**CREATE A CHATBOT IN PYTHON**

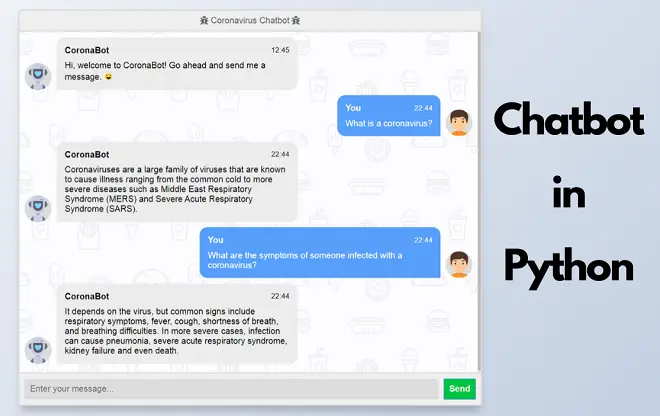
By Akash.S – 411421205002

(B.Tech/ Information Technology, 3rd year)

Domain name: Artificial Intelligence

# Phase-2 Document Submission

**Project:** To create a Chatbot in Python that provides exceptional customer service,answering user queries (diabetes) on a website.



**Introduction:**

**- Chatbots are a technology designed to facilitate human-computer interaction using natural language (spoken human language).**

**- Modern chatbots serve various purposes, acting as assistants for online shopping, website guidance, or general knowledge conversation partners.**

**- Despite being tailored to specific domains, many chatbots follow a similar flow, processing a user's input against their knowledge base and responding accordingly, resembling a search engine's behavior.**

**- Some chatbots use techniques to remember the recent conversation topic but lack the ability to recall the entire conversation flow.**

**- We propose an architectural design for a chatbot with the ability to remember the entire conversation flow. This chatbot is intended for diabetic patients to assist them in their daily diabetes control activities.**

**Natural Language Processing (NLP):**

In the information provided, several common symptoms associated with Type 2 diabetes are outlined. These symptoms can act as indicators for individuals to assess their health and consider consulting a healthcare professional for further evaluation and potential diagnosis.

1. \*\*Frequent Urination:\*\*

People with Type 2 diabetes often experience an increase in the frequency of urination, particularly noticeable during the night. This occurs due to the body's attempt to remove excess sugar through urine.

2. \*\*Increased Thirst:\*\*

Unusual and persistent thirst is a common symptom of Type 2 diabetes. The body's response to high blood sugar levels often includes an increased need for fluids.

3. \*\*Fatigue:\*\*

Persistent fatigue and a general lack of energy are frequently reported by individuals with Type 2 diabetes. Fluctuations in blood sugar levels can contribute to this feeling of constant tiredness.

4. \*\*Blurred Vision:\*\*

High levels of blood sugar can cause temporary changes in vision, resulting in blurred or distorted eyesight. This symptom is a result of the sugar affecting the lenses of the eyes.

5. \*\*Slow Wound Healing:\*\*

Type 2 diabetes can affect the body's ability to heal wounds and fight off infections. Cuts and wounds may take a longer time to heal compared to individuals without diabetes.

6. \*\*Unexplained Weight Loss:\*\*

Despite maintaining a regular diet, some individuals with Type 2 diabetes may experience unanticipated weight loss. This occurs due to the body using muscle and fat for energy when it cannot access glucose properly.

7. \*\*Tingling or Numbness:\*\*

Neuropathy, a type of nerve damage often associated with diabetes, can lead to sensations of tingling or numbness, typically felt in the hands or feet. It is a result of the effect of high blood sugar on the nerves.

It's important to note that symptoms may vary from person to person. If any of these symptoms are experienced or suspected, seeking medical advice and undergoing proper evaluation is crucial for an accurate diagnosis and appropriate management of Type 2 diabetes.

**Example:**

pip install nltk import nltk from nltk.chat.util import Chat, reflections # Define patterns and responses for the chatbot patterns = [

(r'(.\*)tell me about Type 2 diabetes(.\*)', [

"Type 2 diabetes is a chronic condition that affects how your body metabolizes glucose.",

"Common risk factors include genetics, obesity, and a sedentary lifestyle.",

"Symptoms may include increased thirst, frequent urination, and fatigue.",

"Managing blood sugar levels through diet, exercise, and medication is crucial for treatment.",

]),

(r'(.\*)symptoms of diabetes(.\*)', [

"The symptoms of diabetes can include increased thirst, frequent urination, fatigue, and more.",

"It's essential to consult a healthcare professional for a proper diagnosis if you suspect diabetes.",

]),

(r'(.\*)how to manage diabetes(.\*)', [

"Managing diabetes involves monitoring blood sugar levels, eating a balanced diet, and regular exercise.",

"Medications and insulin may also be prescribed by a healthcare provider.",

]),

(r'(.\*)help(.\*)', [

"I can provide information about diabetes. Just ask me a specific question, and I'll do my best to help!",

]),

]

# Create and start the chatbot def diabetes\_chat():

print("Hello! I'm your diabetes chatbot. How can I assist you today?") chatbot = Chat(patterns, reflections) chatbot.converse() if \_\_name\_\_ == "\_\_main\_\_": nltk.download("punkt") diabetes\_chat()

**Data Preparation:**

* Load the dataset into a suitable data structure (e.g., Pandas DataFrame).

* Examine the dataset to understand its structure and distribution.

* Preprocess the data by removing unnecessary characters, converting text to lowercase, and handling any missing values**.**

**Example:**

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('/kaggle/input'): for filename in filenames:

print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"

# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session import json with open('/kaggle/input/mental-health-conversational-data/intents.json', 'r') as f:

data = json.load(f) df = pd.DataFrame(data['intents'])

df

tag patterns responses

1. greeting [Hi, Hey, Is anyone there?, Hi there, Hello, H... [Hello there. Tell me how are you feeling toda...
2. morning [Good morning] [Good morning. I hope you had a good night's s...
3. afternoon [Good afternoon] [Good afternoon. How is your day going?]
4. evening [Good evening] [Good evening. How has your day been?] 4 night [Good night] [Good night. Get some proper sleep, Good night...

... ... ... ...

... ... ... ...

... ... ... ...

1. fact-28 [What do I do if I'm worried about my mental h... [The most important thing is to talk to someon...
2. fact-29 [How do I know if I'm unwell?] [If your beliefs , thoughts , feelings or beha...
3. fact-30 [How can I maintain social connections? What i... [A lot of people are alone right now, but we d...
4. fact-31 [What's the difference between anxiety and str... [Stress and anxiety are often used interchange...
5. fact-32 [What's the difference between sadness and dep... [Sadness is a normal reaction to a loss, disap...
6. rows × 3 columns

dic = {"tag":[], "patterns":[], "responses":[]} for i in range(len(df)):

ptrns = df[df.index == i]['patterns'].values[0] rspns = df[df.index == i]['responses'].values[0] tag = df[df.index == i]['tag'].values[0] for j in range(len(ptrns)): dic['tag'].append(tag) dic['patterns'].append(ptrns[j]) dic['responses'].append(rspns)

df = pd.DataFrame.from\_dict(dic)

df

tag patterns responses

0 greeting Hi [Hello there. Tell me how are you feeling toda... 1 greeting Hey [Hello there. Tell me how are you feeling toda...

1. greeting Is anyone there? [Hello there. Tell me how are you feeling toda...
2. greeting Hi there [Hello there. Tell me how are you feeling toda...
3. greeting Hello [Hello there. Tell me how are you feeling toda...

... ... ... ...

... ... ... ...

... ... ... ...

1. fact-29 How do I know if I'm unwell? [If your beliefs , thoughts , feelings or beha...
2. fact-30 How can I maintain social connections? What if... [A lot of people are alone right now, but we d...
3. fact-31 What's the difference between anxiety and stress? [Stress and anxiety are often used interchange...
4. fact-32 What's the difference between sadness and depr... [Sadness is a normal reaction to a loss, disap...
5. fact-32 difference between sadness and depression [Sadness is a normal reaction to a loss, disap...
6. rows × 3 columns

df['tag'].unique() array(['greeting', 'morning', 'afternoon', 'evening', 'night', 'goodbye',

'thanks', 'no-response', 'neutral-response', 'about', 'skill',

'creation', 'name', 'help', 'sad', 'stressed', 'worthless',

'depressed', 'happy', 'casual', 'anxious', 'not-talking', 'sleep',

'scared', 'death', 'understand', 'done', 'suicide', 'hate-you',

'hate-me', 'default', 'jokes', 'repeat', 'wrong', 'stupid',

'location', 'something-else', 'friends', 'ask', 'problem',

'no-approach', 'learn-more', 'user-agree', 'meditation',

'user-meditation', 'pandora-useful', 'user-advice',

'learn-mental-health', 'mental-health-fact', 'fact-1', 'fact-2',

'fact-3', 'fact-5', 'fact-6', 'fact-7', 'fact-8', 'fact-9',

'fact-10', 'fact-11', 'fact-12', 'fact-13', 'fact-14', 'fact-15', 'fact-16', 'fact-17', 'fact-18', 'fact-19', 'fact-20', 'fact-21',

'fact-22', 'fact-23', 'fact-24', 'fact-25', 'fact-26', 'fact-27',

'fact-28', 'fact-29', 'fact-30', 'fact-31', 'fact-32'], dtype=object)

**Exploratory Data Analysis:**

* Analyze the distribution of intents in the dataset.

* Visualize the frequency of different intents using a bar plot from the Plotly library.

* The x-axis can represent the intents, and the y-axis can represent the count of patterns or responses associated with each intent.

**Example:**

import plotly.graph\_objects as go intent\_counts = df['tag'].value\_counts() fig = go.Figure(data=[go.Bar(x=intent\_counts.index, y=intent\_counts.values)]) fig.update\_layout(title='Distribution of Intents', xaxis\_title='Intents', yaxis\_title='Count') fig.show()

**Pattern and Response Analysis:**

* Explore the patterns and responses associated with each intent.

* Calculate the average number of patterns and responses per intent.

* Visualize this information using a Plotly bar plot, where the x-axis represents the intents, and the y-axis represents the average count of patterns or responses.

* Interpret the plot to understand the varying degrees of complexity and diversity in patterns and responses across different intents.

**Example:**

df['pattern\_count'] = df['patterns'].apply(lambda x: len(x)) df['response\_count'] = df['responses'].apply(lambda x: len(x)) avg\_pattern\_count = df.groupby('tag')['pattern\_count'].mean() avg\_response\_count = df.groupby('tag')['response\_count'].mean()

fig = go.Figure()

fig.add\_trace(go.Bar(x=avg\_pattern\_count.index, y=avg\_pattern\_count.values, name='Average Pattern Count'))

fig.add\_trace(go.Bar(x=avg\_response\_count.index, y=avg\_response\_count.values, name='Average Response Count'))

fig.update\_layout(title='Pattern and Response Analysis', xaxis\_title='Intents', yaxis\_title='Average Count') fig.show()

**Intent Prediction Model:**

* Split the dataset into training and testing sets.

* Implement a machine learning or deep learning model suitable for intent prediction, such as a text classification model.

* Vectorize the text data (e.g., using TF-IDF or word embeddings) and train the model using the patterns as input and the corresponding intents as target variables.

* Evaluate the model's performance on the testing set using appropriate metrics like accuracy, precision, recall, and F1-score.

* Visualize the model's performance using a Plotly bar plot, where the x-axis represents the evaluation metrics, and the y-axis represents the corresponding scores.

* Interpret the plot to analyze the effectiveness of the intent prediction model.

**Example:**

from sklearn.model\_selection import train\_test\_split from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.svm import SVC from sklearn.metrics import classification\_report import plotly.graph\_objects as go

# Split the dataset into training and testing sets X = df['patterns'] y = df['tag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Vectorize the text data using TF-IDF vectorizer = TfidfVectorizer() X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test) # Train a Support Vector Machine (SVM) classifier model = SVC()

model.fit(X\_train\_vec, y\_train) # Predict intents for the testing set y\_pred = model.predict(X\_test\_vec) # Evaluate the model's performance report = classification\_report(y\_test, y\_pred, output\_dict=True, zero\_division=0)

# Convert float values in the report to dictionaries

report = {label: {metric: report[label][metric] for metric in report[label]} for label in report if isinstance(report[label], dict)} # Extract evaluation metrics labels = list(report.keys()) evaluation\_metrics = ['precision', 'recall', 'f1-score']

metric\_scores = {metric: [report[label][metric] for label in labels if label in report] for metric in evaluation\_metrics}

# Visualize the model's performance using a Plotly bar plot fig = go.Figure() for metric in evaluation\_metrics:

fig.add\_trace(go.Bar(name=metric, x=labels, y=metric\_scores[metric])) fig.update\_layout(title='Intent Prediction Model Performance',

xaxis\_title='Intent', yaxis\_title='Score', barmode='group') fig.show()

**Prediction Model Deployment:**

* Once satisfied with the model's performance, deploy the intent prediction model in a chatbot framework.

* Utilize the trained model to predict intents based on user input in real-time.

* Implement an appropriate response generation mechanism to provide relevant and empathetic responses based on the predicted intents.

**Example:**

def predict\_intent(user\_input): # Vectorize the user input user\_input\_vec = vectorizer.transform([user\_input])

# Predict the intent intent = model.predict(user\_input\_vec)[0] return intent

# Function to generate responses based on predicted intents def generate\_response(intent):

# Implement your logic here to generate appropriate responses based on the predicted intents

if intent == 'greeting':

response = "Hello! How can I assist you today?"

elif intent == 'farewell': response = "Goodbye! Take care."

elif intent == 'question':

response = "I'm sorry, I don't have the information you're looking for." else:

response = "I'm here to help. Please let me know how I can assist you."

return response # Example usage

while True: # Get user input user\_input = input("User: ")

# Predict intent intent = predict\_intent(user\_input) # Generate response response = generate\_response(intent) print("Chatbot:", response)

Chatbot: Hello! How can I assist you today?

Chatbot: Hello! How can I assist you today?

Chatbot: I'm here to help. Please let me know how I can assist you.

**Conclusion:**

* The availability of a well-structured dataset encompassing various conversations related to diabetes health provides a valuable resource for training chatbot models to offer emotional support to individuals dealing with anxiety and depression. By utilizing intents, patterns, and responses, the models can learn to understand user messages and generate empathetic and relevant replies.

* The use of such models in chatbot frameworks holds great potential for providing accessible and compassionate support to those in need of diabetes health assistance. By simulating the behavior of a therapist, these chatbots can offer guidance, answer frequently asked questions, and provide general advice to individuals experiencing anxiety and depression.

* The insights and knowledge gained from this dataset, researchers and developers can contribute to the development of chatbots that serve as virtual companions, offering emotional solace and alleviating some of the burdens faced by individuals seeking Diabetes health support.

* Overall, the dataset and the subsequent training of chatbot models enable the creation of innovative tools that bridge the gap in Diabetes health care, providing individuals with a readily available resource for emotional support and guidance.