Abstract

Project AI Use Case
DLMAIPAIUC01
Task 3: Weather Chatbot

Alessandro Casonato (9215675)

The weather chatbot has been developed in Python using an object-oriented programming approach. The following components are involved using the zero-shot classifier "bart-large-mnli" to identify the intent of the user's query (in parenthesis can be found the file name where the component is declared).

- **IntentClassifier**(`intent_classifier.py`): this class is the one using the zero-shot classifier to understand the user's intent among the supported ones, which are "weather", "temperature", "rain", "wind", "humidity" and "help".
- **EntityExtractor**(`entity_extraction.py`): it extracts the city and date from the user's query when needed. The extraction happens using a combination of regular expressions and logic with stopwords.
- **WeatherAPI**(`weather_api.py`): this class contains the methods used to interact with Open Weather map, an external service that exposes endpoints that can be called to retrieve weather data. The calls are made using a free tier API key, which, in comparison to the paid version, is limited only in the amount of requests that can be done per day.
- **ResponseBuilder**(`response_builder.py`): it build the response to provide to the user, using the data gathered before from the other components.
- **WeatherChatbot** (`chatbot.py`): this class is responsible of using the intent classifier, the entity extractor, the weather API component and the response builder. All these components are used to provide the correct response to the user's query.

We can also find the following files:

- `main.py` : the file containing the main logic that uses the other components.
- `tests`: contains all the unit tests, executable using the library "`pytest`" through the command `python -m pytest tests/`
- `readme.md` : a markdown file containing some instructions for setting up the environment to execute the chatbot correctly in the local environment.
- `requirements.txt` : file used to install all the dependencies through the command pip install -r requirements.txt
- `.gitattributes` and `.gitignore`: files used to keep the repository clean and in order on Git.

Here’s the complete flow explained, starting from the main.py file:

1. the **WeatherChatbot** is instantiated, which in turn instantiate its internal instances of the other components
2. the program enters in a loop, in which the user is prompted to provide a query
3. the user inserts a query through the terminal: if it states “exit” or “quit” (case insensitive), the chat is closed. Otherwise, the query is passed to the chatbot’s “`handle_input`” function.
4. The “`handle_input`” function calls the **IntentClassifier**’s “`classify`” method, passing the query as input. This method outputs the user’s intent if found among those supported, otherwise returns null. In case “null” is returned by “`classify`”, the chatbot returns “I'm not sure how to help with that.” and it doesn’t execute the next steps.
5. If the intent found is “help”, then the ResponseBuilder is called passing the intent, so that it can provide immediately the proper response.
6. If the intent found is not help, but not null, then the **EntityExtractor**’s “`extract`” method is called to try to extract a city and a date from the query passed as parameter.
7. The date is extracted using the following method imported from the “`dateparser`” package:
   ```
   parsed = search_dates(user_input,
   settings={'PREFER_DATES_FROM': 'future'})
   ```
8. The city is extracted using the following RegEx:
   ```
   re.search(r"\b(?:in|at)\s+([A-ZÀ-Ùa-zà-ù\s]+)",
   cleaned_input)
   ```
   This means that it tries to get as the city the word(s) found after an “in” or “at” not contained in other words, supporting spaces in between (ex. “in New York”).
9. If the city is not found, then the chatbot returns to the user "Please include a city in your question (e.g., 'What's the weather like in London?').”.
10. If the city is found, then the WeatherAPI component is called, passing the city as input. If the extract method found a date then it’s called “`get_forecast`”, otherwise it’s called the method “`get_current`”,
11. Using all the data gathered from all the previous steps, the ResponseBuilder is then called to provide a properly formatted answer to the user, containing the data requested depending on the user's intent.