Creating a chatbot in Python

Overview:

This is the phase 1 document for the project “creating a chatbot in python” from IBM on the Naan Mudhalvan Scheme. This project is useful for web developers and software developers as this a integration project with the software and website. This projects goal is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application.

Project Title: “Create a Chatbot in Python” **Problem**

Definition:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver highquality support to users, ensuring a positive user experience and customer satisfaction.

SOFTWARE COMPONENT:

The software used in our project is Google Colab or Jupiter Notebook. **Dataset Used:**

https://www.kaggle.com/datasets/grafstor/simple-dialogs-forchatbot

What is Chatbot?

Chat bots are computer programs that can simulate conversation with human users. They are often used in customer service applications, where they can answer questions, provide support, and resolve issues. Chat bots can also be used in other applications, such as education, entertainment and gaming.

CHALLENGES:

Chat bots are relatively new technology and they still face a number of challenges. One of the biggest challenges is that chat bots after have difficulty understanding natural language. This is because human language is complex and nuanced, and chat bots are still under development in this area. As a result, chat bots may misunderstand user request, which can lead to frustration and confusion.

Another challenge is that chat bots often lack context awareness. This means that they may not be able to understand the full context of a conversation, which can make it difficult for them to provide relevant and helpful responses. For example, a chat bot may not be able to understand the user's intent if they do not explicitly state it.

Finally, there are security and privacy concerns associated with chat bots. This is because chat bots may collect and store sensitive personal data about users, such as their name, email address and phone number. This data could be compromised if the chat bot is not properly designed and implemented.

PROBLEM STATEMENTS:

- **LIMITED UNDERSTANDING OF NATURAL LANGUAGE**:

Chat bots have difficulty understanding the nuances of human language, which can lead to misunderstandings and frustrations for users.

- **LACK OF CONTEXT AWARENESS:**

Chat bots often lack the ability to understand context of a conversation, which can make it difficult for them to provide relevant and useful responses.

- **INABILITY TO HANDLE COMPLEX TASKS:**

Chat bots are typically limited to performing simple tasks, such as answering FAQs or providing customer support. They may have difficulty handling more complex tasks, such as troubleshooting technical problems or negotiating contracts.

Design of Chatbot:

Functionality: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.

User Interface: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.

Natural Language Processing (NLP): Implement NLP techniques to understand and process user input in a conversational manner.
Responses: Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.
Integration: Decide how the chatbot will be integrated with the website or app.

Testing and Improvement: Continuously test and refine the chatbot's performance based on user interactions.

PROBLEM RESOLVE STATEMENTS:

- Ensure the chat bot’s responses are accurate and relevant by fine-tuning its language model and understanding of user model.
- Handle ambiguous user queries effectively by using context or asking clarifying questions.
- Troubleshoot and resolve integration issues with external systems or APIs.
- Optimize code and infrastructure for performance to prevent slow or unresponsive behaviour.
- Implement security measures to protect against potential attacks and vulnerabilities.
- Scale infrastructure to handle a high

Program:

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import string
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
pad_sequences


# Download NLTK data
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

# Load data
with open('dialogs.txt', 'r', encoding='utf-8') as f:
    raw_data = f.read()

# Preprocess data
def preprocess(data):
    # Tokenize data
    tokens = nltk.word_tokenize(data)

    # Lowercase all words
    tokens = [word.lower() for word in tokens]

    # Remove stopwords and punctuation
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in
stop_words and word not in string.punctuation]
```

```
    # Lemmatize words
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in
tokens]

    return tokens

# Preprocess data
processed_data = [preprocess(qa) for qa in
raw_data.split('\n')]

# Set parameters
vocab_size = 5000
embedding_dim = 64
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size = len(processed_data)

# Create tokenizer
tokenizer = Tokenizer(num_words=vocab_size,
oov_token=oov_tok)
tokenizer.fit_on_texts(processed_data)
word_index = tokenizer.word_index

# Create sequences
sequences = tokenizer.texts_to_sequences(processed_data)
padded_sequences = pad_sequences(sequences,
maxlen=max_length, padding=padding_type,
truncating=trunc_type)
```

```
# Create training data
training_data = padded_sequences[:training_size]
training_labels = padded_sequences[:training_size]

# Build model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
input_length=max_length),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(vocab_size,
activation='softmax')
])


# Compile model
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])



# Train model
num_epochs = 50
history = model.fit(training_data, training_labels,
epochs=num_epochs, verbose=2)
```

```
# Define function to predict answer
def predict_answer(model, tokenizer, question):
    # Preprocess question
    question = preprocess(question)
    # Convert question to sequence
    sequence = tokenizer.texts_to_sequences([question])
    # Pad sequence
    padded_sequence = pad_sequences(sequence,
maxlen=max_length, padding=padding_type,
truncating=trunc_type)
    # Predict answer
    pred = model.predict(padded_sequence)[0]
    # Get index of highest probability
    idx = np.argmax(pred)
    # Get answer
    answer = tokenizer.index_word[idx]
    return answer

# Start chatbot
while True:
    question = input('You: ')
    answer = predict_answer(model, tokenizer, question)
    print('Chatbot:', answer)
```

Conclusion:

Hence the project of creating the chatbot using python is done successfully using python library like nltk and tensorflow.