**Finance Chatbot**

**File: DataChatBot.json**

Break down the components of the DataChatBot.json file:

1. **Tags**: Tags represent different categories or intents that the chatbot can recognize. Each tag corresponds to a specific type of user query or request. For example, tags could include "greeting," "finance\_information," "investment\_advice," etc.

**Tags**:

- Tags represent different intents or categories of user queries.

- Example tags:

- "greeting": Intended for user greetings.

- "finance\_information": Pertaining to requests for financial information.

- "investment\_advice": For queries related to investment advice.

2. **Patterns**: Patterns are expressions or phrases that users might input when interacting with the chatbot. These patterns help the chatbot understand the user's intent by recognizing common language patterns. For instance, for the "greeting" tag, patterns might include "Hi," "Hello," "Hey there," etc.

**Patterns**:

- Patterns are expressions or phrases used by users.

- They help the chatbot understand user intent.

- Example patterns:

- For "greeting" tag: "Hi," "Hello," "Hey there," etc.

- For "finance\_information" tag: "What is the current stock price of Apple?"

"Show me the latest financial news."

3. **Responses**: Responses are the pre-defined messages or actions that the chatbot will deliver in response to user input. Each pattern is associated with one or more responses, allowing the chatbot to provide appropriate answers or actions based on the detected intent. Responses can include text messages, links to resources, or instructions for further interaction.

**Responses**:

- Responses are pre-defined messages or actions delivered by the chatbot.

- Each pattern is associated with one or more responses.

- Example responses:

- For "greeting" tag: "Hello! How can I assist you today?"

- For "finance\_information" tag: "The current stock price of Apple is $XXX."

**File: speak.py**

This document provides an overview of the speak.py file used in the finance chatbot project.

1. Import Library:

- Import the `pyttsx3` library for text-to-speech conversion.

2. Define Function:

- `Speak(audio)`: This function takes a string as input and speaks it aloud.

3. Initialize Text-to-Speech Engine:

- Initialize the text-to-speech engine using `pyttsx3.init('sapi5')`.

- Set the voice property to the first available voice and the speaking rate to 150 words per minute.

4. Print and Speak:

- Print the provided text as "Buddy: [audio]".

- Use the text-to-speech engine to speak the

**File: train.py**

This document provides an overview of the train.py file used in the finance chatbot project.

1. Import Libraries:

- Import necessary libraries for data processing, machine learning, and natural language processing tasks.

2. Initialize Lemmatizer:

- Initialize the WordNet lemmatizer from NLTK for lemmatizing words.

3. Load Intents:

- Load intents from the DataChatBot.json file containing predefined patterns, tags, and responses for the chatbot.

4. Process Intents:

- Extract patterns, tags, and responses from intents.

- Tokenize patterns, lemmatize words, and preprocess data.

5. Save Processed Data:

- Save processed words and classes to binary files for future use.

6. Prepare Training Data:

- Prepare training data by creating a bag of words representation for each document and converting tags into one-hot encoded vectors.

7. Build Neural Network Model:

- Build a Sequential model using Keras with Dense layers and ReLU activation functions.

8. Compile Model:

- Compile the model using categorical cross-entropy loss and stochastic gradient descent (SGD) optimizer.

9. Train the Model:

- Train the model using the prepared training data with specified epochs and batch size.

- Save the trained model.

10. Print Training Completion Message:

- Print a message indicating that training is completed.

**File: chatbot.py**

This document provides an overview of the chatbot.py file used in the finance chatbot project.

1. Import Libraries:

- Import necessary libraries for data processing, machine learning, natural language processing, and text-to-speech conversion.

2. Load Pre-Trained Model and Tokenizer:

- Load the pre-trained model and tokenizer from disk, which have been previously trained on processed data.

3. Define Functions:

- `clean\_up\_sentence(sentence)`: Tokenizes and lemmatizes input sentences.

- `bag\_of\_words(sentence)`: Converts sentences into a bag of words representation.

- `predict\_class(sentence)`: Predicts the intent of input sentences.

- `get\_response(intent\_list, intents\_json)`: Retrieves a random response corresponding to the predicted intent.

4. Chat Loop:

- Start an infinite loop to continuously accept user input.

- Predict the intent of user input and generate a response based on the predicted intent.

- Convert the response to speech using text-to-speech conversion.

- Terminate the loop if the user enters "quit".