CREATE A CHATBOT IN PYTHON

PHASE-4

DEVELOPMENT PART (2) :

Building the chatbot by integrating it into a web app using Flask.

Introduction: Welcome to Our Chatbot Web App

In a world where technology continues to transform the way we interact with information and services, we are excited to introduce you to our Chatbot Web App. With this innovative platform, we aim to provide you with a seamless and interactive experience that leverages the power of artificial intelligence to assist and engage with you.

At [Your Company Name], we understand that the modern user seeks quick and efficient solutions. Our Chatbot Web App is designed to do just that - understand your queries, provide relevant information, and engage in conversations, all in real-time. Whether you have questions about our products, need assistance, or simply wan

USES OF FLASK TOOL

Flask is used for developing web applications using python, implemented on Werkzeug and Jinja2. Advantages of using Flask framework are: There is a built-in development server and a fast debugger provided.

USES OF FLASK FOR PYTHON

It offers a straightforward and adaptable method for developing Python-based web applications and APIs (Application Programming Interfaces). Flask is renowned for its straightforward design, which gives developers the freedom to select the elements they desire and customise their apps to meet their needs.

FEATURE ENGINEERING :

Feature engineering for a chatbot using Flask involves creating a robust set of functionalities that enable effective communication and interaction between the user and the chatbot. Some key aspects to consider include:

Input Processing: Preprocess the user's input, which can involve tasks like tokenization, stemming, and lemmatization, to ensure the chatbot can understand and respond appropriately.

Intent Recognition: Implement techniques such as natural language understanding (NLU) and natural language processing (NLP) to discern the user's intent from their input. This can involve using machine learning models like intent classifiers.

Context Management: Develop mechanisms to maintain context across multiple interactions, ensuring that the chatbot can remember previous conversations and provide coherent responses.

Response Generation: Create a mechanism to generate appropriate responses based on the user's input and the chatbot's understanding. This can involve the use of predefined templates, machine learning models, or a combination of both.

User Experience Enhancement: Implement features like error handling, sentiment analysis for understanding user emotions, and integration of multimedia elements to enhance the user experience.

To implement these features using Flask, you can create routes to handle different functionalities and integrate them into the web application. You would also need to connect the chatbot logic with the Flask server to facilitate communication between the user interface and the chatbot's backend

MODEL TRAINING FOR CHATBOT USING FLASK :

Training a chatbot model using Flask typically involves the following steps:

Data Collection and Preprocessing: Gather a substantial dataset of conversational examples, preprocess the data by tokenizing and cleaning the text, and prepare it for training.

Model Selection:

Choose an appropriate model architecture for the chatbot, such as a retrieval-based model, generative model, or a combination of both.

Model Training:

Train the selected model on the preprocessed dataset. This step involves feeding the data into the model, adjusting the model parameters, and optimizing the model's performance.

Evaluation:

Assess the performance of the trained model by evaluating metrics such as accuracy, perplexity, or F1 score, depending on the type of chatbot and the task it is intended to perform.

Deployment with Flask:

Integrate the trained model into a Flask web application by creating appropriate routes and connecting the model with the Flask server to enable real-time communication between the user interface and the chatbot.

During the model training process, it is crucial to monitor the model's performance, fine-tune it as needed, and ensure that it can

handle various user inputs effectively while providing coherent and contextually relevant responses

EVALUATION

Evaluating a chatbot using Flask involves assessing its performance in terms of how effectively it can understand user inputs and provide relevant and coherent responses.

Here are some key steps for evaluating a chatbot using Flask:

Test Data Preparation:

Prepare a set of test data that includes various types of user inputs and the expected corresponding responses.

Testing Routes:

Implement Flask routes that simulate user interactions with the chatbot, allowing you to input test data and evaluate the chatbot's responses.

Metric Calculation:

Calculate relevant metrics such as accuracy, precision, recall, F1 score, or perplexity, depending on the type of chatbot and the specific task it is designed to perform.

User Experience Assessment:

Solicit feedback from users to gauge their satisfaction with the chatbot's responses, taking into account factors such as the relevance of the answers, the chatbot's ability to maintain context, and overall user experience.

Error Analysis:

Analyze errors and inconsistencies in the chatbot's responses to identify areas for improvement, such as refining the training data, adjusting the model architecture, or enhancing the natural language processing capabilities.

By evaluating the chatbot's performance using Flask, you can gain insights into its strengths and weaknesses, which can then be used to refine and enhance its capabilities for improved user interactions.

PROGRAM

```
[1]: import numpy as np # linear algebra
     import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

     # Input data files are available in the read-only "../input/" directory # For example,
     running this (by clicking run or pressing Shift+Enter) will list␣ ↪all files under the input
     directory

     import os
     for dirname, _, filenames in os.walk('dialogs.txt'):
         for filename in filenames:
             print(os.path.join(dirname, filename))

[2]: import numpy as np
```

```
import random
import string
import nltk

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

```
[3]: f = open("dialogs.txt","r",errors = 'ignore')
     raw_doc = f.read()
     raw_doc = raw_doc.lower()
```

```
[4]: nltk.download('punkt')
     nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\rohig\AppData\Roaming\nltk_data…
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\rohig\AppData\Roaming\nltk_data…
[nltk_data] Package wordnet is already up-to-date!
```

```
[4]: True
```



```
[5]: sent_token = nltk.sent_tokenize(raw_doc)
     word_token = nltk.word_tokenize(raw_doc)
     print(f"Number of sentence : {len(sent_token)}")
     print(f"Number of Words in : {len(word_token)}")
```

```
Number of sentence : 8272
Number of Words in : 60702
```

```
[6]: lemmer = nltk.stem.WordNetLemmatizer()
     remove_punct_dict = {ord(punct):None for punct in string.punctuation} def
     lem_token(tokens):
         return [lemmer.lemmatize(token) for token in tokens]

     def lem_normalize(text):
         return lem_token(nltk.word_tokenize(text.lower().
       ↪translate(remove_punct_dict)))
```

```
[7]: GREET_INPUT = ("hello,hi,sup")
     GREET_RESPONSE = ['Hi',"Hello","I am glad you are talking to me!"]

     def greet(sentence):
         for word in sentence.split():
             if word.lower() in GREET_INPUT:
                 return random.choice(GREET_RESPONSE)
```

[8]:
```
def response(user_response):
    sent_token.append(user_response)
    tfidfvec = TfidfVectorizer(tokenizer = lem_normalize,stop_words = 'english') tfidf =
    tfidfvec.fit_transform(sent_token)
    vals = cosine_similarity(tfidf[-1],tfidf)
    idx = vals.argsort()[0][-2]

    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    sent_token.remove(user_response)
    if (req_tfidf == 0):
        return " I am Sorry! I dont understand you"
    else:
        return str(sent_token[idx])
```

[9]:
```
flag = True
print("BOT : My Name is BOThi, Let's Have Conversation! If you want to exit any␣
  ↪time, just type Bye! ")
print("\n")
while (flag==True):
    user_response = input("You : ")
```



```
    if (user_response != "bye"):
        if (user_response == 'thanks'):
            flag = False
            print("BOT : You are welcome..")
        else:
            if (greet(user_response) != None):
                print("BOT :" +"\t"+greet(user_response))
            else:
                print("BOT L",end = "")
                print(response(user_response))
        print("\n")

    else:
        flag = False
        print("BOT : Goodbye! Take Care")
```

```
BOT : My Name is BOThi, Let's Have Conversation! If you want to exit any time, just
type Bye!


You : hello
BOT : I am glad you are talking to me!


You : what is python
BOT L
```

D:\Anaconda\Lib\site-packages\sklearn\feature_extraction\text.py:525: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
D:\Anaconda\Lib\site-packages\sklearn\feature_extraction\text.py:408: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated tokens ['ha', 'le', 'u', 'wa'] not in stop_words.
  warnings.warn(

I am Sorry! I dont understand you

You : what happend to you
BOT L I am Sorry! I dont understand you

You : what
BOT L I am Sorry! I dont understand you

You : why
BOT L I am Sorry! I dont understand you

You : have a lunch
BOT Land then i made lunch.

You : are you feel good
BOT Lgood!

You : then thank you
BOT Lthank you very much.

You : hi
BOT : Hello

You : bye
BOT : Goodbye! Take Care

Conclusion :

Developing a chatbot using Flask entails creating a robust and interactive conversational interface that can understand user inputs and provide relevant and coherent responses. By implementing key features such as input processing, intent recognition, context management, response generation, and user experience enhancement, you can ensure a seamless and engaging chatbot experience.

Through model training, evaluation, and continuous refinement, you can optimize the chatbot's performance and enhance its ability to handle a variety of user queries and interactions. Integrating the chatbot into a Flask web application enables real-time communication between the user interface and the chatbot's backend, facilitating an efficient and user-friendly experience.

Overall, the combination of Flask and a well-trained chatbot model can provide a powerful and versatile platform for building intelligent and effective conversational agents that can cater to a diverse range of user needs and preferences.