**AI Chatbot Using Python:**

**Problem Definition:**

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

**Design Thinking:**

1. Functionality: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.
2. User Interface: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.
3. Natural Language Processing (NLP): Implement NLP techniques to understand and process user input in a conversational manner.
4. Responses: Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.
5. Integration: Decide how the chatbot will be integrated with the website or app.
6. Testing and Improvement: Continuously test and refine the chatbot's performance based on user interactions

**Preprocessing The Data Set:**

Data preprocessing is a crucial step in building a chatbot, as it ensures that the dataset used for training and natural language understanding (NLU) is clean and properly formatted. Here are common data preprocessing steps in chatbot development:

1. **Data Cleaning:**
   * **Remove Duplicates:** Check for and remove duplicate entries in your dataset. Duplicates can lead to biased training and affect the chatbot's performance.
   * **Handle Missing Values:** Address any missing or null values in your dataset. Missing data can lead to issues during training.
2. **Text Cleaning and Normalization:**
   * **Lowercasing:** Convert all text to lowercase to ensure consistency.
   * **Remove Special Characters:** Remove unnecessary special characters, punctuation, and symbols that don't carry significant meaning for the chatbot.
   * **Tokenization:** Split sentences into individual words or tokens. Tokenization is essential for text analysis.
   * **Stemming and Lemmatization:** Reduce words to their root form to improve the chatbot's ability to understand variations of a word.
3. **Handling Imbalanced Data:**
   * If your dataset has imbalanced classes, you may need to oversample the minority class or under sample the majority class to ensure balanced training.
4. **Remove Stop Words:**
   * Stop words like "a," "an," "the," and other common words don't typically carry much meaning and can be removed from the text.
5. **Entity Recognition:**
   * Identify and mark entities (e.g., names, dates, locations) in your dataset. Many chatbot frameworks allow you to specify and label entities in your training data.
6. **Feature Engineering:**
   * Create additional features if needed. For instance, you might want to extract features like sentiment scores or named entity types to improve the chatbot's understanding of user input.
7. **Intent and Response Mapping:**
   * Ensure that each user input is correctly mapped to the intended intent and associated response.
8. **Dataset Splitting:**
   * Split your dataset into training, validation, and test sets. The training set is used to train the chatbot, the validation set helps tune hyperparameters, and the test set is used to evaluate the chatbot's performance.
9. **Data Format Conversion:**
   * Convert your dataset into a format suitable for your chatbot framework. For example, Rasa uses Markdown-style training data, while other frameworks may have different data format requirements.
10. **Remove Noise:**
    * Remove any irrelevant or noisy data that doesn't contribute to the chatbot's training.
11. **Data Augmentation:**
    * In some cases, you might consider data augmentation techniques to generate additional training examples, especially if your dataset is small.

**Dataset Link:**

<https://drive.google.com/file/d/1lFech-MF0AVgyUhMf3Brb4x-vSSMoTDl/view?usp=drive_link>

**Model Training:**

Training a chatbot involves several key steps. Below are the general steps involved in training a chatbot:

1. **Define Objectives and Use Cases:**
   * Clearly define the objectives and use cases for your chatbot. Understand what tasks or functions the chatbot should perform and who the target users are.
2. **Data Collection:**
   * Gather or generate training data, including conversations, user messages, and corresponding responses. High-quality and relevant training data is crucial for chatbot success.
3. **Data Preprocessing:**
   * Clean and preprocess the training data. This may involve removing duplicates, irrelevant information, and handling special characters or formatting issues.
4. **Select a Chatbot Framework or Platform:**
   * Choose a chatbot development framework or platform. Popular options include Dialogflow, Microsoft Bot Framework, Rasa, and custom solutions.
5. **Natural Language Processing (NLP) Model Selection:**
   * Choose a suitable NLP model for your chatbot. Pre-trained models like GPT-3, BERT, or others can be fine-tuned for your specific chatbot task.
6. **Model Training:**
   * Train the NLP model using the preprocessed training data. Fine-tuning a pre-trained model can help it understand and generate human-like responses.
7. **Conversation Flow Design:**
   * Design the conversation flow by defining user intents, entities, and expected responses. Create a dialogue tree or flowchart to map out the conversation structure.
8. **Implement Logic and Actions:**
   * Develop the chatbot's logic and actions. Write code to handle user input, trigger appropriate responses, and potentially integrate with external APIs or databases.
9. **User Interface Integration:**
   * Implement the user interface for users to interact with the chatbot. This could be on a website, messaging app, voice interface, or any other platform.
10. **Testing and Evaluation:**
    * Thoroughly test the chatbot to ensure it understands user inputs, provides relevant responses, and handles various scenarios. Gather user feedback and make improvements as necessary.
11. **Deployment:**
    * Deploy the chatbot to the chosen platform or application so that users can start using it.
12. **Monitoring and Maintenance:**
    * Continuously monitor the chatbot's performance, gather user feedback, and make updates to improve its capabilities and accuracy. Regularly update the model to adapt to changing language trends.
13. **Data Privacy and Security:**
    * If your chatbot handles sensitive information, ensure that you have robust security measures in place to protect user data and that you comply with data privacy regulations.
14. **Scaling:**
    * As the chatbot's user base grows, be prepared to scale your infrastructure to handle increased user load.

**Code for Training:**

import random

import json

import pickle

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout

from tensorflow.keras.optimizers import SGD

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = []

classes = []

documents = []

ignore\_letters = ['?', '!',',','.']

for intent in intents['intents']:

    for pattern in intent['patterns']:

        word\_list = nltk.word\_tokenize(pattern)

        words.extend(word\_list)

        documents.append((word\_list,intent['tag']))

        if intent['tag'] not in classes:

            classes.append(intent['tag'])

words = [lemmatizer.lemmatize(word) for word in words if word not in ignore\_letters]

words = sorted(set(words))

classes = sorted(set(classes))

pickle.dump(words, open('words.pkl', 'wb'))

pickle.dump(classes, open('classes.pkl', 'wb'))

training = []

output\_empty = [0] \* len(classes)

for document in documents:

    bag =[]

    word\_patterns = document[0]

    word\_patterns = [lemmatizer.lemmatize(word.lower()) for word in word\_patterns]

    for word in words:

        bag.append(1) if word in word\_patterns else bag.append(0)

    output\_row = list(output\_empty)

    output\_row[classes.index(document[1])] = 1

    training.append([bag, output\_row])

random.shuffle(training)

training = np.array(training)

train\_x = list(training[:, 0])

train\_y = list(training[:, 1])

model = Sequential()

model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train\_y[0]), activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)

model.save('chatbotmodel.h5', hist)

print('Done')

**Main Program :**

import random

import json

import pickle

import numpy as np

import nltk

from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import load\_model

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl', 'rb'))

classes = pickle.load(open('classes.pkl', 'rb'))

model = load\_model('chatbotmodel.h5')

def clean\_up\_sentence(sentence):

    sentence\_words = nltk.word\_tokenize(sentence)

    sentence\_words = [lemmatizer.lemmatize(word)  for word in sentence\_words]

    return sentence\_words

def bag\_of\_words(sentence):

    sentence\_words= clean\_up\_sentence(sentence)

    bag = [0] \* len(words)

    for w in sentence\_words:

        for i, word in enumerate(words):

            if word == w:

                bag[i] = 1

    return np.array(bag)

def predict\_class(sentence):

    bow = bag\_of\_words(sentence)

    res = model.predict(np.array([bow]))[0]

    ERROR\_THRESHOLD = 0.25

    results = [[i,r] for i, r in enumerate(res) if r > ERROR\_THRESHOLD]

    results.sort(key=lambda  x:x[1], reverse=True)

    return\_list = []

    for r in results:

        return\_list.append({'intent': classes[r[0]], 'probability': str(r[1])})

    return return\_list

def get\_response(intents\_list,intents\_json):

    tag= intents\_list[0]['intent']

    list\_of\_intents =intents\_json['intents']

    for i in list\_of\_intents:

        if i['tag'] == tag:

            result = random.choice(i['responses'])

            break

    return result

print("|============= Welcome to College Equiry Chatbot System! =============|")

print("|============================== Feel Free ============================|")

print("|================================== To ===============================|")

print("|=============== Ask your any query about our college ================|")

while True:

    message = input("| You: ")

    if message == "bye" or message == "Goodbye":

        ints = predict\_class(message)

        res = get\_response(ints, intents)

        print("| Bot:", res)

        print("|===================== The Program End here! =====================|")

        exit()

    else:

        ints = predict\_class(message)

        res = get\_response(ints, intents)

        print("| Bot:", res)