1. Project Structure

The project is organized into several components:

- Training Data: Multiple JSON files containing intents, patterns, and responses.

- Model Training: A script to train the chatbot model using Sentence-BERT and Nearest Neighbors.

- IPC (Indian Penal Code) Module: A separate module to handle legal queries related to IPC sections.

- Web Interface: A Flask-based web server to serve the chatbot and IPC query functionality.

- Terminal Interface: A command-line interface for interacting with the chatbot.

- HTML/CSS/JavaScript: Frontend code for the chatbot and IPC query interface.

---

2. Key Components

A. Training the Chatbot Model

- Data Loading: The `load_data()` function loads training data from multiple JSON files.

- Text Preprocessing: The `preprocess_text()` function tokenizes, removes stopwords, and cleans the input text.

- Data Augmentation: The `augment_data()` function (placeholder) is designed to enhance the dataset with paraphrased sentences.

- Model Training:

  - Uses Sentence-BERT (`all-MiniLM-L6-v2`) to generate embeddings for the training data.

  - Trains a Nearest Neighbors model to find the closest matching intent based on semantic similarity.

  - Saves the trained model, embeddings, and label encoder using `pickle`.

B. IPC Module

- Data Loading: Loads IPC data from a CSV file.

- Text Preprocessing: Cleans and tokenizes the "Offense" column for better matching.

- Search Methods:

  - BM25: For keyword-based search.

  - Sentence-BERT: For semantic similarity.

  - TF-IDF: For vector space search.

- Combined Search: Combines results from all three methods with weighted scores to provide the best match.

C. Web Interface

- Flask Server: Hosts the chatbot and IPC query endpoints.
  - `/chat`: Handles chatbot queries.
  - `/query`: Handles IPC-related queries.
- Frontend:
  - Chatbot Interface: A responsive web interface with typing effects and real-time responses.
  - IPC Interface: A user-friendly interface to search for IPC sections and offenses.

D. Terminal Interface

- Interactive Chat: Allows users to interact with the chatbot via the terminal.
- Typing Effect: Simulates a typing effect for chatbot responses.

E. Utility Functions

- Background Music: Plays background music during model loading.
- Server Status Check: Continuously checks the status of the Flask server.
- Offline Responses: Provides fallback responses when the server is offline.

---

3. Execution Flow

Step 1: Training the Model

1. Load Data: The `load_data()` function reads training data from JSON files.
2. Preprocess Text: The `preprocess_text()` function cleans and tokenizes the input text.
3. Generate Embeddings: Sentence-BERT generates embeddings for the training data.
4. Train Nearest Neighbors: The Nearest Neighbors model is trained on the embeddings.
5. Save Model: The trained model, embeddings, and label encoder are saved to a `.pkl` file.

Step 2: Running the Chatbot

1. Load Model: The `load_model()` function loads the trained model and associated data.
2. Preprocess User Input: The user's input is cleaned and tokenized.
3. Generate Embedding: Sentence-BERT generates an embedding for the user's input.
4. Find Nearest Neighbor: The Nearest Neighbors model finds the closest matching intent.
5. Generate Response: A response is selected from the matching intent's responses.

Step 3: IPC Query Handling

1. Preprocess Query: The user's query is cleaned and tokenized.

2. Search Methods:

   - BM25: Scores documents based on keyword matches.

   - Sentence-BERT: Computes semantic similarity.

   - TF-IDF: Computes vector space similarity.

3. Combine Results: Scores from all methods are combined to find the best match.

4. Return Response: The best match and similar sections are returned.

Step 4: Web Interface

1. Start Flask Server: The Flask app hosts the chatbot and IPC query endpoints.

2. Frontend Interaction:

   - Users interact with the chatbot or IPC search interface.

   - Queries are sent to the Flask server via API calls.

3. Display Results: Responses are displayed in the web interface.

Step 5: Terminal Interface

1. Start Terminal Chat: The `terminal_chat()` function initiates the chatbot in the terminal.

2. User Interaction: Users type messages, and the chatbot responds with typing effects.

---

4. Real-Time Example

Chatbot Interaction

- User Input: "Hi, how are you?"

- Processing:

  1. Preprocess text: "hi how are you"

  2. Generate embedding using Sentence-BERT.

  3. Find nearest neighbor using Nearest Neighbors.

  4. Select response: "I'm doing well, thank you! How about you?"

- Output: "Chatbot: I'm doing well, thank you! How about you?"

IPC Query

- User Input: "What is the punishment for theft?"
- Processing:
  1. Preprocess text: "punishment theft"
  2. Search using BM25, Sentence-BERT, and TF-IDF.
  3. Combine scores and find the best match.
  4. Return IPC section, offense, and punishment details.
- Output:
  ```
  Best Match:
  IPC Section: 378
  Offense: Theft
  Punishment: Imprisonment up to 3 years or fine or both
  ```

---

5. Complex Example

Multi-Turn Conversation

- User Input 1: "Tell me about IPC section 302."
- Chatbot Response: "IPC Section 302 deals with murder. The punishment is life imprisonment or the death penalty."
- User Input 2: "Is it bailable?"
- Chatbot Response: "No, IPC Section 302 is non-bailable."

Fallback to Offline Mode

- Scenario: Flask server is offline.
- User Input: "What's the weather today?"
- Chatbot Response: "I don't have access to real-time weather data. You might want to check a weather app or website."

---

6. FAQs

1. What is the purpose of this chatbot?

   - The chatbot is designed to assist users with general queries and provide information about IPC sections.

2. How does the chatbot understand user queries?

   - It uses Sentence-BERT for semantic understanding and Nearest Neighbors for intent matching.

3. Can the chatbot handle real-time data?

   - No, it cannot access real-time data like weather or news. It relies on pre-trained models and static data.

4. What happens if the server is offline?

   - The chatbot falls back to offline responses stored in the `offlineResponses` array.

5. How accurate is the IPC search?

   - The IPC search combines BM25, Sentence-BERT, and TF-IDF for high accuracy.

6. Can I add new intents to the chatbot?

   - Yes, you can add new intents to the training data JSON files and retrain the model.

7. How do I run the chatbot locally?

   - Run the `web_server.py` script to start the Flask server and open the HTML file in a browser.

8. What technologies are used in this project?

   - Python, Flask, Sentence-BERT, Nearest Neighbors, NLTK, and Tailwind CSS.

9. Can I customize the chatbot's responses?

   - Yes, you can modify the responses in the training data JSON files.

10. How do I deploy this chatbot?

    - You can deploy the Flask app on platforms like Heroku, AWS, or Google Cloud.

---

7. Troubleshooting

1. Model not loading:

   - Ensure the `.pkl` file exists in the correct path.

   - Check for missing dependencies using `pip install -r requirements.txt`.

2. IPC data not found:

   - Verify that the `ipc_sections.csv` file is in the `datasets` directory.

3. Flask server not starting:

   - Check if port 5000 or 8080 is already in use.

   - Ensure all dependencies are installed.

4. Offline responses not working:

   - Verify that the `offlineResponses` array is correctly defined in the JavaScript code.

5. Typing effect not working:

   - Ensure the `type_effect()` function is correctly implemented in the terminal interface.

……. Created By Praneeth :: Crafted By Praveen …….