

CREATE A CHATBOT IN PYTHON

TEAMMEMBER

810621104031 : SUMITHRA.A

Phase-4 submission document

Project Tittle : **Chatbot in python**

Phase 4 : Development part 2

Topic : continue building the chatbot python model by feature engineering, model training and evaluation.



Chatbot in python

Introduction:

- ❖ A chatbot (conversational interface, AI agent) is a computer program that can understand human language and converse with a user via a website or a messaging app. Chatbots can handle various tasks online from answering simple questions and scheduling calls to gathering customer feedback. Brands use bots to automate their [business processes, speed up customer service, and lower support costs](#). The best thing about chatbots is that it streamlines the communication process for companies. In this comprehensive guide, we will continue develop into the construction of a fundamental components: feature selection, model training, and evaluation.
- ❖ Feature selection is the process of identifying and selecting the most relevant features form a dataset to improve the performance of a chatbot model. This is an important step in chatbot as it can help to reduce overfitting and improve the generalization ability of the model.
- ❖ Model training is the process of feeding the select features to a chatbot algorithm and allowing it to learn the relationship between the features and target variable ([i.e., chatbot](#)). Once the model is trained, it can be used to predict the chatbot in python, given their features.

Model evaluation is the process of assessing the performance of a trained python model on a held-out test data set. This is important to ensure that the model is generalizing well and then it is overfitting the trained data.

Dataset for chatbot:

When it comes to training a chatbot, you'll need a dataset that consists of conversations between users and the chatbot. Here are some popular datasets that you can use:

1. **Cornell Movie-Dialogs Corpus:** This dataset contains a large collection of movie scripts. It's often used for training chatbots as it provides dialogues between different characters.
2. **Twitter Dialog Corpus:** This dataset contains conversations from Twitter. It's useful if you want to train a chatbot to understand and generate tweets.
3. **Ubuntu Dialogue Corpus:** This dataset contains conversations from the Ubuntu forums where users ask technical questions and receive answers. It's a good choice if you're interested in building a technical support chatbot.
4. **Persona-Chat Dataset:** This dataset provides conversations where each speaker has a predefined persona. It's useful for training chatbots to have consistent personalities.
5. **Daily Dialog Dataset:** It's a dataset designed for training chatbot models to generate human-like responses in conversations on a wide array of topics.
6. **Multi WOZ Dataset:** This dataset focuses on task-oriented dialogue, particularly in the domain of booking hotels and restaurants.
7. **Open Subtitles Dataset:** It's a large collection of subtitles from movies and TV shows. It's commonly used for training chatbots due to its vast and diverse dialogue data.

8. **Conversational Intelligence Challenge (Conv AI):** This dataset is used for the Conversational Intelligence Challenge, and it involves training on a mixture of dialogue from Reddit and the Open Subtitles dataset.
9. **Empathetic Dialogues Dataset:** This dataset is designed to train chatbots to generate empathetic responses. It contains conversations where one person shares a problem and the other responds empathetically.
10. **Schema-Guided Dialogue Dataset:** The dataset contains dialogues that are focused on interactions with a virtual assistant to accomplish tasks like finding a restaurant or booking a ride.
 - Remember, while using these datasets, it's important to consider the licensing and usage terms. Some may have specific requirements or restrictions.
 - Additionally, depending on your specific application, you might want to create a custom dataset that is more tailored to your domain or target audience. This could involve collecting and annotating conversations relevant to your use case.

Overview of the process:

Creating a chatbot in Python typically involves several steps. Here's an overview of the process:

1. Define the Purpose and Scope:

- Determine the purpose of your chatbot (e.g., customer support, information retrieval, entertainment, etc.).
- Define the specific tasks or questions your chatbot will handle.

2. Choose a Framework or Library:

- Decide whether you want to build a rule-based chatbot, a retrieval-based chatbot, or a generative chatbot.

- Popular libraries and frameworks for chatbot development in Python include:
 - ✓ NLTK (Natural Language Toolkit)
 - ✓ spacey
 - ✓ TensorFlow
 - ✓ Py Torch
 - ✓ Rasa

3.Data Collection and Preprocessing:

- Gather a dataset or corpus of text relevant to your chatbot's purpose. Preprocess the data, which may include tasks like tokenization, lemmatization, and removing noise (like punctuation, special characters, etc.).

4.Training Data Preparation:

- For a rule-based or retrieval-based chatbot, create a set of predefined responses or rules for different user inputs.
- For a generative chatbot, you'll need to create a training set consisting of input-output pairs.

5.Model Selection and Training:

- Depending on the type of chatbot, select an appropriate model architecture (e.g., rule-based, retrieval-based, sequence-to-sequence, transformer-based, etc.).
- Train the model using the pre-processed data.

6. Integration with a Messaging Platform:

- Choose a platform where your chatbot will be deployed. This could be a website, a messaging app (e.g., Facebook Messenger, Slack, etc.), or any other platform.
- Integrate the chatbot with the chosen platform using APIs or SDKs.

7.Handle User Input:

- Receive user input from the integrated platform.
- Preprocess the input (tokenization, cleaning, etc.) if necessary.

Process user input:

- Use the trained model or predefined rules to generate a response based on the user input.
- For generative models, you might use techniques like beam search or nucleus sampling to generate responses.

8.Generate and Send Response:

- Format the generated response and send it back to the user via the messaging platform.

9.Feedback and Continuous Learning (Optional):

- Collect feedback from users to improve the chatbot's performance.
- If applicable, implement mechanisms for the chatbot to learn from this feedback and improve over time.

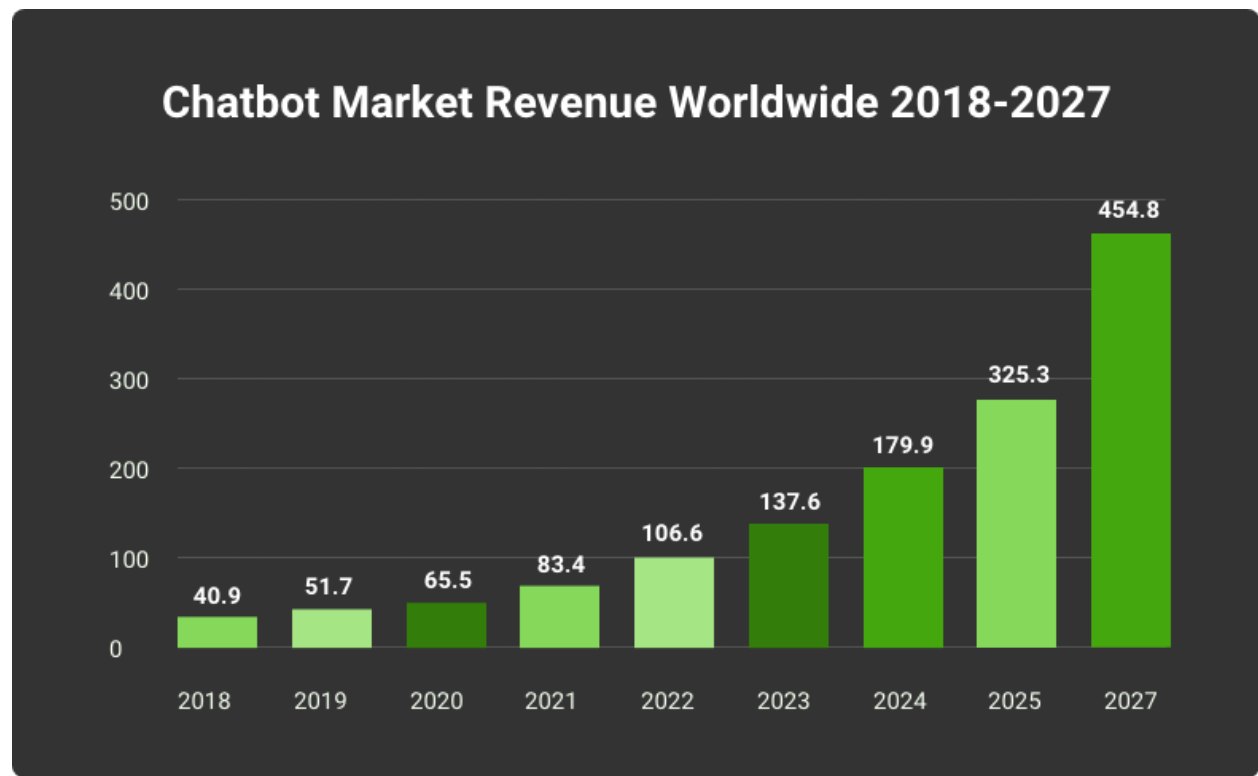
10.Testing and Deployment:

- Thoroughly test the chatbot to ensure it handles a variety of inputs gracefully and provides accurate responses.
- Deploy the chatbot on the desired platform.

11.Monitoring and Maintenance:

- Keep an eye on the chatbot's performance and user feedback.
- Make updates and improvements as needed.

Feature selection:



- ❖ Feature selection for a chatbot can vary widely depending on the specific use case, but here's an example of a basic table that outlines some common features and their descriptions:

| Feature | Description |
|-----------------------------------|---|
| Natural Language Processing (NLP) | Enables the chatbot to understand and generate human-like text. |

| Feature | Description |
|------------------------|---|
| Intent Recognition | Identifies the user's intention or purpose behind a message. |
| Entity Recognition | Extracts specific pieces of information (e.g., dates, names) from user input. |
| Sentiment Analysis | Determines the sentiment (positive, negative, neutral) of user messages. |
| Context Management | Maintains context across multiple interactions for a coherent conversation. |
| Multi-language Support | Ability to understand and respond in multiple languages. |
| Predefined Responses | Allows the bot to provide canned responses for common queries. |

Model training:

Step 1: Data Collection:

- The first step is to gather a large dataset of conversations. These conversations will serve as the training data for your chatbot. You can use existing datasets or create your own.

Python

```
# Example: Creating a simple dataset conversation = [ {
```

```
# Example: Creating a simple dataset
```

```
Conversations = [
```

```
{
```

```
    "input" : "Hi",
```

```
    "output": "Hello",
```

```
},
```

```
{
```

```
    "input" : "What's your name?",
```

```
    "output" : "I am a chatbot.",
```

```
},
```

```
#Add more conversations....
```

```
]
```

Step 2: Model Selection:

- Choose a suitable model architecture for your chatbot. Popular choices include transformer-based models like GPT-3, GPT-2, or their variants. For this example, let's use the OpenAI GPT-3 model.

Step 3: Model Training:

- You don't train models like GPT-3 from scratch. Instead, you fine-tune them on your specific task. You'll need to follow the specific guidelines provided by the model's documentation for fine-tuning.

Step 4: Integration:

- After training, you can integrate the model into your application. You'll need an API key or credentials to access the model.

Python

```
Import openai
```

```
Api-key = 'YOUR_API_KEY'
```

```
Openai.api_key = api_key
```

```
Def get_chatbot_response(input_text):
```

```
    Response = openai.Completion.create
```

```
    (
```

```
        Engine= "Davinci",
```

```
        Prompt= input_text,
```

```
        Max_token=50
```

```
    )
```

```
Return response.choice[0].text.strip()
```

```
#Example usage
```

```
User_input = "Hi"
```

```
Bot_response = get_chatbot_response(user_input=_response)
```

```
Print(bot_response)
```

Step 5: Fine-tuning (Optional):

- Fine-tuning can be done to improve the performance of the model on your specific task. This involves further training on your custom dataset.

Step 6: Testing and Evaluation:

- Test your chatbot with various inputs to ensure its providing accurate and relevant responses. You can use metrics like BLEU score, ROUGE score, or conduct user studies for evaluation.
- Keep in mind that deploying a chatbot in a real-world scenario may involve additional steps such as handling user interactions, error handling, and scalability considerations.
- Remember to respect privacy and ethical guidelines when working with chatbots, especially if they're going to interact with users in a sensitive context.

Dataset in to features and variables:

- ❖ Dividing a dataset into features and target variables is a crucial step in training a machine learning model, including a chatbot. Here's a step-by-step guide on how to do it:

1. Import Libraries:

- First, you'll need to import the necessary libraries. In Python, common libraries for this task are pandas for data manipulation and numpy for numerical operations.

```
 python  
import pandas as pd  
import numpy as np
```

2.Load and Inspect Data:

- Load your dataset using appropriate functions. For example, if your data is in a CSV file, you can use `pd.read_csv()` in pandas.

```
 python  
  
data = pd.read_csv('your_dataset.csv')
```

- Inspect the data using `data.head()` to understand its structure and content.

3.Identify Features and Target:

- Determine which columns represent features (inputs) and which one represents the target variable (output) of your chatbot model. In a chatbot context, features can include things like user messages, context, intent, etc. The target is usually the expected response.

4. Separate Features and Target:

- Once you've identified the columns, you can create separate variables for features and target.

 [python](#)

```
features = data[['feature1', 'feature2', ...]] # List the columns that are features.
```

```
target = data['target_variable'] # Assuming 'target_variable' is the column name for your target
```

5. Convert Data to Numpy Arrays (Optional):

- Depending on the machine learning library you're using, you may need to convert your features and target into numpy arrays. Many libraries (like scikit-learn) work well with numpy arrays.

 [python](#)

```
X = features.values
```

```
y = target.values
```

6. Train-Test Split (Optional):

- If you're planning to evaluate your chatbot model's performance, you should split your data into training and testing sets.

 [Python](#)

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Here, test_size=0.2 means that 20% of the data will be used for testing, and random_state ensures reproducibility.

7.Finalize Data Preparation:

- Depending on your specific application and the libraries you're using, you might need to perform additional preprocessing steps like tokenization, normalization, or encoding categorical variables.

Your Model:

- With your features and target variables prepared, you can now proceed to train your chatbot model using an appropriate algorithm (like a neural network, a transformer-based model, etc.).
- Remember, the specifics of data preparation can vary depending on the exact nature of your chatbot and the machine learning libraries you're using. Always consult the documentation for the libraries you're working with for any additional requirements or best practices.

A Brief Insight to Various Domains of Chatbot

Testing:

❖ When we start working on chatbot testing, it usually involves the below-mentioned types of testing domains:

- ✓ Answering
- ✓ Conversational Flow
- ✓ Error Management
- ✓ Intelligence
- ✓ Intelligence Onboarding
- ✓ NLP Model
- ✓ Navigation
- ✓ Personality
- ✓ Response Time
- ✓ Speed

- ✓ Security
- ✓ Understanding
- ❖ However, achieving the finest results from these testing domains needs the right application of the testing techniques which involves agile and developer testing practices.

Implement of NLP techniques

- ❖ Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on enabling computers to understand and process human language. Implementing NLP in AI applications can unlock a wide range of capabilities, from chatbots and virtual assistants to sentiment analysis and language translation. Here are the key steps to implement NLP in AI applications:

1. Define the Objective:

Begin by clearly defining the objective of your AI application. Determine the specific NLP tasks you want to achieve, such as sentiment analysis, named entity recognition, text classification, or machine translation. This will guide your implementation process.

2. Data Collection and Preparation:

Collect a diverse and representative dataset that aligns with your NLP objective. Preprocess the data by cleaning and standardizing it, removing noise, and addressing any inconsistencies. This step is crucial to ensure accurate and reliable results from your NLP models.

3. Choose an NLP Framework or Library:

Select an NLP framework or library that aligns with your programming language and requirements. Popular choices include Natural Language Toolkit (NLTK) and spaCy in Python, and Stanford NLP in Java. These libraries provide tools and functions to perform various NLP tasks efficiently.

4. Tokenization:

Tokenization is the process of breaking down text into smaller units, such as words or sentences. Apply tokenization to your dataset, segmenting the text into

meaningful units for further analysis and processing. This step serves as a foundation for subsequent NLP tasks.

5. Text Preprocessing:

Perform text preprocessing techniques, such as removing stop words (common words with little significance), stemming (reducing words to their base form), and lemmatization (reducing words to their dictionary form). These techniques help reduce noise and improve the quality of your NLP analysis.

Chatbot testing tools:

- ❖ Since chatbot testing needs to offer a pleasing user experience to anyone visiting the website, working on the various domains and practices needs access to the right tools. Here are a few good tools that you may consider for your chatbot testing project:

- **Bot analytics**

Bot analytics is an AI-enabled tool that works on conversational analytics while capturing engagement. This tool is made to enhance capability for A/B testing, lead interactions through sentiment analysis, and more.

- **Chatbot test**

Chatbot test is a free-to-use tool that comes with 120 questions to assess chatbot experience. This tool works well on all the above-defined domains of chatbot testing.

- **Dimon**

Dimon is a tool that you can use to test the conversational flow of your chatbot along with the user experience. Besides, this tool can be used for integrating chatbots with social media platforms like Facebook, Messenger, etc.

- **Developer Testing**

This is a more direct form of testing, which is meant to validate and verify tests by defining answers to user queries in advance. This type of testing is simple and

works by checking the precision of answers given by the chatbot against any random questions.

➤ **Agile and Regular Testing**

Chatbots are all about agile technology, as it helps to attain the viability required after every loop. Such technology can aid with error-handling functions and prevent bugs with rapid iterations.

Chatbot evaluation:



- Reading and evaluating a sample of conversations manually
- Calculating the automation rate and the savings in time and resources

- Using metrics such as activation rate, session duration, sessions per user, voluntary user engagement, retention rate, goal completion rate, revenue growth, confusion rate, etc.
- Comparing the chatbot's responses with the ground-truth responses using n-grams, cosine-similarity, BLEU score, perplexity, etc.

Conclusion:

- ❖ Chatterbot is a python library that helps create engage in a conversation. To create a chatbot using chatterbot and test if it is working properly, using the using the Chatterbot library! Your chatbot isn't a smarty plant just yet, but everyone has to start somewhere. You already helped it grow by training the chatbot with pre-processed conversation data from a WhatsApp chat export.