Chapter 6

Building Chatbots I

Introduction to chatbot

- A chatbot is a conversational software application designed to simulate human conversation that run on messaging apps.
- Conversational software is not a new idea, the first command line application was created in the 1960s.
- Conversational bots have been becoming exponentially more popular in marketing
- Chatbot technology uses NLP and AI to understand what a human needs and adapt its response to help end-users reach a desired outcome (like a virtual assistant).

How do chatbots work?

- Chatbots are powered by pre-programmed responses, artificial intelligence, or both. Based on the applied mechanism, they process a user's question to deliver a matching answer.
- Two main types of chatbots:
  - Rule-based chatbots
  - AI Chatbots

Rule-based chatbots

- Provide **answers based on a set of if/then rules** that are defined and implemented by a chatbot designer.
- Provide **matching answers** only when users use a keyword or a command they were programmed to answer.
- Rule-based conversational interfaces **can't learn from past experiences**. They respond based on what they know at that moment.
- Rule-based bots are the cheapest to build.
- Example: “How can I reset my password?”
    - First, **looks for keywords** in the sentence
    - Then, **matches keywords** (‘reset,' and ‘password') with responses available in its database to provide the answer.

AI Chatbots

- Can freely communicate with users
- **Need to be well-trained** and equipped with predefined responses to get started.
- **Learn from past conversations**, don't need to be updated manually later.
- **Better conversationalists than the rule-based** because they take advantage of
    - Machine Learning (ML) allows bots to identify patterns of user input, make decisions, and learn from past conversations.
    - Natural Language Processing (NLP) helps bots understand how humans communicate, understand the context of the conversation even if a person makes a spelling mistake.
    - The sentiment analysis helps a chatbot understand users' emotions.

Content

- Implementing smalltalk
- Learn how to use **regular expressions** and **machine learning** to extract meaning from free-form text.
- Build chatbots that can
  - query a database
  - plan a trip
  - and help you order coffee.

EchoBot I

USER: Hello!

BOT: I can hear you, you said: ‘Hello!’

USER: How are you?

BOT: I can hear you, you said: ‘How are you?’

EchoBot II

USER: Hello!

BOT: I can hear you, you said: ‘Hello!’

```
def respond(message):
    return "I can hear you! you said: {}".format(message)
def send_message(message):
    # calls respond() to get response
    print(respond(message))
send_message("hello!")
```

- A ‘**respond**’ function: use to take a message as an argument and returns an appropriate response.
- A `**send_message**’ function which prints what the user just said, gets the response by calling the respond function, and then prints the bots response.

EchoBot III

```
import time
time.sleep(5)
```

- It just doesn't **feel natural** because a response come back immediately, so we can create a delay by importing the `time` module
- In this example we've created a half-second delay.

Let's practice!

EchoBot I

- Hello, World!
- You'll begin learning how to build chatbots in Python by writing two functions to build the simplest bot possible: EchoBot.
    - EchoBot just responds by replying with the same message it receives.
- In this exercise, you'll define a function that responds to a user's message.
- In the next exercise, you'll complete EchoBot by writing a function to send a message to the bot.

EchoBot I

- Write a function called **respond()** with a single parameter **message** which returns the bot's response. To do this, concatenate the strings **"I can hear you! You said: "** and **message**.
- Store the concatenated strings in **bot_message**, and return this result.

EchoBot I

I can hear you! You said: hello!

```
bot_template = "BOT : {0}"
user_template = "USER : {0}"

# Define a function that responds to a user's message: respond
def ____(____):
    # Concatenate the user's message to the end of a standard bot response
    bot_message = "____" + ____
    # Return the result
    return ____

# Test function
print(respond("hello!"))
```

EchoBot II

- Having written your **respond()** function, you'll now define a function called **send_message()** with a single parameter **message** which logs the **message** and the bot's response.

EchoBot II

- Use the **user_template** string's **.format()** method to include the user's **message** into the user template, and print the result.
- Call the **respond()** function with the message passed in and save the result as **response**.
- Log the bot's **response** using the **bot_template** string's **.format()** method.
- Send the message **"hello"** to the bot.

EchoBot II

```
USER : hello
BOT : I can hear you! You said: hello
```

```python
# Create templates
bot_template = "BOT : {0}"
user_template = "USER : {0}"

# Define a function that sends a message to the bot: send_message
def ____(____):
    # Print user_template including the user_message
    print(____.format(____))
    # Get the bot's response to the message
    response = ____(____)
    # Print the bot template including the bot's response.
    print(____.format(____))

# Send a message to the bot
send_message("____")
```

Creating a personality

Why personality?

- Most chatbots are embedded in a messaging app that people are comfortable using to talk to their friends.
- To make a bit of smalltalk for users, before trying out any 'functionality' that they came for.
- Makes chatbots and voice assistants more accessible and fun to use.
- You users will expect it!.

Smalltalk

```
responses = {
    "what's your name?": "my name is EchoBot",
    "what's the weather today?": "it's sunny!"
}
def respond(message):
    if message in responses:
        return responses[message]
respond("what's your name?")
```

```
my name is EchoBot
```

- Notice that if there isn't a matching message, the `return` keyword will never be reached, so the function will return None.

Including variables

```
responses = {
    "what's today's weather?": "it's {} today"
}
weather_today = "cloudy"
def respond(message):
    if message in responses:
        return responses[message].format(weather_today)
respond("what's today's weather?")
```

```
it's cloudy today
```

Choosing responses

```
responses = {
    "what's your name?": [
        "my name is EchoBot",
        "they call me EchoBot",
        "the name's Bot, EchoBot"
    ]
}
import random
def respond(message):
    if message in responses:
        return random.choice(responses[message])
respond("what's your name?")
```

```
the name's Bot, EchoBot
```

Asking questions

```python
responses = ["tell me more!", "why do you think that?"]
import random
def respond(message):
    return random.choice(responses)
respond("I think you're really great")
```

```
tell me more!
```

- A great way to keep users engaged is to ask them questions, or invite them to go into more detail.
- Instead of using a default message like "**I'm sorry, I didn't understand you**", you can use some phrases that invite further conversation.
- Questions are a great way to achieve this. "Why do you think that?", "How long have you felt this way?", and "Tell me more!" are appropriate responses to many different kinds of message.

Let's practice!

Chitchat

- Uses a dictionary with the questions as keys and the correct responses as values.
- This means the bot only respond correctly if the message matches exactly, which is a big limitation.
- Define a **respond()** function which takes in a **message** argument, checks if the **message** has a pre-defined response, and returns the response in the **responses** dictionary if there is a match, or the **"default"** message otherwise.

Chitchat

```
# Define variables
name = “Bot"
weather = "cloudy"
# Define a dictionary with the predefined responses
responses = {
  "what's your name?": "my name is {0}".format(name),
  "what's today's weather?": "the weather is {0}".format(weather),
  "default": "default message"
}
# Return the matching response if there is one, default otherwise
def ____(____):
    # Check if the message is in the responses
    if ____ in ____:
        # Return the matching message
        bot_message = ____[____]
    else:
        # Return the "default" message
        bot_message = ____["____"]
    return bot_message
```

```
BOT : Hi!
USER : what's your name?
BOT : my name is Bot
```

- Add **send_message()** function
- Add **bot_template** variables
- Test by call **send_message(“…”)**

Adding variety

- It can get a little boring hearing the same old answers over and over.
- In this exercise, you'll add some variation.
- If you ask your bot how it's feeling, the likelihood that it responds with "oh I'm great!" or "I'm very sad today" should be equal.
- You'll use the **random** module - specifically **random.choice(ls)** - which randomly selects an element from a list **ls**.
- A dictionary called **responses**, which maps each message to a list of possible responses, has been defined for you.

Adding variety

- Import the **random** module.

Adding variety

```
# Import the random module
_______???_________
name = "Bot"
weather = "cloudy"
# Define a dictionary containing a list of responses for each message
responses = {
  "what's your name?": [
      "my name is {0}".format(name),
      "they call me {0}".format(name),
      "I am {0}".format(name)
   ],
  "what's today's weather?": [
      "the weather is {0}".format(weather),
      "it's {0} today".format(weather)
    ],
  "default": ["default message"]
}
```

Adding variety

- If the message is in responses, use **random.choice()** in the **respond()** function to choose a random matching response.
- If the **message** is not in **responses**, choose a random default response.

Adding variety

```
# Use random.choice() to choose a matching response
def respond(message):
    # Check if the message is in the responses
    if message in responses:
        # Return a random matching response
        bot_message = ____.____(____[____])
    else:
        # Return a random "default" response
        bot_message = ____.____(____["____"])
    return bot_message
```

Adding variety

- Adding some variety makes your bot much more fun to talk to.
- Now, 'Run Code' and use **send_message()** (which utilizes the **respond()** function) to ask the bot "what's your name?".

```
BOT : Hi!
USER : what's your name?
BOT : my name is Bot
USER : what's your name?
BOT : my name is Bot
USER : what's your name?
BOT : I am Bot
```

- User can keep typing and chatting until they type "bye."

Asking questions

- Asking questions is a great way to create an engaging conversation by responding to statements with a question and responding to questions with answers.
- Create a dictionary of **responses** with **"question"** and **"statement"** as keys and lists of appropriate responses as values.

Asking questions

- Define a **respond()** function which takes in **message** as an argument, and uses the string's **.endswith()** method to check if a **message** ends with a question mark.
- If the **message** does end with a question mark, choose a random **"question"** from the responses dictionary. Else, choose a random **"statement"** from the responses.

```python
# Define a dictionary containing a list of
responses for each message
responses = {
    'statement': [
        'tell me more!',
        'why do you think that?',
        'how long have you felt this way?',
        'I find that extremely interesting',
        'can you back that up?', 'oh wow!',
        ':)'
    ],
    'question': [
        "I don't know :(",
        'you tell me!'
    ]
}
```

Asking questions

```
import random
def respond(message):
    # Check for a question mark
    if ____:
        # Return a random question
        return ____(____["____"])
    # Return a random statement
    return ____(____["____"])
# Send messages ending in a question mark
send_message("what's today's weather?")
send_message("what's today's weather?")
# Send messages which don't end with a
question mark
send_message("I love building chatbots")
send_message("I love building chatbots")
```

- Create templates (**bot_template**, **user_template)**
- Define a dictionary containing a list of **responses** for each message
- Define a function that sends a message to the bot: **send_message**

```
USER : what's today's weather?
BOT : I don't know :(
USER : what's today's weather?
BOT : you tell me!
USER : I love building chatbots
BOT : :)
USER : I love building chatbots
BOT : how long have you felt this way?
```

- User can keep typing and chatting until they type "bye."

Text processing with regular expressions

Regular expressions

- Use for matching messages against known patterns
- Use for extracting key phrases
- Use for transforming sentences grammatically.

Pattern matching

USER: “Do you remember when you ate strawberries in the garden?”

BOT: “How could I forget when I ate strawberries in the garden?”

- The magic of the system relied on giving the “impression” that the bot had understood you, the underlying logic was simple.
- The subject of this example is
  - We are asking about “memories”.
  - The memory itself, of eating strawberries in the garden.
- If we pick apart how the response is generated, we see that it's quite simple.

Pattern matching

```python
import re
pattern = "do you remember .*"
message = "do you remember when you ate strawberries in the garden"
match = re.search(pattern, message)
if match:
    print("string matches!")
```

```
string matches!
```

Extracting key phrases

```python
import re
pattern = "if (.*)"
message = "what would happen if bots took over the world"
match = re.search(pattern, message)
```

```python
match.group(0)
```

```
‘if bots took over the world’
```

```python
match.group(1)
```

```
‘bots took over the world’
```

- A group is just a substring that we can retrieve after matching the string against the pattern.
- We use the match object's `group` method to retrieve the parts of the string that matched.
- Index 0 is the whole string.
- Index 1 is the group we defined by including the parentheses (...) in the pattern.

Grammatical transformation

```
import re
def swap_pronouns(phrase):
    if 'I' in phrase:
        return re.sub('I', 'You', phrase)
    if 'my' in phrase:
        return re.sub('my', 'your', phrase)
    else:
        return phrase

print(swap_pronouns("I walk to school"))
```

```
You walk to school
```

Let's practice!

Extracting key phrases

- The way the program appears to understand what you told it.
- In this exercise, you will match messages against some common patterns and extract phrases using re.search().
- Create a dictionary called rules, which matches the following patterns:
    - "do you think (.*)"
    - "do you remember (.*)"
    - "I want (.*)"
    - "if (.*)“

```
rules = {
    'do you think (.*)': [
        'if {0}? Absolutely.',
        'No chance'],
    'do you remember (.*)': [
        'Did you think I would forget {0}',
        "Why haven't you been able to forget {0}",
        'What about {0}',
        'Yes .. and?'],
    'I want (.*)': [
        'What would it mean if you got {0}',
        'Why do you want {0}',
        "What's stopping you from getting {0}"],
    'if (.*)': [
        "Do you really think it's likely that {0}",
        'Do you wish that {0}',
        'What do you think about {0}',
        'Really--if {0}']
}
```

Extracting key phrases

- Iterate over the **rules** dictionary using its **.items()** method, with **pattern** and **responses** as your iterator variables.
- Use **re.search()** with the **pattern** and **message** to create a **match** object.
- If there is a match, use **random.choice()** to pick a **response**.
- If **'{0}'** is in that **response**, use the **match** object's **.group()** method with index **1** to retrieve a phrase.

```
# Define match_rule()
def match_rule(rules, message):
    response, phrase = "default", None
    # Iterate over the rules dictionary
    for ____, ____ in ____:
        # Create a match object
        match = ____
        if match is not None:
            # Choose a random response
            response = ____
            if '{0}' in response:
                phrase = ____
    # Return the response and phrase
    return response.format(phrase)
# Test match_rule
print(match_rule(rules, "do you remember your last birthday"))
```

- User can keep typing and chatting until they type "bye."

Pronouns

- Transform the extracted phrases from first to second person and vice versa.
- In English, conjugating verbs is simply swapping, for example "me" and 'you', "my" and "your".
- In this exercise, you'll define a function called **replace_pronouns()** which uses **re.sub()** to map "me" and "my" to "you" and "your" (and vice versa) in a string.

Pronouns

- If **'me'** is in **message**, use **re.sub()** to replace it with **'you'**.
- If **'my'** is in **message**, replace it with **'your'**.
- If **'your'** is in **message**, replace it with **'my'**.
- If **'you'** is in **message**, replace it with **'me'**.

```python
# Define replace_pronouns()
def replace_pronouns(message):
    message = message.lower()
    if 'me' in message:
        # Replace 'me' with 'you'
        return ____
    if 'my' in message:
        # Replace 'my' with 'your'
        return ____
    if 'your' in message:
        # Replace 'your' with 'my'
        return ____
    if 'you' in message:
        # Replace 'you' with 'me'
        return ____
    return message

print(replace_pronouns("my last birthday"))
print(replace_pronouns("go with me to Florida"))
print(replace_pronouns("I had my own castle"))
```

```
your last birthday
go with you to florida
i had your own castle
```

Putting it all together

- Put everything from the previous exercises together.
- Create **match_rule()**, **send_message()**, **replace_pronouns()** functions, **templates** and **rules** dictionary

Putting it all together

- Get a **response** and **phrase** by calling **match_rule()** with the **rules** dictionary and **message**.
- Check if the **response** is a template by seeing if it includes the string **'{0}'**. If it does:
  - Use the **replace_pronouns()** function on phrase.
  - Include the **phrase** by using **.format()** on **response** and overriding the value of **response**.

```
USER : do you remember my last birthday
BOT : Did you think I would forget your last birthday
```

```python
# Define respond()
def respond(message):
    # Call match_rule
    response = ____
    phrase = ____

    if '{0}' in response:
        # Replace the pronouns in the phrase
        phrase = ____
        # Include the phrase in the response
        response = ____
    return response

# Send the messages
send_message("do you remember my last birthday")
send_message("do you think humans should be worried about AI")
send_message("I want a robot friend")
send_message("what if you could be anything you wanted")
```

Questions

Reference:
https://campus.datacamp.com/
https://www.chatbot.com/blog/chatbot-guide/