chatbot

March 4, 2024

1 Quastion & Answering Chatbot

Python 3.7.13

Source: https://chatterbot.readthedocs.io/en/stable/index.html In this report, we try to explain the process and methodologies used in the development of an intelligent chatbot system leveraging the Python programming language, specifically using the ChatterBot library. The reason for this project lies in the exploration of chatbot technologies as a means to augment real-time customer interaction across a spectrum of industries. The focal point of this investigation is the ChatterBot library's capacity to facilitate the creation of self-learning chatbot systems with a minimalistic approach to coding.

Our objective is to explain the process of building a self-learning chatbot from the ground up, showcasing the simplicity with which Python can facilitate the creation of sophisticated AI-driven applications. We introduce the foundational steps in setting up a basic chatbot framework using the ChatterBot library. The emphasis, however, is placed on the critical phase of training our chatbot, where the quality and specificity of the training data significantly influence the bot's performance and its ability to deliver meaningful interactions. This dataset is designed to enhance a chatbot's ability to address mental health queries, emphasizing the significance of emotional, psychological, and social well-being. It covers the essentials of mental health, its impact on daily life, and the importance of maintaining it for productivity, relationship health, and effective coping mechanisms. Recognizing the prevalence of mental health issues, the dataset also underscores the availability of help and the potential for recovery, aiming to provide comprehensive support through the chatbot interface.

Furthermore, the document details the technical processes involved in chatbot development, including the instantiation of a command-line chatbot, refinement of its response mechanism through training, and the procedural steps for exporting, cleansing, and employing chat history as training material. Insight is also provided into the mechanisms by which the ChatterBot library manages training data, alongside a discourse on potential future enhancements, notably the integration of real-user interactions as a means to continuously evolve the chatbot's conversational capabilities.

In summation, this report archives our expedition through the project and also aims to serve as an instructive compendium for individuals aspiring to navigate the domain of chatbot development using Python's ChatterBot library. The attempt seeks to underscore the practicality and accessibility of deploying Python for the creation of sophisticated, AI-driven technological solutions, thereby contributing to the ongoing discourse on the integration and application of chatbots within digital infrastructures.

[ ]:

[ ]:

[ ]:
```
!python3.7 -m pip install chatterbot==1.0.4 pytz
```

Simple Chatterbot Example We first import modules and libraries we need, such as ChatBot and ListTrainer from chatterbot. Then we create an instance of the ChatBot class and store it into a variable (chatbot here). To train our chatbot to give appropiate response, we start with small amout of data. We simply write a conditional statement usuing while loop that the loop will keep going as long as user have not entered special charachter or string via input, in the loop, user type their message, we store it in a variable named 'query' check if the user did or did not user the special charachter we defined to exit the loop, if they did loop will break and our program ends, if not, we call 'get_response' fucntion from our chatbot instance earlier and pass our 'query' which is the user message to the fucntion and what it will return is the response to the message, the accuracy and quality of the responses depends on the quality and the amount of the data our chatbot got trained with.

By running the next block, ChatterBot might download some data and language models associated with NLTK.

[ ]:
```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

chatbot = ChatBot("Rohi&Bahadir")

trainer = ListTrainer(chatbot)

# trainer.train([
#     "Hi, can I help you?",
#     "Sure, I'd like to book a flight to Iceland.",
#     "Your flight has been booked."
# ])

def the_interface(chatbot):
    exit_conditions = (":q", "quit", "exit")
    while True:
        query = input(">>>  ")
        print(f"You: {query}")
        if query in exit_conditions:
            break
        else:
            print(f"Bot: {chatbot.get_response(query)}")

# the_interface(chatbot)
```

As we said erlier, the chatbot did not trained well yet, has very limited responses, but, by keep running it and have conversation eith it, it do remember your previous conversation.

Chatterbot require database to store all inputs and connect them with possible responses, to do that, by default, it uses SQLite database file.

Chatterbot will create a file named "db.sqlite3" to mainly store all inputs and possible corresponding answers, and two other with same name expect they ending with "wal" and "-shm" which they are temporary support files.

1.1 Train chatterbot

ChatterBot comes with a data utility module that can be used to train chat bots. At the moment there is training data for over a dozen languages in this module. Contributions of additional training data or training data in other languages would be greatly appreciated.

[ ]:
```
from chatterbot.trainers import ChatterBotCorpusTrainer

trainer = ChatterBotCorpusTrainer(chatbot)

# Train based on english greetings corpus
trainer.train("chatterbot.corpus.english.greetings")

# Train based on the english conversations corpus
trainer.train("chatterbot.corpus.english.conversations")
```

1.1.1 Preparing Data for Training

in this part, we want to read out data set file using pandas,to do that we first need to instsall pandas module in our envirement, and that import it.

[ ]:
```
import pandas as pd
```

in this case, we used a Mnetal Health FAQ dataset, containing 98 qaustions and their corresponding answers.

[ ]:
```
df = pd.read_csv('data/Mental_Health_FAQ.csv') #from kaggle
df
```

Text cleaning and Normalization Python is case sensetive programming language, which means the string 'Hello' and 'helLO' are two different words, so to solve this problem we make all the strings to lowers case, and remove all punctuation (using a string substitution function from "re" module, this function searches for all occurrences of a pattern in a string and replaces them with a specified replacement string. If the pattern is not found, the string is returned unchanged) by writing the function 'normalize_text' (the name of the function could be anything) and than we store it in a variable and return it from the function.

In the next step, we use out function to apply those changes in our pandas DataFrame using ".apply()" function from pandas library, it allows you to apply the function along an axis of the DataFrame or to all elements of a Series

[ ]:
```
import re
```

```
def simplified_normalize_text(text):
    if not isinstance(text, str):
        return ''
    text = text.lower()  # Convert text to lowercase
    # text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    return text

# Normalize the 'Questions' and 'Answers' columns
df['Questions'] = df['Questions'].apply(simplified_normalize_text)
df['Answers'] = df['Answers'].apply(simplified_normalize_text)

# Display the first few rows to verify changes
df.head()
```

1.1.2 Removing Duplicates and Missing Values

[ ]:
```
df.drop(columns=df.columns[0], inplace=True)
df
```

To check if our DataFrame has any missing or duplicated value, we can use ".isnull()" function to count any missing value, and ".duplicated()" for any duplicate data in DataFrame

[ ]:
```
missing_values = df.isnull().sum()
duplicate_questions = df['Questions'].duplicated().sum()
```

As we can see, out DataFrame has no missing or duplicated value

[ ]:
```
missing_values, duplicate_questions
```

[ ]:
```
# from chatterbot import ChatBot
# chatbot = ChatBot("rohi&bahadir")
```

[ ]:
```
from chatterbot.trainers import ListTrainer

training_data = df.apply(lambda row: [row['Questions'], row['Answers']],␣
 ↪axis=1).tolist()

flattened_training_data = [item for pair in training_data for item in pair]

trainer = ListTrainer(chatbot)

trainer.train(flattened_training_data)
```

[ ]:
```
# the_interface(chatbot)
```

Train with another dataset

[ ]:
```
file_path = "data/train.csv"

df2 = pd.read_csv(file_path)
df2
```

[ ]:
```
# Apply the simplified normalization to the 'question' and 'answer' columns
df2['Context'] = df2['Context'].apply(simplified_normalize_text)
df2['Response'] = df2['Response'].apply(simplified_normalize_text)

# Display the first few rows to verify the simplified normalization
df2.head()
```

[ ]:
```
training_data = df2.apply(lambda row: [row['Context'], row['Response']],␣
↪axis=1).tolist()

flattened_training_data = [item for pair in training_data for item in pair]

trainer = ListTrainer(chatbot)

trainer.train(flattened_training_data)
```

[ ]:
```
file_path = "data/Conversation.csv"
df3 = pd.read_csv(file_path)
df3
```

[ ]:
```
df3.drop(columns=df3.columns[0], inplace=True)

# Normalize the 'Questions' and 'Answers' columns
df3['question'] = df3['question'].apply(simplified_normalize_text)
df3['answer'] = df3['answer'].apply(simplified_normalize_text)
```

[ ]:
```
df3
```

[ ]:
```
training_data = df3.apply(lambda row: [row['question'], row['answer']], axis=1).
↪tolist()

flattened_training_data = [item for pair in training_data for item in pair]

trainer = ListTrainer(chatbot)

trainer.train(flattened_training_data)
```

[ ]:
```
# the_interface(chatbot)
```

1.1.3 Chatbot Evaluation

[ ]:
```
test_dataset = pd.read_csv('data/train.csv')
test_dataset
```

[ ]:
```

```

[ ]:
```
# test_dataset.drop(columns=test_dataset.columns[0], inplace=True)
# test_dataset['Questions'] = test_dataset['Questions'].
 ↪apply(simplified_normalize_text)
# test_dataset['Answers'] = test_dataset['Answers'].
 ↪apply(simplified_normalize_text)
test_dataset
```

[ ]:
```
test_data_tuples = list(test_dataset.itertuples(index=False, name=None))
```

[ ]:
```
def evaluate_chatbot(chatbot, test_data):
    """
    Evaluates the chatbot's accuracy based on a set of test question-answer␣
 ↪pairs,
    with normalization applied to both questions and expected answers for a more
    flexible comparison.

    Parameters:
    - chatbot: The chatbot instance to be evaluated.
    - test_data: A list of tuples, where each tuple contains a question and its␣
 ↪expected answer.

    Returns:
    - accuracy: The accuracy of the chatbot as a percentage.
    """
    correct_answers = 0

    for i, (question, expected_answer) in enumerate(test_data):
        response = chatbot.get_response(question).text
        normalized_response = response
        normalized_expected = expected_answer

        # Debug print to see the comparison
        print(f"Q{i}: '{question}'")
        print(f"Expected: '{normalized_expected}'")
        print(f"Response: '{normalized_response}'")

        if normalized_response == normalized_expected:
            correct_answers += 1
        else:
            print(f"Mismatch at Q{i}.\n")
```

```
    total_questions = len(test_data)
    accuracy = (correct_answers / total_questions) * 100

    return accuracy

# Assuming your chatbot instance is named `chatbot`
accuracy = evaluate_chatbot(chatbot, test_data_tuples)
print(f"Chatbot accuracy: {accuracy}%")
```

[ ]:
```
# correct_responses = sum(1 for i, response in enumerate(
#     responses) if response == test_dataset['Answers'][i])
# accuracy = correct_responses / len(test_dataset) * 100
# print(f"Accuracy: {accuracy}%")
```

[ ]:
```

```