**IBM- Naan Mudhalvan Project**

Student Name : Dharanika.S

Register Number : 620821243028

Branch : B.TECH AI&DS

Year : 3rd year

Topic : Artificial Intelligence

Title : To Create a Chatbot Using Python

College : Gnanamani College of Technology

**TO CREATE A CHATBOT USING PYTHON**

**ABSTRACT:**

This project focuses on the development of a chatbot using Python, leveraging its versatile libraries and tools. The process entails data preprocessing, natural language understanding, and dialogue management. Python's capabilities are harnessed for text processing, while machine learning models enhance the chatbot's conversational abilities. The result is a Python-based chatbot that can engage in meaningful and context-aware conversations, opening up numerous possibilities for application in various domains.

**UNDERSTANDING THE PROBLEM:**

Lots of data is created when people visit websites. The tricky part is turning all that data into useful information. This problem is about making sense of the data to make websites better for users.

**DESIGN THINKING:**

Existing chatbot systems have made significant strides in automating customer interactions and providing round-the-clock support. However, several critical challenges and limitations persist within these systems, hindering their effectiveness and user satisfaction.

**PROJECT DEFINITION:**

Developing a chatbot using Python involves several key steps and methodologies to ensure a successful project. Here's a high-level project methodology for chatbot development using Python:

1.     Project Initiation and Planning:

·        Define the project objectives, scope, and goals.

·        Identify the target audience and their specific needs.

·        Create a project plan, including timelines, milestones, and resource allocation.

·        Establish key performance indicators (KPIs) to measure the chatbot's success.

2.     Data Collection and Preprocessing:

·        Gather relevant data sources, including text corpora, FAQs, and historical chat logs.

·        Clean and preprocess the data to remove noise, handle missing values, and standardize the format.

·        Annotate the data for intent recognition, entity extraction, and dialogue context.

3.     Natural Language Understanding (NLU):

·        Choose and implement NLU techniques such as tokenization, part-of-speech tagging, and named entity recognition using Python libraries like spaCy or NLTK.

·        Train machine learning models for intent classification and entity recognition. Popular choices include support vector machines (SVM), recurrent neural networks (RNNs), or transformer-based models like BERT.

4.     Dialogue Management:

·        Implement a dialogue management system to handle conversations. Options include rule-based systems, finite state machines, or reinforcement learning-based approaches.

·        Design a conversation flow and state transitions to ensure coherent interactions.

·        Use Python to build and maintain the dialogue manager.

5.     Response Generation:

·        Create response templates for common user queries.

·        Employ natural language generation (NLG) techniques to generate dynamic and context-aware responses.

·        Python libraries like GPT-3 or NLTK can assist in NLG.

6.     Integration with External Systems:

·        Develop connectors and APIs to integrate the chatbot with external systems, databases, or web services.

·        Ensure secure and efficient data exchange using Python frameworks like Flask or Django.

7.     User Interface (UI):

·        Design a user-friendly interface for the chatbot, which could be a web application, mobile app, or embedded widget.

·        Implement the UI using Python frameworks such as Flask, Django, or popular JavaScript libraries if required.

8.     Testing and Evaluation:

·        Conduct extensive testing to ensure the chatbot's accuracy, functionality, and usability.

·        Use Python testing frameworks like pytest and libraries like NLTK for automated testing.

·        Collect user feedback and iteratively improve the chatbot's performance based on user input.

9.     Deployment and Scaling:

·        Deploy the chatbot on a suitable hosting platform, such as cloud services like AWS or Azure.

·        Implement scalability measures to handle increased user loads.

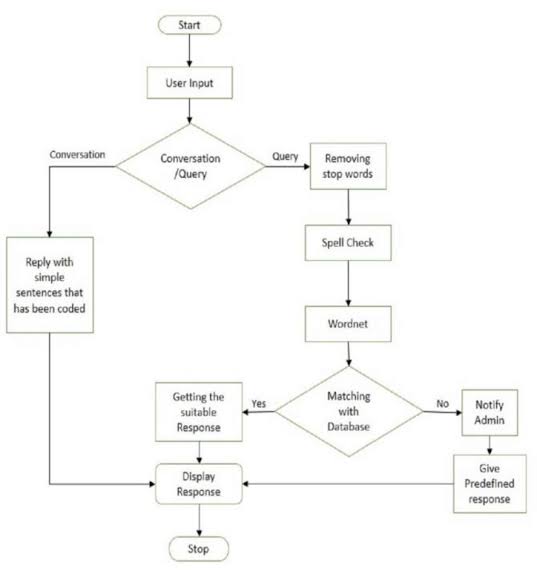
·        Monitor the chatbot's performance and usage in real-time.

10. Maintenance and Continuous Improvement:

·        Regularly update the chatbot's knowledge base and training data.

·        Monitor user interactions and analyze chat logs to identify areas for improvement.

·        Continuously refine the chatbot's NLU and dialogue management based on user feedback and evolving requirements.



**Designing a chatbot in Python involves several steps. Here’s a high-level overview of the process:**

***Define the Purpose and Scope:***

Determine what your chatbot will do. Is it for customer support, information retrieval, or entertainment? Define the scope of its capabilities.

***Choose a Framework or Library:***

You can use libraries like NLTK, spaCy, or frameworks like Rasa, ChatterBot, or the Python Telegram Bot API, depending on your requirements.

***Data Collection and Preprocessing:***

Gather and preprocess data. This may include creating a dataset of user queries and responses if you’re building a rule-based chatbot or collecting training data for machine learning models.

Select a Chatbot Type:

Decide if your chatbot will be rule-based (predefined responses) or machine learning-based (natural language understanding and generation).

Machine Learning Model (If Applicable):

If building an ML-based chatbot, train a model using techniques like sequence-to-sequence models or transformer architectures. Consider using pre-trained models for NLP tasks.

Create a User Interface:

Design a user interface for your chatbot, which can be a web-based chat interface, a mobile app, or integrated into an existing platform.

***Implement Natural Language Processing (NLP):***

Integrate NLP capabilities to understand user input, such as intent recognition and entity extraction.

***Implement Logic and Responses:***

Define the logic for how your chatbot responds to user inputs. For rule-based bots, this involves creating a set of rules. For ML-based bots, this involves generating responses based on learned patterns.

***Testing and Evaluation:***

Test your chatbot extensively to ensure it works as expected. Gather user feedback and continuously improve its responses.

***Deployment:***

Deploy your chatbot to a server or cloud platform. Make it accessible to users through your chosen user interface.

***Monitoring and Maintenance:***

Monitor your chatbot’s performance and address issues as they arise. Regularly update and improve its capabilities.

***Security and Privacy:***

Ensure that your chatbot handles user data securely and follows privacy regulations.

***Load the Dataset:***

Load dataset into Python, which could be in formats like CSV, JSON, or plain text.

***Text Cleaning:***

Remove any irrelevant characters, symbols, or special characters.

Convert text to lowercase to ensure consistency.

***Tokenization:***

Split the text into individual words or tokens. This helps the model understand the structure of sentences.

***Stopword Removal:***

Remove common words (e.g., “the”, “and”, “is”) that don’t contribute much meaning to the text.

***Lemmatization or Stemming:***

Reduce words to their base or root form to normalize the text (e.g., “running” to “run”). You can choose either lemmatization or stemming based on your preference.

Handling Abbreviations and Acronyms:

Expand abbreviations and acronyms to their full forms for consistency and better understanding.

***Handling Contractions:***

Expand contractions (e.g., “can’t” to “cannot”) for uniformity in the text.

***Removing Duplicates:***

Eliminate duplicate entries to ensure data quality and avoid bias during training.

***Text Vectorization:***

Convert the text data into a numerical format suitable for model training. Common approaches include using bag-of-words, TF-IDF (Term Frequency-Inverse Document Frequency), or embeddings like Word2Vec or GloVe.

***Padding and Truncation:***

Ensure that all input sequences have a consistent length by either padding shorter sequences or truncating longer ones.

***Dataset Splitting:***

Divide the dataset into training, validation, and testing sets for model training and evaluation.

Def load\_Dataset(data,size=None):

If(size!=None):

Y,X=data[:size]

Else:

Y,X=data

X\_tokenizer=tokenize(X)

Y\_tokenizer=tokenize(y)

X\_tensor=vectorization(X\_tokenizer,X)

Y\_tensor=vectorization(y\_tokenizer,y)

Return X\_tensor,X\_tokenizer, y\_tensor, y\_tokenizer

Size=30000

Data=preprocessed\_answers,preprocessed\_questions\

X\_tensor,X\_tokenizer, y\_tensor, y\_tokenizer=load\_Dataset(data,size)

# Calculate max\_length of the target tensors

Max\_length\_y, max\_length\_X = y\_tensor.shape[1], X\_tensor.shape[1],

.X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_tensor, y\_tensor, test\_size=0.2)

# Show length

Print(len(X\_train), len(y\_train), len(X\_val), len(y\_val))

2980 2980 745 745

Data preprocessing

The basic text processing in NLP are:

1. Sentence Segmentation

2. Normalization

3. Tokenization

1.Segmentation:

In[1] :

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’)

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

Warnings.warn(f”A NumPy version >={np\_minversion} and <{np\_maxversion}”

Data=open(‘simple-dialogs-for-chatbot/dialogs.txt’,’r’).read()

In[2] :

#paried list of question and corresponding answer

QA\_list=[QA.split(‘\t’) for QA in data.split(‘\n’)]

Print(QA\_list[:5])

[[‘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.”]]

In[3] :

Questions=[row[0] for row in QA\_list]

Answers=[row[1] for row in QA\_list]

In[4] :

Print(questions[0:5])

Print(answers[0:5])

[‘hi, how are you doing?’, “I’m fine. How about yourself?”, “I’m pretty good. Thanks for asking.”, ‘no problem. So how have you been?’, “I’ve been great. What about you?”]

[“I’m fine. How about yourself?”, “I’m pretty good. Thanks for asking.”, ‘no problem. So how have you been?’, “I’ve been great. What about you?”, “I’ve been good. I’m in school right now.”]

2. Normalization:

To reduce its randomness, bringing it closer to a predefined “standard”

In[5] :

Def remove\_diacritic(text):

Return ‘’.join(char for char in unicodedata.normalize(‘NFD’,text)

If unicodedata.category(char) !=’Mn’)

In[6] :

Def preprocessing(text):

#Case folding and removing extra whitespaces

Text=remove\_diacritic(text.lower().strip())

#Ensuring punctuation marks to be treated as tokens

Text=re.sub(r”([?.!,¿])”, r” \1 “, text)

#Removing redundant spaces

Text= re.sub(r’[“ “]+’, “ “, text)

#Removing non alphabetic characters

Text=re.sub(r”[^a-zA-Z?.!,¿]+”, “ “, text)

Text=text.strip()

#Indicating the start and end of each sentence

Text=’<start> ‘ + text + ‘ <end>’

Return text

Preprocessed\_questions=[preprocessing(sen) for sen in questions]

Preprocessed\_answers=[preprocessing(sen) for sen in answers]

Print(preprocessed\_questions[0])

Print(preprocessed\_answers[0])

<start> hi , how are you doing ? <end>

<start> I m fine . how about yourself ? <end>

3.Tokenization:

In[7] :

Def tokenize(lang):

Lang\_tokenizer = tf.keras.preprocessing.text.Tokenizer(

Filters=’’)

#build vocabulary on unique words

Lang\_tokenizer.fit\_on\_texts(lang)

Return lang\_tokenizer

**CODING:**

**app.py**

from flask import Flask, render\_template, request, jsonify

import json

app = Flask(\_name\_)

# Load QA data from qa\_data.json

with open('qa\_data.json', 'r') as file:

qa\_data = json.load(file)

@app.route("/")

def index():

return render\_template("index.html")

@app.route("/ask", methods=["POST"])

def ask():

user\_question = request.json["question"]

answer = qa\_data.get(user\_question, "I don't know the answer to that question.")

return jsonify({"answer": answer})

if \_name\_ == "\_main\_":

app.run(debug=True)from flask import Flask, render\_template, request, jsonify

import json

app = Flask(\_name\_)

# Load QA data from qa\_data.json

with open('qa\_data.json', 'r') as file:

qa\_data = json.load(file)

@app.route("/")

def index():

return render\_template("index.html")

@app.route("/ask", methods=["POST"])

def ask():

user\_question = request.json["question"]

answer = qa\_data.get(user\_question, "I don't know the answer to that question.")

return jsonify({"answer": answer})

if \_name\_ == "\_main\_":

app.run(debug=True)

**DATA.JSON**

{

"What is your name?": "My name is ChatBot.",

"How are you?": "I'm just a computer program, so I don't have feelings, but thanks for asking!",

"Who created you?": "I was created by a talented programmer.",

"What is the weather today?": "I'm sorry, I cannot provide real-time weather information.",

"Tell me a joke.": "Why did the computer catch a cold? Because it had too many windows open!",

"What is the meaning of life?": "The meaning of life is a philosophical question that has different answers for different people.",

"Can you sing a song?": "I'm afraid I can't sing, but I can help you find song lyrics!",

"How do you work?": "I work by processing natural language and providing responses based on predefined patterns and data.",

"Tell me a fun fact.": "A group of flamingos is called a 'flamboyance.'",

"Do you dream?": "No, I don't dream. I'm always awake and ready to assist you!",

"What's your favorite color?": "I don't have personal preferences, but I can provide information about colors if you'd like.",

"Who won the World Series in 2020?": "The Los Angeles Dodgers won the 2020 World Series.",

"What's the capital of France?": "The capital of France is Paris.",

"Tell me about yourself.": "I am a computer program designed to assist with answering questions and providing information.",

"How tall is Mount Everest?": "Mount Everest is approximately 29,032 feet (8,849 meters) tall.",

"What's the largest planet in our solar system?": "The largest planet in our solar system is Jupiter.",

"What is the Pythagorean theorem?": "The Pythagorean theorem states that in a right triangle, the square of the hypotenuse is equal to the sum of the squares of the other two sides.",

"Who wrote 'Romeo and Juliet'?": " 'Romeo and Juliet' was written by William Shakespeare.",

"hi": "Hello there!",

"bye": "Goodbye!",

"How fast can a cheetah run?": "A cheetah can run at speeds of up to 75 miles per hour (120 kilometers per hour)."

}

**INDEX.HTML**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<title>ChatBot</title>

</head>

<body>

<div class="chat-container">

<div class="chat-box">

<div class="chat" id="chat">

<!-- Chat messages will appear here -->

</div>

</div>

<div class="input-container">

<input type="text" id="user-input" placeholder="Type your question...">

<button id="submit-btn">Ask</button>

</div>

</div>

<script src="{{ url\_for('static', filename='script.js') }}"></script>

</body>

</html>

**SCRIPT.JS**

document.addEventListener("DOMContentLoaded", () => {

const chatBox = document.getElementById("chat");

const userInput = document.getElementById("user-input");

const submitBtn = document.getElementById("submit-btn");

function addMessage(message, isUser = false) {

const messageDiv = document.createElement("div");

messageDiv.classList.add("message", isUser ? "user" : "bot");

messageDiv.innerText = message;

chatBox.appendChild(messageDiv);

chatBox.scrollTop = chatBox.scrollHeight;

}

submitBtn.addEventListener("click", () => {

const userMessage = userInput.value;

if (userMessage) {

addMessage(You: ${userMessage}, true);

userInput.value = "";

// Send user's question to the server and get a bot response

fetch("/ask", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ question: userMessage }),

})

.then((response) => response.json())

.then((data) => {

const botResponse = Bot: ${data.answer};

addMessage(botResponse, false);

})

.catch((error) => console.error("Error:", error));

}

});

});

**STYLE.CSS**

/\* Reset default margins and paddings \*/

body, h1, h2, p, ul, li {

margin: 0;

padding: 0;

}

/\* Set a background color for the page \*/

body {

background-color: #f4f4f4;

font-family: Arial, Helvetica, sans-serif;

}

/\* Style the chat container \*/

.chat-container {

max-width: 400px;

margin: 0 auto;

background-color: #ffffff;

border: 1px solid #ccc;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

border-radius: 5px;

overflow: hidden;

}

/\* Style the chat box \*/

.chat-box {

max-height: 400px;

overflow-y: auto;

}

/\* Style chat messages \*/

.message {

padding: 10px;

margin: 5px;

border-radius: 5px;

}

/\* Style user messages \*/

.user {

background-color: #f0f0f0;

text-align: right;

}

/\* Style bot messages \*/

.bot {

background-color: #d9f7e6;

}

/\* Style the input container \*/

.input-container {

padding: 10px;

background-color: #f4f4f4;

border-top: 1px solid #ccc;

display: flex;

justify-content: space-between;

}

/\* Style the user input field \*/

#user-input {

flex-grow: 2;

padding: 5px;

border: none;

border-radius: 3px;

outline: none;

}

/\* Style the submit button \*/

#submit-btn {

padding: 5px 10px;

background-color: #007bff;

color: #fff;

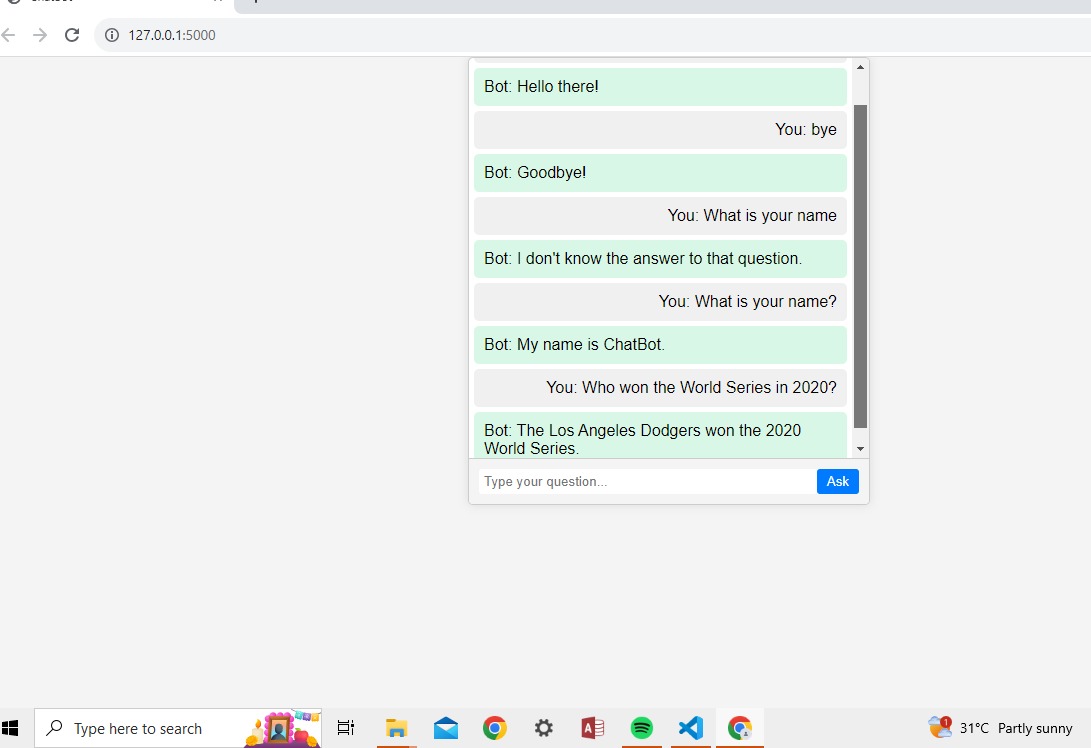
border: none;

border-radius: 3px;

cursor: pointer;

}

**OUTPUT:**



**CONCLUSION:**

In conclusion, the preprocessing phase is a crucial step in creating a chatbot using Python. It involves several key tasks, including text cleaning, tokenization, stemming or lemmatization, stop word removal, and handling special characters. This phase helps in preparing the text data for natural language processing tasks and ensures that the chatbot can understand and respond effectively to user input. Additionally, data preprocessing can also involve the collection and structuring of training data and the creation of a knowledge base or dataset to train the chatbot. By paying attention to the preprocessing phase, you can improve the accuracy and performance of your chatbot, making it a more efficient and user-friendly conversational agent.