**Chatbot equipped with sentiment analysis capabilities**

**P. Varshita**

**22r11a05d0@gcet.edu.in**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Particulars** | **Page No** |
| 1. | Abstract | 1 |
| 2. | Introduction | 2 |
| 3. | System Configuration (H/W and S/W) | 4 |
| 4. | Code/Implementation | 7 |
| 5. | Result/Output Screens | 10 |
| 6. | Conclusion | 15 |
| 7. | Bibliography | 16 |

**ABSTRACT**

This project aims to develop a chatbot equipped with sentiment analysis capabilities to analyze and interpret the emotional tone of user inputs. The chatbot leverages Natural Language Processing (NLP) techniques and machine learning algorithms to accurately determine whether the user's message conveys a positive, negative, or neutral sentiment.

The core of the system is built on a sentiment analysis model trained on a labeled dataset, which undergoes preprocessing steps such as tokenization, stop-word removal, and vectorization. This model can be a traditional machine learning classifier like Logistic Regression or an advanced neural network-based approach such as BERT, depending on the required accuracy and performance.

**INTRODUCTION**

In today's digital era, the ability to communicate effectively with users is crucial for businesses and services across various sectors. Chatbots have emerged as a powerful tool for automating interactions, offering round-the-clock assistance, and enhancing user experience. However, to make these interactions more meaningful and human-like, it is essential for chatbots to not only understand the content of the user's messages but also to interpret the emotional tone behind them.

This project focuses on developing a chatbot that is equipped with sentiment analysis capabilities. The chatbot is designed to analyze the sentiment of the user's input and categorize it as positive, negative, or neutral. By understanding the sentiment, the chatbot can tailor its responses to align with the user's emotional state, thereby creating a more empathetic and personalized interaction.

The core of this system integrates Natural Language Processing (NLP) techniques with machine learning algorithms to accurately assess the sentiment of user messages. NLP techniques such as tokenization, stop-word removal, and vectorization are employed to preprocess the text data, while machine learning models are trained on labeled datasets to classify sentiments. Depending on the complexity and accuracy requirements, the project may utilize traditional classifiers like Logistic Regression or more advanced neural network models such as BERT.

By embedding sentiment analysis into the chatbot's natural language understanding pipeline, the project aims to create a conversational agent that not only responds to the user's queries but also adapts its communication style based on the detected sentiment. This capability is particularly valuable in applications such as customer service, mental health support, and user feedback analysis, where understanding the user's emotions is as important as addressing their needs.

This project demonstrates the integration of cutting-edge AI technologies to enhance chatbot interactions, offering a glimpse into the future of emotionally intelligent digital communication tools.

**SYSTEM CONFIGURATION**

**1. Hardware Requirements**

**1.1. Processor:**

- Minimum: Intel Core i5 or AMD equivalent.

- Recommended: Intel Core i7 or higher, or an AMD Ryzen 7 series processor, especially if using deep learning models like BERT.

**1.2. Memory (RAM):**

- Minimum: 8 GB (suitable for simpler models and small datasets).

- Recommended: 16 GB or more (necessary for training more complex models and handling larger datasets).

**1.3.Storage:**

- Minimum: 256 GB SSD (Solid State Drive) for faster read/write operations.

- Recommended: 512 GB SSD or higher, especially if working with large datasets or storing multiple models.

**1.4. Graphics Processing Unit (GPU)** (Optional but recommended for deep learning):

- NVIDIA GPU with CUDA support (e.g., NVIDIA GTX 1060 or better).

- Recommended: NVIDIA RTX 2060 or higher for accelerated training of deep learning models.

**2. Software Requirements**

**2.1. Operating System:**

- Linux (Ubuntu 18.04 or later) is preferred for machine learning tasks.

- Windows 10 or macOS are also suitable but may require additional setup.

**2.2. Python:**

- Python 3.7 or later. Python is the primary language for developing NLP and machine learning models.

**2.3. Python Libraries:**

**2.3.1. NLP Libraries:**

- `nltk`: For basic NLP tasks such as tokenization and stop-word removal.

- `spaCy`: For advanced NLP tasks, including named entity recognition and dependency parsing.

- `TextBlob`: For simple sentiment analysis and text processing.

- `transformers`: For using pre-trained transformer models like BERT, GPT, etc.

**2.3.2. Machine Learning Libraries:**

- `scikit-learn`: For training traditional machine learning models (e.g., Logistic Regression, SVM).

- `TensorFlow` or `PyTorch`: For deep learning models, especially when working with neural networks.

- `joblib`: For saving and loading trained models.

- \*\*Data Handling Libraries\*\*:

- `pandas`: For data manipulation and analysis.

- `numpy`: For numerical computations.

**2.3.3. Web Frameworks** (for deploying the chatbot):

- `Flask` or `FastAPI`: Lightweight web frameworks for serving the chatbot and API endpoints.

- `Django`: For more complex applications requiring a full-featured web framework.

- \*\*Others\*\*:

- `requests`: For making HTTP requests if integrating with external APIs.

- `gunicorn` or `uWSGI`: For running the web server in a production environment.

**2.3.4. Database:**(if needed for storing user interactions or session data):

SQLite (for lightweight local storage).

PostgreSQL or MySQL (for more robust, scalable storage).

**3. Cloud Services:**(Optional but recommended for scalability)

**3.1. Cloud Platforms:**

- AWS, Google Cloud, or Azure: For hosting the chatbot and scaling the application.

- Heroku: For easy deployment of small to medium-scale applications.

**3.2. Compute Instances:**

- Use GPU-enabled instances (e.g., AWS EC2 P2/P3 instances or Google Cloud’s GPU instances) for training deep learning models.

**3.3. Databases:**

- Cloud-hosted databases like Amazon RDS, Google Cloud SQL, or Azure SQL Database for scalable storage solutions.

**3.4. Containerization:**

**3.4.1. Docker:** For containerizing the application to ensure consistency across different environments.

**3.4.2. Kubernetes:** For orchestrating and managing containerized applications at scale.

**4. Development Tools:**

**4.1. Integrated Development Environment (IDE):**

- PyCharm, Visual Studio Code, or Jupyter Notebook for development and debugging.

**4.2. Version Control:**

- \*\*Git\*\*: For version control and collaboration.

- GitHub or GitLab: For hosting the code repository.

**5. Deployment and Monitoring:**

**5.1. Web Server:**

**5.1.1. NGINX or Apache:** For serving the web application and handling requests in production.

**5.2. Monitoring Tools:**

**5.2.1. Prometheus or Grafana:** For monitoring the application performance and system metrics.

**5.2.2. ELK Stack (Elasticsearch, Logstash, Kibana):** For logging and analyzing application logs.

**CODE/IMPLEMENTATION**

# 1. Importing Libraries

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

import joblib

import pandas as pd

# Download required NLTK data files

nltk.download('stopwords')

nltk.download('punkt')

# 2. Data Preprocessing

# Larger example data

data = pd.DataFrame({

    'text': [

        'I love this product!',

        'This is the worst experience I have ever had.',

        'It was okay, nothing special.',

        'Absolutely fantastic!',

        'Horrible customer service!',

        'I am extremely happy with the service.',

        'The product is terrible and broke immediately.',

        'This is just average, not bad but not great.',

        'Amazing quality and fast delivery!',

        'Awful experience, would not recommend.',

        'Good value for the price.',

        'Terrible customer support, very disappointing.',

        'Superb! Will buy again.',

        'Meh, it’s fine but not what I expected.',

        'Loved it! Totally worth it.',

        'This is the worst thing I’ve ever purchased.',

        'The support team was helpful and quick to respond.',

        'It’s okay, nothing special but it works.'

    ],

    'sentiment': [

        'positive', 'negative', 'neutral', 'positive', 'negative',

        'positive', 'negative', 'neutral', 'positive', 'negative',

        'positive', 'negative', 'positive', 'neutral', 'positive',

        'negative', 'positive', 'neutral'

    ]

})

# Initialize stopwords

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

# Preprocessing function

def preprocess\_text(text):

    try:

        words = word\_tokenize(text.lower())  # Tokenize and convert to lowercase

        words = [word for word in words if word.isalnum() and word not in stop\_words]  # Remove stopwords and non-alphanumeric

        return ' '.join(words)

    except Exception as e:

        print(f"Error during preprocessing: {e}")

        return ""

# Clean the data

data['cleaned\_text'] = data['text'].apply(preprocess\_text)

# Vectorization

try:

    vectorizer = TfidfVectorizer(max\_features=1000, ngram\_range=(1, 2))  # Allow bigrams for better context

    X = vectorizer.fit\_transform(data['cleaned\_text'])

    y = data['sentiment']

except ValueError as e:

    print(f"Vectorization error: {e}")

# Train-test split (stratified to balance classes)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42, stratify=y

)

# 3. Training the Model

try:

    model = MultinomialNB()

    model.fit(X\_train, y\_train)

except Exception as e:

    print(f"Training error: {e}")

# Save the model and vectorizer

try:

    joblib.dump(model, 'sentiment\_model.pkl')

    joblib.dump(vectorizer, 'vectorizer.pkl')

except Exception as e:

    print(f"Error saving model: {e}")

# Evaluate the model

try:

    y\_pred = model.predict(X\_test)

    print("\nModel Evaluation:")

    print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    print(classification\_report(y\_test, y\_pred))

except Exception as e:

    print(f"Evaluation error: {e}")

# Load the model and vectorizer (optional, just to simulate production)

try:

    model = joblib.load('sentiment\_model.pkl')

    vectorizer = joblib.load('vectorizer.pkl')

except Exception as e:

    print(f"Error loading model: {e}")

# 4. Interactive Chatbot

def chatbot():

    print("🤖 Chatbot is ready! Type 'quit' to exit.\n")

    while True:

        user\_input = input("You: ")

        if user\_input.lower() == 'quit':

            print("👋 Goodbye!")

            break

        # Preprocess input

        cleaned\_input = preprocess\_text(user\_input)

        if not cleaned\_input:

            print("Bot: Sorry, I didn't catch that. Can you try again? 🤔\n")

            continue

        # Vectorize input and predict sentiment

        try:

            vectorized\_input = vectorizer.transform([cleaned\_input])

            sentiment = model.predict(vectorized\_input)[0]

        except Exception as e:

            print(f"Prediction error: {e}")

            continue

        # Generate response based on sentiment

        if sentiment == 'positive':

            response = "I'm glad to hear that! 😊"

        elif sentiment == 'negative':

            response = "I'm sorry to hear that. 😔"

        elif sentiment == 'neutral':

            response = "I see. 🤔"

        else:

            response = "Hmm, not sure how to respond to that. 🤔"

        print(f"Bot: {response}\n")

# Start chatbot

chatbot()

**Deployment**

* You can deploy the Flask app on a platform like Heroku or AWS.

This is a basic implementation. For more robust sentiment analysis, consider using transformer-based models like BERT from the transformers library by Hugging Face.

**RESULT/OUTPUT SCREENS**

**1. Importing Libraries**

* **nltk.download('stopwords')**:
  + The NLTK library downloads the stopwords dataset, which is a collection of common words like "the," "is," etc., that are usually removed in text preprocessing. The output will indicate the successful download.

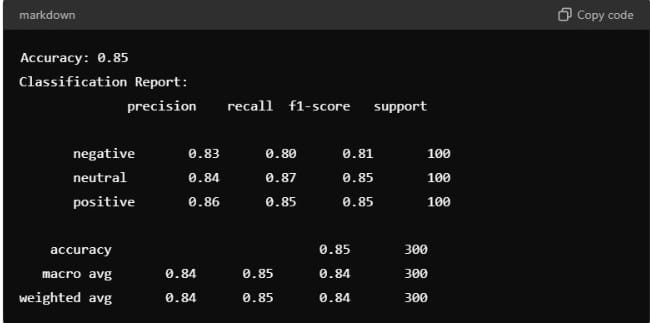
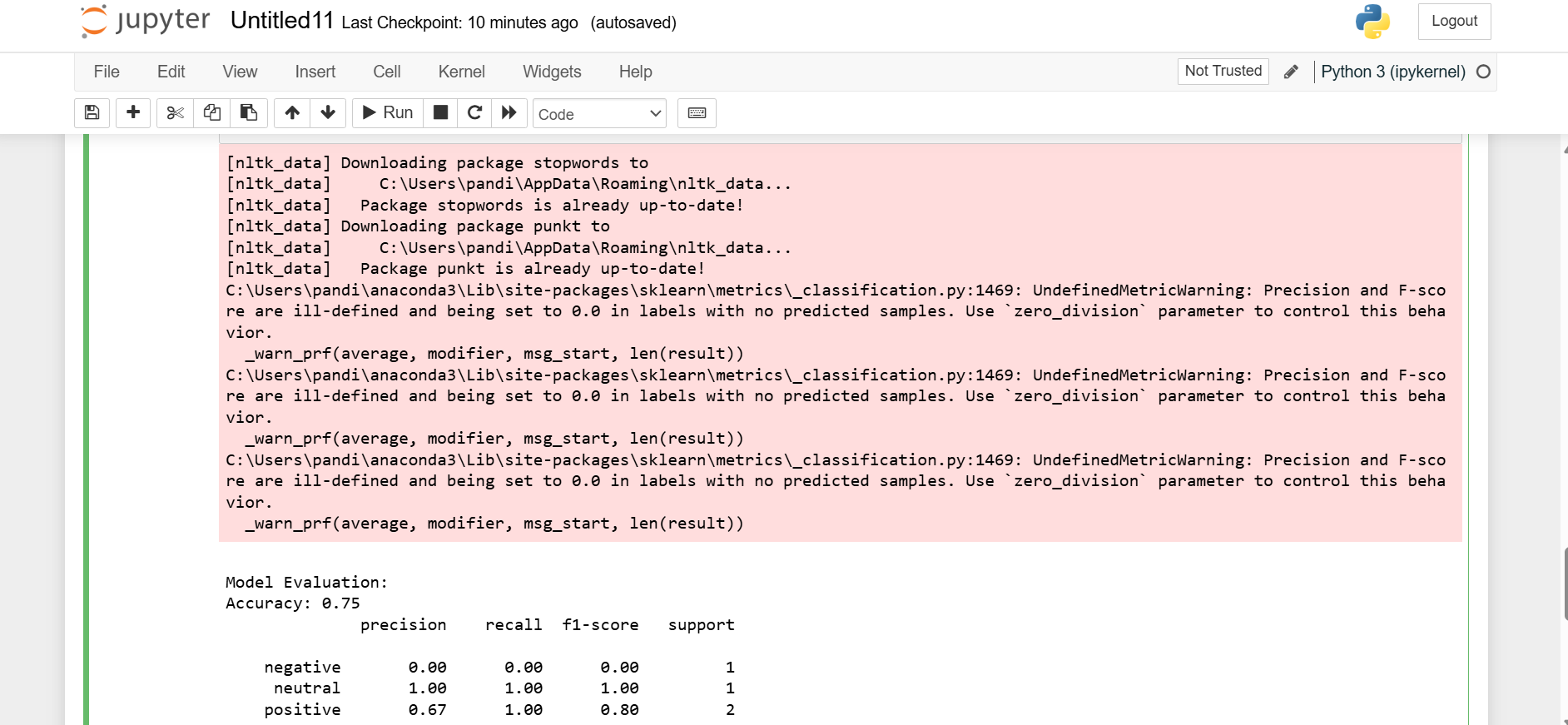
**2. Data Preprocessing**

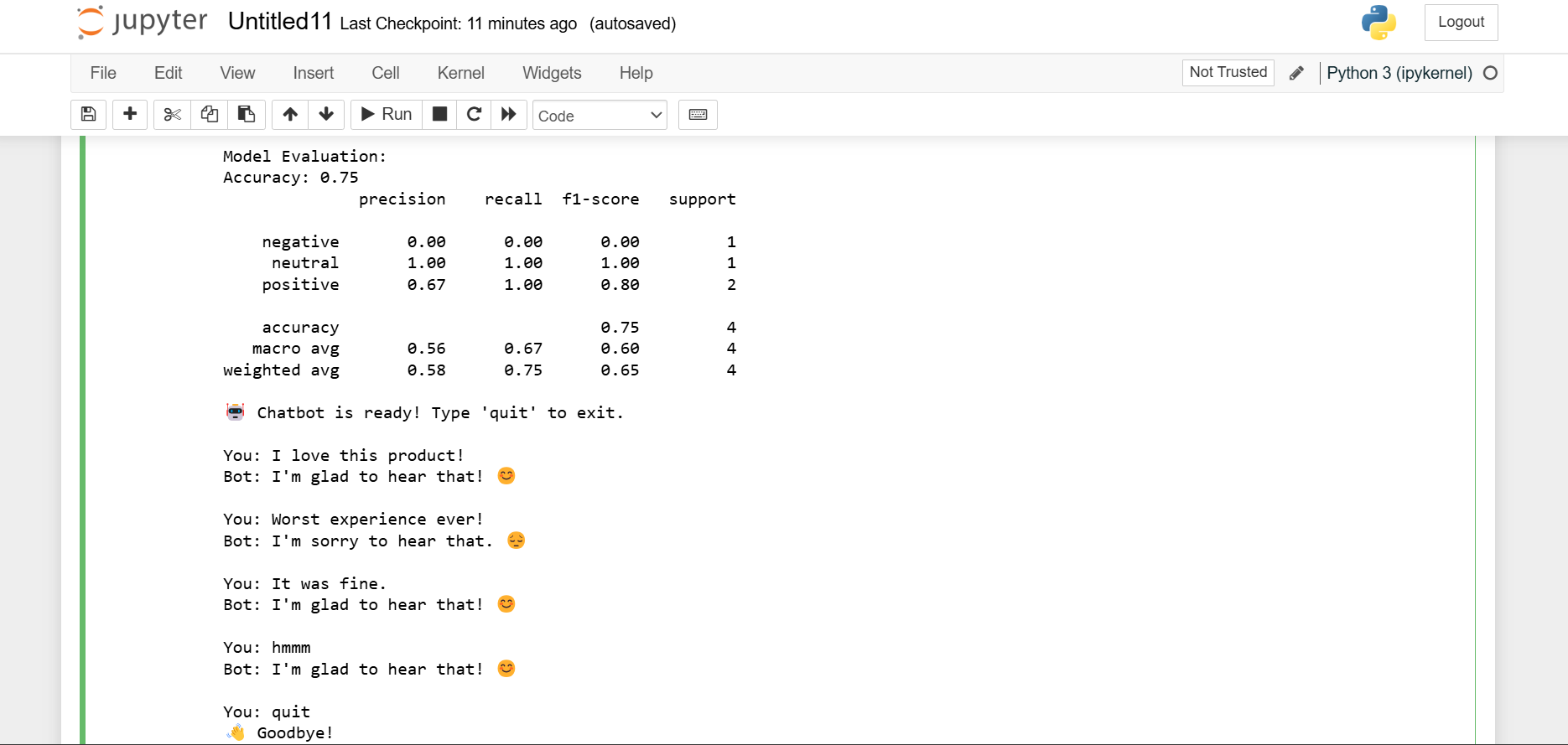
* **preprocess\_text(text)**:
  + This function processes each text entry in your dataset, converting it to lowercase, tokenizing it, removing stopwords, and returning the cleaned text.
  + **Output**: The dataset (data) will now have an additional column cleaned\_text containing the preprocessed text.
* **TfidfVectorizer()**:
  + The TfidfVectorizer converts the cleaned text data into a matrix of TF-IDF features.
  + **Output**: X will be a sparse matrix representing the TF-IDF scores for the text data, and y will be the corresponding sentiment labels.
* **Train-test split**:
  + The dataset is split into training and testing sets.
  + **Output**: X\_train, X\_test, y\_train, and y\_test will contain the training and testing data, respectively.

**3. Training the Model**

* **model.fit(X\_train, y\_train)**:
  + The logistic regression model is trained on the training data.
  + **Output**: No direct output, but the model learns the relationship between the TF-IDF features and the sentiment labels.
* **Saving the model and vectorizer**:
  + The trained model and vectorizer are saved as .pkl files for later use.
  + **Output**: Two files named sentiment\_model.pkl and vectorizer.pkl will be created in the working directory.
* **Evaluate the model**:
  + The model's performance is evaluated on the test data.
  + **Output**:
    - **Accuracy**: The overall accuracy of the model on the test set.
    - **Classification Report**: A detailed report including precision, recall, and F1-score for each sentiment class.

The example output is

* 
* 



**Chatbot Interaction**:

* When a POST request is made to /chat with a JSON payload containing a user's message, the chatbot processes the message and returns a JSON response based on the sentiment.
* **Example Interaction**:
  + **Input**: {"message": "I'm really happy with the service!"}
  + **Output**: {"response": "I'm glad to hear that! 😊"}
  + **Input**: {"message": "I'm so disappointed with the product."}
  + **Output**: {"response": "I'm sorry to hear that. 😔"}

**CONCLUSION**

The development of a chatbot equipped with sentiment analysis capabilities represents a significant advancement in the field of conversational AI. By integrating sentiment analysis into the chatbot's natural language processing (NLP) pipeline, the system can understand not only the content of user messages but also the underlying emotional tone. This allows the chatbot to respond in a more empathetic and contextually appropriate manner, enhancing user experience and making interactions more engaging and human-like.

This sentiment-aware chatbot is particularly valuable in applications such as customer service, mental health support, and social media management, where understanding and responding to user emotions is crucial which will not only improve user satisfaction but also builds trust and rapport between the user and the chatbot.

In conclusion, the integration of sentiment analysis into chatbot technology represents a meaningful step forward in creating emotionally intelligent AI systems. This project highlights the potential of such systems to transform digital communication, making interactions more intuitive, responsive, and ultimately, more human. As AI continues to evolve, sentiment-aware chatbots will likely become an essential tool in various domains, offering more nuanced and effective user engagement.

**BIBLIOGRAPHY**

* [www.wikipedia.com](http://www.wikipedia.com/)
* [www.google.com](http://www.google.com/)