



Chapter 6

Building Chatbots II

Understanding intents and entities

This topic is all about the topic of natural language understanding (NLU).
NLU is a subfield of NLP that is usually concerned with converting freeform text into structured data within a particular domain.

An example

A restaurant booking bot should
be able to **understand a sentence**

I'm looking for
an American fast food restaurant
in the center of town

NLU

Intent:

- restaurant_search

Entities:

- Cuisine: American fast food
- Area: Center

Sure! What about
KFC on Main St.

Database

Then, query a database or an API
to find matching results

To do this, we need to identify the **intent** of the message, and extract a set of relevant **entities**.

Intents

- An **intent** is a broad description of **what a person is trying to say**.
 - For example, "hello", "hi" are all ways that people might `greet` your bot.
- There are many different ways someone might express the intent that described with a `**restaurant_search**`,
 - I'm hungry
 - Show me good pizza spots
 - I want to take my friend out for sushi

Intents

- There is no correct way to assign intents to sentences. The 'correct' answer depends on your application. For example,
 - If you expand your bot's capabilities so that it can actually **book a table for you**, the sentence "I want to take my friend out for sushi" might better be described as a ``request_booking`` intent than as a ``restaurant_search``.

Entities

- The **second part** of NLU is to extract **`entities`** from the text.

“Book a table for **June 10th** at a **sushi** restaurant in **New York City**”

- In the restaurant search example, this means identifying
 - ‘**june 10th**’ as a date,
 - ‘**sushi**’ as a cuisine type, and
 - ‘**new york city**’ as a location.
- A well-studied problem in NLP is “**NER**”. This is almost exactly the same problem, the difference that
 - NER usually aims to find '**universal**' entities like the names of people, organizations, dates, etc.
 - Bots often want a '**narrower**' definition of entities that are specific to their domain.

Using regular expressions

- Use regex to look for keywords in text.
- Build expressions which match any one of a set of keywords by using
 - the pipe '|' operator and
 - add the word **boundary expression** "\b" that there shouldn't be any alphanumeric characters on either side of our keyword.

```
re.search(r"\b(hello|hey|hi)\b", "hey there!") is not None
```

True

```
re.search(r"\b(hello|hey|hi)\b", "which one?") is not None
```

False

Using regex for entity recognition

```
pattern = re.compile('[A-Z]{1}[a-z]*')  
message = """Mary is a friend of mine, she studied at Oxford  
and now works at Google"""  
pattern.findall(message)
```

['Mary', 'Oxford', 'Google']

- Create a pattern object using the `re.compile` method
- This pattern matches exactly **one upper case letter** and 0 or more lower case letters.

Let's practice!

Intent classification with regex

- Begin by implementing a very simple technique to recognize intents - looking for the presence of keywords.
- Create a dictionary **'keywords'**. It has
 - the intents "greet", "goodbye", and "thankyou" as keys, and
 - lists of keywords as the corresponding values.
 - For example, keywords["greet"] is set to ["hello", "hi", "hey"].
- Create a second dictionary, **responses**, indicating how the bot should respond to each of these intents. It also has a default response with the key "default".
- Create function **send_message()**, along with the bot and user templates.

Intent classification with regex

- Create a dictionary `'keywords'`.
- Iterate over the keywords dictionary, using intent and keys as your iterator variables.
- Use `'|'.join(keys)` to create regular expressions to match at least one of the keywords and pass it to `re.compile()` to compile the regular expressions into pattern objects. Store the result as the value of the patterns dictionary.

```
# Define a dictionary 'keywords'.
keywords = {'greet': ['hello', 'hi', 'hey'], 'goodbye': ['bye',
'farewell'], 'thankyou': ['thank', 'thx']}
# Define a dictionary of patterns
patterns = {}
# Iterate over the keywords dictionary
for ____, ____ in ____:
    # Create regular expressions and compile them into pattern objects
    patterns[intent] = ____
# Print the patterns
print(patterns)
```

```
{'greet': re.compile('hello|hi|hey'), 'goodbye': re.compile('bye|farewell'),
'thankyou': re.compile('thank|thx')}
```

Intent classification with regex

- Define a function to find the intent of a message.
- Iterate over the **intents** and **patterns** in the **patterns** dictionary using its **.items()** method.
- Use the **.search()** method of **pattern** to look for keywords in the **message**.
- If there is a match, return the corresponding **intent**.
- Call your **match_intent()** function inside **respond()** with **message** as the argument.

```
# Define a function to find the intent of a message
```

```
def match_intent(message):  
    matched_intent = None  
    for intent, pattern in ____:  
        # Check if the pattern occurs in the message  
        if ____:  
            matched_intent = ____  
    return matched_intent
```

```
# Define a respond function
```

```
def respond(message):  
    # Call the match_intent function  
    intent = ____  
    # Fall back to the default response  
    key = "default"  
    if intent in responses:  
        key = intent  
    return responses[key]
```

```
# Send messages
```

```
send_message("hello!")  
send_message("bye byeee")  
send_message("thanks very much!")
```

```
responses = {'greet': 'Hello you!  
:)', 'goodbye': 'goodbye for now',  
'thankyou': 'you are very  
welcome', 'default': 'default  
message'}
```

```
# Create templates
```

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

```
# Define a function that sends a message to the bot:
```

```
def send_message(message):  
    # Print user_template including the user_message  
    print(user_template.format(message))  
    # Get the bot's response to the message  
    response = respond(message)  
    # Print the bot template including the bot's response.  
    print(bot_template.format(response))
```

```
USER : hello!  
BOT : Hello you! :)  
USER : bye byeee  
BOT : goodbye for now  
USER : thanks very much!  
BOT : you are very welcome
```

Entity extraction with regex

- Use another simple method for finding a person's name in a sentence, such as "hello, my name is David Copperfield".
- Look for the keywords "**name**" or "**call(ed)**", and find capitalized words using regex and assume those are names.
- This exercise is to define a **find_name()** function to do this.

Entity extraction with regex

- Use **re.compile()** to create a pattern for checking if "name" or "call" keywords occur.
- Create a pattern for finding capitalized words.
- Use the **.findall()** method on `name_pattern` to retrieve all matching words in **message**.
- Call your **find_name()** function inside **respond()**.


```
# Define find_name()
def find_name(message):
    name = None
    # Create a pattern for checking if the keywords occur
    name_keyword = _____
    # Create a pattern for finding capitalized words
    name_pattern = _____
    if name_keyword.search(message):
        # Get the matching words in the string
        name_words = _____
        if len(name_words) > 0:
            # Return the name if the keywords are present
            name = ' '.join(name_words)
    return name
```

```

# Define respond()
def respond(message):
    # Find the name
    name = _____
    if name is None:
        return "Hi there!"
    else:
        return "Hello, {0}!".format(name)

# Send messages
send_message("my name is David Copperfield")
send_message("call me Ishmael")
send_message("people call me Cassandra")
send_message("I walk to school")

```

```

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

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

```

```

USER : my name is David Copperfield
BOT : Hello, David Copperfield!
USER : call me Ishmael
BOT : Hello, Ishmael!
USER : people call me Cassandra
BOT : Hello, Cassandra!
USER : I walk to school
BOT : Hi there!

```



Building a Virtual Assistant

Building a virtual assistant can range from relatively easy to incredibly complex, based on how sophisticated you want the functionality to be.

An intelligent virtual assistant (IVA) or intelligent personal assistant (IPA)

- A software agent that can perform tasks or services for an individual based on commands or questions.
- "Chatbot" is sometimes used to refer to **virtual assistants** accessed by online chat that is exclusively for entertainment purposes.
- Some **virtual assistants** are able to interpret human speech and respond via synthesized voices.
- **Users** can
 - ask their assistants questions,
 - control home automation devices and media playback via voice, and
 - manage other basic tasks such as email, to-do lists.

The virtual assistant (VA)

- NLP enables chatbots to understand language as humans speak it
- VA doesn't just read the words, but can
 - understand the intent and
 - understand the context of the question/conversation.

This lets the interaction flow as a conversation instead of as a question-answer session.

- Chatbots that use NLP can converse with users like they would with a human agent, and get answers similarly.

Let's practice!

Building a virtual assistant in Python

- To understand the basic functionality of a relatively simple virtual assistant before treading the deeper.
- First, install the relevant modules and libraries:

```
pip install pyttsx3  
pip install SpeechRecognition  
pip install PyAudio
```

- **SpeechRecognition**: It's one of the Python libraries for recognizing and processing human speech.
- **Pyttsx3**: It's a text-to-speech conversion library in Python.

Building a virtual assistant in Python

- Import the modules and libraries

```
import pyttsx3  
import speech_recognition as sr  
import webbrowser  
import datetime
```


Building a virtual assistant in Python

- The “Assistant” function (pytsx3)
 - To define “who” or “what” your assistant is
 - To determine its voice for this simple virtual assistant.
- Toggle between male and female voices by switching 0 and 1 in the voices[].id.
- “runAndWait” function controls the queue and makes the speech audible in the system.

```
def assistant(audio):  
    engine = pyttsx3.init()  
    # getter: To get the current  
    # engine property value  
    voices = engine.getProperty('voices')  
    # setter method  
    # [0] for male voice  
    # [1] for female voice  
    engine.setProperty('voice', voices[1].id)  
    # Method governing assistant's speech  
    engine.say(audio)  
    # Blocks/processes queued commands  
    engine.runAndWait()
```

Building a virtual assistant in Python

- The Greeting function
 - Write any phrase that you want the virtual assistant to use.

```
def greeting():  
    # This is a simple greeting and  
    # it informs the user that the  
    # assistant has started  
    assistant("Hello, I am your Virtual Assistant.  
How Can I Help You")
```

Building a virtual assistant in Python

- The main body

```
def core_code():  
    # First, we will call greeting  
    # to mark the starting  
    greeting()  
  
core_code()
```

The audioinput function: accepting verbal commands

- Determine how the assistant process the verbal commands.
- Set a microphone as the “Sound Recognition” source

```
def audioinput():
    # this function is all about taking the audio input from the user
    aud = sr.Recognizer()
    with sr.Microphone() as source:
        print('listening and processing')
        # The pause is optional here
        aud.pause_threshold = 0.7
        audio = aud.listen(source)
        # Using try (for valid commands) and exception for when the assistant
        # doesnt "catch" the command
        try:
            print("understanding")
            # en-eu is simply for the accent here english we can use 'en-GB' or 'en-au'
            # for UK and Australian accents
            phrase = aud.recognize_google(audio, language='en-us')
            print("you said: ", phrase)
        except Exception as exp:
            print(exp)
            print("Can you please repeat that")
            return "None"
    return phrase
```

- Add this code to `core_code()` function to test `audioinput()`

```
while (True):  
    # changing the query to lowercase  
    # ensures it works most of the time  
    phrase = audioinput().lower()  
  
    if "what is your name" in phrase:  
        assistant("I am your nameless virtual assistant")  
        continue  
  
    # trigger/condition to exit the program  
    elif "bye" in phrase:  
        assistant("Exiting. Have a Good Day")  
        exit()
```

- The day functions: telling the day

```
def theDay():  
    # This function is for the day  
    day = datetime.datetime.today().weekday() + 1  
    # assigning a number makes it a bit cleaner  
    Day_dict = {  
        1: 'Monday', 2: 'Tuesday',  
        3: 'Wednesday', 4: 'Thursday',  
        5: 'Friday', 6: 'Saturday',  
        7: 'Sunday'  
    }  
    if day in Day_dict.keys():  
        weekday = Day_dict[day]  
        print(weekday)  
    assistant("it's " + weekday)
```

```
while (True):  
    ...  
    elif "what day is it" in phrase:  
        theDay()  
        continue
```


- The time functions: telling the time

```
def theTime():  
    # This function is for time  
    time = str(datetime.datetime.now())  
    # time needs to be sliced for  
    # better audio comprehension  
    print(time)  
    hour = time[11:13]  
    min = time[14:16]  
    assistant("The time right now is" + hour + "Hours and" + min + "Minutes")
```

```
while (True):  
    ...  
    elif "what time is it" in phrase:  
        theTime()  
        continue
```

Building a virtual assistant in Python

- use the Webbrowser module to open any website

```
while (True):  
    ...  
    elif "open google" in phrase:  
        assistant("Opening Google ")  
        webbrowser.open("www.google.com")  
        continue
```

Building a virtual assistant in Python

- use the Wikipedia module to search for a topic within Wikipedia

```
while (True):  
    ...  
    elif "wiki" in phrase:  
        # to pull information from Wiki  
        assistant("Checking the wikipedia ")  
        phrase = phrase.replace("wiki ", "")  
        # it will limit the summary to four lines  
        result = wikipedia.summary(phrase, sentences=4)  
        assistant("As per wikipedia")  
        assistant(result)  
        continue
```



Questions

Reference:

<https://campus.datacamp.com/>

<https://medium.datadriveninvestor.com/>