CREATING A CHATBOT USING PYTHON

TEAM MEMBER

510521104027:S.MONIKA

PHASE-1: PROJECT

OBJECTIVES:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

PROBLEM DEFINITION:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

DESIGN THINKING:

Functionality: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.

User Interface: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.

Natural Language Processing (NLP): Implement NLP techniques to understand and process user input in a conversational manner.

Responses: Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.

Integration: Decide how the chatbot will be integrated with the website or app.

Testing and Improvement: Continuously test and refine the chatbot's performance based on user interactions.

DATA SET LINK:

https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

ABSTRACT:

The project describes the creation of a chatbot using Python programming language. The chatbot is designed to interact with users, answer common questions and provide support for various services. The chatbot is built using natural language processing techniques and machine learning algorithms such as text classification and sentiment analysis. The project involves collecting and processing data, building the chatbot model, training the model using a dataset and testing the chatbot's performance. The implemented chatbot is capable of understanding user inputs, identifying the user's intent and providing a suitable response. The chatbot can also learn from interactions with users and improve its performance over time. Overall, this project offers a practical approach to building a chatbot using Python, providing a foundation for further development and customization.

STEPS TO CREATE A CHATBOT:

- Build a command-line chatbot with Chatterbot
- Train the chatbot to customize its responses
- Retain the chatbot with industry-specific data

Choose a chatbot development framework:

There are several chatbot development frameworks available in Python that provide pre-built libraries, tools, and APIs for creating a chatbot. Some popular frameworks include ChatterBot, Rasa, and BotStar.

Goals of chatbot:

- Customer Service: One of the primary goals of chatbots is to act as an automated customer service representative, providing quick and accurate responses to customer queries and issues.
- Information Retrieval: Chatbots can be programmed to retrieve information from databases or other sources in response to user queries.
- Personalization: Chatbots can personalize interactions with users by using information such as their name, location, or previous interactions.
- Transactions: Chatbots can be used to facilitate transactions such as ordering products, booking appointments or making reservations.
- Automation: Chatbots can automate routine tasks such as answering frequently asked questions, sending updates to customers, or providing reminders.

Functionalities of chatbot:

- Natural Language Processing: Python has several packages such as SpaCy, NLTK, and TextBlob that provide natural language processing capabilities such as sentiment analysis, entity recognition, and text classification.
- Dialog Management: Python allows for the creation of a dialog management system that makes it possible to manage and maintain complex conversations with users.
- Integration with APIs: Python can be used to integrate chatbots with third-party APIs, allowing them to access external data sources and provide more accurate information to users.

Train the model:

Use the pre-processed data to train the chatbot's model. Some popular natural language processing models include rule-based models, machine learning models like Naive Bayes, SVMs, and deep learning models like Recurrent Neural Networks (RNNs).

Build the chatbot:

Use the trained model to build the chatbot. For instance, if using Chatterbot, you can define the chatbot object, add a training set, train the chatbot, and finally, test the bot to check if it's responding as expected.

Building:

```
import tensorflow as tf

from sklearn.model_selection import train_test_split

#nlp processing

import unicodedata

import re

import numpy as np

import warnings

warnings.filterwarnings('ignore')

data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r').read()

#paried list of question and corresponding answer

QA_list=[QA.split('\t') for QA in data.split('\n')]

print(QA_list[:5])
```

Output:

```
[[hi, how are you doing?  i'm fine. how about yourself?

i'm fine. how about yourself?     i'm pretty good. thanks for asking.
```

i'm pretty good. thanks for asking no problem. so how have you been?

no problem. so how have you been? i've been great. what about you?

i've been great. what about you? i've been good. i'm in school right now.

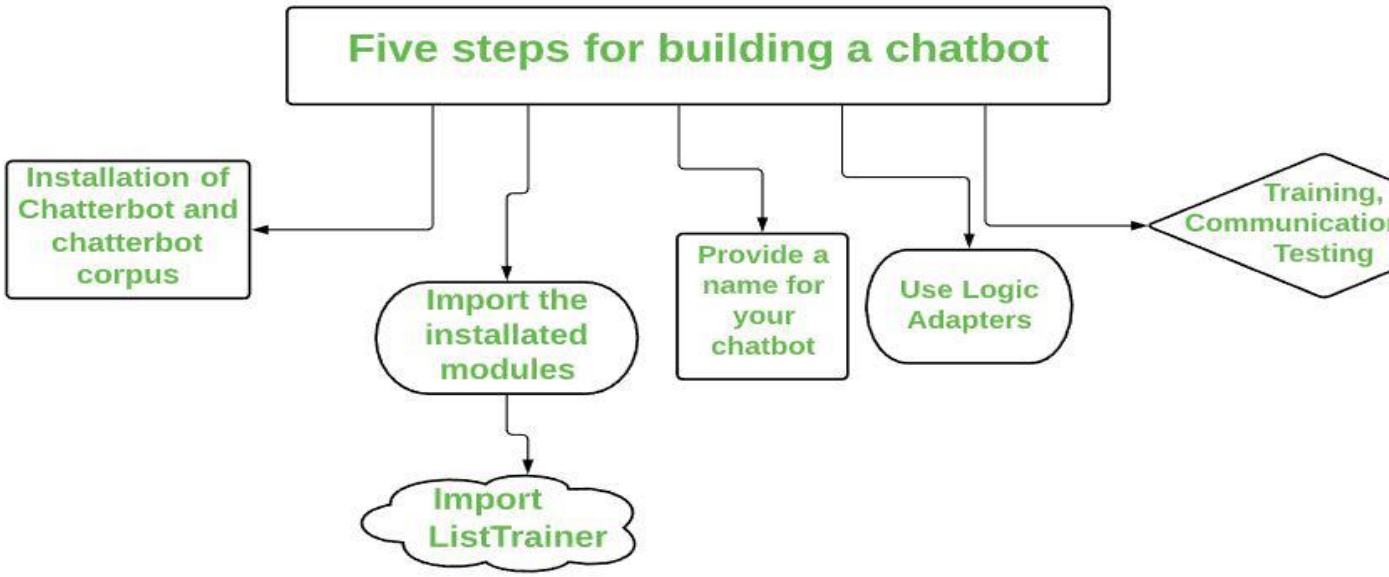
i've been good. i'm in school right now. what school do you go to?

what school do you go to? i go to pcc.

Test and refine the chatbot:

Test your chatbot on a small group of users and continuously refine it based on user feedback to improve its accuracy and responsiveness.

Flowchart for Chatbot:



Model selection:

- Rule-based Model: This model relies on predefined rules and patterns to generate responses. It is relatively simple to implement, but it may lack the ability to handle complex conversations and understand context.
- Retrieval Model: This model uses a predefined set of responses and matches user queries to the most appropriate response based on similarity measures.

It can be effective for simple chatbots but might struggle with understanding nuanced queries.

- Generative Model: This model generates responses based on training data and can handle complex conversations. An example of a popular generative model is the Sequence-to-Sequence (Seq2Seq) model utilizing Long Short-Term Memory (LSTM) networks.

Evaluation:

1. **Accuracy:**
    - The chatbot should be able to accurately understand and respond to user queries. It should be able to correctly interpret the user's intent and provide relevant and correct information.
2. **Natural Language Understanding (NLU):**
    - The chatbot should be able to accurately understand and process natural language input from users. It should have good NLU capabilities to understand user queries of different formats and variations.
3. **Response Time:**
    - The chatbot should be able to provide responses in a reasonable amount of time. Slow response times can be frustrating for users and impact the overall user experience.
4. **User Experience:**
    - The chatbot should provide a smooth and seamless user experience. It should be able to engage users effectively and make the conversation feel natural.
5. **Integration:**
    - The chatbot should be able to integrate with other systems or APIs to fetch data or perform tasks. It should be able to access external resources and provide comprehensive responses to user queries.