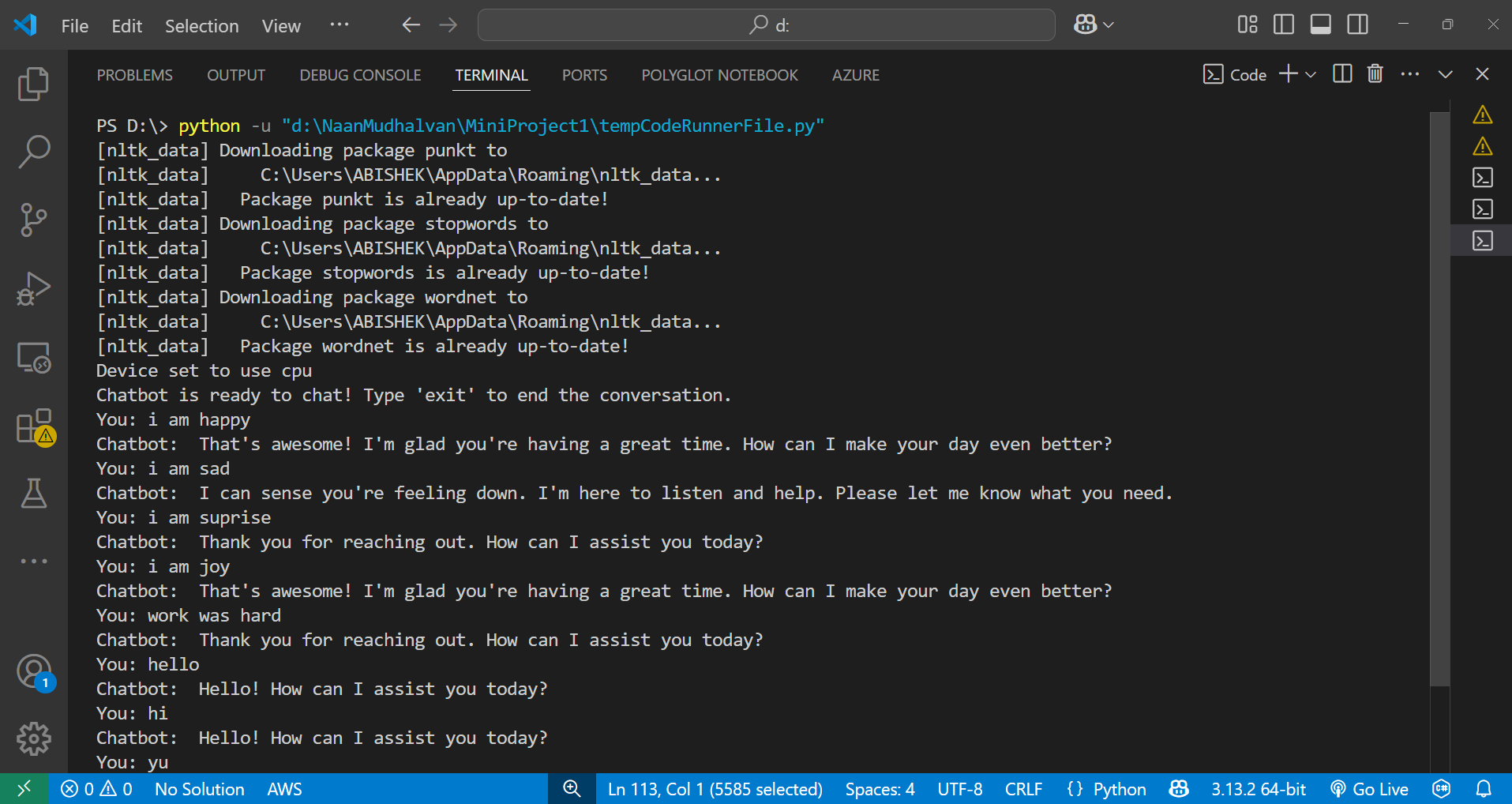
**5.Custom Emotion -DataSet**

****

**6. TESTING AND OPTIMIZATION**

* Unit Testing: Key functions (text processing, emotion classification) are tested for accuracy.
* Fine-Tuning Validation: Model performance is evaluated on a validation dataset to ensure accurate emotion detection.
* Error Handling: Proper exception handling for missing datasets, invalid inputs, and model loading errors.

**7.Output:**

****

**8.CONCLUSION**

The Emotion-Aware Chatbot effectively demonstrates the application of Natural Language Processing (NLP) and deep learning in recognizing and responding to user emotions. By leveraging a fine-tuned DistilRoBERTa model, the chatbot can accurately detect emotions in user text and provide empathetic, context-aware responses. The implementation showcases the potential of combining NLP and deep learning for enhanced user interaction, making it a valuable tool for customer support and other communication-based applications.