Create a chatbot in python....

naan· mudhalvan· project

Introduction:

Python chatbots have exploded in popularity in the technology and commercial sectors during the last several years. These clever bots are skilled at mimicking natural human languages and talking with humans that businesses use across several industrial sectors. From e-commerce companies to healthcare facilities, it appears as though everyone is utilizing this handy tool to generate business benefits.

What is a Chatbot?

A chatbot is a kind of artificial intelligence-based software meant to communicate with humans in their natural languages. These chatbots often interact via audio or textual means and mimic human languages to connect with humans in a human-like manner. A chatbot is, without a doubt, one of the most important uses of natural language processing

A ChatBot is merely software by which humans can interact with each other. Examples include Apple's Siri, Amazon's Alexa, Google Assistant, and Microsoft's Cortana.

A. Interaction of User for asking the name

First, we will ask Username

```
print("BOT: What is your name?")

user_name = input()
```

B. Response of ChatBot

I assigning questions and answers ChatBot must ask us, for example, we have assigned three variables with different answers, most important point to note it can be used in code and can be also updated automatically by just changing in values of variables

```
name = "Bot Number 286"

monsoon = "rainy"

mood = "Smiley"

resp = {

"what's your name?": [
```

"They call me {0}".format(name),

"I usually go by {0}".format(name),

"My name is the {0}".format(name)],

"what's today's weather?": [

"The weather is {0}".format(monsoon),

"It's {0} today".format(monsoon)],

"how are you?": [

"I am feeling {0}".format(mood),

"{0}! How about you?".format(mood),

"I am {0}! How about yourself?".format(mood),],

"": [

"Hey! Are you there?",

"What do you mean by these?",

],

"default": [

"This is a default message"] }

c. Another Function

Sometimes the questions added are not related to available questions, and sometimes some letters are forgotten to write in the chat. At that time, the bot will not answer any questions, but another function is forward.

```
def real(xtext):

if "name" in xtext:

ytext = "what's your name?"

elif "monsoon" in xtext:

ytext = "what's today's weather?"

elif "how are" in xtext:

ytext = "how are you?"

else:
```

```
ytext = ""

return ytext
```

D. Sending back the message function

def send_message(message):

```
print((message))

response = res(message)

print((response))
```

E. Final Step to break the loop

```
while 1:

my_input = input()

my_input = my_input.lower()

related_text = real(my_input)

send_message(related_text)

if my_input == "exit" or my_input == "stop":

break
```

Preprocess a dataset for a chatbot using Python.

Creating a chatbot in Python involves several steps, and preprocessing the dataset is a crucial part of it. Below is a simplified guide on how to preprocess a dataset for a chatbot using Python.

1. Choose a Dataset:

Select a dataset that fits your chatbot's purpose. Common chatbot datasets include conversational data, question-answer pairs, or dialogue datasets. Ensure the data is in a structured format like CSV, JSON, or plain text.

2. Import Libraries:

```
import pandas as pd
```

```
import numpy as np

import re

from sklearn.model_selection import train_test_split
```

3. Load and Inspect Data:

```
# Assuming your data is in a CSV file
data = pd.read_csv('your_dataset.csv')


# Check the structure of your data
print(data.head())
```

4. Text Cleaning:

Clean the text data to remove noise and irrelevant information. Common cleaning steps include converting to lowercase, removing special characters, and handling contractions.

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    # Add more cleaning steps as needed
    return text


data['cleaned_text'] = data['text_column'].apply(clean_text)
```

5. Tokenization:

Tokenize the text into individual words or subwords. You can use tokenization libraries like NLTK or Spacy.

```
from nltk.tokenize import word_tokenize


data['tokenized_text'] = data['cleaned_text'].apply(word_tokenize)
```

6. Train-Test Split:

Split the dataset into training and testing sets

```
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

7. Prepare Input-Output Pairs:

Create input-output pairs for training the chatbot. Depending on your chatbot type, you

might have single-turn or multi-turn conversations

```
# For a simple question-answer chatbot

X_train = train_data['tokenized_text'].values

y_train = train_data['response_column'].values


X_test = test_data['tokenized_text'].values

y_test = test_data['response_column'].values
```

8. Vectorization:

Convert the tokenized text into numerical vectors using techniques like TF-IDF or word embeddings.

```
from sklearn.feature_extraction.text import TfidfVectorizer


tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

9. Build and Train the Model:

Use a machine learning model or a deep learning framework (like TensorFlow or PyTorch) to build and train your chatbot model.

10. Evaluate and Test:

Evaluate the performance of your chatbot on the test set and make adjustments as needed

```
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test_tfidf)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

Program:

```
pip install chatterbot

from chatterbot import ChatBot

from chatterbot.trainersimport ChatterBotCorpusTrainer

from chatterbot.trainers import ListTrainer

chatbot = ChatBot('MyBot')

trainer = ChatterBotCorpusTrainer(chatbot)

trainer.train('chatterbot.corpus.english')

list_trainer = ListTrainer(chatbot)

ist_trainer.train([

'Hi there!',

'Hello!',

'How are you?',

'I am good, thank you!'

,'What is your name?',

'I am a chatbot.'])

response = chatbot.get_response('Hi')

print(response)

python chatbot.py
```

Output:

```
you : Hi there!

chatbot : Hello!

you : How are you?

chatbot : I am good, thanks!

you : What is your name?

chatbot : I am a chatbot
```

Certainly! Creating a chatbot involves multiple steps, including feature engineering, model training, and evaluation. Below is a basic outline of a simple chatbot project using Python. For this example, we'll use a rule-based approach instead of a machine learning model for simplicity.

Step 1: Install Required Libraries

pip install nltk

Step 2: Import Libraries

import nltk

from nltk.chat.util import Chat, reflections

Step 3: Define Pairs for Chat

```
pairs = [
    [
        r"my name is (.*)",
        ["Hello %1! How can I help you today?",]
    ],
    [
        r"what is your name",
        ["I am a chatbot. You can call me ChatGPT.",]
    ],
    [
        r"how are you",
        ["I'm doing well, thank you!", "I'm great, thanks for asking.",]
```

```
    ],
    [
        r"(.*) (good|well|fine)",
        ["That's awesome to hear!", "Great! How can I assist you today?",]
    ],
    [
        r"quit",
        ["Bye, take care. See you soon!", "Goodbye!"]
    ]
]
```

Step 4: Create Chatbot

```
def simple_chatbot():
    print("Hello! I'm your chatbot. Type 'quit' to exit.")
    chat = Chat(pairs, reflections)
    chat.converse()

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

Step 5: Run the Chatbot

Run the Python script, and the chatbot will interact with the user based on the defined patterns and responses.

```
python your_chatbot_script.py
```

This is a basic example using rule-based responses. For more sophisticated chatbots, you might want to explore machine learning approaches, such as using natural language processing (NLP) libraries like spaCy or training a neural network model.

Output:

you : hi my name is aj

chatbot : Hello!how can i help u today

you : what is your name?

chatbot: I am a chatbot. You can call me ChatGPT

you : how are you?

chatbot :I'm doing well, thank you!", "I'm great, thanks for asking..

you:good|well fine

chatbot:That's awesome to hear!", "Great! How can I assist you today?

you:"quit",

chatbot:Bye, take care. See you soon!", "Goodbye!"

---

create a simple chatbot project in Python.

Step 1: Project Setup

1.Install Dependencies:
Install necessary libraries, such as NLTK or SpaCy for natural language processing.

```
pip install nltk
```

Import Libraries:

* Import the required libraries in your Python script.

```
import nltk
from nltk.chat.util import Chat, reflections
```

Step 2: Data Preparation

Define Pairs:

Define pairs of patterns and responses for your chatbot.

```
pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, how can I help you today?",]
    ],
    # Add more pattern-response pairs
    # ...
]
```

Step 3: Feature Engineering

Define Reflections:

Define reflections to customize the way your chatbot responds.

```
reflections = {
    "i am": "you are",
    "i was": "you were",
    "i": "you",
    "i'm": "you are",
    "i'd": "you would",
    "i've": "you have",
    "i'll": "you will",
```

```
}
```

Step 4: Model Training

Train the Chatbot:
Use the Chatclass from NLTK to train your chatbot.

```
chatbot = Chat(pairs, reflections)
```

Step 5: User Interaction

User Input and Interaction:
Get user input and let the chatbot respond

```
print("Hi, I'm your chatbot. Type 'exit' to end the conversation.")

while True:

  user_input = input("You: ")

  if user_input.lower() == 'exit':

    break

  response = chatbot.respond(user_input)

  print("Bot:", response)
```

Step 6: Evaluation

Testing:
Test your chatbot with various inputs to see how well it performs.

Step 7: Improvement

Iterate and Improve:
Based on testing and user feedback, iterate on your chatbot to improve its performance.

That's a basic outline to get you started. You can customize and expand upon this structure based on your project requirements. Feel free to add more sophisticated natural language processing techniques or use pre-trained models for better performance.

Remember to explore NLTK or other NLP libraries for more advanced features and capabilities

Creating a fully-featured chatbot involves a more complex implementation. Below is an example using the Rasa framework, a powerful open-source framework for building conversational AI:

```
pip install rasa
```

Now you can create a basic Rasa chatbot:

Initialize a Rasa Project:

```
rasa init
```

```
cd
```

```
your_project_directory
```

Define Intents and Stories:

Edit data/nlu.mdto define intents and examples, and data/stories.mdto create dialog flows.

Create Actions:

Implement custom actions in the actions.pyfile. Actions are Python functions that can interact with external APIs or perform any action you define.

Train the Model:

Train your chatbot using the following command:

```
rasa train
```

Run the Chatbot:

Start the chatbot with the following command:

```
rasa run -m models --enable-api --cors "*"--debug
```

Interact with the Chatbot:You can interact with your chatbot by sending HTTP requests to

http://localhost:5005/webhooks/rest/webhook.

You can use tools like curlor create a simple front-end application to communicate with the chatbot.

Remember, building a sophisticated chatbot often involves continuous improvement, adding more training data, refining intents, and potentially integrating with external APIs.

For more details and advanced features, refer to the official Rasa documentation: Rasa Documentation.

Building complex chatbots can also involve machine learning models and natural language processing libraries like spaCy or TensorFlow, depending on your requirements. The choice of the framework depends on the complexity of your chatbot and your specific needs.