CREATE A CHATBOT IN PYTHON

SUBMITTED BY

SHANMUGAPPRIYK(311521106089)

SUBIDSHA S C(311521106100)

SWARNALAKSHMI E(311521106102)

SOUNDHARYA B(311521106094)

PHASE 3 SUBMISSION :DEVELOPMENT PART 1

Introduction:

Chatbot is a computer program that humans will interact with in natural spoken language and including artificial intelligence techniques such as NLP (Natural language processing) that makes the chatbot more interactive and more reliable.

Based on the recent epidemiological situation, the increasing demand and reliance on electronic education has become very difficult to access to the university due to the curfew imposed, and this has led to limited access to information for academics at the university.

This project aims to build a chatbot for Admission and Registration to answer every person who asks about the university, colleges, majors and admission policy.

Importing packages:

To create a simple chatbot in Python, you can use a variety of packages and libraries, including natural language processing (NLP) tools and frameworks. Here's a basic example using the NLTK library and scikit-learn:

1. First, you need to install the necessary libraries if you haven't already. You can do this using pip:

```bash
pip install nltk scikit-learn
```

2. Next, you can create a Python script to import the required packages, load a dataset for creating a chatbot, and perform basic preprocessing. In this example, we'll use a simple dataset of predefined responses for specific inputs.

```python
import nltk
import numpy as np
import random
import string

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download the NLTK data
nltk.download('punkt')
nltk.download('wordnet')

# Define a set of responses or dataset
dataset = {
    "hello": ["Hi there!", "Hello!", "Hey!"],
```

```
    "how are you": ["I'm good, thanks for asking.", "I'm doing well.", "I'm fine, how about you?"],
    "what's your name": ["I'm just a chatbot.", "You can call me ChatGPT.", "I'm your virtual assistant."],
    "bye": ["Goodbye!", "See you later!", "Bye now!"],
}

# Preprocess the dataset
def preprocess(text):
    text = text.lower()
    text = ''.join([char for char in text if char not in string.punctuation])
    return text

# Create a function to generate a response
def generate_response(user_input):
    user_input = preprocess(user_input)
    response = "I'm sorry, I don't understand."

    # Check for a known input
    for key in dataset:
        if user_input in key:
            response = random.choice(dataset[key])
            break

    return response

# Main loop for chat
while True:
```

```
    user_input = input("You: ")
    if user_input.lower() == 'exit':
        print("Chatbot: Goodbye!")
        break
    response = generate_response(user_input)
    print("Chatbot:", response)
```

This is a basic example of a chatbot using predefined responses. In a more advanced chatbot, you would typically use machine learning models, such as deep learning-based approaches like transformers, and more extensive datasets. You can also integrate the chatbot into different platforms or applications as needed.

In the previous example, I provided a simple dataset of predefined responses for specific inputs directly in the code. If you want to load an external dataset, you can modify the code to read the dataset from a file, such as a JSON or CSV file. Here's how you can load a dataset from a JSON file:

PREPROCESSING THE DATA:

1. Create a JSON file (e.g., `chatbot_dataset.json`) with your dataset. The JSON file might look like this:

```json
{
    "hello": ["Hi there!", "Hello!", "Hey!"],
    "how are you": ["I'm good, thanks for asking.", "I'm doing well.", "I'm fine, how about you?"],
    "what's your name": ["I'm just a chatbot.", "You can call me ChatGPT.", "I'm your virtual assistant."],
    "bye": ["Goodbye!", "See you later!", "Bye now!"]
}
```

2. Modify the Python script to load the dataset from the JSON file:

```python
import nltk
import numpy as np
import random
import string
import json

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Download the NLTK data
nltk.download('punkt')
nltk.download('wordnet')

# Load the dataset from a JSON file
with open('chatbot_dataset.json', 'r') as file:
    dataset = json.load(file)

# ... rest of the code remains the same
```

With this modification, your chatbot will use the dataset provided in the `chatbot_dataset.json` file for responses. You can easily expand or customize your dataset by editing the JSON file without changing the code.

DEVELOPING A CHATBOT:

Building a Chatbot in Python using TensorFlow and NLTK

Step 1: Set up a Development Environment

To get started, you'll need to set up your development environment. This tutorial uses Python 3. You can download Python 3 from the official website (https://www.python.org/downloads/) and install it on your machine. Once Python is installed, you can install the necessary packages by running these commands in your terminal or command prompt:

bash

Copy code

pip install nltk

pip install numpy

pip install tensorflow

Step 2: Define the Problem Statement

The first step in building a chatbot is to define the problem statement. In this tutorial, we will create a simple chatbot that can answer basic questions about a specific topic. The chatbot should understand questions and provide relevant answers.

Step 3: Collect and Preprocess Data

Next, you need to collect and preprocess data. For this tutorial, we will use a dataset of questions and answers related to programming. You can download the dataset from this link.

Once you have the dataset, use NLTK to preprocess the data. Here's the code to preprocess the data:

python

Copy code

import nltk

from nltk.stem import WordNetLemmatizer

```
from nltk.corpus import stopwords
import string

# Download NLTK data
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

# Load data
with open('data.txt', 'r', encoding='utf-8') as f:
    raw_data = f.read()

# Preprocess data
def preprocess(data):
    tokens = nltk.word_tokenize(data)  # Tokenize data
    tokens = [word.lower() for word in tokens]  # Lowercase all words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words and word not in
string.punctuation]  # Remove stopwords and punctuation
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]  # Lemmatize words
    return tokens

# Preprocess data
processed_data = [preprocess(qa) for qa in raw_data.split('\n')]
```

In this code, we download the necessary NLTK data, load the dataset, and preprocess it by tokenizing, lowercasing, removing stopwords and punctuation, and lemmatizing words.

Step 4: Train a Machine Learning Model

The next step is to train a machine learning model using TensorFlow. We will use the processed data to train a neural network. Here's the code to train the model:

python

Copy code

```
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences


# Set parameters

vocab_size = 5000

embedding_dim = 64

max_length = 100

trunc_type = 'post'

padding_type = 'post'

oov_tok = "<OOV>"

training_size = len(processed_data)


# Create tokenizer

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)

tokenizer.fit_on_texts(processed_data)

word_index = tokenizer.word_index


# Create sequences

sequences = tokenizer.texts_to_sequences(processed_data)

padded_sequences = pad_sequences(sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type)
```

```
# Create training data
training_data = padded_sequences[:training_size]
training_labels = padded_sequences[:training_size]

# Build model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])

# Compile model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train model
num_epochs = 50
history = model.fit(training_data, training_labels, epochs=num_epochs, verbose=2)
```

In this code, we set model parameters, create a tokenizer, generate sequences, and pad them. We then build a neural network model using TensorFlow's Keras Sequential API, compile the model, and train it for 50 epochs.

Step 5: Build the Chatbot Interface

Next, we create a simple command-line interface for our chatbot. Here's the code:

python

Copy code

```
# Define function to predict answer
def predict_answer(model, tokenizer, question):
    # Preprocess question
    question = preprocess(question)
    # Convert question to sequence
    sequence = tokenizer.texts_to_sequences([question])
    # Pad sequence
    padded_sequence = pad_sequences(sequence, maxlen=max_length, padding=padding_type, truncating=trunc_type)
    # Predict answer
    pred = model.predict(padded_sequence)[0]
    # Get index of highest probability
    idx = np.argmax(pred)
    # Get answer
    answer = tokenizer.index_word[idx]
    return answer

# Start chatbot
while True:
    question = input('You: ')
    answer = predict_answer(model, tokenizer, question)
    print('Chatbot:', answer)
```

In this code, we define a predict_answer function that preprocesses the user's question, converts it to a sequence, and predicts an answer using the trained model and tokenizer. We

then create a simple command-line interface for the chatbot that takes user input, predicts answers, and displays them.

Step 6: Test the Chatbot

Now, you can run the code to test your chatbot. Start a conversation with it and see how it responds. Keep in mind that the chatbot's responses may not be meaningful or coherent due to the simplicity of the model and the limited training data.

CONCLUSION:

In this example, we've created a simple chatbot in Python using the NLTK library and a predefined dataset of responses. The chatbot takes user input, preprocesses it, and searches for a matching input in the dataset to generate an appropriate response. This basic implementation serves as a starting point for building more advanced chatbots.

To enhance this chatbot, you can consider the following improvements:

1. Integration with NLP frameworks: Employ more advanced NLP tools and frameworks, such as spaCy or Hugging Face Transformers, to enable your chatbot to better understand and generate human-like responses.

2. Machine Learning: Train your chatbot on a larger and more diverse dataset or use machine learning techniques, including deep learning models like GPT, to create a chatbot with improved language understanding and generation capabilities.

3. Real-time interaction: Develop a chatbot that can be integrated into websites, applications, or messaging platforms for real-time interactions with users.

4. Personalization: Implement user-specific responses and personalized recommendations by storing user data and preferences.

5. Extensive datasets: Use larger and more comprehensive datasets to make your chatbot more versatile and context-aware.

Remember that the key to a successful chatbot lies in its ability to understand user input,

provide relevant responses, and continuously learn and adapt to user interactions. This example serves as a foundation for creating more sophisticated and capable chatbots for a wide range of applications.