CS4047 Computational Intelligence Assessment

Siyana Pavlova

1.Fuzzy Rule Based Systems

The term Fuzzy Logic originated from Lotfi Zadeh in 1965. Up to now a number of applications have been found for fuzzy logic. Examples include camera, washing machines, helicopter control and even facial pattern recognition.

Fuzzy Logic is a form of multivalued logic which deals with the degree of truth of a fact or belief. The truth values that variables can take can be any real number between 0 and 1; 0 being completely false and 1 being completely true. For example, we can have P(Speed is fast) = 0.4 meaning that 'Speed is fast' is 40% true.

When dealing with Fuzzy Logic, we use fuzzy sets as opposed to crisp sets. In a crisp set, an element either belongs or does not belongs to the set. In fuzzy sets, however, an element can belong to the set to a degree. For example, the speed of 90km/h is 0.4 fast, i.e. the degree with which it belongs to the fuzzy set 'fast' is 0.4. If we plot the degree of belonging with respect to speed, the curve that we will get is called a membership function (MF).

There are a number of different MF formulations, e.g. Gaussian, Triangular. For the purpose of this assessment, we will use a Trapezoidal MF formulation. This formulation can be represented by a 4-tuple (a, b, α, β) where the MF ($\mu_{\alpha}(x)$) is as follows:

$$\mu_a(x) = \begin{cases} 0 & x < a-\alpha \\ \alpha^{-1}(x-a+\alpha) & x \in [a-\alpha \;\; a] \\ 1 & x \in [a \;\; b] \\ \beta^{-1}(b+\beta-x) & x \in [b \;\; b+\beta] \\ 0 & x > b+\beta \end{cases}$$

Let us now consider three fuzzy sets for speed: slow, medium and fast. “Speed” is also called a linguistic variable and “slow”, “medium” and “fast” would be its linguistic values. They can be represented by a 4-tuple formulation:

slow	0	30	0	20
medium	50	50	20	30
fast	70	...	20	30

We can now plot these MFs on a graph.

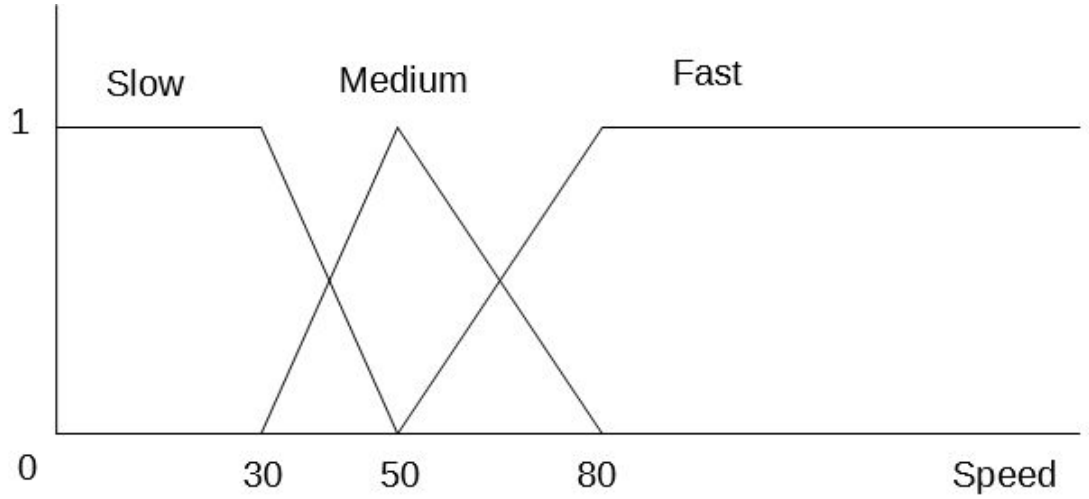

Fig.1. A sample graph for Speed

Similarly, we can construct a fuzzy set for a the quality of a road and give it the following 4-tuples:

bad	0	3	0	2
average	5	6	2	2
good	8	10	2	0

These MFs can be plotted on a graph as in Fig.2.

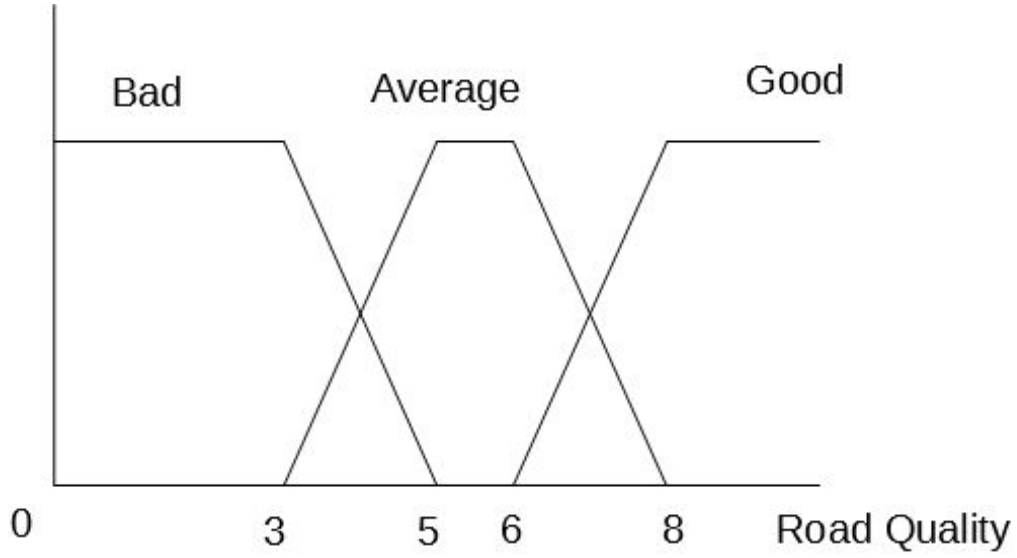

Fig.2. A sample graph for Road Quality

We have a speed graph and a road quality graph, so we can now say:
If the speed is high and the road is bad then the change in speed is 'decrease'.

This is called an if-then rule and its general format is **If x is A then y is B**. There can be more than one antecedent in a rule and they are connected by conjunctions - **AND** or **OR**.

This leads us to the problem posed by this assessment: given a set of if-then rules, a number of variables with linguistic values and 4-tuples to represent them, and a real world example (the speed is 85 and the road quality is 7), can we find an approximate value for the output variable of the rules.

A solution to such a problem can be found by using a Fuzzy Rule Based System. Such a system can be decomposed into smaller parts, thus making it easier to handle. A possible decomposition is splitting the system into a fuzzifier, an inference engine and a defuzzifier.

The job of the fuzzifier is to convert the crisp real world value (the speed is 85) into a linguistic value using the membership functions provided and to determine the degree to which the real world value belongs to a linguistic class. If the speed is 85, this would be translated to 1.0 'fast', meaning it is completely fast. However, if we were given a value which lies between two classes, the case would be different. Let us take speed is 35. In this

case, we can calculate, using the formulae in Fig. 1 that the speed is 0.75 ‘slow’ and 0.25 ‘medium’.

Once the crisp input has been fuzzified, the inference engine can use these fuzzy values and the set of rules provided to produce a fuzzy output. Let us imagine one of our rules is “If the speed is slow then the change in speed is ‘increase’”. If our speed is 0.75 ‘slow’, then the change in speed will be 0.75 ‘increase’. However, rules are not always this simple and may contain more than one antecedent. When dealing with such rules, we have to consider the conjunction between them. **AND** means taking the minimum of the fuzzy values in the rule, whereas **OR** means taking the maximum. There may be rules in our set whose outputs refers to the same linguistic value of the output variables. In this case, since rules are independent of one another, the maximum of the resulting fuzzy values is taken. At the end of the inference process, we have one fuzzy value for each different output value of the rules and are ready to defuzzify the results.

The job of the defuzzifier is to convert the fuzzy outputs into crisp outputs using the membership functions. There are different approaches to defuzzifying which produce slightly different results. In this assessment I have considered two defuzzification methods, namely the Center of Gravity (CoG) and Dilation of the Aggregate (DoA) methods.

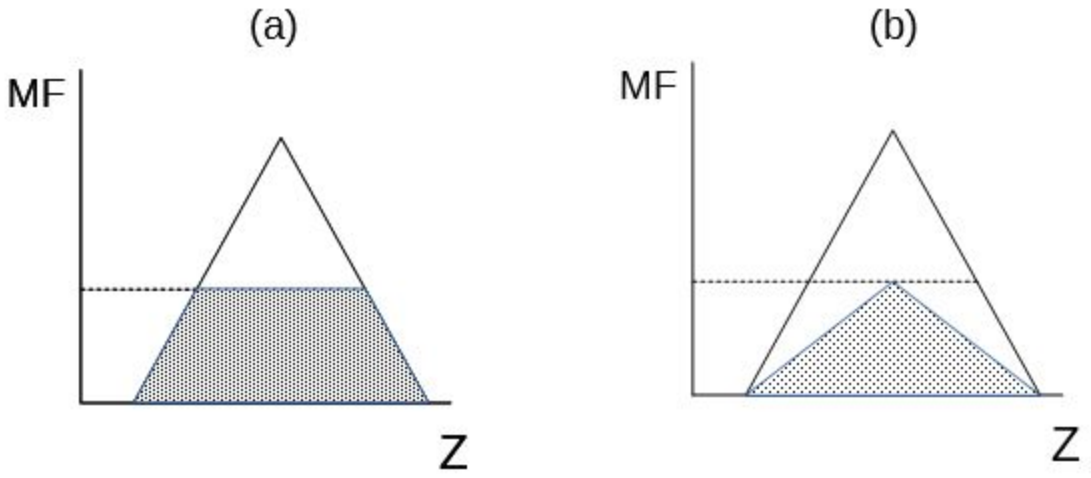

Fig.3. Comparison between (a) the CoG and (b) the DoA methods

When trying to find the crisp value that corresponds to the fuzzy value of our output, we need to consider the weight average for each of the linguistic values as found by the inference engine. The Center of Gravity and the Dilation of the Aggregate methods differ with respect to the areas they consider for calculating the weighted average. The CoG method uses the area inside the membership function that is under the membership value

calculated by the inference engine for that function as seen in Fig.3(a). The DoA method, on the other hand, considers the area of a triangle with base the bottom of the membership function, i.e. $(b + \beta - (a - \alpha))$ and height, the membership value calculated by the inference engine, thus placing the apex of such a triangle on the line of the membership value, as shown in Fig.3(b).

The weighted average is then calculated as the sum of each such area whose membership value is non-zero multiplied by the center of the membership function, i.e. $(b + \beta + (a - \alpha))/2$, divided by the sum of all the areas whose membership value is non-zero. The end result is the answer of the system.

2. System Design

My system will follow the structure of a fuzzy rule-based system described above. This means it will consist of a fuzzifier, an inference engine and a defuzzifier. Furthermore, as it needs to read all the data from a file, it will have a component which reads the data and records it in a format which the rest of the components can use.

2.1 File Parser

The problem is provided to the program in the form of a text file where all the rules, the variables and their 4-tuple fuzzy representation and the real world values of the variables are given. The file parser is the first part of the system. It reads the file given line by line and records the data provided in a from which can be used by the other parts of the system. The data is split as follows:

- rules - an array of dictionaries, where each dictionary is a rule
- variables - a dictionary of dictionaries, where each inner dictionary represents a variable and its 4-tuple fuzzy representation
- given - an array of dictionaries where each dictionary represents a variable and its real world value as given in the text document

2.2 Fuzzifier

The fuzzifier uses the variables dictionary and the given array to construct a membership functions dictionary. It uses the 4-tuple a, b, alpha and beta values to calculate and save the membership functions values for the real values for the given variables.

2.3 Inference Engine

The inference engine examines the firing of the rules in the system. It uses the rules dictionary and the membership functions dictionary to calculate the membership value for each class of the output variable.

2.4 Defuzzifier

The system uses two defuzzification methods thus allowing the user to compare their results, namely the center of gravity (CoG) defuzzification method and the dilation of the aggregate (DoA) method to find an approximate value for the output variable.

3. Implementation of the Components

The system has been implemented in Python 2.7.

3.1 File Parser

The implementation of the file parser can be found under the *readData* function in the *readData.py* file. The *readData* function takes the name of a text file, e.g. 'example.txt', parses the file and creates a rules array, a variables dictionary and a given array.

The rules array is an array of dictionaries where each dictionary has the following keys: 'connective', 'result', 'variables'.

The variables dictionary contains a key for each variable and the value of each such key is a dictionary too. The keys in the inner dictionary are the classes of the variable and their values themselves are also dictionaries with keys 'a', 'b' ,'alfa' and 'beta'.

The given array is an array of dictionaries where each dictionary has a 'name' and a 'value' key whose values correspond to the real world values for the variables provided in the problem.

Sample results for the file parser can be found in the *readDataSampleOutput.txt*[1].

[1] All the sample output files can be found in the *sampleOutputs* folder

3.2 Fuzzifier

The implementation of the fuzzifier can be found under the *fuzzify* function of the *fuzzifier.py* file. The function takes a variables dictionary with a structure as described 3.1 and a given array as described there too and produces a membership functions dictionary. The fuzzifier calculates the membership value for each class of each given variable based on the real world values provided. Sample results for the fuzzifier can be seen in *fuzzifierSampleOutput.txt*.

3.3 Inference Engine

The implementation of the inference engine can be found under the function *infer* of the *inference.py* file. The function takes a rules array with a structure as described in 3.1 and a membership functions dictionary with a structure as described 3.2. For each rule, the function finds the minimum of the membership class values for each part of the rule if the connector of that rule is "and" and the maximum of the same if the connector is "or". If the class of the result is the same for different rules, the maximum value for that class is taken. The membership values for the result are recorded in a dictionary with a key the name of the output variable and another dictionary as a value which, on its hand, has the membership classes as keys and the calculated real world values as their values. Sample results for the inference engine can be seen in *inferenceSampleOutput.txt*.

3.4 Defuzzifier

The implementation of the centre of gravity defuzzification method can be found under the *defuzzifyCoG* function of the *defuzzifier.py* file. The function takes a variables dictionary as described in 3.1 and another dictionary which contains the real world values of the output variable as described in 3.3. The function goes through each membership class of the output variable whose real world value is not 0. The final result is found using the Center of Gravity method, while always taking the 0 of the x-axis as a point of reference for the answer and then returned.

Similarly the implementation of the dilation of the aggregate defuzzification method can be found under the *defuzzifyDoA* function of the same file. This works in the same way as the *defuzzifyCoG* function, the only difference being that the area are calculated using the dilation of the aggregate method.

Sample outputs for the defuzzifiers can be seen in *defuzzifierCoGSampleOutput.txt* and *defuzzifierDoASampleOutput.txt* for the CoG and DoA methods respectively.

4. Implementation of the Complete System

The complete system is essentially a combination of the components described in 3. executed in a sequence. The file parser runs first producing a rules array, a variables dictionary and a given array. The fuzzifier is then run with the variables dictionary and the given array produced by the file parser as an input and producing a membership functions dictionary as its output. Then follows the inference engine using the rules array from the file parser and the membership functions dictionary from the fuzzifier as input and producing a dictionary of the real world values for the membership classes of the output variable. In order to allow the user to compare both the CoG and DoA fuzzification methods, I run both the *fuzzifyCoG* and *fuzzifyDoA* functions. They both take the variables dictionary and the output of the inference engine as their input and return the final answer of the system as an output.

When run in this sequence, the system can take a text file containing the rules, membership functions and real life values of a system and produce an answer. Sample results for the entire system are provided in the table below. I have included the outputs for both CoG and DoA fuzzification. The file 'tippingRulebase.txt' contains the tippingRuleBase provided as an example for this assessment, 'currentRulebase.txt' contains the Current and Temperature rule base provided in the lectures and the tutorial of the course and 'drugRulebase.txt' contains the heart rate and respiration rate example, the solution of which can be found in the answer sheet for the fuzzy logic tutorial of this course.

Rule Base	**Expected CoG Output**	**CoG Output**	**Expected DoA Output**	**DoA Output**
'tippingRulebase.txt'	108.26	108.26086957	105.56	105.55555556
'currentRulebase.txt'	-31.04	-31.04624716	-33.87	-33.87096774
'drugRulebase.txt'	2.53	2.53653445	2.48	2.48148148

The table above shows the expected output for each example, calculated manually with a precision of up to two decimal places for both CoG and DoA, and the CoG and DoA outputs of the system rounded up to eight decimal places. As it can be seen from the table, the system produces the expected results. There are only minor differences. For example, the Expected CoG value for 'currentRulebase.txt' is -31.04, whereas the value obtained by the system is -31.04624716, which would be rounded up to -31.05. However, this difference is due to the lower precision used when the expected value was calculated manually.

5. Testing and Evaluation

A more exhaustive testing has been made for each of the rulebases above by varying the real world values for each variable. The process was automated and the results have been included in Excel files, which can be found in the *testingAndEvaluation* folder.

tippingRulebase.xls presents the output obtained when varying the 'driving' variable with a step 5 and the 'journey_time' variable with step 1.

currentRulebase.xls presents the output obtained when varying the temperature by 20 and the current by 1.

drugRulebase.xls presents the output obtained when varying the heart rate by 10 and the respiration rate by 1.

The results provided show how the output varies depending on the real world values of the variables and is not simply boolean. It can be clearly seen that the regions where the real world value of one of the variables falls in only one membership class, the output tends to stay constant. However, when the real world value of one (or more) of the variables is between two classes, a clear trend is observed as to how the output changes.

5.1 Limitations

Despite the accurate results provided by the system, there are a number of limitations to it. A very important limitation is that the system would work in this form only if the files passed to it are in the specific format specified in the assignment. A number of other limitations include:

- The system accepts only rules which have one type of a conjunction (only AND or only OR)
- The system can handle only one output variable.
- The system would always give an output of 0 if the real world value of a variable does not fall in any of the classes for that variable
- From above follows that if the real world values do not fall into any of the rules specified, the system will output 0. For example, in the currentRulebase example, there is no rule specifying what happens if the temperature is low and/or the current is medium. Therefore the real world example of temperature = 0 and current = 10 would produce an output of 0.

- If there is a mistake in the input file, the system may throw an error or give a wrong answer to the problem.
- The system will fail if the name of any of the variables given consists of two or more separate words (avoided by the use of underscore for this version of the system)
- The system lacks a friendly user interface thus making it difficult for non-experts to work with it.

A future version of the system would have to build upon the existing one and address all these issues. However, considering the scope of this assignment, the system has achieved its purpose.

5.2 Conclusion

The system provides an output for a Fuzzy Logic rule base presented to it in the required format. Furthermore, it provides results for two different defuzzification methods, thus allowing the end user to compare the output. The values for both methods align with the expected output thus showing an accurate and complete system.