Fuzzy Based Rule System

Grant Christie

1 Fuzzy Logic Overview

Fuzzy Logic is a form of many valued logic where the truth values of variables may be any real number between 1 and 0 as opposed to Boolean Logic where the values may only be 1 or 0. This method of reasoning is more akin to the way humans make decisions as they consider all the possibilities in between the traditional digital values YES and NO. The architecture of a Fuzzy Logic System has four main parts:

- Fuzzifier - This transforms the systems inputs into fuzzy sets.
- Knowledge Base - This stores the IF THEN rules provided by experts.
- Inference Engine - This performs the reasoning process by making fuzzy inference on the inputs and IF THEN rules.
- Defuzzifier - This transforms the fuzzy set acquired from the inference engine into a crisp value.

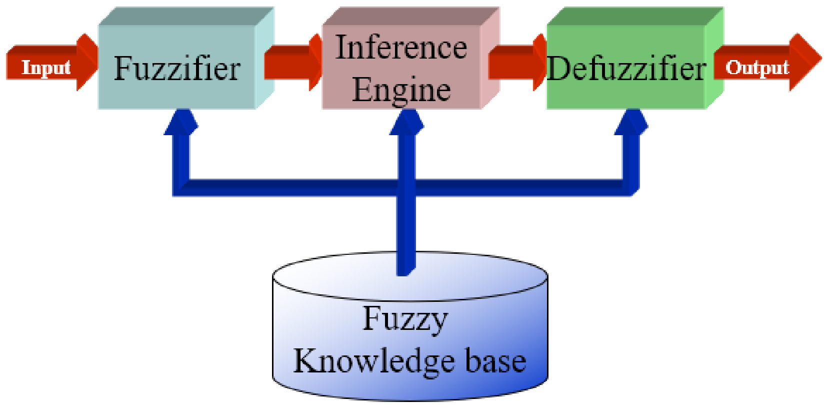

Figure 1: Fuzzy Logic System Architecture

As shown above the Fuzzifier turns a crisp input into a fuzzy set, a fuzzy set is simply a set which has a fuzzy boundary. For example, if we have a set A of young people, in a crisp set an element either belongs in the set A or does not belong in the set A. Therefore, there needs to be a value in which anything below that number is classed as young and anything above that number is classed as old, say 40. However, in the real world it may seem odd to classify a 39 year old and a 5 year old in the same manner, this is where fuzzy sets help. Fuzzy logic makes use of membership functions to determine how much a given value belongs to a set. Membership functions map each element to a value between 1 and 0 which quantifies the degree of membership of the element in the fuzzy set. An important thing to note is that membership functions are subjective measures so different systems may provide different results depending on their implementation. Therefore, in this example we give 40 the value 0.5, this represents the middle point of young and not young. A person who is age 1 is likely to have the value 1 as they are definitely young, however a person aged 20 may have the value 0.75 as they are starting to become older and now belong less to the set A. As can be seen if the 0.5 value had been set at a different age, say 30, the membership value of the 20 year old would not be the same.

Once the system has used the membership functions to convert the crisp data into a fuzzy set the rule base within the inference engine is then used to evaluate the rules. Each rule must be examined and for each rule the likelihood that that rule will be executed is calculated. To determine this the main part of the rule that must be looked at is the logical operator, AND or OR. If the AND operator is used, then the minimum value must be used and if the OR operator is being used then the maximum value must be used. After each rule has been examined and a value or values have been obtained for each, the system must then combine the results to have a single value for each one. As each of the rules in the rule base are to be treated independently from one another the system uses OR which again equates to finding the MAX of the given values.

Finally, the system uses these fuzzy values to perform defuzzification. To do this the system will use the following equation:

$$x = \frac{\sum A_i X_i}{\sum A_i}$$

Where, A_i= areas of the membership regions. X_i = centres of the membership regions.

To calculate the area of the membership regions the system must multiply the length of the base of the curve by the relevant fuzzy values and then multiply the result of that by 0.5. The centre is simply the halfway point of the membership region.

2 Design Document

For my fuzzy logic implementation, I will split my system into 5 separate modules:

- File Reader
- Membership Calculator
- Rule Reader
- Rule Firer
- Defuzzifier

2.1 File Reader

The first part of my system will be the file reader. This component will be responsible for reading the input file and dissecting it into its different parts for use by the system. The system will split the file into the following parts:

- Rule Base Name
- Rules

- Tuples
- Real Values

Algorithm 1: File Reader

```
Data: .txt file to be read
Result: Rule base split into rulebase name, rules, tuples and real vaues
begin;
Read text file;
Split file using line breaks;
Rulebase name = first split section;
Rules = second split section split by line breaks;
Real values = last section split by line breaks;
foreach item in the last section do
    Split the item into it's label and value;
end
foreach item in the third section do
    Group the fuzzy sets by using the relevant header;
end
```

2.2 Membership Calculator

The Membership Calculator takes in a 4 tuple (a,b,alpha,beta) and a value and calculates the degree to which the value belongs to the fuzzy set created by the 4 tuple on a scale of 0 to 1.

Algorithm 2: Membership Calculator

```
Data: a,b $\alpha$, $\beta$, value
Result: Membership Value
begin;
if value $< a - \alpha$ then
    return 0 ;
else if value $\in [a - \alpha, a]$ then
    return (value - a + $\alpha$)/$\alpha$ ;
else if value $\in [a, b]$ then
    return 1 ;
else if value $\in [b, b + \beta]$ then
    return ( b + $\beta$ - value)/$\beta$ ;
else if value $> b + \beta$ then
    return 0 ;
```

2.3 Rule Reader

The rule reader operates similarly to the file reader, it takes the set of rules acquired from the file reader and uses regular expressions to split the rule into its various components. These parts are the Rule ID, the variable names, the values that correspond to the variables, the operator being used and the output response.

Algorithm 3: Rule Reader

```
  Data: Single text rule
  Result: A dictionary containing ID, variables, values and output
1 begin;
2 if rule matches then
3     ID = matched ID;
4     variables = matched variables;
5     values = matched values;
6     operator = matched operator;
7     output = matched output;
8     output value = matched output value;
9 return dictionary: (ID, variables, values, operator, output, output
   value)
```

2.4 Rule Firer

This module will examine each rule against the values created by the membership calculator. Depending on the operator used by the rule the system will either find the maximum value (OR) or the minimum value(AND). It then uses max to acquire a single value for each set group that will be used for defuzzification.

Algorithm 4: Rule Firer

```
Data: List of read rules
Result: Output value for each group
begin;
foreach rule do
    extract variable name and their values;
    if The rule operator is AND then
        OperatorValue = MIN(values);
    else
        OperatorValue = MAX(values);
    Append OperatorValue to dictionary against the output group of
      the rule;
end
foreach Value in dictionary do
    FinalValue = MAX(value) if FinalValue $\neq$ 0 then
        Add FinalValue to FiredRules array;
end
return FiredRules array
```

2.5 Defuzzifier

This module calculates the final crisp value based on the values obtained by the rule firer. The defuzzifier calculates this value by taking the sum of the areas multiplied by the centres divided by the sum of the areas.

Algorithm 5: Defuzzifier

```
Data: Fired Rule Values
Result: Defuzzified Value
begin;
foreach Fired Rule Value do
    Base = (b + $\beta$) = (a - $\alpha$);
    Area = 0.5 * Base * Fired Rule Value;
    Centre = (a-$\alpha$) + Base/2
end
Defuzzified Value = (Sum of Areas * Centres)/(Sum of Areas)
```

3 Implementation

3.1 File Reader

The individual file reader module can be found in File Reader.py. When executing the file the script will ask the user for the name of the file they wish to have the system read. The script will then execute and return to the user a dictionary which contains the keys: Rule_Base, Rules, Sets, Real_Values and their corresponding values.

3.2 Membership Calculator

The individual membership calculator module can be found in Membership.py. When executing the file the script will ask the user for the a, b, alpha, beta and value to generate the membership value and return it to the user.

3.3 Rule Reader

The individual rule reader module can be found in Rule Reader.py. When executing the file the script will ask the user for the rule they wish to be read. This rule must follow the pattern indicated in the assignment specification: Rule [x] if the <variable>is <value>[and—or] <variable>is <value>then the will be <variable><value>If an input is entered that does not fit this format an error message will be returned. If the input is accepted by the script then it will return a dictionary to the user with the following keys: ID, Variables, Values, Operator, Output. Each key will have its corresponding values populated from the relevant section in the inputted rule.

3.4 Rule Firer

In order for this module to work it is necessary that the previous three modules are combined to create a partially complete system. This is required because we need to have a set of rules to work with and we need to be able to calculate membership values. When executing the partial system the script, Firing Rules.py, will ask the user for the name of the file they wish to be used, this is the functionality that was created in the first module, File Reader.py. Within this file should be the rules and variables outlined within the assignment specification in the correct format. The script will use these rules and values to calculate single values for each outcome set. In example.txt the outcome sets are small, moderate and big under the identifier tip.

3.5 Defuzzifier

Like the rule firer module the defuzzifier can not work without combining the previous four modules together. The defuzzifier requires rules to be fired to defuzz the values returned and the rule firer cannot work without the first three modules. The defuzzifier module can be found in Defuzz.py, when executing the file the script will again ask the user for the filename of the file they wish to be used by the script. The defuzzifier will take the values returned from the rule firer module and perform a series of calculations shown in section 2.5 to return the final defuzzified value to the user.

4 Testing

4.1 File Reader

Test Number	Document Tested	Expected Result	Actual Result
1	example.txt	Dictionary 1 in expectedfile.txt should be returned.	Output matches dictionary 1.
2	example2.txt	Dictionary 2 in expectedfile.txt should be returned.	Output matches dictionary 2.
3	errorexample.txt	Script should terminate with no output.	Script terminates.
4	Any filename that does not exist, empty.txt for example	Script should terminate with no output.	Script terminates.

4.2 Membership Calculator

To perform this test I will use the following 4 tuple: Middle-Age (50,60,15,5) with the following ages: 20,37,55,63,85

Test Number	Inputs	Expected Result	Actual Result
1	50,60,15,5,20	0	0
2	50,60,15,5,37	0.13	0.13
3	50,60,15,5,55	1	1
4	50,60,15,5,63	0.4	0.4
5	50,60,15,5,85	0	0

4.3 Rule Reader

Test Number	Document Tested	Expected Result	Actual Result
1	Rule 1 If the driving is good and the journey_time is short then the tip will be big	{'ID': '1', 'Variables': ['driving', 'journey_time'], 'Values': ['good', 'short'], 'Operator': 'and', 'Output': {'tip': 'big'}}	{'ID': '1', 'Variables': ['driving', 'journey_time'], 'Values': ['good', 'short'], 'Operator': 'and', 'Output': {'tip': 'big'}}
2	Rule1 If the driving is good and the journey_time is short then the tip will be big	Invalid Rule	Invalid Rule
3	Empty String	Invalid Rule	Invalid Rule
4	rule 4 if the temperature is low and the current is low then the result will be increaseCurrent	{'ID': '4', 'Variables': ['temperature', 'current'], 'Values': ['low', 'low'], 'Operator': 'and', 'Output': {'result': 'increaseCurrent'}}	{'ID': '4', 'Variables': ['temperature', 'current'], 'Values': ['low', 'low'], 'Operator': 'and', 'Output': {'result': 'increaseCurrent'}}
5	Rule 4 If driving is bad and the journey_time is long then the tip is small	Invalid Rule	Invalid Rule

4.4 Rule Firer

Test Number	Document Tested	Expected Result	Actual Result

1	example.txt	Big = 0.1
		Moderate = 0.8
			Big = 0.1
			Moderate = 0.8

2	example2.txt	Reduce Current = 0.7 No Change = 0.3	Reduce Current = 0.7 No Change = 0.3
3	errorexample.txt	Script should terminate with no output.	Script terminates.
4	Any filename that does not exist, empty.txt for example	Script should terminate with no output.	Script terminates.

4.5 Defuzzifier

Test Number	Document Tested	Expected Result	Actual Result
1	example.txt	65	65
2	example2.txt	Reduce Current = 0.7 No Change = 0.3	Reduce Current = 0.7, No Change = 0.3
3	errorexample.txt	Script should terminate with no output.	Script terminates.
4	Any filename that does not exist, empty.txt for example	Script should terminate with no output.	Script terminates.

The only difference here between expected and actual results is the accuracy of the floating point number in test number 1. The system returns a more accurate number so there is no problem with the mismatch.

4.6 Full System

To test the exhaustively test the full systems reliability, I will use the both rule base found in example.txt and change the real world values to ensure the correct results is returned in different circumstances. As a general rule, tests where driving is good and journey times are short should always return a higher value than those that represent the opposite. The first few tests will be based on the more extreme scenarios and then the rest will deviate slightly to see how the output value changes. If the journey time increases the tip should be less and vice versa. If the driving quality increases the tip should also increase and vice versa. There are also instances where given the real world values where the rule firing will only return one potential membership, in this example small:1. If given this value, the system will return the maximum value of that tuple in the rule base. For this test set those max values are 50 for small, 100 for moderate and 150 for big.

Test Number	Real World Values	Result

1	journey_time = 9
	driving = 65
		105.55555555555556

2	journey_time = 4
	driving = 77
	(This represents a very good scenario)
		120.6896551724138

3	journey_time = 19
	driving = 10
	(This represents a very bad scenario)
		50

4	journey_time = 12
	driving = 35
	(This represents a very average scenario)
		87.5

5	journey_time = 6
	driving = 80
		114.28571428571429

6	journey_time = 8
	driving = 80
		108.33333333333333

7	journey_time = 8
	driving = 70
		112.5

8	journey_time = 5
	driving = 100
		125

9	journey_time = 20
	driving = 20
		50

10	journey_time = 10
	driving = 50
		100

As can be seen the system behaves as one would expect from a Fuzzy Logic system, as the real world values change the tip either increase or decreases as would be expected given the rules within the rule base.

5 Evalulation

5.1 Limitations

- Rulebase must follow the layout outlined in the specification strictly.
- If there is an issue within the rulebase file the system will terminate abruptly.
- Rules must follow a strict pattern.
- Only one type of conditional operator may be used within a single rule.

5.2 Future Work

- Improve user interface and add visualisation of the data.
- Improve error catching.
- Allow for flexibility in rulebase and rule formatting

5.3 Conclusion

As it stands given correctly formatted inputs the system returns a calculated defuzzified output which is fairly accurate, there can be a small margin of difference between values the system returns and hand calculated values bu the margin is never too large for this to become an issue. This is also somewhat to be expected as it is in the nature of fuzzy logic to not have a ”fixed” defuzzified value.