Haritha – The Chat bot

Project Report

Introduction:

Many Popular online food delivery platforms that operate in several countries utilize chatbots to enhance their customer experience and provide quick and efficient support to their users. This project aims to create a mini - Chatbot named **Haritha** with the intension to mimic the chatbots of food delivery platforms

Motivation:

The purpose of creating mini chatbots that mimic food delivery apps was to enhance the user experience, streamline customer support processes, and optimize resource allocation. These chatbots offer a familiar interface for users, allowing them to interact seamlessly and obtain information efficiently. By automating routine tasks, the chatbots free up human resources to focus on more complex inquiries, resulting in improved customer satisfaction and faster response times.

Data set:

This dataset consists of a collection of intents and their corresponding patterns and responses. Each intent represents a specific topic or user query that the chatbot is designed to understand and respond to. The intents in this dataset cover a variety of topics related to customer support in the context of a service or food delivery platform. Some of the topics covered include greetings, farewells, expressing gratitude, general information about the chatbot, account creation, order tracking, complaints, invoice requests, quality and quantity issues with food, packaging problems, delivery agent behavior, order-related inquiries, customer care contact information, order cancellation, refunds, operational hours, discounts, customization options, payment methods, delivery charges, delivery time, ordering from multiple restaurants, item unavailability, and more.

Procedure:

The implemented chatbot in the provided code is a retrieval-based chatbot. It retrieves pre-existing responses from a predefined set of responses based on the identified intent using techniques like bag-of-words representation and similarity matching. It does not generate responses dynamically based on the conversation context. It is a Task-oriented (declarative) chatbot

1.Import libraries and load intents data.

2.Preprocess the data: tokenize, lemmatize, and create vocabulary and class lists.

3.Generate training data: create a bag of words representation for each pattern and associate it with the corresponding class.

4.Build and train a deep learning model using the training data.

5.Define utility functions for text cleaning, bag of words generation, and predicting classes.

6.Enter a loop to interact with the user:

- Get user input.
- Predict intents based on the input and retrieve a response.
- Display the response.
- Repeat until an exit command is given.

Model:

The training model used in the code has the following architecture:

1. Input Layer: Receives the bag of words representation of the patterns.
2. Dense Layer 1: Fully connected layer with 128 units and ReLU activation.
3. Dropout Layer 1: Regularization to prevent overfitting (dropout rate: 50%).
4. Dense Layer 2: Fully connected layer with 64 units and ReLU activation.
5. Dropout Layer 2: Regularization (dropout rate: 30%).
6. Output Layer: Number of units equal to the unique classes, with softmax activation.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 128)               21248

 dropout_10 (Dropout)        (None, 128)               0

 dense_16 (Dense)            (None, 64)                8256

 dropout_11 (Dropout)        (None, 64)                0

 dense_17 (Dense)            (None, 30)                1950

=================================================================
Total params: 31,454
Trainable params: 31,454
Non-trainable params: 0
```

The model uses the categorical cross-entropy loss function, which is suitable for multi-class classification problems, and the Adam optimizer (`tf.keras.optimizers.legacy.Adam`). The optimizer adjusts the learning rate during training to optimize the model's parameters.

Overall, the architecture follows a sequential model structure, where the layers are stacked on top of each other in a linear manner. The use of dense layers with ReLU activation and dropout regularization helps the model learn and generalize patterns from the input data, leading to accurate classification results.

Tech Stack used:

The code you provided uses the following tech stack:

1. Python: The code is written in Python, which is a popular programming language for machine learning and natural language processing tasks.
2. JSON: The **json** library is used to load the intents from a JSON file. JSON (JavaScript Object Notation) is a lightweight data interchange format commonly used for storing and transferring data.
3. NLTK (Natural Language Toolkit): NLTK is a Python library for natural language processing. In this code, NLTK is used for tokenizing text, lemmatizing words, and downloading necessary resources like wordnet.
4. NumPy: NumPy is a Python library for numerical computing. It provides high-performance multidimensional array objects and tools for working with arrays. In this code, NumPy is used for creating and manipulating arrays.
5. TensorFlow: TensorFlow is an open-source deep learning library developed by Google. It provides a flexible framework for building and training machine learning models. In this code, TensorFlow is used to create and train a sequential neural network model.
6. Keras: Keras is a high-level neural networks API written in Python and running on top of TensorFlow. It provides an easy-to-use interface for building and training deep learning models. The code uses Keras to define the architecture of the neural network model and compile it with the appropriate optimizer and loss function.
7. Matplotlib: Matplotlib is a plotting library for Python. It provides a wide variety of customizable plots and charts. In this code, Matplotlib is used to plot the loss and accuracy values during the training process.

These libraries and tools are commonly used in natural language processing and machine learning tasks and provide the necessary functionality for building and training a chatbot model.

Concepts implemented :

The project you provided applies several key concepts and techniques in natural language processing (NLP) and machine learning. Here are the main concepts used in the code:

1. Tokenization: The code utilizes tokenization, which is the process of breaking down text into individual tokens or words. It uses the **nltk.word_tokenize()** function from NLTK to tokenize the patterns and user input.
2. Word Lemmatization: Lemmatization is the process of reducing words to their base or root form. The code applies lemmatization using the WordNet lemmatizer from NLTK (**WordNetLemmatizer**). It helps to handle different word forms and reduces the vocabulary size.
3. Bag-of-Words (BoW) Model: The code implements a Bag-of-Words representation, where each pattern is converted into a binary vector indicating the presence or absence of words from the vocabulary. It creates a BoW representation for each pattern and encodes the associated class label.
4. Deep Learning with Neural Networks: The code builds a deep learning model using the **Sequential** API from Keras (which runs on top of TensorFlow). The model

architecture consists of dense layers with ReLU activation and dropout regularization. It uses the softmax activation in the output layer for multi-class classification.

5. Categorical Cross-Entropy Loss: The code employs categorical cross-entropy as the loss function for training the neural network model. This loss function is commonly used for multi-class classification problems.
6. Adam Optimizer: The code utilizes the Adam optimizer (**tf.keras.optimizers.legacy.Adam**) for updating the weights of the neural network during training. The optimizer adjusts the learning rate automatically and helps optimize the model parameters.
7. Training and Validation: The code splits the training data into features (**train_X**) and labels (**train_y**). It then trains the model on the training data for a specified number of epochs and uses the validation set to monitor the loss and accuracy during training.
8. Plotting Training History: After training the model, the code uses Matplotlib to visualize the training history by plotting the loss and accuracy values over epochs. This provides insights into the model's performance and helps in evaluating and fine-tuning the model.
9. Intent Classification: The code applies the trained model for intent classification. It tokenizes the user input, converts it into a bag of words representation, and uses the model to predict the associated intents. It selects the most probable intents based on a predefined threshold and retrieves a random response for the selected intent.

These concepts and techniques are fundamental in NLP and machine learning and are commonly used for building chatbots, text classification models, and other NLP applications.

Challenges:

Creating a dataset for this chatbot involves several challenges. Here are some of the challenges commonly faced in the process:

1. Data Collection: Gathering relevant and high-quality data.
2. Annotation and Labeling: Ensuring accurate labeling of intents and entities.
3. Balancing the Dataset: Representing various intents and query types equally.
4. Language and Context Variations: Accounting for different expressions and contexts.
5. Handling Ambiguity: Training the chatbot to handle ambiguous queries.
6. Regular Updates: Maintaining and enhancing the dataset over time.
7. Privacy and Legal Considerations: Adhering to privacy regulations.
8. Cost and Time Constraints: Resource-intensive process with associated costs.

Addressing these challenges requires a systematic approach, domain expertise, and ongoing efforts to refine and improve the dataset for optimal chatbot performance.

Impact:

The potential impact of this project can be seen in the following aspects:

1. Chatbot Applications: The project provides a foundation for building chatbot applications. By training the model on specific intents and responses, the chatbot can

effectively understand and respond to user queries, providing assistance and information in a conversational manner. This can be valuable in customer support, virtual assistants, and various other interactive applications.

2. Natural Language Understanding: The project demonstrates techniques for natural language understanding (NLU). By tokenizing, lemmatizing, and creating a bag-of-words representation, the model can interpret and classify user inputs based on their intents. This capability can be extended to more complex NLU tasks, such as sentiment analysis, topic classification, and entity recognition.
3. Personalization and Customization: The chatbot model can be trained on specific datasets, allowing for personalization and customization. By adding more intents, patterns, and responses relevant to a particular domain or application, the chatbot can provide tailored and context-aware interactions, enhancing user experience and engagement.
4. Scalability and Adaptability: The deep learning architecture employed in the project allows for scalability and adaptability. With a larger and more diverse training dataset, the model can be trained to handle a wider range of intents and user inputs.
5. Automation and Efficiency: By automating responses to common user queries, the chatbot reduces the need for human intervention, leading to increased efficiency and productivity. It can handle a large volume of inquiries simultaneously, providing quick and consistent responses. This can be especially beneficial in scenarios where repetitive tasks or information retrieval are involved.

Overall, this project demonstrates the power of NLP and deep learning techniques in developing intelligent chatbot systems with various real-world applications.

Results :

```
User: hello
1/1 [==============================] - 0s 63ms/step
Bot: Hello
User: what is your name
1/1 [==============================] - 0s 28ms/step
Bot: You can call me Haritha.
User: i need a help
1/1 [==============================] - 0s 22ms/step
Bot: Yes, sure! How can I support you?
User: what will be the delivery charge for an order
1/1 [==============================] - 0s 22ms/step
Bot: Delivery charges may vary depending on the restaurant and your location. Some restaurants offer free delivery, while others may have a nominal fee.
User: how to get the order id
1/1 [==============================] - 0s 24ms/step
Bot: Your order ID can be found in the confirmation email or SMS you received after placing the order. It is usually a unique alphanumeric code.
User: what is customer care no.
1/1 [==============================] - 0s 26ms/step
Bot: You can contact our customer care team at 9444070589. They are available to help you with any issues or inquiries you may have.
User: ok thanks
1/1 [==============================] - 0s 32ms/step
Bot: Any time!
User: quit
Bot: Goodbye!
```

With 500 Epochs : loss: 0.0219 and accuracy: 0.9900

Future extension :

The future extensions of this project could include:

1. Increasing the training data: Adding more diverse and representative data to the training dataset can improve the performance and accuracy of the chatbot. This can involve collecting more user queries and intents to cover a wide range of scenarios.
2. Exploring different neural network architectures: Experimenting with different architectures, such as recurrent neural networks (RNNs) or transformer models, can provide insights into which model performs best for the chatbot application. This can involve adjusting the network layers, activation functions, and other hyperparameters to optimize the model's performance.
3. Incorporating emojis: Including emojis in the chatbot's responses can add a touch of personality and enhance the user experience. Emojis can help convey emotions, reinforce context, or provide visual cues, making the interaction more engaging and expressive.
4. Integration with web and app: Extending the chatbot's functionality by integrating it with web and mobile applications can provide users with seamless access to the chatbot across different platforms. This can involve building APIs or using chatbot frameworks that allow integration with various channels, such as websites, messaging platforms, or mobile apps.
5. The project could expand by integrating speech capabilities through an API. This would allow the chatbot to process text-based responses, handle speech input, and provide speech output. By leveraging a speech recognition API, users can convert spoken queries into text, making interaction more natural and convenient. Similarly, integrating a text-to-speech (TTS) API would convert responses into spoken words, providing an interactive and accessible experience. This would expand the chatbot's functionality and adapt to different user preferences and accessibility needs, enhancing usability and user satisfaction.

References:

https://www.section.io/engineering-education/creating-chatbot-using-natural-language-processing-in-python/

https://towardsdatascience.com/a-simple-chatbot-in-python-with-deep-learning-3e8669997758

https://towardsdatascience.com/how-to-build-your-own-chatbot-using-deep-learning-bb41f970e281

https://medium.com/@sajalvictorious83/how-does-swiggy-chatbot-work-167c98ee51c0

https://www.techwithtim.net/tutorials/ai-chatbot/chat-bot-part-4