DEVELOPMENT PART-2

Feature engineering is a crucial step in developing a chatbot using Python.

*****Abstract Explanation:*****

It is the process of transforming raw text data into meaningful numerical representations that can be used by a chatbot model. This involves extracting key information from text and converting it into features that the model can understand. Abstraction and modulation can be achieved through the following steps:

1. **Text Preprocessing:** Before feature extraction, preprocess the text by tokenizing (splitting into words or subwords), lowercasing, removing punctuation, and handling special cases like stemming or lemmatization.

2. **Feature Extraction:** This step involves identifying relevant information in the text and converting it into numerical features. Some common feature extraction techniques include:

- **Bag of Words (BoW):** Create a matrix where each row represents a document, and each column represents a unique word in the entire dataset. The cell values represent the word frequency in the document.

*- **TF-IDF (Term Frequency-Inverse Document Frequency):** Similar to BoW, but it considers the importance of a word in a document relative to its importance in the entire dataset.*

*- **Word Embeddings (e.g., Word2Vec or GloVe):** Pre-trained word embeddings can be used to convert words into dense vector representations.*

*- **Custom Features:** Create custom features based on domain-specific information, like sentiment scores, named entity recognition, or specific keywords.*

*3. **Modulation:** Modulation involves fine-tuning and optimizing the features for the specific chatbot task. This may include dimensionality reduction, feature scaling, or feature selection techniques.*

*4. **Feature Vector:** After modulation, you should have a feature vector for each document (or input) that represents the data in a format suitable for machine learning models.*

Now, let's provide a Python code example for creating a simple chatbot using these concepts:

```python
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity

# Sample dataset
corpus = [
    "What is a chatbot?",
    "How do chatbots work?",
    "Can you give me an example of a chatbot?",
    "What are the applications of chatbots?",
    "Tell me about Python programming.",
]

# Text preprocessing and feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)

# User input
user_input = "What are chatbot applications?"

# Preprocess user input
user_input_tfidf = tfidf_vectorizer.transform([user_input])

# Calculate cosine similarity between user input and dataset
cosine_similarities = cosine_similarity(user_input_tfidf,
tfidf_matrix)
```

 # Get the most similar response

most_similar_index = np.argmax(cosine_similarities)

response = corpus[most_similar_index]

 # Print the chatbot response

print("Chatbot: " + response)

```

**TEXT PROCESSING:**

*\*\*Abstraction of Text Processing:\*\**

*Text processing for a chatbot involves several key tasks:*

*1. \*\*User Input:\*\* Receive and interpret user messages.*

*2. \*\*Preprocessing:\*\* Clean and tokenize the user's message.*

*3. \*\*Intent Recognition:\*\* Determine the user's intent or what they want.*

*4. \*\*Entity Recognition:\*\* Identify specific entities (e.g., dates, locations) in the message.*

*5. \*\*Response Generation:\*\* Construct a meaningful response based on the intent and entities.*

*6. \*\*Output:\*\* Send the response to the user.*

*\*\*Python Code Example for Text Processing:\*\**
```

```bash
pip install nltk spacy
python -m spacy download en_core_web_sm
```

Now, let's write Python code for text processing:

```python
import nltk
from nltk.tokenize import word_tokenize
import spacy

# Initialize NLTK and spaCy
nltk.download('punkt')
nlp = spacy.load("en_core_web_sm")

# Sample intents and entities
intents = {
    "greet": ["hi", "hello", "hey"],
    "bye": ["goodbye", "bye", "see you"],
    "weather": ["weather", "forecast"],
}

entities = {
```

```
    "location": ["New York", "Los Angeles", "San Francisco"],
}

# User input
user_input = "What's the weather like in New York?"

# Tokenize user input
tokens = word_tokenize(user_input)

# Intent recognition
intent = None
for key, value in intents.items():
    if any(word in tokens for word in value):
        intent = key
        break

# Entity recognition
doc = nlp(user_input)
entities_found = [ent.text for ent in doc.ents if ent.label_ == "GPE"]
# Assuming "GPE" represents locations

# Response generation
if intent == "greet":
    response = "Hello there!"
```

```
elif intent == "bye":
    response = "Goodbye! Have a great day!"
elif intent == "weather" and entities_found:
    response = f"You asked about the weather in {entities_found[0]}."
else:
    response = "I'm not sure how to respond to that."

# Output
print("User: " + user_input)
print("Chatbot: " + response)
```

Output:

```
User: What's the weather like in New York?
Chatbot: You asked about the weather in New York.
```

FEATURE EXTRACTION:

Let's start with feature extraction:

```python
# Feature Extraction
```

```
def extract_features(user_input):
    features = {}
    # You can extract features from user_input here
    # For example, you might want to check for specific keywords
    if 'hello' in user_input:
        features['greeting'] = True
    elif 'goodbye' in user_input:
        features['farewell'] = True
    # Add more features as needed
    return features
```

Now, let's move on to abstraction:

```python
# Abstraction
def abstract_features(features):
    abstraction = 'unknown'
    # Define abstraction rules based on extracted features
    if 'greeting' in features:
        abstraction = 'greeting'
    elif 'farewell' in features:
        abstraction = 'farewell'
    # Add more rules as needed
```

```
    return abstraction
```

Lastly, modulation:

```python
# Modulation
def modulate_response(abstraction):
    response = 'I don't understand.'
    # Define responses based on the abstraction
    if abstraction == 'greeting':
        response = 'Hello! How can I assist you?'
    elif abstraction == 'farewell':
        response = 'Goodbye! Have a great day!'
    # Add more responses as needed
    return response
```

Here's how you can use these functions to create a chatbot:

```python
while True:
    user_input = input("You: ")
    features = extract_features(user_input)
```

abstraction = abstract_features(features)

response = modulate_response(abstraction)

print("Bot:", response)

```

## *MODULATION:*

*\*\*Step 1: Install NLTK and other dependencies\*\**

*You need to install NLTK and download the necessary corpora for this example. You can do this using pip:*

```bash

*pip install nltk*

```

Step 2: Create a Python Chatbot

```python

*import nltk*

*import random*

*from nltk.chat.util import Chat, reflections*

*# Define some simple rules for the chatbot*

```
pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, how can I help you today?", "Hi %1, what can I do for you?"],
    ],
    ["(what is your name|who are you)?", ["I am a chatbot.", "I'm just a bot."]],
    [
        r"how are you ?",
        ["I'm good, thanks for asking.", "I'm doing well, how about you?"],
    ],
    [
        r"(.*) (good|well|fine)",
        ["That's great to hear!", "Awesome!", "I'm happy for you."],
    ],
    [
        r"(.*) (help|support|assistance)",
        ["Sure, I can help you with that. How can I assist you?", "How may I assist you?"],
    ],
    [
        r"(.*) (thanks|thank you)",
        ["You're welcome!", "No problem, happy to help."],
```

```
    ],
    [
        r"(bye|goodbye|exit|quit)",
        ["Goodbye! Have a great day.", "See you later!"],
    ],
]

# Create a chatbot instance
chatbot = Chat(pairs, reflections)

# Start the conversation with the chatbot
print("Hello! I'm your chatbot. You can ask me anything. To exit, type 'bye'.")

while True:
    user_input = input("You: ")
    if user_input.lower() == "bye":
        print("Chatbot: Goodbye!")
        break
    response = chatbot.respond(user_input)
    print("Chatbot:", response)
```

Step 3: Run the Chatbot

Execute the script in your terminal or Python environment. You can interact with the chatbot by typing messages, and it will respond according to the defined rules.

Here's an example conversation with the chatbot:

```
*Hello! I'm your chatbot. You can ask me anything. To exit, type 'bye.'*
*You: Hi, my name is John.*
*Chatbot: Hello John, how can I help you today?*
*You: How are you doing?*
*Chatbot: I'm good, thanks for asking.*
*You: I need some assistance.*
*Chatbot: Sure, I can help you with that. How can I assist you?*
*You: Thanks!*
*Chatbot: You're welcome!*
*You: Goodbye*
*Chatbot: Goodbye!*
```

FEATURE VECTOR:

```python
```

```
import nltk
from nltk.chat.util import Chat, reflections

# Define reflection pairs for personal pronouns
reflections = {
    "i am": "you are",
    "i was": "you were",
    "i": "you",
    "my": "your",
    "me": "you",
    "you are": "I am",
    "you were": "I was",
    "your": "my",
    "yours": "mine",
    "you": "me",
}

# Define patterns for the chatbot
patterns = [
    (r"hi|hello|hey", ["Hello!", "Hi there!"]),
    (r"what is your name?", ["I am a chatbot.", "You can call me ChatGPT."]),
    (r"how are you?", ["I'm just a computer program, so I don't have feelings, but I'm here to help!"]),
```

```
    (r"(.*) help (.*)", ["I can help you with a variety of tasks. Please ask me a specific question."]),
    (r"(.*) your name?", ["I'm ChatGPT, a chatbot."]),
    (r"(.*) (thank you|thanks)", ["You're welcome!", "No problem."]),
    (r"quit|bye|exit", ["Goodbye!", "Bye for now."]),
]

# Create a chatbot using the defined patterns and reflections
chatbot = Chat(patterns, reflections)

# Main chat loop
print("Hello! I'm your chatbot. You can start a conversation or type 'quit' to exit.")
while True:
    user_input = input("You: ")
    if user_input.lower() == 'quit':
        print("Chatbot: Goodbye!")
        break
    response = chatbot.respond(user_input)
    print("Chatbot:", response)
```

In this code:

1. We define a list of patterns, where each pattern is a regular expression paired with a list of possible responses.

2. We use reflections to handle personal pronouns and make the conversation more engaging.

3. We create a chatbot instance using the `Chat` class from NLTK, passing in our patterns and reflections.

4. The chatbot enters a loop where it waits for user input and responds accordingly.

Here's an example interaction:

```
Hello! I'm your chatbot. You can start a conversation or type 'quit' to exit.
You: Hello
Chatbot: Hi there!
You: What is your name?
Chatbot: I'm a chatbot.
You: How are you?
Chatbot: I'm just a computer program, so I don't have feelings, but I'm here to help!
You: Thank you
```

Chatbot: You're welcome!

You: Bye

Chatbot: Goodbye!

```

## *MODEL TRAINING:*

**\*\*Abstraction of Model Training:\*\***

**1. \*\*Data Collection and Preprocessing\*\*:**

**- Gather chat data, which can include conversations, user queries, and responses.**

**- Preprocess the data, cleaning and structuring it for training.**

**2. \*\*Model Selection\*\*:**

**- Choose a chatbot model type, such as rule-based, retrieval-based, or generative (e.g., using Rasa, Seq2Seq, or Transformer models).**

**3. \*\*Training the Model\*\*:**

**- For rule-based chatbots, define rules and responses.**

**- For retrieval-based models, build an index for responses.**

**- For generative models, use a large dataset for training.**

**4. \*\*Evaluation\*\*:**

**- Evaluate the model using metrics like accuracy, F1 score, or human evaluation.**
```

5. **Fine-Tuning:**

- Make adjustments based on evaluation results.

6. **Deployment:**

- Deploy the chatbot to the desired platform, such as a website, messaging app, or social media.

****Python Code for a Simple Rule-Based Chatbot:****

Below is a basic Python code for a rule-based chatbot that responds to predefined rules and patterns:

```python
import random

# Define chatbot rules
chatbot_rules = {
    "hello": ["Hi there!", "Hello!", "Hey!"],
    "how are you": ["I'm just a chatbot, but I'm here to help!", "I don't have feelings, but thanks for asking."],
    "bye": ["Goodbye!", "See you later!", "Bye now!"]
}

# Function to generate a response
def chatbot_response(user_input):
    user_input = user_input.lower()
```

```
    for pattern, responses in chatbot_rules.items():
        if pattern in user_input:
            return random.choice(responses)
    return "I'm not sure how to respond to that."

# Main loop for the chatbot
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = chatbot_response(user_input)
    print("Chatbot:", response)
```

Sample Output:

```
You: hello
Chatbot: Hi there!

You: How are you doing?
Chatbot: I'm just a chatbot, but I'm here to help!

You: What's the weather like today?
Chatbot: I'm not sure how to respond to that.
```

You: bye

Chatbot: Goodbye!

You: exit

Chatbot: Goodbye!

```

## EVALUATION:

## **1. Abstraction:**

## Abstraction is the process of identifying the essential features and functionalities of your chatbot. This involves defining the chatbot's purpose, the types of interactions it will support, and the core components needed.

## **2. Modulation of Evaluation:**

```python

## import random

## # Define a list of predefined rules for the chatbot

## rules = [

```
    {
        'input': ['hi', 'hello', 'hey'],
        'output': ['Hello!', 'Hi there!', 'Hey! How can I assist you?']
    },
    {
        'input': ['how are you', 'how are you doing'],
        'output': ['I am just a chatbot, but thanks for asking!', 'I'm doing well, thanks! How can I help you?']
    },
    {
        'input': ['bye', 'goodbye', 'see you later'],
        'output': ['Goodbye!', 'See you later!', 'Have a great day!']
    },
    {
        'input': ['help', 'instructions'],
        'output': ['I can answer questions and have general conversations. Just type your question or say hi!']
    },
    {
        'input': ['weather', 'temperature'],
```

```
        'output': ['I'm sorry, I can't provide real-time weather information. You can check a weather website or app.']
    },
    {
        'input': ['thanks', 'thank you'],
        'output': ['You're welcome!', 'No problem. How else can I assist you?']
    },
    {
        'input': ['default'],
        'output': ["I'm not sure I understand. Can you please rephrase your question?"]
    }
]

def chatbot_response(user_input):
    user_input = user_input.lower()
    for rule in rules:
        if any(keyword in user_input for keyword in rule['input']):
            return random.choice(rule['output'])
    return random.choice(rules[-1]['output'])  # Default response if no rules matched
```

```
# Main chat loop
print("Chatbot: Hello! How can I assist you today?")
while True:
    user_input = input("You: ")
    if user_input.lower() in ['exit', 'quit', 'bye']:
        print("Chatbot: Goodbye!")
        break
    response = chatbot_response(user_input)
    print("Chatbot:", response)
```

Sample Output:

```
Chatbot: Hello! How can I assist you today?
You: Hi
Chatbot: Hi there!
You: What's the weather like today?
Chatbot: I'm sorry, I can't provide real-time weather information. You can check a weather website or app.
You: Thank you
```

Chatbot: You're welcome!

You: Have a nice day

Chatbot: I'm not sure I understand. Can you please rephrase your question?

You: Bye

Chatbot: Goodbye!

```
```