CS 4395 Human Language Technologies
Dmitrii Obideiko
Suraj Janakiraman

Weather Forecast Chat Bot - Part 2

General description of the project

We built a weather forecast chatbot that gives the user various weather information.

As of right now, the chatbot has the following capabilities:

1. Tell the minimum, maximum, or general temperature for the next 40 days
2. Tell the weather on a particular day
3. Tell you what the humidity will be on a particular day
4. Tell you what pressure will be on a particular day
5. Remember the city that the user first used to look up any weather information. After a city is mentioned at least once, the user doesn't have to type it ever again unless they decide to look up the weather information of a different city.
6. Understand the common language that people use to look up the weather on a particular day. For example, "What's the weather this Sunday in Plano?" Or what's the temperature next Tuesday in London?

System description

User Input

The system prompts the user to write a message to the bot as well as displays what the bot can do. Until the user enters a city, the chatbot will continuously write to the user to include a city in their sentence. Once the user includes a city in their sentences, the user doesn't have to write the name of the city. The chatbot then returns a response based on the question type.

Processing User Input

In order to break a sentence into multiple words, we tokenized the user input using one of the functions in the nltk library, which is word_tokenize(text). Then we removed all stop words except the word "next" as we didn't need them. The reason why we didn't remove the word "stop", even though it's one of the stop words, is because this word is used quite often when someone looks up the weather. For example, "what is the weather like next Monday?".

After that, we found bigrams by using one of the nltk's functions, called ngrams(). We used the bigrams because for this chatbot, in particular, oftentimes 2 words have more meaning compared to if they were looked at separately. For example, it's useful to know what "next Monday" or "minimum temperature" mean. By going through each bigram, we could find information about what kind of question the user wanted to ask, what weekday the user is

referring to, as well as which day. To find what kind of question the user was asking, we were looking for specific words like “weather” or “humanity”. If the program found one of those words, it assumed that the question was about them. The program also found phrases like “next Tuesday” and “minimum temperature” by looking for something after the word “next” and “minimum”. Using bigrams was efficient as it gave us easy access to neighboring words.

In order to find the location (the city), we tagged all tokens based on the part of speech (for ex. noun, verb). To tag all tokens we used an nltk function called pos_tag(). Then we used another nltk function called ne_chunk, which chunks the input into groups. Then we used the function called hasattr() to check if the chunk has an attribute “label” which indicates that it’s a location.

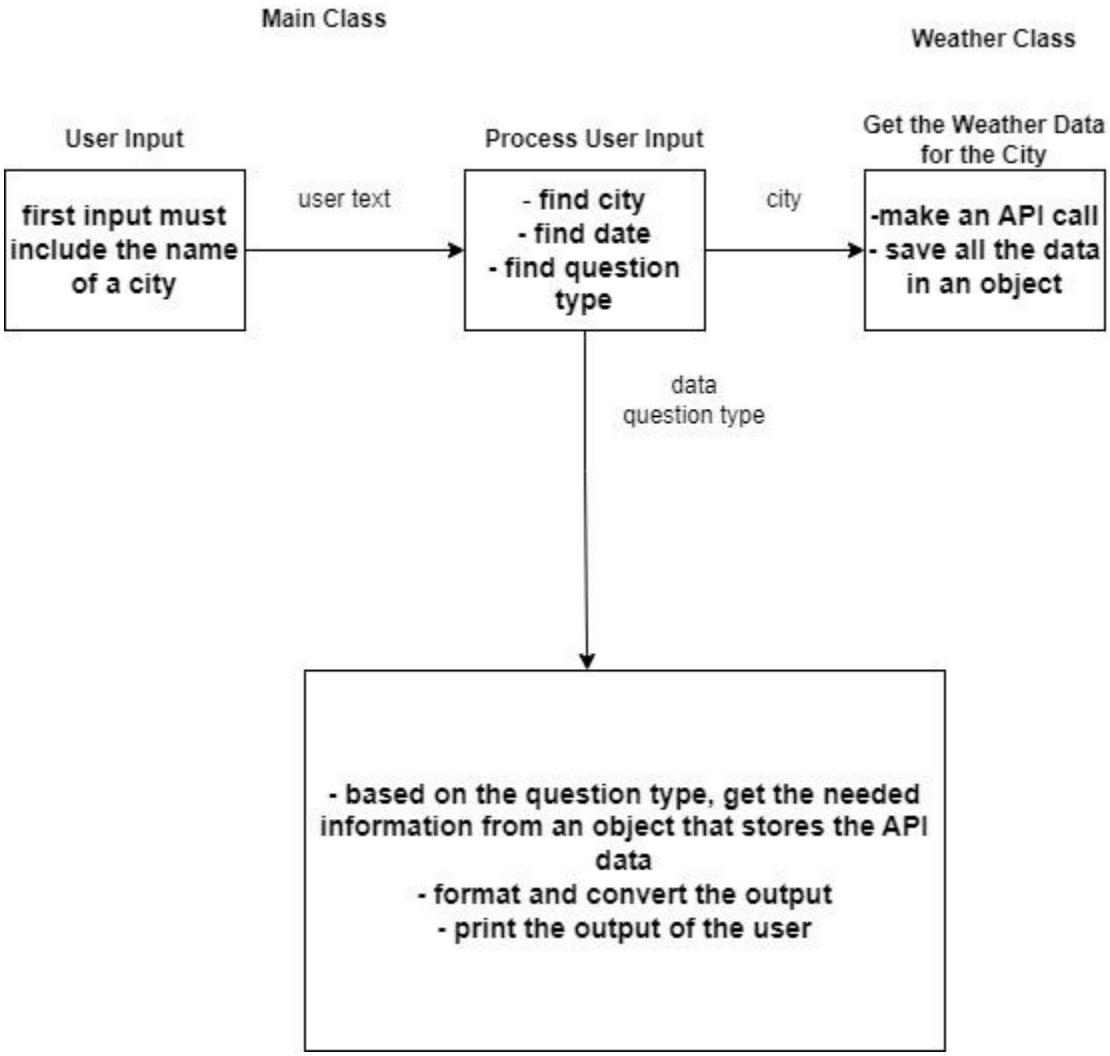
After the day, location, and question type were identified, the output was sent to a function called ChatBotOutput() to let it decide what to write to the user based on the provided output.

Chat Bot Output

After receiving the day, location, and question type

The chatbot output may vary depending on the question type. For example, if the question type is a weather question, the response starts with “It looks like:” followed by a weather description (e.g. scattered clouds, etc.). Depending on the question the chatbot calls the corresponding function of a created object that stores all information received from an API call about the weather. The chatbot also calculates the index of the day, where 0 is today. For example, if today is Tuesday, and the user asked for the weather for next Tuesday, the program will find that it’s 8. This chatbot then looks up the 8th object found in a JSON file that API Server sent to the chatbot when a request was made. The chatbot also formats or converts any data that it receives. For example, when it gets a temperature from an API call in Kelvin, it converts it to Fahrenheit. If the user asks the chatbot a question that it doesn’t understand, it will print “Sorry, I am not sure I understand”. The chatbot stops listening to user input once the user prints “quit”.

Diagram (note: software used to draw this diagram is called diagrams.net (draw.io)).



Sample dialogue interactions

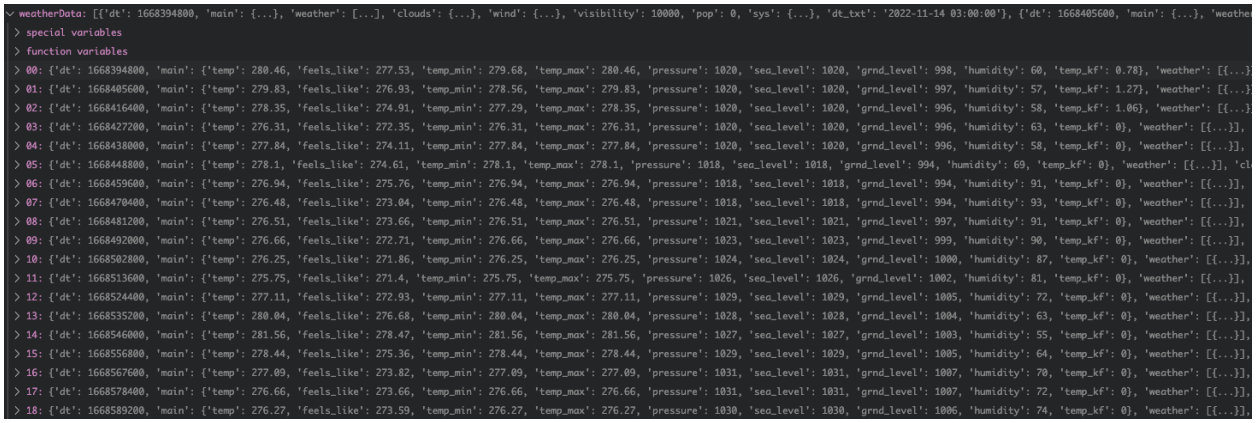
```
Hello! I am a Weather Forcast Chat Bot!
What I can do:
1. Tell minimum, maximum, or general temperature for the next 40 days
2. Tell the weather on a particular day
3. Tell you what what humidity will be on a particular day
4. Tell you what pressure will be on a particular day
~ Note: You do not have to tell me the name of a city every time, I can remember that ~
~ To quit the chat, type "bye" ~

[User]: What's the weather like in Plano right now?
[Chat Bot]: It looks like:  clear sky
[User]: What's the weather next Friday?
[Chat Bot]: It looks like:  light rain
[User]: What's the lowest temperature next Tuesday?
[Chat Bot]: The temperature is 42.0 F
[User]: What humidity is next Monday in London?
[Chat Bot]: The humidity is 92 %
[User]: What pressure is next Tuesday?
[Chat Bot]: The pressure is 1010 Hg
[User]: bye
```

Appendix of knowledge Base

All information received from an API is stored in an object named “weather data”

Example #1



Example #2

```
{
  "cod": "200",
  "message": 0,
  "cnt": 40,
  "list": [
    {
      "dt": 1661871600,
      "main": {
        "temp": 296.76,
        "feels_like": 296.98,
        "temp_min": 296.76,
        "temp_max": 297.87,
        "pressure": 1015,
        "sea_level": 1015,
        "grnd_level": 933,
        "humidity": 69,
        "temp_kf": -1.11
      },
      "weather": [
        {
          "id": 500,
          "main": "Rain",
          "description": "light rain",
          "icon": "10d"
        }
      ],
      "clouds": {
        "all": 100
      },
      "wind": {
        "speed": 0.62,
        "deg": 349,
        "gust": 1.18
      },
      "visibility": 10000,
      "pop": 0.32,
      "rain": {
        "3h": 0.26
      },
      "sys": {
        "pod": "d"
      },
      "dt_txt": "2022-08-30 15:00:00"
    },
```


Appendix of sample user models that were created

After the user mentions the name of the city, the program memorizes the name of the city as well as any information received for that particular city to avoid making unnecessary calls to an API. As you can see in the code below, the program checks if the user already provided the name of a city. In the second screenshot, you can see what it stores after the user provides a city name. It stores an object that stores all information received from an API about that city. The program updates weather information about the city if the user again provides the name of a different city.

```
weatherData = None
userInput = ''
while userInput != 'bye':
    userInput = input('[User]: ')
    if userInput == 'bye':
        break
    city, questionType, partOfDay, weekDay, monthDay = processInput(userInput)
    if not weatherData and city == '':
        print('[Chat Bot]: I can not tell you anything until you tell me the name of a city')
    else:
        # get weather data for the next 40 days
        if len(city) > 0:
            weatherData = WeatherInfo(city)
        # generete chat output
        ChatBotOutput(weatherData, questionType, partOfDay, weekDay, monthDay)
```

```
weatherData: <WeatherInfo.WeatherInfo object at 0x11fb0ec50>
> special variables
> function variables
API_KEY: '0df4f32b4fe92f13f3c242fb49765758'
lat: 33.0136764
lon: -96.6925096
```

Evaluation of the chatbot - strengths & weaknesses

Strengths

The major strength of the chatbot is that it can understand the most common phrases that people use when they want to ask something about the weather. For example, most people prefer to ask questions like “what’s the weather like next Monday” rather than “what’s the weather like on November 14th”. It also checks if the user imputed the name of a city. The chatbot is particularly good at identifying if a word is a city or not. It applied to various names that cities have: whether they consist of just 1 word or 2 words.

Weaknesses

One of the major weaknesses is that grammar really matters to a chatbot. If the user makes a grammatical mistake, the chatbot might misinterpret or not understand what the user is trying to ask. Another weakness is that the chatbot looks for particular words. For example, if the user wants to find out what is the minimum temperature for a particular day, they have to type something that contains “minimum temperature”. If they type “min” temperature, the program

will not understand it. In addition, the user cannot ask 2 questions at the same time. If the user asks 2 questions, the program will ignore the second question.