E-Commerce Chatbot

MACHINE LEARNING AND NLP

Faizan Ahmed

Contents

Project Name: E-commerce Chatbot

Problem Statement

A chatbot powered by artificial intelligence is referred to as an AI chatbot (AI). AI chatbots, in contrast to conventional chatbots, can comprehend user inquiries using natural language processing (NLP) and provide insightful responses, enhancing the general user experience. In this project, we will be developing an e-commerce chatbot specifically designed for websites like (anyStore), ensuring answers to users' questions related to their purchases, accounts, payments, tracking, and more. Users can learn about offers and advantages of making payments online from the bot. Chatbots on e-commerce websites provide answers to frequently asked questions, collect user evaluations, and handle challenging client inquiries. These are simply made to reduce the clutter a customer might run into when shopping.

Methodology

There are many kinds of chatbots accessible, but the following category is implemented:

- Text-based chatbot: A text-based chatbot responds to user inquiries using a text-based interface.

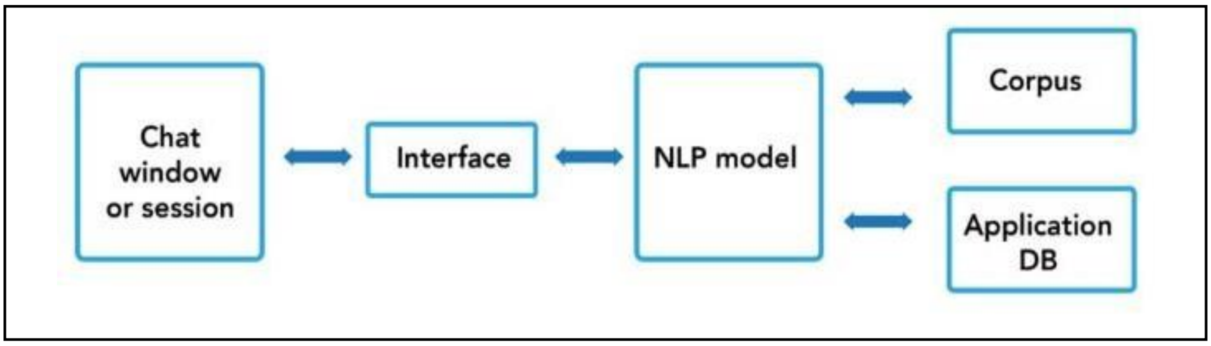
The design of chatbots primarily follows two methods, which are as follows:

- **In a rule-based method**, a bot responds to queries in accordance with some pre-trained rules. The rules specified might range in complexity from very simple to highly complicated. Simple inquiries are handled by the bots, but complicated ones are not.
- **Self-learning bots**, which employ a few machine learning-based techniques, are unquestionably more effective than rule-based bots.

Typical chatbot architecture should consist of the following:

- Chat window/ session/ or front-end application interface.
- The deep learning model for Natural Language Processing [NLP].
- Corpus or training data for training the NLP model.

Architecture of the chatbot:



Back-End working of the Chatbot:

Architecture Overview:

The chatbot's architecture can be divided into the following components:

Data Preprocessing:

- The NLTK library is used for natural language processing tasks such as tokenization and lemmatization.
- Intents are loaded from a JSON file (intents.json) containing patterns and corresponding tags.
- Words are tokenized, lemmatized, and processed to create a vocabulary and a set of classes.

Model Training:

- The training data is prepared by converting input patterns into numerical vectors using a bag-of-words approach.
- A neural network model is constructed using the Keras library. The model consists of densely connected layers with ReLU activation functions and dropout layers to prevent overfitting.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss function.
- Training data is fed into the model for a specified number of epochs and batch size.

Model Evaluation and Persistence:

- Model performance metrics such as accuracy are tracked during training.
- Once training is complete, the model is saved to a file (chatbot_model.h5) for later use.

Chatbot Interaction:

- At runtime, the saved model is loaded along with the preprocessed vocabulary and classes.
- User input is tokenized and converted into a numerical vector using the same bag-of-words approach used during training.
- The model predicts the intent of the user input, and the appropriate response is retrieved from the predefined intents.
- The chatbot's response is presented to the user, completing the interaction loop.

Functionality:

- The chatbot is capable of the following functionalities:

- Understanding user inquiries using natural language processing techniques.
- Providing insightful responses based on predefined intents related to e-commerce tasks such as purchases, account management, payments, and order tracking.
- Handling frequently asked questions and providing relevant information to users.
- Collecting user feedback and improving the user experience over time.
- Educating users about offers and benefits of online payments.
- Reducing clutter and enhancing the shopping experience by addressing challenging customer inquiries.

Implementation Details:

- The implementation is based on Python programming language.
- Key libraries and frameworks used include NLTK for natural language processing, Keras for building and training neural network models, and JSON for storing intent data.
- The chatbot's training process involves converting textual input into numerical vectors, constructing a neural network model, training the model on labeled data, and persisting the trained model for future use.
- During runtime, the chatbot loads the trained model and necessary data files, processes user input, predicts the intent, and retrieves an appropriate response from the predefined intents.

Front-End Implementation

Architecture Overview:

Importing Libraries:

- The code starts by importing necessary libraries such as Streamlit for building the user interface, NLTK for natural language processing, and Keras for loading the trained chatbot model.
- Loading Resources:

- The pre-trained chatbot model (chatbot_model.h5), vocabulary (words.pkl), classes (classes.pkl), and intents (intents.json) are loaded into memory.
- Session State Initialization:

- The session state is used to maintain chat history (chat_history) and suggestions for the user (suggestions). If they don't exist, they are initialized.

Chatbot Functions:

- clean_up_sentence: Tokenizes and lemmatizes user input for processing.
- bow: Converts user input into a bag-of-words representation for prediction.
- predict_class: Predicts the intent of user input using the loaded model.
- getResponse: Retrieves a response from the predefined intents based on the predicted intent.
- Streamlit GUI:

- The Streamlit GUI consists of a title, text input for user prompts, a submit button, a sidebar for suggestions, and a container for displaying the chat history.
- User input is stored in the session state and displayed in the chat history.
- Upon submission, the chatbot processes the user input, generates a response, and updates the chat history accordingly.
- Suggestions for the next possible question are displayed in the sidebar as buttons. Clicking on a suggestion updates the user input field.

Functionality:

User Interaction:

- Users interact with the chatbot by entering prompts in the text input field provided in the frontend interface.
- These prompts can include questions, inquiries, or requests related to their shopping experience on the e-commerce platform.

Chatbot Response:

- Upon receiving a user prompt, the chatbot's backend model processes the input using natural language processing techniques.
- It predicts the intent of the user input and generates an appropriate response based on the predefined intents stored in the dataset.
- The response is then displayed in the chat history container in the frontend interface, allowing users to view the chatbot's reply.

Suggestions for the Similar Questions:

- In addition to providing responses, the chatbot may also suggest similar possible questions or actions for the user.
- These suggestions are based on the current context of the conversation and aim to

guide the user in continuing the interaction flow smoothly.
- Suggestions are displayed in the sidebar as clickable buttons, allowing users to easily select and follow up with the recommended questions or actions.

Implementation Details:

- The implementation is based on Python programming language.
- Streamlit is used to create the user interface, allowing for easy deployment and interaction.
- The chatbot model is loaded using Keras, and NLTK is used for text preprocessing.
- User input is processed, and the predicted intent is used to retrieve a response from predefined intents stored in a JSON file.
- The GUI components are dynamically updated based on user interactions and chatbot responses.

Code snippets:

Backend code:

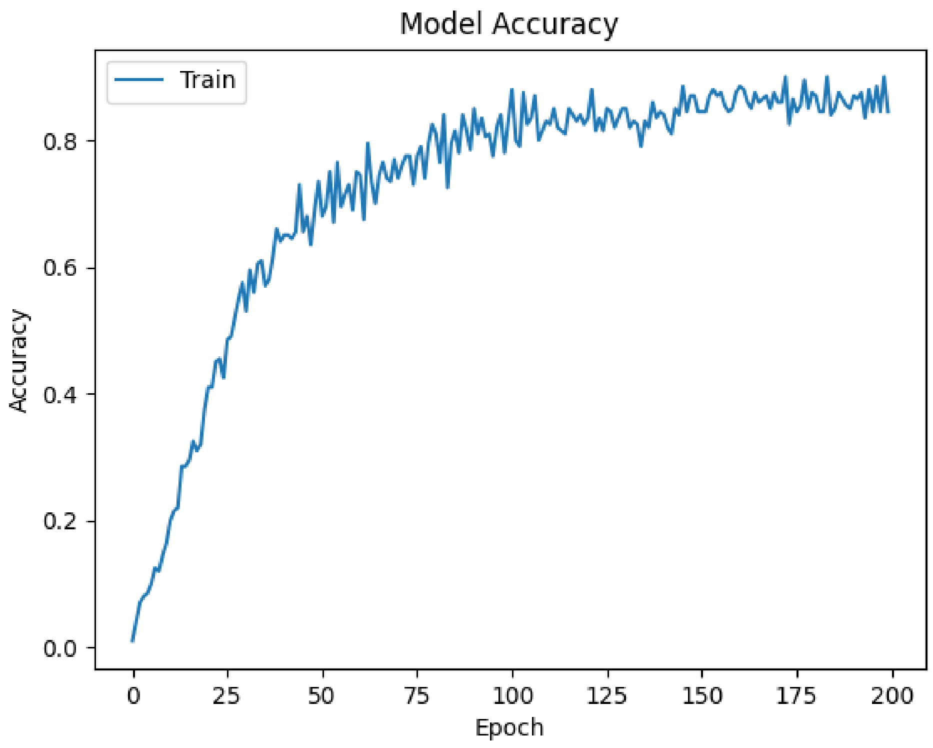
```python
import random  # Importing the random module
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle
import numpy as np
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout

lemmatizer = WordNetLemmatizer()
nltk.download('punkt')
nltk.download('wordnet')

# Load intents from JSON (correcting the path)
intents = json.loads(open('intents.json').read())

# Initialize lists
words = []
classes = []
documents = []
ignore_words = ['?', '!']

# Process intents
for intent in intents['intents']:
    for pattern in intent['patterns']:
        # Tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        # Add documents in the corpus
        documents.append((w, intent['tag']))
        # Add to classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

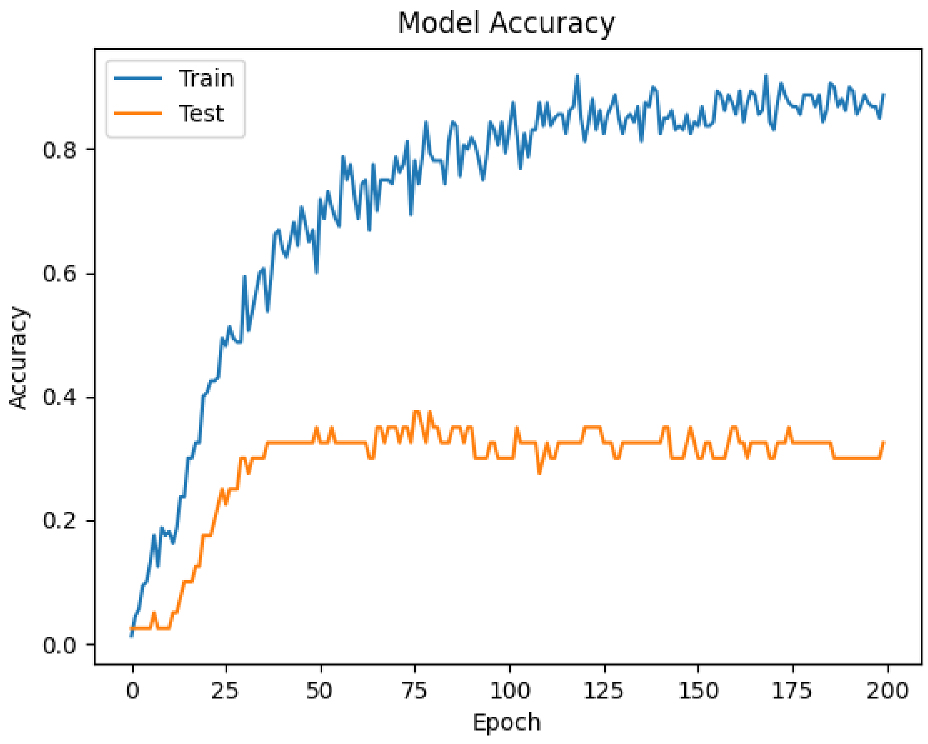
```python
# Lemmatize, lowercase, and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in
ignore_words]
words = sorted(list(set(words)))
classes = sorted(list(set(classes)))

# Save words and classes to pickle files
pickle.dump(words, open('words.pkl', 'wb'))
pickle.dump(classes, open('classes.pkl', 'wb'))

random.shuffle(documents)

# Initialize lists for training
train_x = []
train_y = []

# Generate training data
for doc in documents:
    bag = [0] * len(words)
    pattern_words = doc[0]
    intent = doc[1]
    for word in pattern_words:
        if word in words:
            bag[words.index(word)] = 1
    train_x.append(bag)
    label = [0] * len(classes)
    label[classes.index(intent)] = 1
    train_y.append(label)

# Convert lists to numpy arrays
train_x = np.array(train_x)
train_y = np.array(train_y)

# Define and compile the model
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Fit the model to the data
history = model.fit(train_x, train_y, epochs=200, batch_size=8, verbose=1)

# Save the model
model.save('chatbot_model.h5')

print("Model training completed. Model saved as 'chatbot_model.h5'.")

# Load the model and necessary files
model = load_model('chatbot_model.h5')
words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))
```

Front-End Code:

```python
import streamlit as st
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle
import numpy as np
from keras.models import load_model

# Load necessary resources
lemmatizer = WordNetLemmatizer()
nltk.download('punkt')
nltk.download('wordnet')
model = load_model('chatbot_model.h5')

words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))
intents = json.loads(open('intents.json').read())

# Initialize chat history and suggestions
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "suggestions" not in st.session_state:
    st.session_state.suggestions = []

# Function to update user input
def update_user_input(value):
    st.session_state.user_input = value

# Chatbot functions
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
    return sentence_words

def bow(sentence, words, show_details=True):
    sentence_words = clean_up_sentence(sentence)
    bag = [0] * len(words)
    for s in sentence_words:
        for i, w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    p = bow(sentence, words, show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return [{"intent": classes[r[0]], "probability": str(r[1])} for r in
results]

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if i['tag'] == tag:
            result = np.random.choice(i['responses'])
            suggestions = i['patterns']  # Suggestions for next possible
question
            break
    return result, suggestions

# Streamlit GUI

st.title("E-Commerce Store Assistant")

def main():
    # Sidebar for suggestions
    st.sidebar.title("Suggestions")

    # Container for chat history
    chat_container = st.container()

    # Text input for user prompt
    if "user_input" not in st.session_state:
        st.session_state.user_input = ""
    user_input = st.text_input("Prompt:",
value=st.session_state.user_input, key="user_input")

    # Clear existing suggestions
    st.session_state.suggestions = []

    # Submit button
    if st.button("Send") and user_input.strip() != "":
        st.session_state.chat_history.append({"user": user_input})
        response, suggestions = chatbot_response(user_input)
        st.session_state.chat_history.append({"bot": response})
        if suggestions:
            st.session_state.suggestions = suggestions

    # Display suggestions as buttons
    for suggestion in st.session_state.suggestions:
        if st.sidebar.button(suggestion):
            update_user_input(suggestion)

    # Display entire chat history
    with chat_container:
        for item in st.session_state.chat_history:
            if "user" in item:
                st.image("user.png", width=32)
                st.markdown(f"{item['user']}")
            elif "bot" in item:
                st.image("aiChatbot.png", width=32)
                st.markdown(f"{item['bot']}")



def chatbot_response(text):
    ints = predict_class(text, model)
    if ints:
        res, suggestions = getResponse(ints, intents)
        return res, suggestions
    else:
        return "I'm sorry, I didn't understand that.", None

if __name__ == "__main__":
    main()
```

Accuracy Graph of Model:





some techniques you can try to increase validation accuracy:

Technique	Description
Data Augmentation	Generate variations of existing data
Regularization	Techniques like dropout, L2 regularization
Hyperparameter Tuning	Optimize learning rate, batch size, architecture
Early Stopping	Stop training when validation loss increases
Batch Normalization	Normalize activations of each layer
Increase Data	Gather more data
Model Architecture	Experiment with different architectures
Transfer Learning	Fine-tune pre-trained models

Conclusion:

Project Overview:

The E-commerce Chatbot project aims to enhance user experience on e-commerce platforms through conversational AI. By leveraging natural language processing (NLP), the chatbot provides insightful responses to user inquiries related to purchases, accounts, payments, and more.

Functionality:
The chatbot's capabilities include understanding user queries, providing informative responses, handling FAQs, and promoting online payment benefits. Suggestions for similar questions aid in smooth conversation flow.

Implementation Details:
Built using Python and Streamlit, the chatbot employs NLTK for NLP and Keras for model training. User input is dynamically processed, intents are predicted, and responses are retrieved, creating a seamless interaction experience.

Future Considerations:
To enhance the chatbot's performance, several techniques can be explored, including data augmentation, regularization, hyperparameter tuning, and model architecture adjustments. Continuous monitoring and iteration based on user feedback are crucial for refining the chatbot's accuracy and user satisfaction. Deployment considerations such as scalability, latency optimization, and security measures should be prioritized for real-world implementation.

In conclusion, the E-commerce Chatbot project demonstrates the potential of AI-driven conversational interfaces to streamline user interactions and improve customer engagement on e-commerce platforms. With ongoing refinement and strategic deployment, the chatbot holds promise for enhancing the shopping experience and driving business growth in the digital marketplace.