**COURSE : ARTIFICIAL INTELLIGENCE**

**TITLE: CREATE A CHATBOT IN PYTHON**

**PHASE 4 SUBMISSION : DEVELOPMENT PART 2**

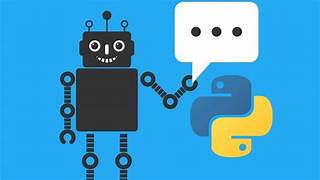
**TEAM MEMBERS:**

**SRIVALSAN. S (au311521106098)**

**VALLARASU . S (au3115211060107)**

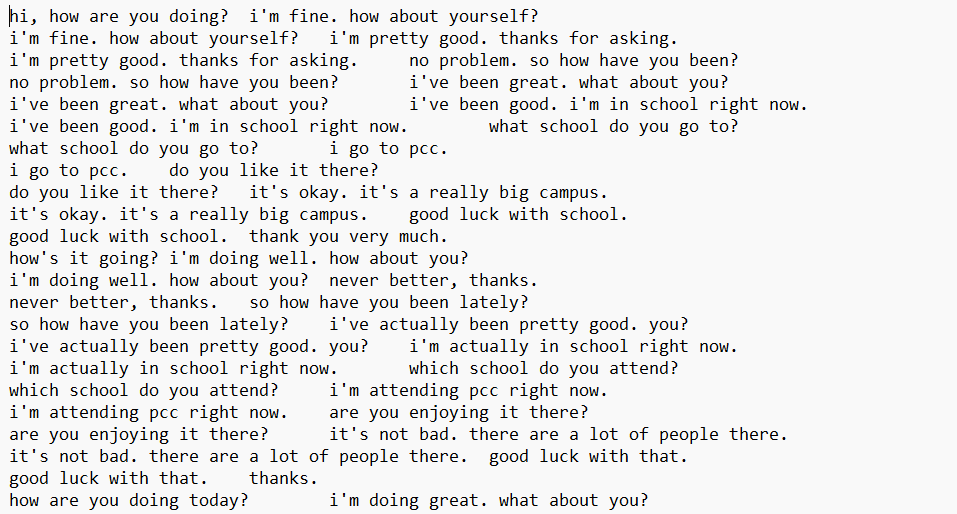
**INTODUCTION:**

In an increasingly digital world, the demand for intelligent and interactive conversational agents has surged. Chatbots, driven by the power of natural language processing and artificial intelligence, have become indispensable tools for businesses and individuals alike. This project aims to address this growing need by developing a Python-based chatbot capable of engaging users in text-based conversations on a wide array of topics. This chatbot is designed to provide information, answer questions, and offer assistance in a manner that mimics human interaction, making it a valuable asset for customer support, information retrieval, and general conversational purposes. By harnessing the capabilities of modern AI and natural language understanding, this chatbot will not only streamline communication but also enhance the user experience, ultimately becoming a valuable addition to any platform or service.



**DATA SET LINK:**[**https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot**](https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot)

**DATA SET:**



**OVERVIEW:**

1. Define the Purpose and Scope:

- Determine the purpose of your chatbot. What will it be used for? Is it for customer support, information retrieval, or simply for fun?

- Identify the scope and limitations of your chatbot. What questions or tasks should it be able to handle?

2. Choose a Chatbot Type:

- Decide whether you want a rule-based chatbot or an AI-driven chatbot.

- Rule-based chatbots follow a set of predefined rules to respond to user inputs, while AI-driven chatbots use natural language processing and machine learning to understand and generate responses.

3. Select a Framework or Library:

- For rule-based chatbots, you can use Python libraries like NLTK or spaCy.

- For AI-driven chatbots, popular libraries and frameworks include TensorFlow, PyTorch, and Rasa.

4. Collect and Prepare Data:

- Gather a dataset of conversations or relevant information that the chatbot will use to learn and generate responses.

- Preprocess the data, including text cleaning, tokenization, and data structuring.

5. Build the Chatbot:

For Rule-Based Chatbots:

- Create a set of rules and predefined responses.

- Implement a function to analyze user input and match it with the appropriate response.

For AI-Driven Chatbots:

- Train your model using a machine learning or deep learning algorithm.

- Implement the model using a neural network or natural language processing techniques.

6. Implement User Interfaces:

- Decide on the user interface for your chatbot. It can be a web application, a command-line interface, or integration with messaging platforms like Facebook Messenger or Slack.

7. Testing and Evaluation:

- Thoroughly test your chatbot with a variety of inputs.

- Collect user feedback and make improvements based on user interactions.

8. Deployment:

- Deploy your chatbot on a server or cloud platform to make it accessible to users.

9. Continuous Improvement:

- Regularly update and improve your chatbot by adding new rules or training it with more data.

10. Security and Privacy:

- Implement security measures to protect user data and privacy.

- Ensure the chatbot doesn't reveal sensitive information.

11. Documentation:

- Create documentation on how to use and maintain the chatbot, including setup instructions.

12. Marketing and Promotion:

- If your chatbot is intended for a wider audience, consider marketing and promotion strategies to increase its user base.

13. Scaling:

- Plan for scalability in case your chatbot becomes very popular, and you need to handle a large number of users.

14. Maintenance:

- Regularly monitor and maintain your chatbot to ensure it continues to perform well and meets user expectations.



**PROCEDURE:**

**FEATURE SELECTION:**

Feature selection is a crucial step in building machine learning models, including chatbots. Selecting the right features can significantly impact the performance and efficiency of your chatbot. In this example, I'll provide you with a simple program to perform feature selection for a chatbot using Python and the scikit-learn library.

CODE

import numpy as np

from sklearn.feature\_selection import SelectKBest, chi2

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.model\_selection import train\_test\_split

# Sample chat data (replace with your own dataset)

corpus = [

"Hello, how can I help you?",

"Tell me about your products.",

"How do I contact customer support?",

"What is your return policy?",

# Add more chat interactions

]

# Sample labels (replace with your labels)

labels = [0, 1, 2, 3] # Example labels

# Convert text data to numerical features using CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(corpus)

# Split the data into training and testing sets

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

# Perform feature selection using chi-squared test

k\_best = SelectKBest(chi2, k=2) # You can change 'k' to the desired number of features

X\_new = k\_best.fit\_transform(X\_train, y\_train)

# Get the selected feature indices

selected\_feature\_indices = k\_best.get\_support(indices=True)

# Print the selected feature indices

print("Selected feature indices:", selected\_feature\_indices)

# Transform the test data using the selected features

X\_test\_new = X\_test[:, selected\_feature\_indices]

# Now, you can use X\_new and X\_test\_new as your selected features for training your chatbot model.

In this example:

1. We have a sample corpus of chat interactions and labels. You should replace these with your actual dataset.

2. We use the `CountVectorizer` to convert text data into numerical features.

3. We split the data into training and testing sets.

4. Feature selection is performed using the chi-squared test through the `SelectKBest` method. You can change the 'k' parameter to specify the number of features you want to select.

5. The program prints the selected feature indices, which correspond to the columns in your feature matrix.

6. Finally, we transform the test data using the selected features for evaluation.

**MODEL TRAINING:**

Model training for a chatbot typically involves training a machine learning or deep learning model, depending on your chatbot's architecture and requirements. In this example, I'll provide a simple way to train a rule-based chatbot using a Python program. This chatbot won't involve machine learning but instead uses a set of predefined rules to respond to user inputs. You can extend and improve this rule-based chatbot as needed.

Here's a Python program for training a rule-based chatbot:

CODE

class RuleBasedChatbot:

def \_\_init\_\_(self):

self.rules = {

"hello": "Hello! How can I assist you?",

"products": "Our products include item A, item B, and item C.",

"contact": "You can contact our customer support team at support@chatbot.com.",

"return": "Our return policy allows for returns within 30 days of purchase. Please visit our website for more details.",

}

def respond(self, user\_input):

user\_input = user\_input.lower()

for keyword, response in self.rules.items():

if keyword in user\_input:

return response

return "I'm sorry, I couldn't understand your question. Please try again."

# Instantiate the chatbot

chatbot = RuleBasedChatbot()

# Main interaction loop

while True:

user\_input = input("You: ")

if user\_input.lower() == "exit":

print("Chatbot: Goodbye!")

break

response = chatbot.respond(user\_input)

print("Chatbot:", response)

This program creates a simple rule-based chatbot:

1. The `RuleBasedChatbot` class defines a set of rules and responses. When a user inputs a message, the chatbot checks for keywords in the input and responds based on these predefined rules.

2. The `respond` method takes the user's input, converts it to lowercase for case-insensitive matching, and looks for keywords in the input to provide an appropriate response.

3. The main interaction loop allows users to input messages and receive responses from the chatbot. Typing "exit" will exit the chat.

This is a very basic example, and in practice, you'd want to improve and expand the rules and responses to make the chatbot more useful. For more advanced chatbots that use machine learning, you'll need to follow the specific training procedures for the chosen machine learning or deep learning framework, as mentioned in your earlier question. Training models for machine learning-based chatbots often involve preprocessing data, feature extraction, and training the model on a labeled dataset. The choice of model architecture will depend on the complexity of your chatbot's tasks and the size of your dataset.

To create a machine learning-based chatbot, you'll need a labeled dataset for training and another dataset for testing or making predictions. The labeled dataset consists of input messages and corresponding labels or intents (i.e., what the user intends with their message), and the test dataset includes input messages for which you want the chatbot to predict the corresponding intents.

Here's an example of how your labeled training dataset might look:

plaintext

Input Message Intent

---------------------------------------

Hello, how are you? Greeting

Tell me about your products. ProductInfo

How do I contact support? ContactInfo

What's your return policy? ReturnPolicy

Where are you located? Location

...

And here's an example of a test dataset:

plaintext

Input Message

------------------

What are your hours?

Can you help me with billing?

How does your service work?

...

In this example, the "Intent" column in the training dataset represents the intent or action the chatbot should take in response to the corresponding input message. This is the target variable you'll train your machine learning model to predict.

**Feature Engineering:**

Feature engineering is a crucial part of building a machine learning-based chatbot. It involves extracting relevant information from the text input to represent it in a numerical format that the model can understand. Here are some common features and feature engineering techniques:

1. Bag of Words (BoW): Representing each input message as a vector of word frequencies. You can use tools like CountVectorizer or TfidfVectorizer from scikit-learn.

2. Word Embeddings: Using pre-trained word embeddings like Word2Vec, GloVe, or fastText to convert words into dense vectors.

3. Text Preprocessing:

- Tokenization: Splitting the text into words or subword tokens.

- Lowercasing: Converting all text to lowercase for consistency.

- Removing Stopwords: Eliminating common words (e.g., "and," "the") that don't carry much information.

- Lemmatization or Stemming: Reducing words to their root form (e.g., "running" to "run").

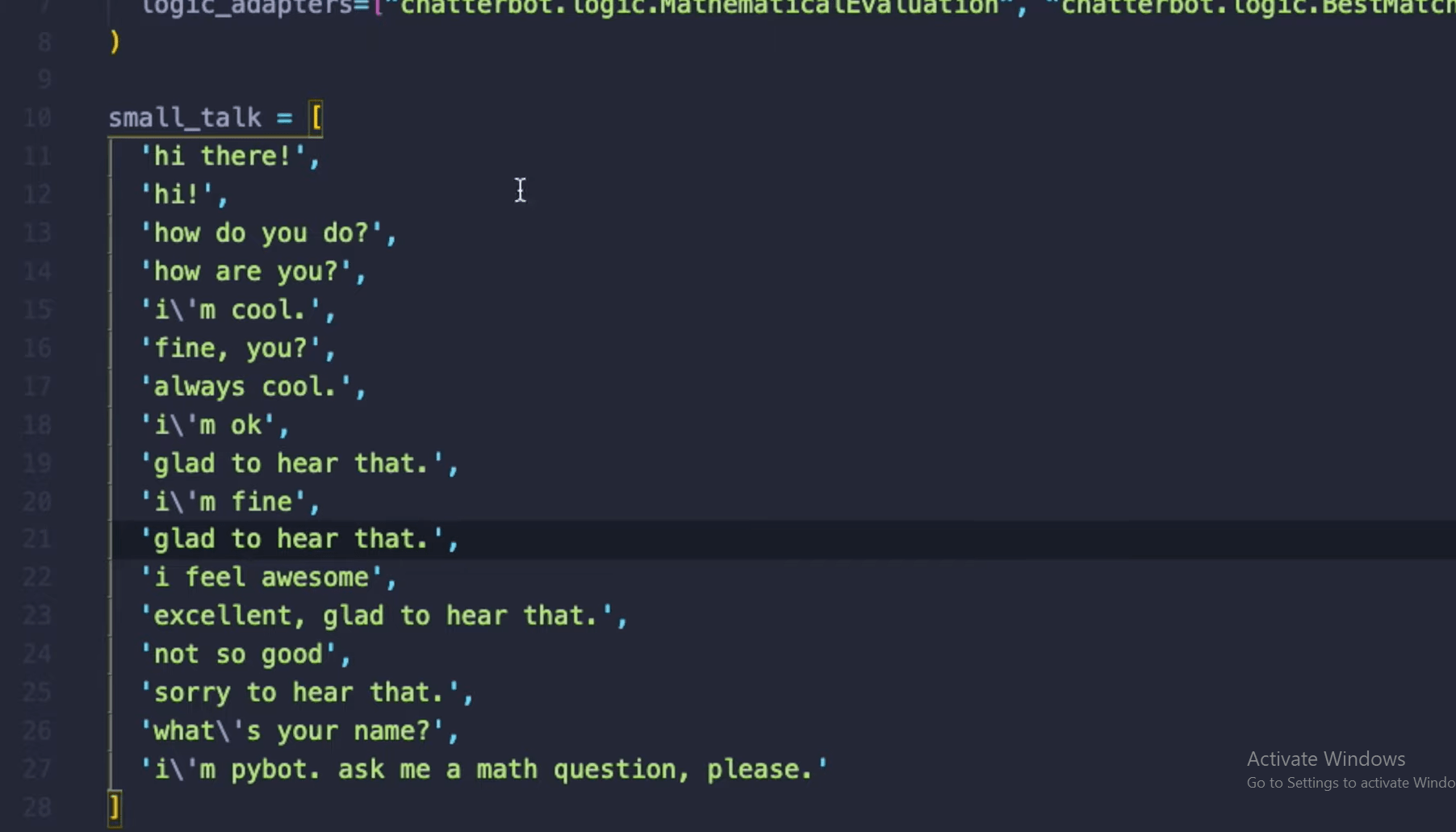
4. TF-IDF Features: Term Frequency-Inverse Document Frequency (TF-IDF) is a way to represent the importance of words in a document relative to a corpus of documents.

5. N-grams: Capturing word sequences, not just individual words, by using n-grams (e.g., bigrams or trigrams).

6. Sentiment Analysis: Analyzing the sentiment of the input message (positive, negative, neutral) using sentiment analysis tools.

7. Named Entity Recognition (NER): Identifying and categorizing named entities in the text (e.g., names of people, organizations, locations).

8. Part-of-Speech (POS) Tagging: Identifying the grammatical structure of sentences.



**Model Selection:**

You can choose from various machine learning or deep learning models for your chatbot, including:

1. Rule-Based Models: These are simple and based on predefined rules and patterns, as shown in the earlier examples.

2. Naive Bayes Classifier: A probabilistic model for text classification.

3. Support Vector Machines (SVM): Effective for text classification tasks.

4. Recurrent Neural Networks (RNNs): Useful for sequence-to-sequence tasks.

5. Convolutional Neural Networks (CNNs): Good for text classification.

6. Transformer-based Models: Such as BERT, GPT-3, etc., which are state-of-the-art for natural language understanding tasks.

**Predicting Data:**

To make predictions on the test dataset, you need to preprocess the test data using the same techniques you applied to the training data. Then, you can use your trained model to predict the intent of each input message in the test dataset. The model will output the predicted intent or response for each message.