CREATE A CHATBOT IN PYTHON

Chatbot:

Chatbots are conversational tools that perform routine tasks efficiently. It Communicates with users using interactive text or speech capabilities. People like them because they help them get through those tasks quickly so they can focus their attention on high-level, strategic, and engaging activities

Problem Definition:

The problem is creating a Chatbot in python that uses ask questions throughout the user's journey and provide information that may persuade the user and create a lead. The Chatbot aims to Communicates with users using interactive text or speech capabilities.

Design Thinking:

1. **Functionality**: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.
2. **User Interface**: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.
3. **Natural Language Processing (NLP):** Implement NLP techniques to understand and process user input in a conversational manner.
4. **Responses:** Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.
5. **Integration:** Decide how the chatbot will be integrated with the website or app.
6. **Testing and Improvement:** Continuously test and refine the chatbot's performance based on user interactions.

Innovative technices

There are two approaches that can be used to develop a chatbot depending on the algorithms and techniques adopted there are rule-based approach, machine learning approach and dialogue management

Innovation:

1. **Rule based approach:**

   These chatbots follow predefined rules and decision trees to respond to user inputs. They are limited to the specific rules and patterns programmed into them and lack the ability to understand natural language process. Rule-based chatbots are relativelysimple to build and are suitable tasks with well-defined interactions.

2. **Machine Learning Approach:**

   These chatbots use machine learning techniques, such as natural language processing (NLP) and neural networks, to understand and generate responses based on patterns in data. Machine learning-based chatbots are more flexible and capable of handling a wider range user inputs but require more extensive development and training.

   The choice between these methods depends on the complexity of the chatbot's intended tasks and the desired level of sophistication in its interactions. Many modern chatbots combine both approaches, using rules for specific tasks and machine learning for more general language understanding.

3. **Dialog Management:**

   Dialog management is the process of controlling the flow of conversation in a chatbot. Various methods, such as rule-based systems, state machines, or reinforcement learning, are used to manage the chatbot's interactions with users. It ensures that the chatbot responds appropriately and maintains context during a conversation.

Ensemble Methods:

Creating a chatbot using python is a task that typically relies on sesimological data and machine learning techniques.

1. **Stacking Models:**

Stack multiple chatbot models in a hierarchical manner, with one model refining the output of another. This can help improve the overall quality of responses

2. **Voting Mechanism:**

Ensemble voting methods, such as majority voting or weighted voting, can be used to combine predictions from multiple models or components within the chatbot. This is helpful for improving classification or decision-making tasks.

3. **Bagging** :

In the Bagging methods can be applied to reduce overfitting and enhance the generalization of chatbot models. Multiple instances of the same model are trained on different subsets of data, and their predictions are aggregated.

4. **Boosting :**

Boosting algorithms, like AdaBoost, can be used to improve the performance of individual classifiers or models within the chatbot. It assigns weights to data points to emphasize misclassified examples, leading to better accuracy.

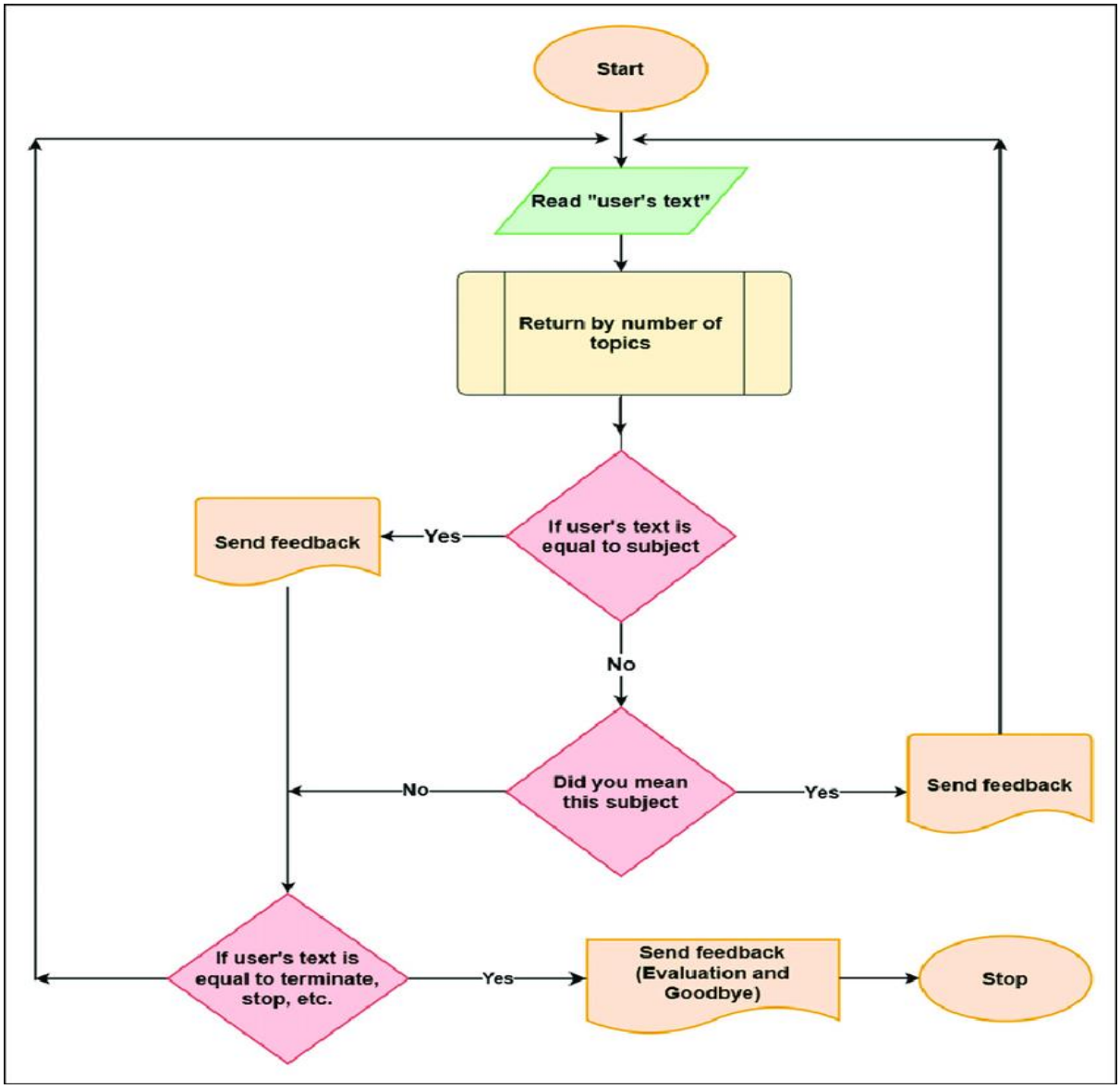
5. **Random Forest:**

Random Forest is an ensemble method based on decision trees. It can be utilized to improve intent classification or other classification tasks in chatbots by aggregating predictions from multiple decision trees.

6. **Model Diversity:**

Ensuring that the chatbot employs diverse models and techniques can enhance its adaptability and robustness, as it can handle a wider range of user inputs and scenarios..

Flowchart:



Libraries:

Here are the key libraries commonly used in chatbot development:

Natural Language Processing (NLP) Libraries:

NLTK (Natural Language Toolkit): It provides tools for tokenization, stemming, lemmatization, part-of-speech tagging, and more**.**

Example:

```
import nltk

sentence = "This is an example sentence."

tokens = nltk.word_tokenize(sentence)

print(tokens)
```

spaCy: spaCy is a popular NLP library known for its speed and accuracy.

Example:

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "SpaCy is a fast and efficient natural language processing library."

doc = nlp(text)

for token in doc:

print(token.text, token.pos_)
```

TextBlob: It provides easy-to-use APIs for common NLP tasks like sentiment analysis and part-of-speech tagging**.**

Example:

```
from textblob import TextBlob

text = "TextBlob is a simple NLP library for Python."

blob = TextBlob(text)

# Sentiment analysis

sentiment = blob.sentiment

print(sentiment)

# Part-of-speech tagging

tags = blob.tags

print(tags)
```

Machine Learning Libraries:

Scikit-learn: It can be used for tasks such as intent classification, sentiment analysis, and entity recognition in chatbots**.**

Example:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import SVC

from sklearn.pipeline import Pipeline

text_clf = Pipeline([('tfidf', TfidfVectorizer()), ('clf', SVC())])

text_clf.fit(X_train, y_train)
```

TensorFlow and Keras: These libraries are used for building and training deep learning models, which can be applied to natural language understanding tasks.

Example:

```
import tensorflow as tf

from tensorflow import keras

model = keras.Sequential()

model.add(keras.layers.Dense(128, input_shape=(X_train.shape[1],), activation='relu'))

model.add(keras.layers.Dense(num_classes, activation='softmax'))
```

Web Frameworks:

Flask or Django: They provide a way to build the chatbot's user interface and handle HTTP requests and responses.

Example:

```
from flask import Flask, request, jsonify

app = Flask(__name)

@app.route('/chat', methods=['POST'])
```

```
def chat():
    user_message = request.json['message']
    # Process the message and generate a response
    return jsonify({"response": chatbot_response})
if __name__ == '__main__':
    app.run()
```

NLP Techniques:

Utilize various NLP techniques to enhance your chatbot's capabilities. Some common techniques include:

- **Text Classification**: To understand the intent of user messages.
- **Named Entity Recognition (NER)**: To extract entities like dates, locations, or product names.
- **Sentiment Analysis:** To determine the sentiment of user messages.
- **Language Modeling**: For generating natural-sounding responses.

Data set:

Here's a brief description of a popular dataset along with its source:

Dataset Name: Iris Dataset

Source:

The Iris dataset is a well-known dataset in machine learning and can be obtained from the UCI Machine Learning Repository: UCI Iris Dataset.

Description:

The Iris dataset is small dataset that contains measurements of four features (sepal length, sepal width, petal length, and petal width) of three different species of iris flowers (setosa, versicolor, and virginica). It consists of 150 instances, with 50 samples from each of the three species. The dataset is commonly used for tasks like classification and clustering in machine learning to demonstrate various algorithms and techniques.

Code:

```
import numpy as np

import pandas as pd

import seaborn as sns

sns.set_palette('husl')

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import metrics

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

data = pd.read_csv('../input/Iris.csv')


IN[2]

data.head()
```

Output:

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLer
0	1	5.1	3.5	1.4
1	2	4.9	3.0	1.4
2	3	4.7	3.2	1.3
3	4	4.6	3.1	1.5
4	5	5.0	3.6	1.4

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

Step 1:

Create a Flask Project

1. Create a new directory for your Flask project.
2. Inside this directory, create a virtual environment:

Python code:

```
python -m venv venv
source venv/bin/activate  # On Windows, use 'venv\Scripts\activate'
```

3. Install Flask:

Python code:

```
pip install Flask
```

Step 2:

Create a Basic Flask Web App

1. Create a file named app.py in your project directory.

2. In app.py, import Flask and create a basic Flask app:

Python code:

```
from flask import Flask, render_template, request
app = Flask(__name)
@app.route('/')
def home():
    return render_template('index.html')
if __name__ == '__main__':
    app.run(debug=True)
```

Step 3:

Create HTML Templates

1. Create a folder named templates in your project directory.

2. Inside the templates folder, create an HTML file named index.html:

HTML code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot</title>
</head>
<body>
    <h1>Chatbot</h1>
    <div id="chat-box"></div>
    <input type="text" id="user-input" placeholder="Type your message">
    <button id="send-button">Send</button>
    <script src="static/chatbot.js"></script>
</body>
```

</html>

Step 4:

Create JavaScript for Chatbot

1. Create a folder named static in your project directory.

2. Inside the static folder, create a JavaScript file named chatbot.js. This file will handle user input and responses from the chatbot.

JavaScript code

```
// chatbot.js
const chatBox = document.getElementById('chat-box');
const userInput = document.getElementById('user-input');
const sendButton = document. getElementById('send-button');
sendButton.addEventListener('click', () => {
    const userMessage = userInput.value;
    chatBox.innerHTML += `<p>User: ${userMessage}</p>`;

    // Send user message to the server for processing
    fetch('/chat', {
        method: 'POST',
        body: JSON.stringify({ userMessage }),
        headers: {
            'Content-Type': 'application/json',
        },
    })
    .then(response => response.json())
    .then(data => {
        const chatbotMessage = data.chatbotMessage;
        chatBox.innerHTML += `<p>Chatbot: ${chatbotMessage}</p>`;
```

```
    });


    userInput.value = '';
});
```

Step 5:

Implement Chatbot Logic in Flask

1. In app.py, add a new route to handle user messages and return chatbot responses.

Python code:

```
import json

# Import your chatbot implementation here

@app.route('/chat', methods=['POST'])

def chat():

    data = json.loads(request.data)

    user_message = data['userMessage']

    # Implement your chatbot logic here

    chatbot_response = get_chatbot_response(user_message)

    return json.dumps({'chatbotMessage': chatbot_response})
```

2. Implement the get_chatbot_response function using your existing chatbot logic. This function should take a user message as input and return the chatbot's response.

Step 6:

Run Your Flask App

1.Run your Flask app using the following command:

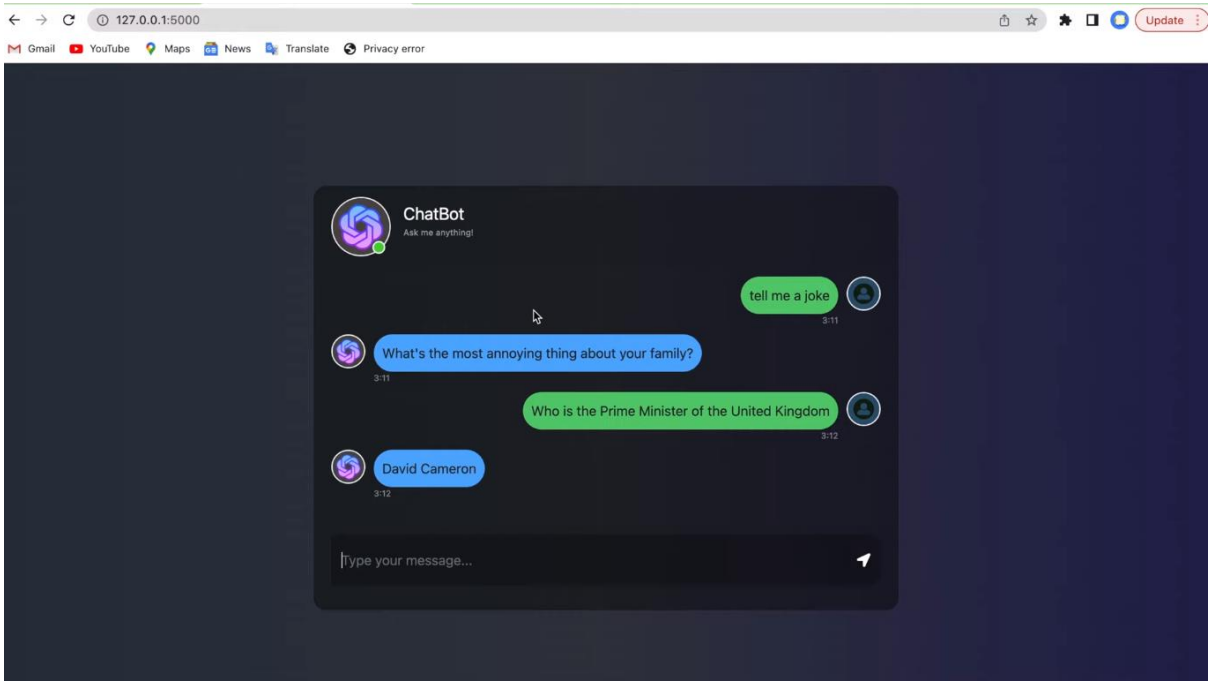
Python Code:

```
python app.py
```

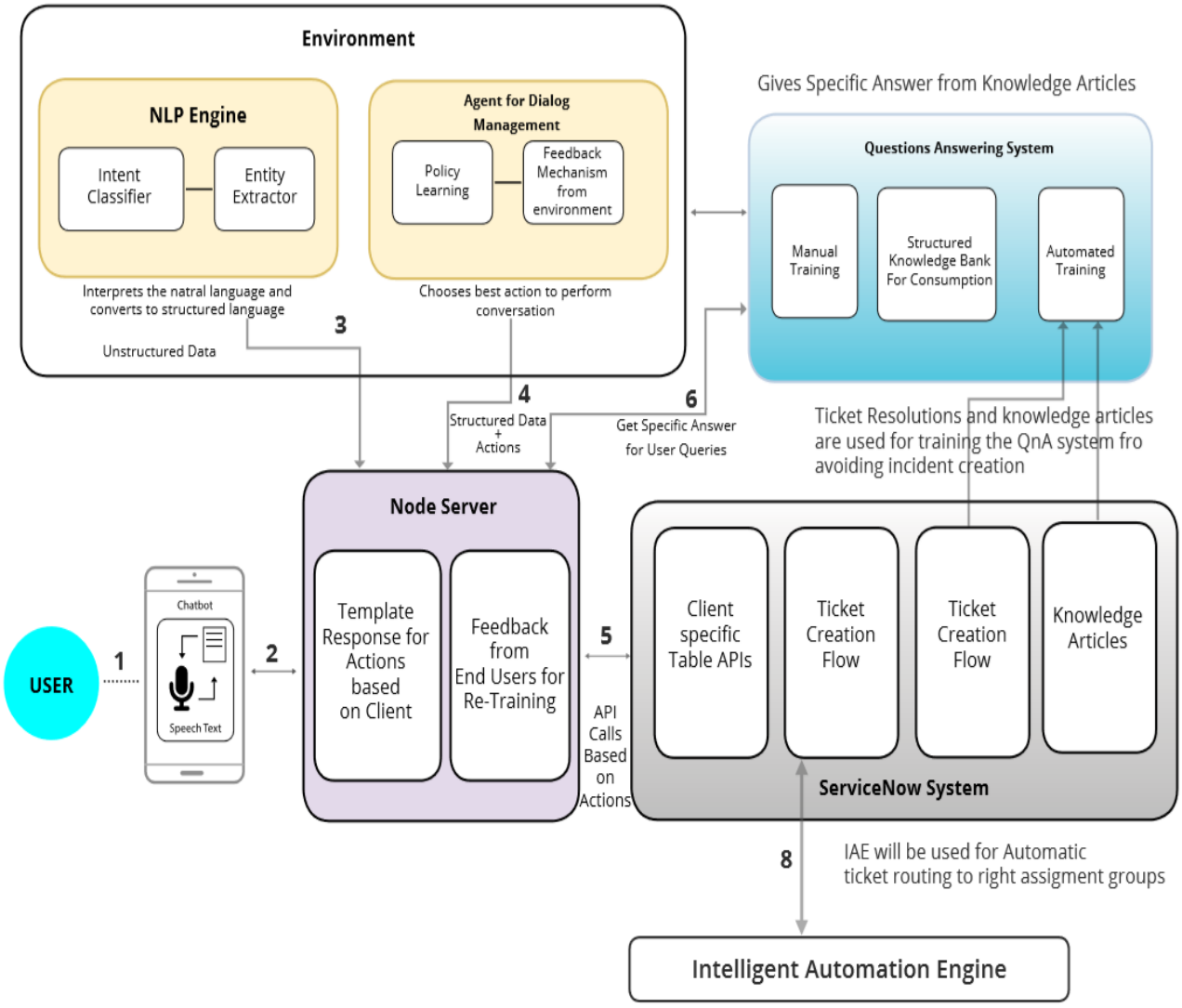
2**.** Access your chatbot web app by opening a web browser and navigating to http:// 127.0.0.1.5000.

Our chatbot is now integrated into a Flask web app. Users can interact with it through the web interface. Make sure you adapt the chatbot logic and responses to suit your specific use case and chatbot implementation. You can also enhance the web interface to make it more interactive and visually appealing

Output:



Architectural Diagram in Chatbot:


Environment
NLP Engine
Intent Classifier
Entity Extractor
Agent for Dialog Management
Policy Learning
Feedback Mechanism from environment
Interprets the natral language and converts to structured language
Chooses best action to perform conversation
Unstructured Data
3
4
Structured Data + Actions
6
Get Specific Answer for User Queries
Gives Specific Answer from Knowledge Articles
Questions Answering System
Manual Training
Structured Knowledge Bank For Consumption
Automated Training
Ticket Resolutions and knowledge articles are used for training the QnA system fro avoiding incident creation
Node Server
Template Response for Actions based on Client
Feedback from End Users for Re-Training
5
API Calls Based on Actions
Client specific Table APIs
Ticket Creation Flow
Ticket Creation Flow
Knowledge Articles
ServiceNow System
USER
1
Chatbot
Speech Text
2
8
IAE will be used for Automatic ticket routing to right assigment groups
Intelligent Automation Engine