PROBLEM DEFINITION

What problems do chatbots solve and what are they used for?

We’ve highlighted some of the core fundamental benefits of AI chatbots. Now, what exactly are the types of problems that this technology can help your business solve?

1. Eliminates the added costs to meet global customer demands

Think of this scenario. You’re an e-commerce business with customers visiting from all corners of the world. Many people don’t even speak your native language and their time zones are approximately half a day ahead or behind from where your headquarters is based. Do you have the human support staff to manage all of those inbound enquiries?

By using a chatbot, you eliminate the need to hire additional staff to engage with these shoppers. Chatbots never sleep and they can be programmed to interact with customers in as many languages as you program them to do. They can provide that front-line support without forcing you to overextend your budget with new or temporary hires.

2. Automates repeat customer support enquiries

Many customer support teams are bogged down with repetitive tasks to answer the same versions of FAQs over and over again. It can lead to burnout and job dissatisfaction that ultimately brings down the level of quality care your brand stands behind.

Since chatbots don’t have human feelings, they don’t respond to repetitive menial tasks with any degree of dissatisfaction. They’re simply programmed to do the job and provide the level of support that buyers need. Most importantly, they can automate many of these tasks, thereby speeding up response times and reducing employee dissatisfaction and/or turnover.

3. Ends sales activity only taking place during working hours

Again, let’s go back to the international example. If you have shoppers visiting your website from another part of the world, they need the same level of service that local shoppers demand. But if your sales team works standard 9-5 hour days, they might not be available for those shoppers visiting from another time zone.

Chatbots provide 24/7 real-time support for all buyers, regardless of language barrier and geographic location. If sales teams are not available to those international visitors, a chatbot can provide that exceptional level of service without question. This way, you avoid missing out on potentially lucrative visits from international buyers.

4. Reduces abandoned carts

In March 2021, nearly 80% of all online orders were abandoned. There are many reasons that people may abandon carts without completing a purchase, but the common denominator is that you lose out on revenue for every abandoned shopping cart.

With AI chatbots, like those powered by Kindly, you have the means to reduce abandoned cart rates and win back some of those shoppers to your website. To take things a step further, by integrating with Kindly’s Nudge conversion rate optimization software, your chatbot carefully recommends the products that are most likely to excite the buyer and motivate them to complete a purchase.

5. Gives customers an accessible to their questions channel to find answers

If people feel their needs are not being met by your brand, they’ll abandon your website and give their business to someone else.

Chatbots are always available to customers, and they can have a coloured design to stand out on the page when not in use. Buyers can easily click on the chatbot pop-up to bring up the conversation and interact with the bot as their needs require. They can do this while shopping through a number of different pages and receive answers to pressing questions without having to go digging for those responses on their own.

DESIGN THINKING

Chatbot in Python

In the past few years, chatbots in the Python programming language have become enthusiastically admired in the sectors of technology and business. These intelligent bots are so adept at imitating natural human languages and chatting with humans that companies across different industrial sectors are accepting them. From e-commerce industries to healthcare institutions, everyone appears to be leveraging this nifty utility to drive business advantages. In the following tutorial, we will understand the chatbot with the help of the Python programming language and discuss the steps to create a chatbot in Python.

Understanding the Chatbot

A Chatbot is an Artificial Intelligence-based software developed to interact with humans in their natural languages. These chatbots are generally converse through auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like way. A chatbot is considered one of the best applications of natural languages processing.

We can categorize the Chatbots into two primary variants: Rule-Based Chatbots and Self-Learning Chatbots.

1.Rule-based Chatbots:The Rule-based approach trains a chatbot to answer questions based on a list of pre-determined rules on which it was primarily trained. These set rules can either be pretty simple or quite complex, and we can use these rule-based chatbots to handle simple queries but not process more complicated requests or queries.

2.Self-learning Chatbots:Self-learning chatbots are chatbots that can learn on their own. These leverage advanced technologies such as Artificial Intelligence (AI) and Machine Learning (ML) to train themselves from behaviours and instances. Generally, these chatbots are quite smarter than rule-based bots. We can classify the Self-learning chatbots furtherly into two categories - Retrieval-based Chatbots and Generative Chatbots.

Retrieval-based Chatbots:A retrieval-based chatbot works on pre-defined input patterns and sets responses. Once the question or pattern is inserted, the chatbot utilizes a heuristic approach to deliver the relevant response. The model based on retrieval is extensively utilized to design and develop goal-oriented chatbots using customized features such as the flow and tone of the bot in order to enhance the experience of the customer.

Generative Chatbots:Unlike retrieval-based chatbots, generative chatbots are not based on pre-defined responses - they leverage seq2seq neural networks. This is constructed on the concept of machine translation, where the source code is converted from one language to another language. In the seq2seq approach, the input is changed into an output.

The first chatbot named ELIZA was designed and developed by Joseph Weizenbaum in 1966 that could imitate the language of a psychotherapist in only 200 lines of code. But as the technology gets more advance, we have come a long way from scripted chatbots to chatbots in Python today.

Chatbot in present Generation

Today, we have smart Chatbots powered by Artificial Intelligence that utilize natural language processing (NLP) in order to understand the commands from humans (text and voice) and learn from experience. Chatbots have become a staple customer interaction utility for companies and brands that have an active online existence (website and social network platforms).

With the help of Python, Chatbots are considered a nifty utility as they facilitate rapid messaging between the brand and the customer. Let us think about Microsoft's Cortana, Amazon's Alexa, and Apple's Siri. Aren't these chatbots wonderful? It becomes quite interesting to learn how to create a chatbot using the Python programming language.

Fundamentally, the chatbot utilizing Python is designed and programmed to take in the data we provide and then analyze it using the complex algorithms for Artificial Intelligence. It then delivers us either a written response or a verbal one. Since these bots can learn from experiences and behavior, they can respond to a large variety of queries and commands.

Although chatbot in Python has already started to rule the tech scenario at present, chatbots had handled approximately 85% of the customer-brand interactions by 2020 as per the prediction of Gartner.

In light of the increasing popularity and adoption of chatbots in the industry, we can increase the market value by learning how to create a chatbot in Python - among the most extensively utilized programming languages globally.

So, let's get begun!

Understanding the ChatterBot Library

ChatterBot is a Python library that is developed to provide automated responses to user inputs. It makes utilization of a combination of Machine Learning algorithms in order to generate multiple types of responses. This feature enables developers to construct chatbots using Python that can communicate with humans and provide relevant and appropriate responses. Moreover, the ML algorithms support the bot to improve its performance with experience.

Another amazing feature of the ChatterBot library is its language independence. The library is developed in such a manner that makes it possible to train the bot in more than one programming language.

Understanding the working of the ChatterBot library

When a user inserts a particular input in the chatbot (designed on ChatterBot), the bot saves the input and the response for any future usage. This information (of gathered experiences) allows the chatbot to generate automated responses every time a new input is fed into it.

The program picks the most appropriate response from the nearest statement that matches the input and then delivers a response from the already known choice of statements and responses. Over time, as the chatbot indulges in more communications, the precision of reply progresses.

Creating a Chatbot using Python

We will follow a step-by-step approach and break down the procedure of creating a Python chat.

We will begin building a Python chatbot by importing all the required packages and modules necessary for the project. We will also initialize different variables that we want to use in it. Moreover, we will also be dealing with text data, so we have to perform data preprocessing on the dataset before designing an ML model.

This is where tokenizing supports text data - it converts the large text dataset into smaller, readable chunks (such as words). Once this process is complete, we can go for lemmatization to transform a word into its lemma form. Then it generates a pickle file in order to store the objects of Python that are utilized to predict the responses of the bot.

Another major section of the chatbot development procedure is developing the training and testing datasets.

Now that we have understood the fundamental concepts of chatbot development we need in Python, let us start with the actual process!

Preparing the Dependencies

The initial step to create a chatbot in Python using the ChatterBot library is to install the library in the system. We can also use a new Python virtual environment for the library installation as a good practice. We can install the library using the pip installer with the help of the following command in a Command prompt or Python terminal:

Syntax:

$ pip install chatterbot

$ pip install chatterbot\_corpus

We can also install the latest development version of the ChatterBot library directly from GitHub. For this, we will have to use the following command:

Syntax:

$ pip install git+git://github.com/gunthercox/ChatterBot.git@master

If some of us would like to upgrade the library, we can use the following command

Syntax:

$ pip install --upgrade chatterbot\_corpus

$ pip install --upgrade chatterbot

Now that the setup is ready, we can move on to the next step in order to create a chatbot using the Python programming language.

Importing the Classes

The second step in the Python chatbot development procedure is to import the required classes.

Let us consider the following snippet of code to understand the same.

File: my\_chatbot.py

# importing the required modules

from chatterbot import ChatBot

from chatterbot.trainers import ListTrainer

Explanation:

In the above snippet of code, we have imported two classes - ChatBot from chatterbot and ListTrainer from chatterbot.trainers.

Creating and Training the Chatbot

The next step is to create a chatbot using an instance of the class "ChatBot" and train the bot in order to improve its performance. Training the bot ensures that it has enough knowledge, to begin with, particular replies to particular input statements.

Let us consider the following snippet of code for the same.

File: my\_chatbot.py

# creating a chatbot

myBot = ChatBot(

    name = 'Sakura',

    read\_only = True,

    logic\_adapters = [

        'chatterbot.logic.MathematicalEvaluation',

        'chatterbot.logic.BestMatch'

        ]

        )

Explanation:

In the above snippet of code, we have defined a variable that is an instance of the class "ChatBot". We have included various parameters within the class. The first parameter, 'name', represents the name of the Python chatbot. Another parameter called 'read\_only' accepts a Boolean value that disables (TRUE) or enables (FALSE) the ability of the bot to learn after the training. We have also included another parameter named 'logic\_adapters' that specifies the adapters utilized to train the chatbot.

While the 'chatterbot.logic.MathematicalEvaluation' helps the chatbot solve mathematics problems, the ` helps it select the perfect match from the list of responses already provided.

Since we have to provide a list of responses, we can perform it by specifying the lists of strings that we can use to train the Python chatbot and find the perfect match for a certain query. Let us consider the following example of responses we can train the chatbot using Python to learn.

File: my\_chatbot.py

# training the chatbot

small\_convo = [

    'Hi there!',

    'Hi',

    'How do you do?',

    'How are you?',

    'I\'m cool.',

    'Always cool.',

    'I\'m Okay',

    'Glad to hear that.',

    'I\'m fine',

    'I feel awesome',

    'Excellent, glad to hear that.',

    'Not so good',

    'Sorry to hear that.',

    'What\'s your name?',

    ' I\'m Sakura. Ask me a math question, please.'

    ]

math\_convo\_1 = [

    'Pythagorean theorem',

    'a squared plus b squared equals c squared.'

    ]

math\_convo\_2 = [

    'Law of Cosines',

    'c\*\*2 = a\*\*2 + b\*\*2 - 2\*a\*b\*cos(gamma)'

    ]

Explanation:

In the above snippet of code, we have defined some list of responses in order to train the chatbot. We can also create and train the chatbot by simple typing an instance of "ListTrainer" and providing it with a list of strings as shown below:

File: my\_chatbot.py

# using the ListTrainer class

list\_trainee = ListTrainer(myBot)

for i in (small\_convo, math\_convo\_1, math\_convo\_2):

    list\_trainee.train(i)

Explanation:

In the above snippet of code, we have created an instance of the ListTrainer class and used the for-loop to iterate through each item present in the lists of responses.

Now, the Python chatbot is ready to communicate.

Communicating with the Python chatbot

We can use the get\_response() function in order to interact with the Python chatbot. Let us consider the following execution of the program to understand it.

Output:

# starting a conversation

>>> print(myBot.get\_response("Hi, there!"))

Hi

>>> print(myBot.get\_response("What's your name?"))

I'm Sakura. Ask me a math question, please.

>>> print(myBot.get\_response("Do you know Pythagorean theorem"))

a squared plus b squared equals c squared.

>>> print(myBot.get\_response("Tell me the formula of law of cosines"))

c\*\*2 = a\*\*2 + b\*\*2 - 2\*a\*b\*cos(gamma)

Explanation:

The above execution of the program tells us that we have successfully created a chatbot in Python using the chatterbot library. However, it is also necessary to understand that the chatbot using Python might not know how to answer all the queries. Since its knowledge and training are still very limited, we have to provide it time and give more training data to train it further.

Training the Python Chatbot using a Corpus of Data

As we move to the final step of creating a chatbot in Python, we can utilize a present corpus of data to train the Python chatbot even further.

Let us consider the following example of training the Python chatbot with a corpus of data given by the bot itself.

File: my\_chatbot.py

from chatterbot.trainers import ChatterBotCorpusTrainer

corpus\_trainee = ChatterBotCorpusTrainer(myBot)

corpus\_trainee.train('chatterbot.corpus.english')

Explanation:

In the above snippet of code, we have imported the ChatterBotCorpusTrainer class from the chatterbot.trainers module. We created an instance of the class for the chatbot and set the training language to English.

Moreover, from the last statement, we can observe that the ChatterBot library provides this functionality in multiple languages. Thus, we can also specify a subset of a corpus in a language we would prefer. Hence, our chatbot in Python has been created successfully.

A complete code for the Python chatbot project is shown below.

Complete Project Code

File: my\_chatbot.py

# importing the required modules

from chatterbot import ChatBot

from chatterbot.trainers import ListTrainer

from chatterbot.trainers import ChatterBotCorpusTrainer

# creating a chatbot

myBot = ChatBot(

    name = 'Sakura',

    read\_only = True,

    logic\_adapters = [

        'chatterbot.logic.MathematicalEvaluation',

        'chatterbot.logic.BestMatch'

        ]

        )

# training the chatbot

small\_convo = [

    'Hi there!',

    'Hi',

    'How do you do?',

    'How are you?',

    'I\'m cool.',

    'Always cool.',

    'I\'m Okay',

    'Glad to hear that.',

    'I\'m fine',

    'I feel awesome',

    'Excellent, glad to hear that.',

    'Not so good',

    'Sorry to hear that.',

    'What\'s your name?',

    ' I\'m Sakura. Ask me a math question, please.'

    ]

math\_convo\_1 = [

    'Pythagorean theorem',

    'a squared plus b squared equals c squared.'

    ]

math\_convo\_2 = [

    'Law of Cosines',

    'c\*\*2 = a\*\*2 + b\*\*2 - 2\*a\*b\*cos(gamma)'

    ]

# using the ListTrainer class

list\_trainee = ListTrainer(myBot)

for i in (small\_convo, math\_convo\_1, math\_convo\_2):

    list\_trainee.train(i)

# using the ChatterBotCorpusTrainer class

corpus\_trainee = ChatterBotCorpusTrainer(myBot)

corpus\_trainee.train('chatterbot.corpus.english')

ABSTRACT:

Certainly! Creating a simple chatbot in Python can be done using various approaches. One common method is using a library like ChatterBot. Here's a basic example to get you started:

From chatterbot import ChatBot

From chatterbot.trainers import ChatterBotCorpusTrainer

# Create a new chatbot

Chatbot = ChatBot(‘SimpleBot’)

# Create a new trainer for the chatbot

Trainer = ChatterBotCorpusTrainer(chatbot)

# Train the chatbot on the English language

Trainer.train(‘chatterbot.corpus.english’)

# Simple interaction loop

Print(“SimpleBot: Hello! I’m SimpleBot. Type ‘exit’ to end the conversation.”)

While True:

User\_input = input(“You: “)

If user\_input.lower() == ‘exit’:

Print(“SimpleBot: Goodbye!”)

Break

R

esponse = chatbot.get\_response(user\_input)

Print(“SimpleBot:”, response)

This basic example uses ChatterBot to create a chatbot that responds to user input based on pre-existing training data in English. You can customize and extend this example to suit your specific requirements.

DEVELOPMENT PART 1

ABSTRACT :

Creating a chatbot in Python involves several steps. Here's a simplified guide to help you get started with the development process. We'll use Python and the popular Natural Language Processing library, NLTK, to build a basic rule-based chatbot. You can extend this foundation with more advanced techniques and libraries as needed.

Step 1: Set Up Your Environment

Before you start coding, make sure you have Python installed on your computer. You'll also need to install NLTK (Natural Language Toolkit), which is a library for working with human language data. You can install it using pip:

bash :

pip install nltk

Step 2: Import Necessary Libraries

In your Python script, import the required libraries:

Python :

import nltk

from nltk.chat.util import Chat, reflections

Step 3: Define Chatbot Responses

You can define the chatbot's responses in a list of patterns and responses. This is a very basic examples

Python :

pairs = [

["hi|hello|hey", ["Hello!", "Hi there!", "Hey!"]],

["how are you", ["I'm a chatbot, so I don't have feelings, but thanks for asking!"]],

["what is your name", ["I'm a chatbot, and you can call me ChatGPT."]],

["bye|goodbye", ["Goodbye!", "Have a great day!"]]

]

Step 4: Create a Chatbot Instance

Create an instance of the chatbot using the Chat class and the defined patterns and responses:

Python :

chatbot = Chat(pairs, reflections)

The reflections dictionary is used for handling pronouns (e.g., "I am" -> "you are"). It comes with some default reflections, but you can extend it if needed.

Step 5: Start the Conversation

Now, you can start a conversation with your chatbot :

Python :

print("Hello! I'm your chatbot. Type 'bye' to exit.")

while True:

user\_input = input("You: ")

if user\_input.lower() == "bye":

print("Chatbot: Goodbye!")

break

else:

response = chatbot.respond(user\_input)

print("Chatbot:", response)

This basic chatbot will respond to simple greetings and questions. You can expand the pairs list with more patterns and responses to make the chatbot more versatile.

Step 6: Improve and Expand

To create a more advanced chatbot, you can explore:

Using machine learning and natural language processing libraries like spaCy or TensorFlow.

Implementing a knowledge base or integrating external APIs for more informative responses.

Adding user authentication and handling more complex interactions.

Developing a user-friendly interface, like a web application or a messaging platform integration.

This is just a starting point, and chatbot development can become quite complex depending on your requirements. However, this basic example should help you get started with a simple rule-based chatbot in Python.

Top of Form

This basic chatbot will respond to simple greetings and questions. You can expand the pairs list with more patterns and responses to make the chatbot more versatile.

Step 6: Improve and Expand

To create a more advanced chatbot, you can explore:

Using machine learning and natural language processing libraries like spaCy or TensorFlow.

Implementing a knowledge base or integrating external APIs for more informative responses.

Adding user authentication and handling more complex interactions.

Developing a user-friendly interface, like a web application or a messaging platform integration.

This is just a starting point, and chatbot development can become quite complex depending on your requirements. However, this basic example should help you get started with a simple rule-based chatbot in Python.

DEVELOPMENT PART 2

1.Define Your Use Case: Decide what purpose your chatbot will serve. Is it for customer support, information retrieval, entertainment, or something else? Understanding the use case is crucial.

2.Choose a Framework or Library: Python offers several libraries and frameworks for building chatbots. Some popular options include:

NLTK (Natural Language Toolkit): Ideal for NLP and text processing.

spaCy: A more modern NLP library.

ChatterBot: A library specifically designed for creating chatbots.

Rasa: An open-source platform for building conversational AI.

3.Data Collection: You'll need a dataset to train your chatbot. This dataset should include examples of user inputs and the corresponding responses. You can manually curate this data or use existing datasets.

4.reprocessing Data: Clean and preprocess your dataset. Tokenize, remove stop words, and perform other necessary text processing steps

5.Choose a Model: Depending on your chosen framework, select an appropriate model. For example, you might use a rule-based system, a retrieval-based model, or a generative model like GPT-3.

6.Training: If you're using machine learning, train your model on the preprocessed data. This step may take time and resources.

7.Integration: Integrate your chatbot with the desired platform or channels. This could be a website, messaging apps, or other interfaces.

8.User Interaction: Implement a mechanism for your chatbot to interact with users. This involves receiving user input, processing it, and generating a response.

9.Testing and Iteration: Test your chatbot extensively. Identify and resolve any issues or inconsistencies in its responses. Continuously improve the chatbot based on user feedback.

10.Deployment: Once you're satisfied with the performance, deploy your chatbot to a server or cloud platform.

11.Monitoring and Maintenance: Regularly monitor the chatbot's performance and user interactions. Update it as needed to adapt to changing user needs.

12.Security and Privacy: Ensure that your chatbot handles user data securely and complies with privacy regulations

13.Scaling: If your chatbot gains popularity, you may need to scale your infrastructure to handle increased usage.

pip install chatterbot

from chatterbot import ChatBot

from chatterbot.trainers import ChatterBotCorpusTrainer

# Create a new chatbot instance

chatbot = ChatBot('MyBot')

# Create a new trainer for the chatbot

trainer = ChatterBotCorpusTrainer(chatbot)

# Train the chatbot on English language data

trainer.train('chatterbot.corpus.english')

# Chat with the bot

print("Bot: Hello! How can I help you today? (Type 'exit' to end)")

while True:

user\_input = input("You: ")

if user\_input.lower() == 'exit':

print("Bot: Goodbye!")

break

response = chatbot.get\_response(user\_input)

print("Bot:", response)

Run the Program: Save the script to a .py file and run it. You can interact with the chatbot by typing messages. Type 'exit' to end the conversation.

Please note that this is a simple rule-based chatbot. For more advanced chatbots with natural language understanding and more context-aware responses, you might consider using other frameworks like Rasa or integrating machine learning models like GPT-3.