NLP – Assessment 1 Chatbot

Assistify

04/12/2024

Max Presley

Introduction

This assessment focuses on natural language processing to create a dialog chatbot. To fulfill this goal, I created a functional chatbot named Assistify. This chatbot is a personal assistant replacement. It features a lot of functionality such as Wikipedia integration to learn about a topic. Weather reports so that the weather does not interfere with your busy day. A meeting scheduler to keep track of your meetings. With these features it makes it easy for a busy person to be prepared for the day and its randomness. The chatbot is not designed for casual conversation like most are, this makes the chatbot practical for work and serves a purpose. Assistify uses a range of NLP techniques and other techniques to make the user experience as smooth as possible. This chatbot is free and is accessible anytime and anywhere, unlike a personal assistant.

NLP Techniques

As previously mentioned, the chatbot uses a range of NLP techniques. A hard decision made during development and planning was whether to use deep learning or use strictly rule-based responses. Each has its advantages and disadvantages. Deep learning is a great way to reply to messages that are unexpected or conversational. If trained well deep learning can guide and respond to the user almost like a human being. However, it requires a lot of data and tuning to get to that level of chatbot understanding. This is not a huge problem, however, due to the time given a rule-based system was chosen. Also, a rule-based system is easier to define suited my intended purpose well. Rule-based systems are great for defining a few use cases and getting an expected and exact answer back.

```
elif locations:
    # Choose the first location mentioned for simplicity
    location = locations[0]
    response = get_weather_update(location.capitalize(), api_key)
elif "weather" in CleanMessage:
    response = "Sure, which city do you want the weather for?"
elif CleanMessage in ["hello", "hi", "hey"]:
    response = "Hello, " + RealName + ". How can I help you today?"
elif "search" in words or "summarize" in words:
```

Even before the response is chosen, cleaning up the user's message must be done. Which is why I use String to remove the punction and convert the text into lowercase. Cleaning up of the input message is important for better understanding and consistency of the messages. This just makes it easier to find a good response.

```
Message = message.lower()
CleanMessage = remove_punctuation(Message)
```

```
def remove_punctuation(text):
    # Create a translation table to map punctuation characters to None
    translator = str.maketrans('', '', string.punctuation)

    # Use translate() method to remove punctuation
    cleaned_text = text.translate(translator)

    return cleaned_text
```

When it comes to getting the location for a weather update, this can be tricky. One way that is very inefficient is to list out all the locations like cities, countries, towns etc. Then search for the location name and the word weather in the message. Obviously, it is inefficient to list thousands of place names and even to just have the most populated places is still challenging. Therefore, a smarter way is needed to extract the location name. One way could be to get the word that starts with a capital letter as we know places must have a capital, but this runs into the error of the user not bothering to capitalize the place name. My solution is to use a library that categorizes each word into things like noun, verb, and location. This way I can cycle through each word and find any word that is a location. This library does not have every location such as New Zealand cities, so I created a list the most populous cities of New Zealand to also search through. This is not a 100% success rate process however it keeps a lot of the bases covered this way.

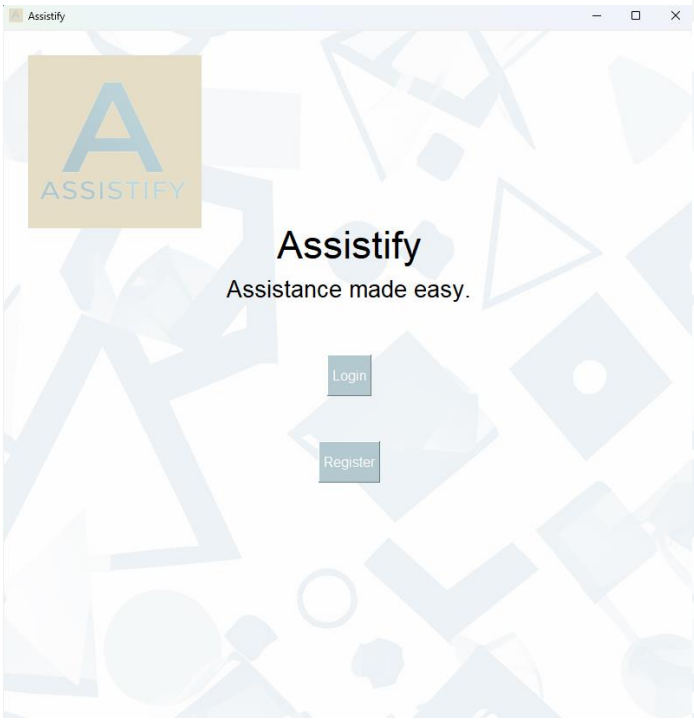
```
locations = []
for ent in doc.ents:
    if ent.label_ == "GPE":
        locations.append(ent.text.lower())
```

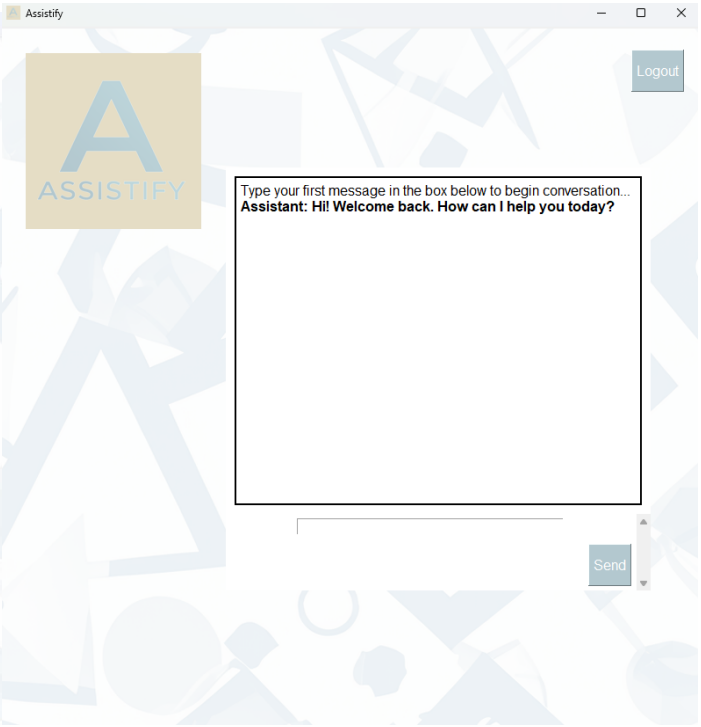
Storage

I key part of the chatbot is storing login information and keeping the meeting information accessible and stored. I introduced a login system to the chatbot so that you can keep your information separate from any other users. The chatbot can access the database and retrieve your name to personalize the experience. A MySQL database was chosen for storage. I chose this because having an online database that is accessible from anywhere is especially useful for syncing data. It could be over-kill and not necessary to have a MySQL database and settle for a local database however local databases are rarely used in projects as you cannot share the same information. The database makes retrieval and writing quite easy and manageable.

Front-End

A front-end GUI was particularly important to me, and I started developing everything with the goal of how it would work with the GUI. I believe the use of a tkinter GUI to be easier to develop than a console based chatbot due to the easy natural of tkinter. I did not worry about loops and recursion. The chatbot made use of the chatbox feature of tkinter. The chatbox made it quite easy to get and send messages. The buttons make directing the user to what they are supposed to do intuitively. All modern applications have a front-end GUI, it has become the norm due to it being easy to use, hence why I use it here.


Assistify
ASSISTIFY
Assistify
Assistance made easy.
Login
Register


Assistify
Logout
ASSISTIFY
Type your first message in the box below to begin conversation...
Assistant: Hi! Welcome back. How can I help you today?
Send