CS24011: Lab 3: A Fuzzy Logic-based Washing Machine

Riza Batista-Navarro
Francisco Lobo

Academic session: 2022-23

Git Tag: `24011-lab3-S-FuzzyLogic`

Preparatory Work

For this exercise, we will use Python 3 and build upon the `frules` library. As a first step, install the library in your environment by issuing the following command

```
$ pip install frules
```

You can find more information about this library at `https://github.com/swistakm/frules`. More specifically, we will use some of its helper functions to accomplish the following.

A Defining membership functions for fuzzy sets

The library, for example, allows us to conveniently define membership functions for car age based on production year, as follows

```
from frules.expressions import Expression as E
from frules.expressions import ltrapezoid, trapezoid, rtrapezoid
# car age expressions
old = E(ltrapezoid(2001, 2008), "old")
new = E(rtrapezoid(2013, 2014), "new")
```

Figure 1 below allows us to visualise the above membership functions.

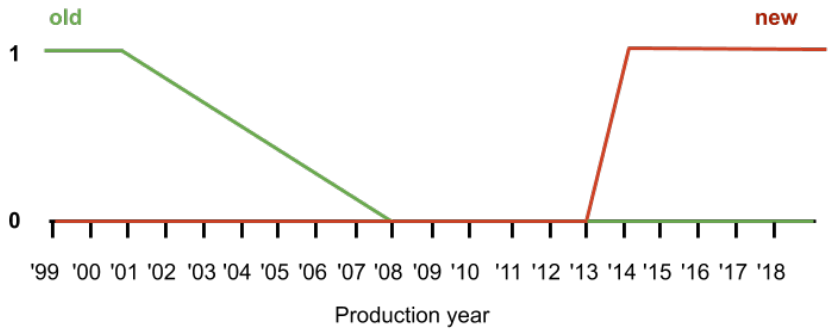

Figure 1: Visualisation of the membership functions for old and new car

B Evaluating the extent to which a crisp value belongs to a fuzzy set

We can, for example, compute "how old" or "how new" a car with production year = 2007 is, as follows

```
from frules.rules import Rule as R
old_set = R(production_year=old)
how_old = old_set.eval(production_year=2007)  #results in 0.14
new_set = R(production_year=new)
how_new = new_set.eval(production_year=2007)  #results in 0.00
```

Introduction to the Problem

A good example of home appliances which make use of fuzzy logic are washing machines[1]. For this exercise, our task is to build a simple fuzzy logic-based system that will help determine how to set the temperature that should be used by the machine, depending on the dirtiness and delicateness of clothing.

Building your Fuzzy Logic System

In your GitLab repo you will find a Python script named `fuzzy_washing_machine.py` which contains some stub code.

You will develop your own version of this script, henceforth referred to as "your solution" in this document, following the tasks outlined below.

During your development, you can for example run the command

```
$ python3 fuzzy_washing_machine.py fuzzify dirty_set 0.9
```

which will execute and show the result of `fuzzify(dirty_set,0.9)`. Similarly, you can use

```
$ python3 fuzzy_washing_machine.py get_rule_output_value 3 0.5
```

which will execute and show the result of `get_rule_output_value(3,0.5)`. The output that you will obtain from the stub code is of course not meaningful.

More generally, you can run the script as follows

```
$ python3 fuzzy_washing_machine.py FUNCTION ARGS...
```

to test each of the functions in your solution. When you complete the exercise, the command

```
$ python3 fuzzy_washing_machine.py \
    get_temperature "*(configure_washing_machine(0.9,6.5))"
```

will obtain the temperature setting for the machine when, for example, the amount of dirt is 0.9 tablespoons and the fabric weight is 6.5 ounces per square yard.

Important: Note that for any task below that is asking you to write a function with a specified function name, you will find the corresponding function signature (i.e., arguments, return type) in the stub code.

[1] `https://ao.com/l/washing_machines-90/1-20/1/`

C Initialisation

C.1 Definition of linguistic variables

We have two input variables, dirtiness and delicateness, with the following linguistic terms

```
T(dirtiness) = {almost_clean, dirty}
T(delicateness) = {very_delicate, delicate, not_delicate}
```

And we have one output variable, temperature

```
T(temperature) = {low, medium, high}
```

C.2 Definition of fuzzy sets

To define our fuzzy sets, we will use the membership functions for each of the input variables dirtiness and delicateness, which are depicted by the trapezoidal shapes in Figures 2 and 3 below, respectively.

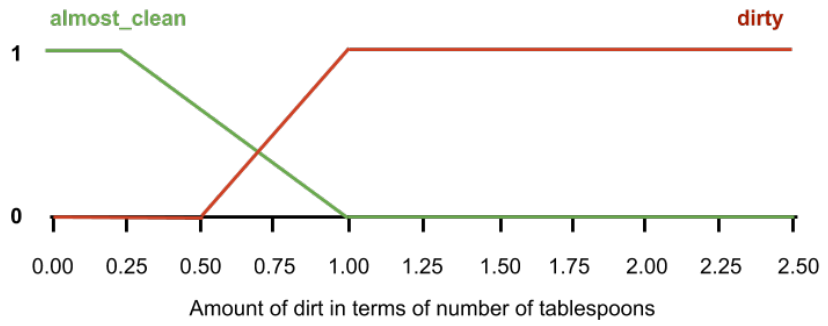

Figure 2: Curve describing the membership function for dirtiness

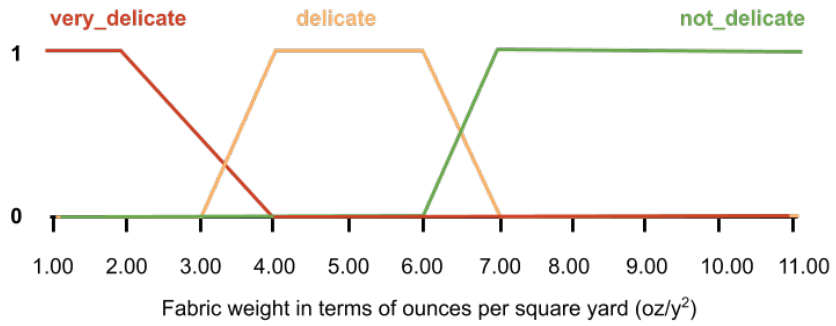

Figure 3: Curve describing the membership function for delicateness

The lines of code that define the membership functions, based on the curves above, have already been provided as part of the stub code. Make sure that you understand these lines.

C.3 Defining rules

Below are the rules that the fuzzy logic system should follow.

```
r1: IF clothing is very_delicate,
    THEN temperature should be low
r2: IF clothing is delicate OR almost_clean,
    THEN temperature should be low
r3: IF clothing is delicate AND dirty,
    THEN temperature should be medium
r4: IF clothing is not_delicate AND dirty,
    THEN temperature should be high
```

Each rule follows the `IF <antecedents> THEN <consequent>` pattern.

D Fuzzification

Now we will encode the rules above based on fuzzification.

D.1 Computation of fuzzified inputs

Task 1: In your solution, write a function called `fuzzify` that will compute the degree to which a crisp input value (i.e., amount of dirt, fabric weight) belongs to a fuzzy set.

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py fuzzify dirty_set 0.9
debug run: fuzzify(dirty_set,0.9)
ret value: 0.8
```

D.2 Evaluation of antecedents

If a rule contains more than one antecedent, their combined value can be computed through either conjunction or disjunction.

Task 2: In your solution, write a function called `get_conjunction` for computing the conjunction (AND) of antecedents in a given rule, and another function called `get_disjunction` for computing their disjunction (OR).

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py get_conjunction 0.8 0.5
debug run: get_conjunction(0.8,0.5)
ret value: 0.5
$ python3 fuzzy_washing_machine.py get_disjunction 0.8 0.5
debug run: get_disjunction(0.8,0.5)
ret value: 0.8
```

Task 3: In your solution, write a function called `get_rule_antecedent_value` that computes the combined value of a rule antecedent (which, in our case, may consist of at most two antecedents).

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py get_rule_antecedent_value \
    None 0.0 very_delicate_set 6.5 "''"
debug run: get_rule_antecedent_value(None,0.0,very_delicate_set,6.5,'')
ret value: 0.0

$ python3 fuzzy_washing_machine.py get_rule_antecedent_value \
    dirty_set 0.9 delicate_set 6.5 "'AND'"
debug run: get_rule_antecedent_value(dirty_set,0.9,delicate_set,6.5,'AND')
ret value: 0.5
```

E Inference

We will use the Sugeno-style fuzzy inference, specifically, the zero-order Sugeno fuzzy model where the output level of each rule is a constant k. The weighted output level is calculated as the product of the rule antecedent value and the output level of the corresponding rule.

Output levels are weighted because not all rules have equal importance. Below is a table showing the same four rules as above, but this time with a corresponding output level.

Rule Number	Rule	Output Level
r1	`IF clothing is very_delicate,` `THEN temperature should be low`	10
r2	`IF clothing is delicate OR almost_clean,` `THEN temperature should be low`	40
r3	`IF clothing is delicate AND dirty,` `THEN temperature should be medium`	60
r4	`IF clothing is not_delicate AND dirty,` `THEN temperature should be high`	100

Note that these output levels — just like the membership functions — are chosen arbitrarily based on user knowledge/experience.

The stub code already provides the above output levels for the four rules, in the form of a dictionary called `rule_weights_dict`, where the keys (integers from 1 to 4) correspond to the rule numbers.

Task 4: In your solution, write a function called `get_rule_output_value` that returns the value of the weighted output level of a rule given an antecedent value, based on the output levels above.

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py get_rule_output_value 3 0.5
debug run: get_rule_output_value(3,0.5)
ret value: 30.0
```

Task 5: In your solution, write a function called `configure_washing_machine` that, given crisp input values for amount of dirt and fabric weight, returns a tuple of 2 lists containing, respectively, the antecedent values and the output values for all rules.

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py \
    configure_washing_machine 0.9 6.5
debug run: configure_washing_machine(0.9,6.5)
ret value: ([0, 0.5, 0.5, 0.5], [0, 20.0, 30.0, 50.0])
```

F Defuzzification

To compute a crisp output value (for temperature), we will use weighted averaging over all rule outputs.

Task 6: In your solution, write a function called `get_weighted_average` that computes the weighted average over all rules. The weighted average is the sum of all the rule outputs divided by the sum of all the rule antecedent values.

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py \
    get_weighted_average "[0,0.5,0.5,0.5]" "[0,20.0,30.0,50.0]"
debug run: get_weighted_average([0,0.5,0.5,0.5],[0,20.0,30.0,50.0])
ret value: 66.66666666666667
```

Task 7: In your solution, write a function called `get_temperature` that computes the actual temperature value that the washing machine should be set to.

You can consider a crisp output value from the previous task as a percentage of the range of possible values for the output variable, in this case, temperature. Assume that the possible temperature range for the washing machine is between 10°C and 90°C (shown in blue below). This means that a crisp output value of 0% is 10°C and a crisp output value of 100% is 90°C.

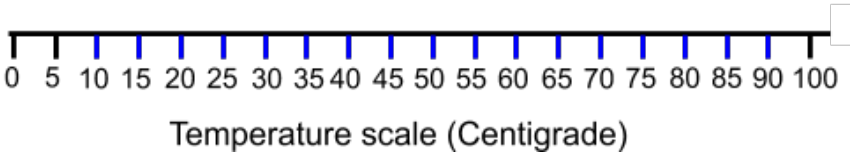

Figure 4: Temperature range

You can verify that your function is correct on the command line. For example, you should obtain the following output

```
$ python3 fuzzy_washing_machine.py \
    get_temperature "*(configure_washing_machine(0.9,6.5))"
debug run: get_temperature(*(configure_washing_machine(0.9,6.5)))
ret value: 63.33333333333334
```

Submission

Please follow the `README.md` instructions in your `COMP24011_2022` Gitlab repo to refresh the files of your `lab3` branch. You will find a file containing stub/skeleton code which is named `fuzzy_washing_machine.py` in which to write the Python code for your solution. When you are ready to push your solution, remember that you **need to** tag your commit with the command

```
$ git tag 24011-lab3-S-FuzzyLogic
```

There must be **only one** file called `fuzzy_washing_machine.py` in your commit. Also, any other files that you include **will be ignored**. General instructions on coursework submission using Gitlab can be found on the CS Handbook.

The deadline for submission is **18:00 on Monday, 28th November**. The lab will be **auto-marked** offline. The marking program will download your code from Gitlab and test it against a reference implementation. (This will take time so expect marking to take at least 2 weeks.)
Important: The skeleton `fuzzy_washing_machine.py` already includes the necessary `import` statements and the definition of the fuzzy sets. You **should not** add any top level code in your solution and only complete the required function definitions.

The skeleton script uses the `if __name__ == '__main__':` construction so that the source file to be run for debugging purposes as well as being imported like a Python module. If you want to use additional debug code, then you **need to** guard it inside this debug construction. The reason is that the auto-marking program will test your solution by importing your code as follows

```
from fuzzy_washing_machine import fuzzify, get_conjunction,
    get_disjunction, get_rule_antecedent_value,
    get_rule_output_value, configure_washing_machine,
    get_weighted_average, get_temperature
```

and any unguarded top level code will affect the testing of your functions.

Mark Scheme

There are 20 marks in total which are spread as shown in the following table.

Task	Function	Marks
1	`fuzzify()`	2
2a	`get_conjunction()`	2
2b	`get_disjunction()`	2
3	`get_rule_antecedent_value()`	3
4	`get_rule_output_value()`	2
5	`configure_washing_machine()`	3
6	`get_weighted_average()`	3
7	`get_temperature()`	3

Each of your functions will be tested on various test cases, and return values compared to the reference implementation. The proportion of fully correct return values will determine your score (out of the possible number of marks for the function shown in the table).