Rule-based Chatbot

Soumaya SABRY

06 November 2024

The assignment must be submitted as notebook or HTML with outputs of each cell visible.

1 Regex

Regular expressions are a sequence of characters that specifies a search pattern in the text. They are extremely useful in extracting information from text such as code, log files, spreadsheets, or even documents for further processing. Most patterns use normal ASCII, which includes letters, digits, punctuation, and other symbols on your keyboard like "*%#$@!*", but Unicode characters can also be used to match any type of international text.

Here you can find a summary of the regex rules and expressions that may help you with the following practical regex exercises.

Regex	Used for matching	Regex	Used for matching
abc...	*Letters such as abc*	{m}	*m Repetitions*
123...	*Digits such as 123*	{m,n}	*m to n Repetitions*
\d	*Any Digit*	*	*Zero or more repetitions*
\D	*Any Non-digit character*	+	*One or more repetitions*
.	*Any Character*	?	*Optional character*
\\.	*Period*	\s	*Any Whitespace*
[abc]	*Only a, b, or c*	\S	*Any Non-whitespace character*
[^abc]	*Not a, b, nor c*	^...$	*Starts and ends*
[a-z]	*Characters a to z*	(...)	*Capture Group*
[0-9]	*Numbers 0 to 9*	(a(bc))	*Capture Sub-group*
\w	*Any Alphanumeric character*	(.*)	*Capture all*
\W	*Any Non-alphanumeric character*	(abc|def)	*Matches abc or def*

For the following questions, you can first check your regex expression by using this website: `https://regex101.com/`. Then, code it in your a Python notebook, and test it using *re* library. Keep in mind that *the best practice of writing regular expressions is to be as specific as possible to ensure that we don't get false positives when matching against the real-world text.*

Write a regex expression that matches:

1. Any word (means only alphanumeric characters) in the paragraph with a length of 4.

2. A duplication in the sentence that contain two same words separated by space, for example " Paris in **the the** spring.".

3. A string containing the email address, for example, *name@gmail.com*. Make 1 big group and 2 nested ones to capture the name and the domain (e.g. gmail) using the same regex pattern.

4. A string that represents time in HH:mm:ss format. HH is hours between 01 and 12. mm and ss represent minutes and seconds, and each is between 00 and 59. Valid examples include 01:26:18 and 11:00:39. Invalid examples include 13:01:01, 00:00:00, and 01:00:60.

5. Write a regex expression that extracts information from a debug file, where we need to capture the filename, method name, and line number from the lines. Capture information from lines that follow the form "at package.class.methodname(filename:linenumber)". Examples::

Match	E/(1553): at widget.List.makeView(ListView.java:1727)
Match	E/(1553): at widget.List.fillDown(ListView:652)
Match	E/(1553): at widget.List.fillFrom(Hello_world:709)
Don't match	W/dalvikvm(1553): threadid=1: uncaught exception
Don't match	E/(1553): FATAL EXCEPTION: main
Don't match	E/(1553): java.lang.StringIndexOutOfBoundsException

6. You are given a string containing a mix of words, symbols, and whitespace. Your task is to use regular expressions to format the string such that:

   - Each word and symbol sequence is separated by a single space.
   - Allowed characters in each sequence include alphanumeric characters, underscores (_), and the symbols `'`, `/`, `+`, `$`, space, and `-`.
   - All other symbols should be separated by a single space.
   - Any extra whitespace should be removed.

   Test your function with the following strings:

   - `"Hello, World!$500 +some_text here-2023."`
   - `"Total is $100+$20 - discount."`
   - `"Python/regex - quick_example!End"`

2 NLTK

Natural language toolkit, also known as NLTK, is used to handle NLP applications, including chatbots. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

Let's use NLTK to create a simple rule-based chatbot. Import the necessary functions from NLTK using the following command: *from nltk.chat.util import Chat, reflections*
You have imported a **chat** function which is a class that contains the complete logic to process the text data that the chatbot receives and find useful information from it, and **reflections** which is a dictionary that contains fundamental input-output pairs. You can examine its contents by simply using *'print(reflections)'*. You can expand this dictionary or create your own, depending on your preferences.
To run the NLTK chatbot, you need to set up rules. Here are a few examples, the first line is what the user says (in raw string form), and the next line is how the chatbot responds. You

can change these pairs to match the questions and answers as you want.The nltk.**chat** function can handle various regex patterns. You can get creative while making your own pairs. Make sure to add at least four more rules to improve the chatbot.

```
    pairs = [
        [   r"my name is (.*)",
            ["Hello %1, How are you today ?",]
        ],
        [   r"hi|hey|hello",
            ["Hello", "Hey there",]
        ],
        [   r"how are you ?",
            ["I'm doing good , and How about You ?",]
        ],
        [   r"what is your name ?",
            ["I am a bot created test. you can call me Bot!",]
        ],
        [    r"sorry(.*)",
            ["Its alright","Its OK, never mind",]
        ],
        [   r"(.*) age?",
            ["I'm a computer program, seriously you are asking me this?",]
        ]
    ]
```

Once you've set up your pairs of rules, you can start the chat process with the following code. The function is quite simple: it begins by greeting the user and asking if they need any assistance. And the conversation starts from here by calling a *Chat* class and passing pairs and *reflections* to it.

```
    def chat():
        print("Hi! I am Bot")
        chat = Chat(pairs, reflections)
        chat.converse()
        # write quit to stop the conversation
```

Great, you created a rule-based chatbot in seconds, but will it be able to answer the weather question? Maybe 🤷‍♂️. As a *bonus* question to this session, try to dig deeper into the source code for NLTK chat function to understand how to get the name of the city and the time required to process the weather API request and reply with an effective answer.

3 Rule-based Weather Bot

Now, it is time to code your own rule-based chatbot. A simple rule-based chatbot will work by searching for specific keywords in inputs given by a user. The keywords will be used to understand what action the user wants to take (user's intent). Once the intent is identified, the bot will then pick out a response appropriate to the intent. This chatbot is limited by the fact that it is based on specific rules to answer the text given by users. But on the other hand, its responses can be almost correct due to the same rules imposed. As a matter of fact, rule-based chatbots are widely used in customer service especially in the banking and e-commerce sector, as they are a safe tool that can interact with customers according to a planned scenario.

A flow of how the chatbot would process inputs is shown below :

$$\begin{tikzcd}\boxed{\text{User Input}} \arrow[r] & \boxed{\text{Search for keywords}} \arrow[r] & \boxed{\text{Match keywords with Intent}} \arrow[r] & \boxed{\text{Pick a response \& Reply}}\end{tikzcd}$$

3.1 Intents & Responses

The list of **keywords** that the bot will be searching for and the dictionary of **responses** will be built up manually based on the specific use case for the chatbot. Let's start by designing a simple one that can have a small conversation with a user. The bot will be able to respond to greetings (Hi, Hello etc.) and will be able to answer questions about names, gender, time ...etc.

The first thing to do is to build a keywords dictionary where the values are the keywords and the key is the corresponding intent. Here is an example:

```
Intent : greetings
Keywords : 'hello', 'howdy', 'hi', 'hullo', 'how do you do'.
```

Don't hesitate to make as many intents as you have in mind such as greetings, ending, time, unknown ..etc. You are required to find at least six intents.

Second, build up the responses dictionary where the values are the pool of sentences/responses that the bot can pick up from and the key is the corresponding intent. In the values, you can add as many sentences as you want to make sure your chatbot has diverse responses which will make it robust when talking to an actual human, not only repeating himself, and as many intents as set in the first dictionary. Here is an example:

```
Intent: greetings
Keywords: 'Hello! How can I help you today.', 'Hi, you look well today', '
   Welcome here', 'Hiii...I am a Bot, how are you?'.
```

3.2 WordNet

Let's try to remake the keywords dictionary in a more robust way using the **WordNet** synonyms from the NLTK python library. This dictionary can be as large as you want. The more keywords you have, the better your chatbot will perform. **WordNet** is a lexical database that defines the semantic relationships between words, which will help us expand our list of keywords without having to manually enter all the words a user could say.

Write a python function that helps you extend your keywords dictionary using WordNet. Be sure to avoid adding any special characters in this dictionary such as question mark or letter with accent ..etc.

3.3 Matcher function

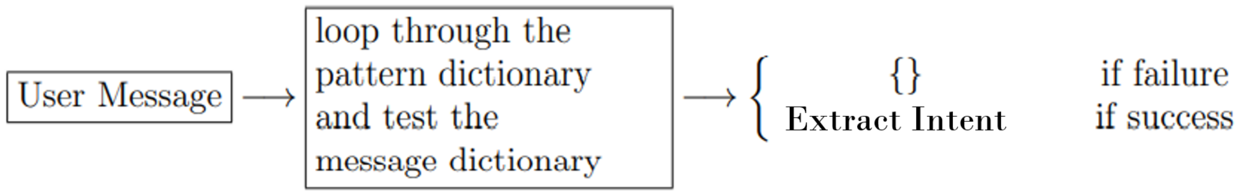
Start by reformatting the keywords in a special syntax that created the regex pattern, you can join them all in a single regex expression like the example below. Improve your regex expression in such a way that it can also understand typos, for example: Hellooo; Helo; Hallo...etc.

```
{'greetings': '.*\\bhullo\\b.*|.*\\bhow-do-you-do\\b.*|.*\\bhowdy\\b.*|.*\\
   bhello\\b.*|.*\\bhi\\b.*'}
```

Question: What does the \b and .* and | in a Regex expression mean and is its use essential here?

Alternative option: You can use methods other than regex for matching. For example, you can use the "pyspellchecker" library to fix typos and then check if the corrected word is in your list of keywords. Any other methods that achieve the same desired outcome are also encouraged.

Now let's match *User Input* and *Bot Responses*. Define a function called *matchPattern* that contains one argument user input (user_msg) where it gets the string written by the user and returns back its intent, if any were found. This function will loop on all Regex patterns to search the user input for keywords stored in the dictionary value field. Then, if a match is found, the current intent is selected, then send to another function that selects an answer randomly from the response dictionary.


User Message
loop through the pattern dictionary and test the message dictionary
{}
Extract Intent
if failure
if success

3.4 Main function

It's time to put it all together and test it, before adding the weather plug-in. Define a switching loop that keeps in place the flow of the conversation. In your main, be sure to check the different options in this list::

1) Match your chatbot response to the user input and ask for its next message.
2) Define a default response to the undefined intention from the user.
3) For the exit case, reply back to the user and exit the console.

3.5 Weather Question

Here you will upgrade your bot into a weather chatbot, which will still be able to respond to greetings (Hi, Hello, etc.) and others, as well as be able to answer questions about the weather on any given day, anywhere in the world.
Start by adding some keywords to your dictionary. Below is an example, don't forget to increase the keyword values with their synonyms using your wordnet function.

```
{
    Intent : " weather question "
    Keywords : 'weather', 'rain', 'temperature', 'sunrise', 'cold'.
}
```

Then, create the various regex pattern that can be asked based on the keywords. Here is an example:

```
{
    Intent: " get weather "
    pattern : "What\\s is\\s the\\s weather\\s like\\s in\\s\\b [a-z]+\\b$"
}
```

3.6 Including entities

Make the regex expression more specific and advance by capturing groups that can identify the main item of the question; such as city and the time.

Be aware that long regular expressions with lots of groups and metacharacters may be hard to read. They can be particularly difficult to manage since adding or deleting a capture group in the middle of the regex will disrupt the groups that follow it. The solution can be to name the group. You can reference the contents of the group with the named backreference *(?P<name>group)*: The question mark, P, and angle brackets are all part of the syntax. This reference captures the match of the group into the backreference "name", which must be an alphanumeric sequence starting with a letter.

Knowing that, let's rewrite the regex pattern of the weather to capture named group, Here is an example:

```
{
    Intent: "Get weather "
    pattern : "What is the weather like in (?P<city>[a-z]+) (?P<time>tomorrow|
   today).$"
}
```

where city and time are the names of the corresponding group in the pattern, which are the entities that your chatbot should use to process this response.

As you might know, the entities (or slots) are the *needed information to be provided by the user in order to help the chatbot to complete the task.* For this, create an empty dictionary of entities as well as a *createEntities* function for extracting the entities from the user message. For instance, this function should extract the information from the following phrase: "what is the weather like in Paris tomorrow?" and if any entities missing, it should alert the chatbot to ask about the missing information.

The output should look as:

```
{
    intent : 'get weather',
    entities :{ city : 'Paris', time : 'tomorrow'}
}
```

You can test your chatbot at this stage before adding the API of the weather.
PS : Modify the Regex pattern to adapt for cities that can include two words as well.

3.7 Weather API

After extracting city names and entities, you need to connect with a weather API to find the weather situation of each city: chatbot ⇔API weather. You can use any API Weather you prefer. We suggest you a list of APIs as below:

- `https://openweathermap.org/`
- `https://developer.accuweather.com/`
- `https://www.climacell.co/weather-api/`

Sign up on their website for getting a free access key to the api. Save your API key in your python code to make the connection. For this, I am proposing the use of the *Requests* python library, you can use other options if you find better 😉.

Next, define a function namely *getWeather*, This function receives at least 'location' & 'date', connects to the api, and gets weather information on the location forecast. The status of the request can be as following ::

- 200: **OK**. Standard response for successful HTTP requests.

- 400: **Bad Request** The server cannot or will not process the request due to an apparent client error. (i.e. invalid API)
- 404: **Not Found** The requested resource could not be found but may be available in the future.
- ... there are many other possible status codes as well to give you specific insights into what happened with your request.

In the end, test your chatbot by starting a conversation and asking it "what's the weather like in Los Angeles" or similar questions.

3.8 Small Project

As a small project, try to finish your weather bot by implementing the following steps or adding your proper ideas to make your chatbot as user-friendly as possible!
After finishing your project upload it to Moodle in the space for Session 2 assignment.

- The API results are detailed information on each city including its exact temperature. Design a method that replies back in a human way. For example, the bot could express temperatures in words such as:
  User:: How is the weather in Paris?
  Bot :: It is very cold in Paris, France, with -2 degrees Celsius.
- You can also add some color which can be only important words in your bot response. Or maybe the color can indicate the temperature as well if cold then Blue if hot then super Red. And why not adding some emoji such as 😶‍🌫️ if it is cloudy, maybe 🥵 or 🥶 if hot or cold.
- You can modify your chatbot in a way that it is capable of answering the following questions:: Is it cold in Berlin today? , Will it be rainy in Paris tomorrow? , Will it be sunny in Paris the day after tomorrow?
- ... be creative 😉

Good Luck ! 💪