Functioning Fuzzy Inference System

Dr. Archie Singh **Khuman**

INSTITUTE OF ARTIFICIAL INTELLIGENCE · GAMES, MATHS & ARTIFICIAL INTELLIGENCE · GH6.57

De Montfort University, Gateway House, Leicester, United Kingdom, LE1 9BH

+44 (0)116 250 6251 | arjab.khuman | arjab singh khuman | archie khuman

"Be inspired or be inspirational - Either way, great things can be accomplished."

Outline for the Presentation...

MATLAB - Getting Started

- We will build upon the previous lab's work and add functionality to our *fuzzy inference system* (**fis**).
- If you did install **MATLAB** on your own devices, make sure that you also included the **Fuzzy Logic Toolbox** as part of the install.
- The **Fuzzy Logic Toolbox** contains the *functions calls* that we will be making use of.
- Without the toolbox, your commands will not be executed, and your scripts will **NOT** compile & run.
- **Make** sure to also watch the *video tutorial*.
- The tutorial will offer an additional *understanding*, and you can follow along in your own time.

Outline for the Presentation...

1. Getting Started
2. Fuzzy Inference System
3. Rule-Base & Inference
4. The Code

Fuzzy Inference System

- For this lab we will begin implementation of a **rule-base** & **inference engine**.
- Your coursework will involve you creating *your own* fuzzy inference systems, they are all started in much the same way.
- You will get to decide on *what* your system will be doing.
- Use your creativity and choose something that genuinely *interests* you.
- Many previous students have gone on to implement fuzzy systems into their *development projects*.
- A fuzzy system is an **AI** paradigm, so *more* marks if it's applicable to your project.

Fuzzy Inference System

- Open the **previous** script you created from the previous lab session, the code can also be found in the lab folder.
- We will be adding to this a *rule-base* & *inference engine*.
- **Make** sure to **watch** the accompanying *video tutorial*.
- By the end of the lab, you will have a *fully functioning*, albeit, very basic **fis**.
- You could use this code as the *basis* of your coursework if you wish.
- We will at some point create a more *complex* fis, more aligned with the coursework requirements.

Outline for the Presentation...

Rule-Base & Inference

- At the heart of every fis is the *rule-base* & *inference engine*.
- The rule-base are effectively rules that *you* have created based on your chosen domain.
- The rule-base will be the embodiment of **YOU**.
- Your systems will only be as *good* as the rules you have created for them.
- The inference is conducted on these rules to provide final *output values*.
- There is **NO** absolute right answer to *how many* rules a system should have.

Rule-Base & Inference

- When you start MATLAB, please remember to navigate to a *directory* that you have *permission* to save to if working from the labs, shouldn't be an issue if installed on your own devices.
- We will begin with using the command window, the **main** window, to enter in commands following the following *prompt*: >>
- We will begin with exploring the declaration for creating a *adding* fuzzy rules to our system.
- Enter the following command at the prompt, **DO NOT** include the >> in your command itself.

```
>> help addrule
```

Rule-Base & Inference

- This function call is *absolutely* needed for when you begin to add rules into your system.
- Reading the explanation, you can see that there are *several* ways in which one can enter rules into a fis.
- We will concentrate on the *numerical representation*, but feel free to use the other methods if you wish.
- The numeric method uses an *array* to pass in the rule arguments.
- Continuing with our **Player Skill** system, we will now begin to add some rules to the system.

Rule-Base & Inference

- Remember, the system is with regards to defining a player's *skill level*.
- There are **2** inputs with **1** output.
- **Input 1 - Player Accuracy**, has **3** fuzzy associated to it:
  Poor, *Average* and *Good*.
- **Input 2 - Damage Output**, has **5** fuzzy associated to it:
  Very Little, *Little*, *Medium*, *High* and *Very High*.
- **Output 1 - Player Skill**, has **3** fuzzy sets associated to it:
  Low Skill, *Average* and *High Skill*.

Rule-Base & Inference

- When creating the rule-base, use *common sense*, and your own *subjectivity*.
- Let us begin with the *left-most* extreme:

  If the player's accuracy is poor, and their damage output is very little, then their player skill should be low.

- Does that make *sense* to you? It makes sense to me.
- Consider the *right-most* extreme:

  If the player's accuracy is good, and their damage output is very high, then their player skill should be high.

Rule-Base & Inference

- What about something down the *middle*:

  If the player's accuracy is average, and their damage output is medium, then their player skill should be average.

- We have just created the basis of **3** very logical rules that follow *reasoning* and *common sense*.
- Using the numeric representation, we will now *code* this into our system.
- Go to your **script file** and go towards the bottom; after the last `addmf` and before the first `subplot` declaration.

Rule-Base & Inference

- Add the following line of code:

```
rule1 = [1 1 1 1 1];
```

- We know that the system has **2** inputs and **1** output.
- $m = \mathbf{2}$, $n = \mathbf{1}$, where m is the number of inputs and n is the number of outputs.
- The position of the $\mathbf{1}^{st}$ value in `rule1` is reference to input **1**.
- The position of the $\mathbf{2}^{nd}$ value in `rule1` is reference to input **2**.
- The position of the $\mathbf{3}^{rd}$ value in `rule1` is reference to output **1**.

$rule1 = [1\ 1\ 1\ 1\ 1];$

- The position of the $\mathbf{4}^{th}$ value in `rule1` is reference to the **weight** of the rule.
- The weight of the each rule is *typically* defaulted to **1**, but can be any value in the range of $[0, 1]$.
- If you multiply anything by **1**, the result does **NOT** change.
- However, if you multiply it by say **0.5**, you will *reduce* the rule firing strength by *half*.
- The position of the $\mathbf{5}^{th}$ value in `rule1` is reference to the **operator**.
- The operator can be either an **AND**(**1**) or an **OR**(**2**).

rule1 = [1 1 1 1 1];

- The **value** of the 1^{st} position in `rule1` is **1**.
- We already know that the position is reference to *input 1*.
- The value **1** itself is referencing the 1^{st} *fuzzy set* of input 1.
- Therefore, we have:

```
rule1 = [Poor, 1 1 1 1]
```

- The **value** of the 2^{nd} position in `rule1` is also a **1**.
- We already know that the position is reference to *input 2*.
- The value **1** itself is referencing the 1^{st} *fuzzy set* of input 2.
- Therefore, we have:

```
rule1 = [Poor, Very Little, 1 1 1]
```

rule1 = [1 1 1 1 1];

- The **value** of the 3^{rd} position in `rule1` is **1**.
- We already know that the position is reference to the *output*.
- The value **1** itself is referencing the 1^{st} *fuzzy set* of the output.
- Therefore, we have:

```
rule1 = [Poor, Very Little, Low Skill, 1 1]
```

- The **value** of the 4^{th} position in `rule1` is also a **1**.
- We already know that the position is reference to the *weight*.
- Therefore, we have:

```
rule1 = [Poor, Very Little, Low Skill, (1), 1]
```

rule1 = [1 1 1 1 1];

- The **value** of the 5^{th} position in `rule1` is also a **1**.
- We already know that the position is reference to the *operator*, with **1** indicating use of the **AND** operator.
- The operator *links* the antecedents of a fuzzy rule together.
- Remember the truth tables associated to **AND** & **OR**.
- Therefore, we have:

```
rule1 = [Poor, Very Little, Low Skill, (1), AND]
```

- `rule1` can be read-off like a *sentence*.

rule1 = [1 1 1 1 1];

- Start with 1^{st} position in the rule, which translates to:

  IF Player Accuracy is Poor...

- Then jump to the 5^{th} position, the **last** position, the operator:

  IF Player Accuracy is Poor AND...

- We now go to the 2^{nd} position in the rule, which translates to:

  IF Player Accuracy is Poor AND Damage Output is Very Little...

- Now we jump to the output position, the 3^{rd} position:

  IF Player Accuracy is Poor AND Damage Output is Very Little THEN Player Skill is Low Skill (1)

- With a weighting of 1 for the rule firing strength.

Rule-Base & Inference

- Let us consider the *second rule* that was discussed:

  If the player's accuracy is good, and their damage output is very high, then their player skill should be high.

- Another sensible rule which follows the *reasoning* of `rule1`.
- Add the following line of code to your **script file**:

```
rule2 = [3 5 3 1 1];
```

- Place it directly under `rule1` in your code.

rule2 = [3 5 3 1 1];

- The **value** of the 1^{st} position in `rule2` is **3**.
- We already know that the position is reference to *input 1*.
- The value **3** itself is referencing the 3^{rd} *fuzzy set* of input 1.
- Therefore, we have:

```
rule2 = [Good, 1 1 1 1]
```

- The **value** of the 2^{nd} position in `rule2` is a **5**.
- We already know that the position is reference to *input 2*.
- The value **5** itself is referencing the 5^{th} *fuzzy set* of input 2.
- Therefore, we have:

```
rule2 = [Good, Very High, 1 1 1]
```

$rule2 = [3\ 5\ 3\ 1\ 1];$

- The **value** of the 3^{rd} position in `rule2` is **3**.
- We already know that the position is reference to the *output*.
- The value **3** itself is referencing the 3^{rd} *fuzzy set* of the output.
- Therefore, we have:

```
rule2 = [Good, Very High, High Skill, 1 1]
```

- The **value** of the 4^{th} position in `rule2` is a **1**.
- We already know that the position is reference to the *weight*.
- Therefore, we have:

```
rule2 = [Good, Very High, High Skill, (1), 1]
```

$rule2 = [3\ 5\ 3\ 1\ 1];$

- The **value** of the 5^{th} position in `rule2` is a **1**.
- We already know that the position is reference to the *operator*, with **1** indicating use of the **AND** operator.
- The operator *links* the antecedents of a fuzzy rule together.
- Remember the truth tables associated to **AND** & **OR**.
- Therefore, we have:

```
rule2 = [Good, Very High, High Skill, (1), AND]
```

- `rule2` can also be read-off like a *sentence*.

rule2 = [3 5 3 1 1];

- Start with 1^{st} position in the rule, which translates to:

  IF Player Accuracy is Good...

- Then jump to the 5^{th} position, the **last** position, the operator:

  IF Player Accuracy is Good AND...

- We now go to the 2^{nd} position in the rule, which translates to:

  IF Player Accuracy is Good AND Damage Output is Very High...

- Now we jump to the output position, the 3^{rd} position:

  IF Player Accuracy is Good AND Damage Output is Very High THEN Player Skill is High Skill (1)

- With a weighting of 1 for the rule firing strength.

Rule-Base & Inference

- The third rule of the system was a rule down the middle:

  If the player's accuracy is average, and their damage output is medium, then their player skill should be average.

- Another sensible rule which follows the *reasoning* of `rule1` & `rule2`.

- Add the following line of code to your **script file**:

```
rule3 = [2 3 2 1 1];
```

- Place it directly under `rule2` in your code.

rule3 = [2 3 2 1 1];

- Start with 1^{st} position in the rule, which translates to:

  IF Player Accuracy is Average...

- Then jump to the 5^{th} position, the **last** position, the operator:

  IF Player Accuracy is Average AND...

- We now go to the 2^{nd} position in the rule, which translates to:

  IF Player Accuracy is Average AND Damage Output is Medium...

- Now we jump to the output position, the 3^{rd} position:

  IF Player Accuracy is Average AND Damage Output is Medium THEN Player Skill is Average (1)

- With a weighting of 1 for the rule firing strength.

Rule-Base & Inference

- With these **3** rules in our system, we have created a very *small* & *simple* rule-base.

- Go back to your **script file** and add the following under the rule declarations:

```
ruleList = [rule1; rule2; rule3];

a = addrule(a, ruleList)

showrule(a)
```

- `ruleList` is a matrix which *holds* each individual rule.

- `addrule` is the command that passes the `ruleList` into our **fis**.

Rule-Base & Inference

- Compile & run the code and see what is displayed in the **Command Window**.
- You can see the fis properties and also the rules, *linguistically* presented.
- We can now *pass* the input values to our system.
- In your **script file** add the following under the `showrule` declaration:

```
Input1 = 10

Input2 = 15
```

- Now Compile & run the code and see what is displayed in the **Command Window**.

Rule-Base & Inference

- Nothing new is displayed, but at least we know it compiles & runs without issue.
- Every **fis** will have to make use of the `evalfis` to return an output.
- Without it, **NO** *inference* will be undertaken.
- In your **script file** add the following under your input declarations:

```
Output = evalfis(a,[Input1, Input2])
```

- `evalfis` requires a reference to the **fis** (a) and the input values.
- `Output` is the variable that will *store* the result of `Input1` & `Input2` being passed through to the **fis**.

Rule-Base & Inference

- Compile & run the code and see what is displayed in the **Command Window**.
- The result of the *evaluation* should be **10.4755**
- Does that seem reasonable? It does to me.
- Change the input values in the **script file** to:

```
Input1 = 80

Input2 = 85
```

- Notice the output jump to **83.9697**

Rule-Base & Inference

- In your **script file** add the following under your `evalis` declaration:

```
ruleview(a)
```

- This will allow you to *inspect* the rule-base and see how the output is being calculated.
- Click and drag on the red lines to *change* the input values.
- Or, simply go to the *Input* text box and enter them in *manually*.
- Make sure to separate each input value with a semi-colon (;).

Rule-Base & Inference

- Change the input values in the **script file** to:

  `Input1 = 10`

  `Input2 = 50`

- The output is now **50**
- Did you notice what was stated in the **Command Window**?
- We do not have any rules in our rule-base to *accommodate* for this specific input combination.
- Try adding more rules to the rule-base to *cater* for such an input combination.

Rule-Base & Inference

- If the system made use of **3** inputs and **1** output, it *could* have the following rule array:

```
rule1 = [1 1 1 1 1 1]
```

- If the system made use of **3** inputs and **2** outputs, it could have the following rule array:

```
rule1 = [3 2 2 3 1 0.75 1]
```

- The rule-base is *indicative* of how you perceive the problem domain.
- Your systems will only be as *good* as your rule-base.
- If the output of your system doesn't seem to be quite right, *modify* your rule-base.

Outline for the Presentation...

1. Getting Started
2. Fuzzy Inference System
3. Rule-Base & Inference
4. The Code

The Code

```
% A declaration of new FIS

a = newfis('Player Skill');

% Declaring a new variable - this is an INPUT(1)

a=addvar(a,'input','Player Accuracy (%)',[0 100]);

% Populating the 1st input variable with ...
membership functions

a=addmf(a,'input',1,'Poor','trapmf',[0 0 15 25]);

a=addmf(a,'input',1,'Average','trimf',[20 50 80]);

a=addmf(a,'input',1,'Good','trapmf',[70 90 100 100]);
```

The Code

```
% Declaring a new variable - this is another INPUT(2)

a=addvar(a,'input','Damage Output (%)',[0 100]);

% Populating the 2nd input variable with ...
membership functions

a=addmf(a,'input',2,'Very Little','trapmf',[0 0 ...
10 20])

a=addmf(a,'input',2,'Little','trimf',[15 25 35]);

a=addmf(a,'input',2,'Medium','trimf',[30 50 70]);

a=addmf(a,'input',2,'High','trimf',[65 75 85]);

a=addmf(a,'input',2,'Very High','trapmf',[80 90 ...
100 100]);
```

The Code

```
% Declaring a new variable - this is an OUTPUT(1)

a=addvar(a,'output','Player Skill',[0 100]);

% Populating the output variable with membership ...
functions

a=addmf(a,'output',1,'Low Skill','trapmf',[0 0 10 ...
25]);

a=addmf(a,'output',1,'Average Skill','trapmf',[20 ...
40 50 70]);

a=addmf(a,'output',1,'High Skill','trapmf',[65 75 ...
100 100]);
```

The Code

```
% The declaration for each rule

rule1 = [1 1 1 1 1];

rule2= [3 5 3 1 1];

rule3 = [2 3 2 1 1];

% A matrix to hold the rule arrays

ruleList = [rule1; rule2; rule3];

% Add the rules to the fis

a = addrule(a, ruleList)
```

The Code

```
% Print the rules to the command window

showrule(a)

% Declare your inputs here:

Input1 = 10

Input1 = 15

% evlafis returns an output

Output = evalfis(a,[Input1, Input2])
```

The Code

```
% The ruleview allows you to see the rule-base

ruleview(a)

% The subplots to visualise the system

subplot(4,1,1),plotmf(a, 'input', 1)

subplot(4,1,2),plotmf(a, 'input', 2)

subplot(4,1,4),plotmf(a, 'output', 1)
```

Functioning Fuzzy Inference System

Dr. Archie Singh **Khuman**

Institute of Artificial Intelligence · Games, Maths & Artificial Intelligence · GH6.57

De Montfort University, Gateway House, Leicester, United Kingdom, LE1 9BH

+44 (0)116 250 6251 | arjab.khuman | arjab singh khuman | archie khuman

"Be inspired or be inspirational - Either way, great things can be accomplished."