CHATBOT DEPLOYMENT WITH IBM CLOUD WATSON ASSISTANT

PHASE 4: DEVELOPMENT –PART 2

This document contains all about feature engineering,model training and evaluation of a chatbot.

FEATURE ENGINEERING:

Feature engineering in the context of chatbots refers to the process of designing and creating features or components that enable a chatbot to understand and respond to user inputs effectively. Feature engineering is essential for improving the chatbot's performance, enhancing its ability to recognize intents, entities, and context, and providing a better user experience. Here are some key aspects of feature engineering in chatbot development:

1.Intent and Entity Recognition:

- Define user intents and entities to improve the chatbot's understanding of user queries.
- For instance, we can define intents like "order_pizza" and entities like "topping" and "size."

```
            "intents": [
  {
      "intent": "order_pizza",
      "examples": [
         {"text": "I want to order a [size] pizza with [topping]."}
      ]
  }
],
"entities": [
  {
      "entity": "size",
      "values": [
         {
            "value": "small",
            "synonyms": ["personal"]
```

```
          },
          {
            "value": "medium",
            "synonyms": ["regular"]
          }
        ]
      },
      {
        "entity": "topping",
        "values": [
          {
            "value": "pepperoni",
            "synonyms": ["pep"]
          },
          {
            "value": "mushroom"
          }
        ]
      }
    ]
```

2. Dialog Flow:

- Design a conversation flow that guides users through the chatbot interaction. Dialog nodes should be used to structure the conversation.

```
"dialog_nodes": [
  {
    "type": "standard",
    "title": "Welcome",
        "output": {
          "generic": [
            {
              "response_type": "text",
              "values": [
                {
                  "text": "Hello! How can I assist you today?"
                }
              ]
            }
          ]
        }
      },
      {
        "type": "response",
        "title": "Order Pizza",
        "output": {
          "generic": [
            {
              "response_type": "text",
              "values": [
                {
                  "text": "Great! I'll order a [size] pizza with [topping] for you."
                }
              ]
            }
          ]
        }
    }
]
```

3. Context Handling:

- ➢ Context variables should be used to store and retrieve information across the conversation.

```
"context": {
    "size": null,
    "topping": null
}
```

4. Response Variations:

- ➢ Multiple responses should be created for each intent to make the conversation more engaging.

```
"output": {
    "generic": [
        {
            "response_type": "text",
            "values": [
                {
                    "text": "Great choice!"
                }
            ]
        },
        {
            "response_type": "text",
            "values": [
                {
                    "text": "Excellent!"
            }
          ]
        }
      ]
    }
```

5. Fallback Mechanisms:

- ➢ A fallback intent and response should be defined to handle user queries that the chatbot doesn't understand.

```
"dialog_nodes": [
  {
    "type": "standard",
    "title": "Fallback",
    "output": {
      "generic": [
        {
          "response_type": "text",
          "values": [
            {
              "text": "I'm sorry, I didn't understand. Can you please rephrase your
request?"
            }
          ]
        }
      ]
    }
  }
]
```

6. Integration with External Services:

- ➢ Integrate with external APIs to fetch real-time information or perform actions. For instance, if the chatbot can order food, integrate it with a restaurant's ordering system.

```
def place_order(order_details):

    # Make API request to the restaurant's ordering system

    response = requests.post('https://restaurant-api.com/place_order', data=order_details)

    return response.json()
```

MODEL TRAINING:

Model training for a chatbot involves teaching the chatbot to understand user inputs, recognize intents, and generate appropriate responses. Here are some techniques commonly used in model training for chatbots in Python, along with sample code to illustrate these techniques:

1. Data Collection and Preprocessing:

- Collect and preprocess training data, which includes user inputs, intents, and entities. Use libraries like spaCy or NLTK for text preprocessing.

```
import spacy


# Load the spaCy NLP model

nlp = spacy.load("en_core_web_sm")


# Tokenization and text processing

text = "Tell me a joke"

tokens = [token.text for token in nlp(text)]
```

2. Intent Recognition:

- Train a machine learning model, such as a support vector machine (SVM) or a neural network, to recognize user intents from their input.

```
from sklearn.svm import SVC

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.pipeline import make_pipeline


# Create a text classification pipeline

classifier = make_pipeline(CountVectorizer(), SVC(kernel='linear'))


# Train the intent recognition model

classifier.fit(train_user_inputs, train_intents)
```

3. Entity Recognition:

- Named Entity Recognition (NER) models should be used to extract entities from user inputs. spaCy provides an NER component for this.

```
import spacy


nlp = spacy.load("en_core_web_sm")


text = "Book a flight to Paris on June 1st."
doc = nlp(text)
entities = [(ent.text, ent.label_) for ent in doc.ents]
```

4. Dialog Flow and Context Handling:

- A dialog manager should be implemented to control the flow of conversation and manage context across multiple turns.

```
context = {}


def handle_user_input(user_input, context):
    intent = classify_intent(user_input)
    if intent == 'book_flight':
        handle_flight_booking(user_input, context)
    elif intent == 'order_food':
        handle_food_order(user_input, context)
    # Update context with relevant information
```

5. Response Generation:

- Responses can be generated using rule-based methods, templates, or natural language generation (NLG) models.

```
def generate_response(intent, entities, context):
    if intent == 'greet':
        return "Hello! How can I assist you today?"

    elif intent == 'book_flight':

        return f"Great! I'll book a flight to {entities['destination']} on {entities['date']}."
```

6. Error Handling:

- Error handling mechanisms can be implemented to address scenarios where the chatbot cannot understand the user input.

```
def handle_errors(user_input):

    return "I'm sorry, I didn't understand. Can you please rephrase your request?"
```

7.Multilingual Support:

- If needed, use language-specific models and data for multilingual support.

```
from transformers import pipeline



translator = pipeline("translation", model="Helsinki-NLP/opus-mt-en-de")

translated_text = translator("Hello, how are you?", src="en", tgt="de")
```

EVALUATION:

Evaluating a chatbot involves assessing its performance, understanding user interactions, and measuring its effectiveness.

1.Intent Recognition Accuracy:

- Assess how accurately the chatbot recognizes user intents. Calculating the intent recognition rate using test data.

```
from sklearn.metrics import accuracy_score

predicted_intents = chatbot.recognize_intents(test_user_inputs)

accuracy = accuracy_score(test_true_intents, predicted_intents)
```

2.Entity Recognition Accuracy:

- Evaluating the chatbot's ability to correctly extract entities from user inputs.

```
from sklearn.metrics import classification_report

entity_results = chatbot.extract_entities(test_user_inputs)
```

```
print(classification_report(test_true_entities, entity_results))
```

3. User Satisfaction Surveys:

- Collecting user feedback through surveys or feedback forms to understand user satisfaction and identify areas for improvement.

```
def collect_user_feedback():
    feedback = input("Please rate your experience from 1 (poor) to 5 (excellent): ")
    # Store feedback in a database or log file
```

4. Response Quality:

- Chatbot should be reviewed responses for accuracy, relevance, and clarity

```
user_input = "Tell me a joke"
response = chatbot.generate_response(user_input)
print("User: ", user_input)
print("Bot: ", response)
```

5. Error Handling Assessment:

- Testing the chatbot's ability to handle unexpected or unclear user inputs.

```
unclear_input = "jshsdadg&3£!?"
response = chatbot.generate_response(unclear_input)
print("User: ", unclear_input)
print("Bot: ", response)
```

6.Task Completion Rate:

- Tracking the chatbot's ability is needed to help users accomplish their tasks.

```
def check_task_completion(user_input, expected_result):
    response = chatbot.generate_response(user_input)
    if response == expected_result:
        print("Task completed successfully.")
```

```
else:
    print("Task not completed.")
```

CONCLUSION:

Thus the examples of feature engineering,model training and evaluation is illustrated in this document for the chatbot deployment.