CREATE A CHATBOT IN PYTHON

GROUP:5

TEAM MEMBER

950921104029:R.RESHMA

PROJECT:Loading and Preprocessing techniques in Chatbot.

OBJECTIVE:

Loading and preprocessing techniques are essential steps in chatbot development. These techniques involve getting your dataset ready for training and ensuring that the data is in a suitable format for the chatbot model. We can either create Data preprocessing is a critical step to make the dataset suitable for Training our dataset or use an existing one. Common choices include dialogue datasets, like Cornell Movie Dialogs Corpus or Twitter conversations. The specific preprocessing techniques we use will depend on the nature of our chatbot and the model we are using. Remember that NLP models like RNNs, LSTMs, Transformers, and retrieval-based models may require different types of preprocessing. Adjust the preprocessing steps based on our model and use case.

LOADING DATA:

"Loading data" in the context of chatbot development refers to the process of acquiring and making available the necessary dataset that will be used for training and, in some cases, testing the chatbot. This dataset typically consists of conversational data, including user messages or questions and their corresponding bot responses. Loading data involves the following steps:

1.DATASOURCE:

Determine where your chatbot's data will come from. This can include scraping websites, using publicly available datasets, or creating your own dataset through data collection tools.

PYTHON PROGRAM:

```
import pandas as pd



# Load your custom dataset from a CSV file

data = pd.read_csv('your_dataset.csv')
```

```
# Ensure your dataset has columns like 'user_message' and 'bot_response'

user_messages = data['user_message']

bot_responses = data['bot_response']



# Now, you can use this data to train and test your chatbot.
```

2.DATA COLLECTION:

Gather the text data you need for your chatbot's training. This data may consist of conversations, question-and-answer pairs, or any text that the chatbot should learn from. Ensure that the data is well-structured and organized. Data collection for building a chatbot in Python involves gathering a dataset of conversational data, which consists of input messages or questions along with their corresponding responses. After collecting the data, you can store it in a suitable format, such as CSV, JSON, or a database, and use Python libraries like to load and preprocess the dataset, as mentioned in the previous responses. Once your data is ready, you can start building and training your chatbot using machine learning or natural language processing techniques, and use it to generate responses to user queries.

PYTHON PROGRAM:

```
# Data Collection

# You can manually collect conversational data or use synthetic data.

conversations = [

    {"user": "Hello", "bot": "Hi there! How can I assist you today?"},

    {"user": "What's the weather like?", "bot": "I'm sorry, I don't have that information."},

    {"user": "Tell me a joke", "bot": "Sure! Why did the chicken cross the road? To get to the other side."},

    # Add more user-bot interactions here

]



# Data Loading

# You can load the collected data into a Python list or a more structured format like a CSV file or a database.

# For this simple example, we're using a list of dictionaries.
# Generating Responses (Rule-Based Chatbot)
def generate_response(user_message):
    # Simple rule-based responses
    if "hello" in user_message.lower():
        return "Hi there! How can I assist you today?"
    elif "weather" in user_message.lower():
        return "I'm sorry, I don't have that information."
    elif "joke" in user_message.lower():
        return "Sure! Why did the chicken cross the road? To get to the other side."
    else:
        return "I'm not sure how to respond to that."


# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = generate_response(user_input)
    print("Chatbot:", response)
```

OUTPUT:

```
You: Hello
Chatbot: Hi there! How can I assist you today?
```

```
You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.



You: Tell me a joke

Chatbot: Sure! Why did the chicken cross the road? To get to the other side.



You: How are you?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

3.DATA FORMATE:

Your data may be stored in various formats, such as CSV, JSON, text files, or databases. Depending on the format, you'll need to use appropriate Python libraries to read and parse the data. The data format for loading a chatbot in Python can vary based on your specific needs and the framework or libraries you're using for chatbot development. However, a common approach is to use structured data, which can be easily loaded and processed using Python libraries like pandas. The data format typically includes two main components.

PYTHON PROGRAM:

```
user_message,bot_response

Hello,Hi there! How can I assist you today?

What's the weather like?,I'm sorry, I don't have that information.

Tell me a joke,Sure! Why did the chicken cross the road? To get to the other side.

import pandas as pd



# Load the dataset from the CSV file

data = pd.read_csv('chatbot_data.csv')
```

```
# Extract user messages and bot responses
user_messages = data['user_message']
bot_responses = data['bot_response']


# Define a function to generate responses based on user input
def generate_response(user_input):
    for i, user_msg in enumerate(user_messages):
        if user_input.lower() in user_msg.lower():
            return bot_responses[i]
    return "I'm not sure how to respond to that."


# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = generate_response(user_input)
    print("Chatbot:", response)
```

OUTPUT:

```
You: Hello
Chatbot: Hi there! How can I assist you today?


You: What's the weather like?
```

```
Chatbot: I'm sorry, I don't have that information.


You: Tell me a joke

Chatbot: Sure! Why did the chicken cross the road? To get to the other side.


You: How are you?

Chatbot: I'm not sure how to respond to that.


You: exit

Chatbot: Goodbye!
```

4.DATA CLEANING:

Clean the data by removing any irrelevant or redundant information. This may involve removing HTML tags, special characters, or anything that doesn't contribute to the chatbot's learning. Data cleaning is an essential step in chatbot development, ensuring that the data used for training and inference is in a suitable format. Data cleaning requirements may vary depending on your specific dataset and use case. More advanced chatbots often require more sophisticated preprocessing and may use techniques such as lemmatization and tokenization.

PYTHON PROGRAM:

```
import pandas as pd

import string


# Load the dataset from the CSV file

data = pd.read_csv('chatbot_data.csv')


# Define a function for data cleaning

def clean_text(text):

    # Remove punctuation and convert to lowercase
    text = ''.join([char for char in text if char not in string.punctuation])
    return text.lower()


# Apply data cleaning to user messages and bot responses
data['user_message'] = data['user_message'].apply(clean_text)
data['bot_response'] = data['bot_response'].apply(clean_text)


# Define a function to generate responses based on user input
def generate_response(user_input):
    for i, user_msg in enumerate(data['user_message']):
        if user_input.lower() in user_msg:
            return data['bot_response'][i]
    return "I'm not sure how to respond to that."


# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = generate_response(clean_text(user_input))
    print("Chatbot:", response)
```

<u>OUTPUT:</u>

```
You: Hello!
Chatbot: Hi there! How can I assist you today?
```

```
You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.



You: Tell me a joke

Chatbot: Sure! Why did the chicken cross the road? To get to the other side.



You: How are you?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

PREPROCESSING DATA:

Preprocessing in the context of chatbot development refers to the techniques and steps used to prepare and clean the data before using it to train and operate the chatbot. Data preprocessing is essential to ensure that the input data is in a suitable format, that it represents the desired context, and that it can be effectively processed by the chatbot's natural language processing (NLP) or machine learning models. Here are some common preprocessing steps in chatbot development:

1.TOKENIZATION:

Tokenization is the process of breaking down text into individual words, phrases, or tokens. This step is crucial for understanding the structure of sentences and conversations. tokenization is a fundamental step in processing user input. Here's an example of how to tokenize user messages and generate output using Python. For this example, we'll use the **nltk** library for tokenization, and we'll simulate a basic rule-based chatbot.

PYTHON PROGRAM:

```
pip install nltk

import nltk

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords


# Download the NLTK data (if not already downloaded)

nltk.download('punkt')

nltk.download('stopwords')


# Sample user messages and responses

user_messages = [

    "Hello, how are you?",

    "Tell me a joke",

    "What's the weather like?",

    "Who won the world series in 2020?",

]


# Tokenize user messages

def tokenize_user_message(user_message):

    tokens = word_tokenize(user_message)

    tokens = [word for word in tokens if word.lower() not in stopwords.words('english')]

    return tokens


# Simulated responses

responses = {

    "hello": "Hi there! I'm doing well. How can I help you?",

    "joke": "Sure! Why did the chicken cross the road? To get to the other side.",

    "weather": "I'm sorry, I don't have that information.",
    "world series": "The Los Angeles Dodgers won the World Series in 2020."
}

# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break

    user_tokens = tokenize_user_message(user_input)
    bot_response = "I'm not sure how to respond to that."

    # Determine the intent and generate a response
    for token in user_tokens:
        if token.lower() in responses:
            bot_response = responses[token.lower()]
            break

    print("Chatbot:", bot_response)
```

OUTPUT:

```
You: Hello, how are you?
Chatbot: Hi there! I'm doing well. How can I help you?

You: Tell me a joke
```

```
Chatbot: Sure! Why did the chicken cross the road? To get to the other side.



You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.



You: Who won the world series in 2020?

Chatbot: The Los Angeles Dodgers won the World Series in 2020.



You: What's the capital of France?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

2.LOWER CASTING:

Converting all text to lowercase helps ensure that the chatbot recognizes words regardless of their capitalization. This is important for consistency in text processing. Lowercasing is a common text preprocessing step in chatbot development to ensure uniformity in text and to facilitate matching user input. Here's an example of how to perform lowercasing and generate output using Python for a simple rule-based chatbot.

PYTHON PROGRAM:

```
# Sample user messages and responses

user_messages = [

    "Hello, how are you?",

    "Tell me a joke",

    "What's the weather like?",

    "Who won the World Series in 2020?",

]
```

```
# Convert user messages to lowercase
user_messages = [message.lower() for message in user_messages]

# Simulated responses
responses = {
    "hello": "Hi there! I'm doing well. How can I help you?",
    "joke": "Sure! Why did the chicken cross the road? To get to the other side.",
    "weather": "I'm sorry, I don't have that information.",
    "world series": "The Los Angeles Dodgers won the World Series in 2020."
}

# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break

    user_input_lower = user_input.lower()
    bot_response = "I'm not sure how to respond to that."

    # Determine the intent and generate a response
    for message in user_messages:
        if message in user_input_lower:
            bot_response = responses[message]

            break



    print("Chatbot:", bot_response)
```

OUTPUT:

```
You: Hello, how are you?

Chatbot: Hi there! I'm doing well. How can I help you?



You: Tell me a joke

Chatbot: Sure! Why did the chicken cross the road? To get to the other side.



You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.



You: Who won the World Series in 2020?

Chatbot: The Los Angeles Dodgers won the World Series in 2020.



You: What's the capital of France?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

3.STEMMING:

Stemming are techniques used to reduce words to their root or base form. This helps the chatbot recognize variations of the same word and reduces the vocabulary size**.** Stemming is a text preprocessing technique that reduces words to their root or base form. In this Python example, we'll use the NLTK library for stemming and generate output using a simple rule-based chatbot. You'll need to install the **nltk** library if you haven't already.

PYTHON PROGRAME:

```
import nltk

from nltk.stem import PorterStemmer



# Download the NLTK data (if not already downloaded)

nltk.download('punkt')



# Initialize the Porter Stemmer

stemmer = PorterStemmer()



# Sample user messages and responses

user_messages = [

    "running",

    "flies",

    "jogging",

    "ran",

    "jogged",

]



# Stem the user messages

stemmed_user_messages = [stemmer.stem(message) for message in user_messages]



# Simulated responses
```

```
responses = {
    "run": "Running is great exercise!",
    "jog": "Jogging is a healthy activity.",
    "fli": "Flies can be quite annoying."
}


# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break


    user_input_lower = user_input.lower()
    user_input_stemmed = stemmer.stem(user_input_lower)
    bot_response = "I'm not sure how to respond to that."


    # Determine the intent and generate a response
    for message in stemmed_user_messages:
        if message in user_input_stemmed:
            bot_response = responses.get(stemmer.stem(message), "I'm not sure how to respond to that.")
            break


    print("Chatbot:", bot_response)
```

OUTPUT:

```
You: Running

Chatbot: Running is great exercise!



You: Jogged

Chatbot: Jogging is a healthy activity.



You: Flies

Chatbot: Flies can be quite annoying.



You: Ran

Chatbot: Running is great exercise!



You: What's the weather like?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

4.PADDING:

If your chatbot uses sequence models, ensuring that all input sequences are of the same length is crucial. You may need to pad shorter sequences with special tokens or truncate longer ones. Padding is typically used when you're working with sequences of text, such as in sequence-to-sequence models or recurrent neural networks. In this example, I'll demonstrate how to add padding to sequences of text and generate output. We'll use the **tensorflow** library to pad sequences and simulate a simple chatbot interaction.

PYTHON PROGRAM:

```
pip install tensorflow
```

```
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences


# Sample user messages and responses

user_messages = [

    "Tell me a joke",

    "What's the weather like?",

    "Who won the World Series in 2020?",

]


# Simulated responses

responses = {

    "joke": "Sure! Why did the chicken cross the road? To get to the other side.",

    "weather": "I'm sorry, I don't have that information.",

    "world series": "The Los Angeles Dodgers won the World Series in 2020."

}


# Create a tokenizer

tokenizer = Tokenizer()

tokenizer.fit_on_texts(user_messages)

word_index = tokenizer.word_index


# Pad sequences

user_sequences = tokenizer.texts_to_sequences(user_messages)
user_sequences = pad_sequences(user_sequences, padding='post')


# Interaction Loop

while True:

    user_input = input("You: ")

    if user_input.lower() == "exit":

        print("Chatbot: Goodbye!")

        break


    # Tokenize and pad the user input

    user_input_sequence = tokenizer.texts_to_sequences([user_input])

    user_input_padded = pad_sequences(user_input_sequence,
maxlen=user_sequences.shape[1], padding='post')


    bot_response = "I'm not sure how to respond to that."


    # Determine the intent and generate a response

    for i, seq in enumerate(user_sequences):

        if tf.reduce_all(tf.equal(user_input_padded[0], seq)):

            bot_response = responses.get(user_messages[i].lower(), "I'm not sure how to respond
to that.")

            break


    print("Chatbot:", bot_response)
```

OUTPUT:

```
You: Tell me a joke
```

```
Chatbot: Sure! Why did the chicken cross the road? To get to the other side.



You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.



You: Who won the World Series in 2020?

Chatbot: The Los Angeles Dodgers won the World Series in 2020.



You: What's the capital of France?

Chatbot: I'm not sure how to respond to that.



You: exit

Chatbot: Goodbye!
```

4.VECTORIZATION:

Convert text data into numerical format, such as word embeddings or TF-IDF (Term Frequency-Inverse Document Frequency) representations, to make it compatible with machine learning models. Vectorization is a key step in converting text data into numerical form, making it suitable for machine learning models. Here's an example of how to perform vectorization and generate output using Python for a simple rule-based chatbot. We'll use the **CountVectorizer** from the scikit-learn library for vectorization.

PYTHON PROGRAM:

```
pip install scikit-learn

from sklearn.feature_extraction.text import CountVectorizer



# Sample user messages and responses

user_messages = [
    "Tell me a joke",
    "What's the weather like?",
    "Who won the World Series in 2020?",
]


# Simulated responses
responses = {
    "joke": "Sure! Why did the chicken cross the road? To get to the other side.",
    "weather": "I'm sorry, I don't have that information.",
    "world series": "The Los Angeles Dodgers won the World Series in 2020."
}


# Create a CountVectorizer
vectorizer = CountVectorizer()
user_message_vectors = vectorizer.fit_transform(user_messages)


# Interaction Loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break


    user_input_vector = vectorizer.transform([user_input])
    bot_response = "I'm not sure how to respond to that."


    # Determine the intent and generate a response

    for i, vector in enumerate(user_message_vectors):

        if (user_input_vector != vector).nnz == 0:

            bot_response = responses.get(user_messages[i].lower(), "I'm not sure how to respond to that.")

            break


    print("Chatbot:", bot_response)
```

OUTPUT:

```
You: Tell me a joke

Chatbot: Sure! Why did the chicken cross the road? To get to the other side.


You: What's the weather like?

Chatbot: I'm sorry, I don't have that information.


You: Who won the World Series in 2020?

Chatbot: The Los Angeles Dodgers won the World Series in 2020.


You: What's the capital of France?

Chatbot: I'm not sure how to respond to that.


You: exit

Chatbot: Goodbye!
```