User Manual

CBR for a cocktail recipe creator

Practical Work 3 - A CBR prototype for a synthetic task

Xavier Cucurull Salamero
xavier.cucurull@estudiantat.upc.edu
Daniel Hinjos García
daniel.hinjos@estudiantat.upc.edu
Fernando Vázquez Novoa
fernando.vazquez.novoa@estudiantat.upc.edu
Estela Vázquez-Monjardín Lorenzo
estela.vazquez-monjardin@estudiantat.upc.edu

🍹 Cocktails CBR 🍸

Master in Artificial Intelligence
Universitat Politècnica de Catalunya
June 2021

Contents

1 Description of the CBR system goals

Welcome to Cocktails CBR! This system is able to suggest a tasty cocktail recipe that adjusts to the user preferences, which may include some considerations such as the desired ingredients, the beverage categories, the alcohol types, the basic beverage tastes or the glass type, among others, as well as some exclusions based on negative preferences.

The system is implemented following a typical Case-Based Reasoning scheme, with all the corresponding main phases (retrieval, adaptation, evaluation and learning), and it has been adapted following the considerations of the authors for the problem at hand.

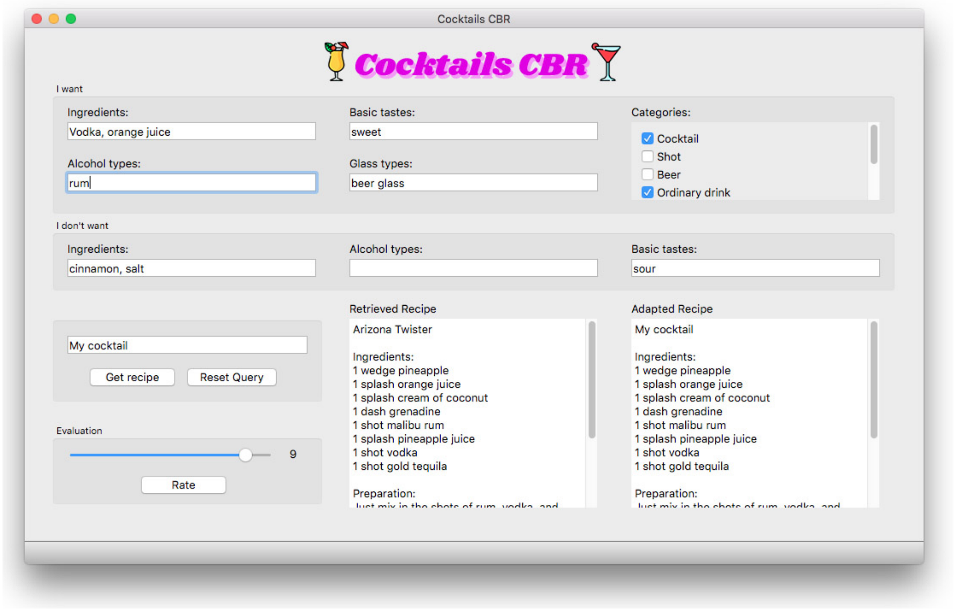
2 Start-Up/Shutdown of the system

This section describes the procedures for starting up and shutting down our system. In order to do so, two different ways of accessing it are available, attending to different levels of intuitiveness and interaction necessities: Graphical User Interface (GUI) and Console Line Interface (CLI). For the last one, the inputs can be specified either via a JSON formatted file or via a personalized console menu. In this section you will find instructions about how to run each one of them.

Note that, for all cases to work properly, all the libraries indicated in `requirements.txt` must be installed in the environment.

2.1 Graphical User Interface (GUI)

In order to open the GUI, simply run "`app/main.py`". Regardless of your OS, you will be greeted with the interface created:



I want

Ingredients: Vodka, orange juice

Basic tastes: sweet

Categories:
- [✓] Cocktail
- [ ] Shot
- [ ] Beer
- [✓] Ordinary drink

Alcohol types: rum

Glass types: beer glass

I don't want

Ingredients: cinnamon, salt

Alcohol types:

Basic tastes: sour

My cocktail

Evaluation 9

Retrieved Recipe

Arizona Twister

Ingredients:
1 wedge pineapple
1 splash orange juice
1 splash cream of coconut
1 dash grenadine
1 shot malibu rum
1 splash pineapple juice
1 shot vodka
1 shot gold tequila

Preparation:
Just mix in the shots of rum, vodka, and

Adapted Recipe

My cocktail

Ingredients:
1 wedge pineapple
1 splash orange juice
1 splash cream of coconut
1 dash grenadine
1 shot malibu rum
1 splash pineapple juice
1 shot vodka
1 shot gold tequila

Preparation:
Just mix in the shots of rum, vodka, and

Figure 1: GUI Interface example

The GUI has several possible configurations worth mentioning. It is implemented so it uses by defect our case library located in `"Data/case_library.xml"`, but another filepath can be specified by going to `"File/Load/Library..."`. It has been implemented this way for scalability issues, in case that future users or developers want to test the system in different datasets. Moreover, a JSON file can be imported in order to directly load existing constraints into the interface by going to `"File/Load/Constraints..."`

Otherwise, the different constraints must be introduced manually. If you want to specify more than one value for each constraint, these values will have to be separated by a comma (ingredients: gin, sugar syrup). After doing so, by pressing "Get recipe" the interface will display both the retrieved and the adapted recipes. If an invalid constraint is selected, an error messages appears so that the user can fix it before pressing "Get recipe" again.

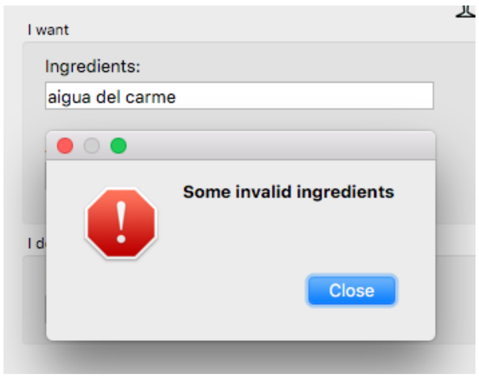


Figure 2: Error message

Once the user is presented with a recipe and in order for the evaluation and learning phases to be completed, the user shall rate the adapted recipe in a range from 0.0 to 10.0 by moving the sliding button and pressing "Rate". Then, the CBR cycle will have finished and another one will start if the user repeats the process. At any point, the constraints manually defined can be exported into a JSON formatted file by selecting `"File/Export Constraints..."`. To shut down the system, the application should be closed as any regular computer program.

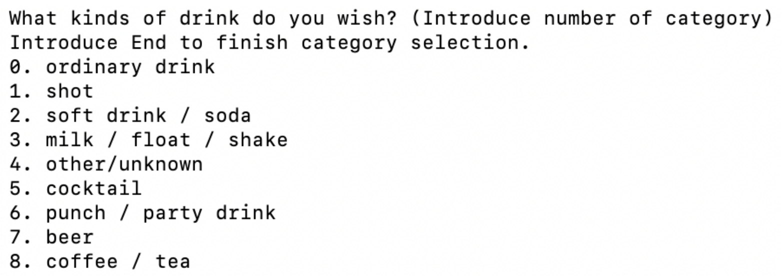
2.2 Command Line Interface (CLI)

Another option, implemented for more expert user profiles, is to execute the system via command line. For that matter, simply write the command `"python main.py` [*caselibrary_filepath*]`"` once you are in the root directory of the project. The only **positional argument** is, indeed, the filepath of the case library, which in our project will always be `"Data/case_library.xml"`. It is implemented this way so that the possibility of using another dataset is plausible.

On the other hand, the presented command has some **optional arguments**:

- `-h, --help`: show a help message and exit.
- `--verbosity`: output verbosity level. Set to 1 to print debug messages.
- `-c [CONSTRAINTS], --constraints [CONSTRAINTS]`: filepath of the JSON formatted file containing the constraints if any.

By specifying a JSON filepath with the last optional argument, the constraints will be automatically passed to the system, which will start the CBR cycle. However, if a JSON filepath is not specified, a menu will greet the user in the command line terminal, who will have to introduce all constraints manually, one by one, from a list of options displayed.



```
What kinds of drink do you wish? (Introduce number of category)
Introduce End to finish category selection.
0. ordinary drink
1. shot
2. soft drink / soda
3. milk / float / shake
4. other/unknown
5. cocktail
6. punch / party drink
7. beer
8. coffee / tea
```

Figure 3: CLI menu, category constraint

A peculiarity of this approach is that each constraint selection finishes when introducing the word “end”. This way, we can specify the number of options that we want, as long as we write “end” at the end, which will transport the user to the next constraint specification. For example, in Figure 3 a plausible response would be to choose both shot and cocktail as categories. For that matter, the commands introduced by keyboard would have to be `"1"-Enter, "5"-Enter, "end"-Enter`. When this process is completed, the constraints will be passed to the system and the CBR cycle will start as well.

To exit the system, you can either wait for the whole iteration to end, which will allow the CBR to learn from it, or terminate the execution at any moment instead.

3 Description of the different functionalities of the system

Apart from the already discussed individual start-up features of each approach, the functionalities of the interface when it comes to constraints definition and user interaction are basically the same in all cases. They can be categorized such as:

- **Positive constraints**: the principal constraints components to specify are *categories*, *ingredients*, *alcohol types*, *basic tastes* and *glass types*. These will be considered to find the most suitable recipe to the user desires.

- **Negative Constraints**: on the contrary, excluding constraints can be also indicated. Specifically, *ingredients*, *alcohol types* and *basic tastes*.

- **Cocktail name**: a personalized name for the new adapted cocktail can be provided. If it is not, the name of the adapted cocktail will be the name of the retrieved cocktail plus “2.0”, representing it as a second version of an already existing recipe.

- **Rate**: a score given to the solution that represents the level of satisfaction of the user.

For each positive or negative constraint, no values, one value or more values can be introduced following the indications already discussed in each interface form. Bear in mind that the more constraints the user provide, the more personalized will be the generated cocktail.

However, not only the values introduced for each constraint must be valid (i.e., they are identified as part of the values stored for each constraint within the system), but also the overall constraints must satisfy a series of logical conditions not managed by the system, so the user must act with consciousness in order for it to work in a reasonable way:

- There must be no direct contradictions between positive/including constraints and negative/excluding constraints for *ingredients*, *alcohol types* and *basic tastes*.
- Moreover, an indirect contradiction can take place for both the *alcohol types* and the *basic tastes* negative constraints. When some of the desired ingredients have an *alcohol type* or a *basic taste* that matches the undesired, the system will not be able to generate a cocktail satisfying all the constraints since it is not possible. Therefore, this situation must be avoided as well.

The rest of the interface functionalities have been covered in the previous section. Additionally, the inner system functionalities are covered through the principal report and are not believed to be a suitable content for this user manual.

4 Examples of use in the concrete application

Please, refer to the principal report of this project in order to visualize some real examples, both manually and automatically executed and in diverse interface formats.