Mental Wellness Chatbot

An AI-Powered Solution for Mental Health Support

[Full Name]

[Christian Brothers University]

PUBH 7160

[Instructor's Name]

March 24, 2025

**Introduction**

This document provides a detailed overview of the AI-powered chatbot backend, which integrates machine learning techniques to classify user queries into predefined intents and return appropriate responses. The chatbot leverages a **Random Forest Classifier** for intent classification and utilizes **Flask** to create an API for handling interactions. The backend ensures dynamic intent management and high scalability for enhanced chatbot functionality.

**Overview of the Backend**

The backend consists of the following core components:

1. **Model Training** - Prepares and trains a **Random Forest model** using a structured dataset containing user queries and corresponding intents.
2. **Flask API** - Provides endpoints for processing user messages, predicting intents, and fetching responses.
3. **Database (JSON Storage)** - Stores predefined intents, patterns, and responses, allowing dynamic updates.
4. **Hyperparameter Tuning** - Optimizes model performance using **GridSearchCV**.
5. **Dynamic Response Handling** - Enables the chatbot to retrieve and return relevant responses based on predicted intent.

**Model Training**

The chatbot model is trained using a dataset stored in **intents.json**, which includes predefined patterns and their corresponding responses. The following steps outline the training pipeline:

**Steps in Training:**

1. **Data Extraction** - Load intents.json and extract text patterns and corresponding intent labels.
2. **Feature Engineering** - Convert text data into numerical representations using **TF-IDF Vectorization** to enhance model comprehension.
3. **Data Splitting** - Partition data into **training (80%)** and **testing (20%)** sets for model evaluation.
4. **Model Training** - Train a **Random Forest Classifier** with class balancing to handle uneven data distribution.
5. **Hyperparameter Optimization** - Utilize **GridSearchCV** to fine-tune model parameters for improved accuracy.
6. **Model Serialization** - Save the trained **Random Forest model** (intent\_model.pkl) and **TF-IDF vectorizer** (vectorizer.pkl) for later use.
7. **Model Evaluation** - Assess model performance using accuracy metrics, a classification report, and a confusion matrix.

**Code Highlights:**

vectorizer = TfidfVectorizer(stop\_words="english", ngram\_range=(1,3), max\_features=5000)

X\_train\_vec = vectorizer.fit\_transform(X\_train)

model = RandomForestClassifier(random\_state=42, class\_weight="balanced")

**Flask API Endpoints**

The backend provides REST API endpoints to facilitate chatbot interactions. These endpoints allow users to communicate with the AI model efficiently.

**1. Chat Endpoint (/chat)**

* Accepts a user message as input.
* Predicts the intent using the trained model.
* Returns an appropriate response from responses.json.

**Example Request:**

{

"message": "I need support"

}

**Example Response:**

{

"response": "How can I assist you today?"

}

**2. Add Intent Endpoint (/add\_intent)**

* Allows dynamic addition of new intents.
* Updates intents.json and responses.json.

**Example Request:**

{

"tag": "greeting",

"patterns": ["Hello", "Hi there"],

"responses": ["Hello! How can I help?"]

}

**Database (JSON Storage)**

**File Structure:**

* **intents.json** - Stores chatbot intents, patterns, and responses.
* **responses.json** - Extracted responses for quick access and retrieval.

**Example JSON Structure:**

{

"intents": [

{

"tag": "greeting",

"patterns": ["Hello", "Hi"],

"responses": ["Hi there! How can I assist you?"]

}

]

}

**How the AI Model Works**

1. **User Input** - The user submits a message via the chatbot interface.
2. **Vectorization** - The message is transformed using **TF-IDF vectorization**.
3. **Intent Prediction** - The trained **Random Forest model** predicts the most relevant intent.
4. **Response Retrieval** - A random response corresponding to the predicted intent is selected and returned to the user.

**Code for Intent Prediction:**

def predict\_intent(user\_input):

user\_input\_vec = vectorizer.transform([user\_input])

prediction = model.predict(user\_input\_vec)

return prediction[0]

**Enhancements & Future Improvements**

To ensure the chatbot remains effective, future iterations will focus on the following improvements:

1. **Improve Intent Classification Accuracy** - Experiment with **deep learning models** (e.g., LSTMs, Transformers) for better intent detection.
2. **Expand the Dataset** - Increase the dataset size by adding more patterns and responses for each intent.
3. **Implement a Database Backend** - Replace JSON storage with a **SQL or NoSQL database** for scalable data management.
4. **Introduce Context Awareness** - Enhance conversation flow by implementing **session-based context tracking**.
5. **Optimize API Performance** - Improve response time using caching mechanisms.
6. **Multi-Language Support** - Extend the chatbot’s capability to handle multiple languages using **NLP translation models**.
7. **Logging & Analytics** - Implement logging to track chatbot interactions and analyze user behavior.

**Conclusion**

This chatbot backend leverages **machine learning and Flask** to provide an intelligent and modular system for handling user interactions. The structured training process ensures accurate intent classification, and the Flask API allows seamless chatbot communication. With ongoing improvements, the chatbot will evolve into a more sophisticated AI-powered assistant.