Review Article

FUZZY Inference System using Python

Prasenjit Nath*
B. Voc. IT Department, B.N.College, Dhubri, Assam, India

Global Journal of Infectious Disease (GJID)

Volume 1 Issue 1, 2020

Article Information
Received date : October 29, 2020
Published date: November 30, 2020

***Corresponding author**
B. Voc. IT Department, B.N.College, Dhubri, Assam, India.

Keywords
Fuzzy Logic; Infectious disease; CODIV-19

Abstract

Most common symptoms of COVID-19 are fever, dry cough, and tiredness but affects different people in different ways. The virus that causes COVID-19 is mainly transmitted through droplets generated when an infected person coughs, sneezes, or exhales. These droplets are too heavy to hang in the air, and quickly fall on floors or surfaces. You can be infected by breathing in the virus if you are within close proximity of someone who has COVID-19, or by touching a contaminated surface and then your eyes, nose or mouth.

This paper is about classification of fuzzy logic application in an infectious disease like COVID-19. Fuzzy logic methods are vastly used for diagnosis of diseases and the key fuzzy logic methods used for the infectious diseases are the fuzzy inference system, rule- based fuzzy logic. This paper using python programming language and developed a Fuzzy logic based expert system to indentified possible COVID-19 cases base on symptoms. Python offers an amazingly powerful and free open-source alternative to traditional techniques and applications. Python have libraries for data analysis and visualization like Pandas, numpy etc. as well as for data visualization like matplotlib.

Introduction

Corona viruses are a family of viruses that can cause illness such as the common cold, Severe Acute Respiratory Syndrome (SARS). The disease it causes is called corona virus disease 2019 (COVID-19). The virus that causes COVID-19 spreads easily among people. So early detection is necessary only then we can stop farther spreading.

This article discuss about the construction of a possible control application using a fuzzy logic in python programming language as well as build a multi-input/output fuzzy inference system . For this purpose I am going to use python_spyder, which is open source software and a powerful Python IDE with advanced editing, interactive testing, debugging and introspection features.

System Architecture

The diagram below illustrates the structure of the application. The design is based on several considerations on Fuzzy Inference Systems as given below:

1. A Fuzzy Inference system will require input and output variables and a collection of fuzzy rules.
2. Both input and output variables will contain a collection of fuzzy sets if the Fuzzy Inference system is of Mamdani technique. In this technique, the precursor, as well as the subsequent section comprises fuzzy statements that reveal the value of the variable.

Input and output variables are very similar, but they are used differently by fuzzy rules. During execution, input variables use the input values to the system to fuzzify their sets, that is they determine the degree of belonging of that input value to all of the fuzzy sets of the variable. Each rule contributes to some extent to the output variables. The totality of this contribution will determine the output of the system (Figure 1).

Fuzzy rules have the structure of the form as follows:

if { antecedent clauses } **then** { consequent clauses }

Therefore, a rule will contain several clauses of antecedent type as well as some clauses of consequent type, which will be of the form as given below: {Variable name} **is** {set name}

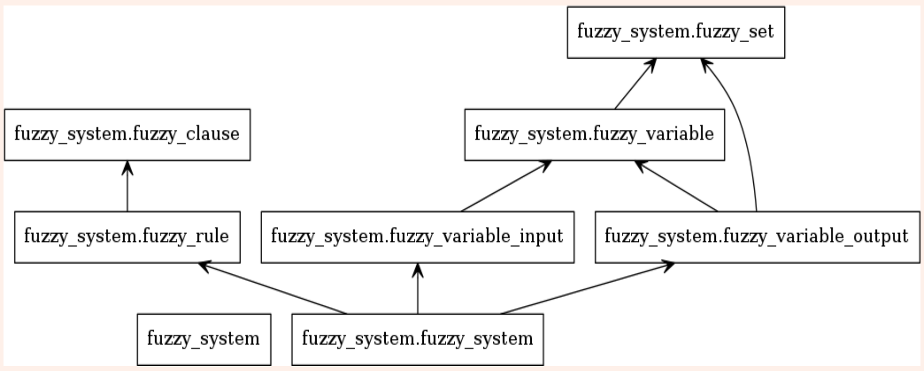

fuzzy_system.fuzzy_set
fuzzy_system.fuzzy_clause
fuzzy_system.fuzzy_variable
fuzzy_system.fuzzy_rule
fuzzy_system.fuzzy_variable_input
fuzzy_system.fuzzy_variable_output
fuzzy_system
fuzzy_system.fuzzy_system

Figure 1: System diagram.

FuzzySet Class

To initiate a FuzzySet the following parameters are requires:

1. Name - the name of the set.
2. Minimum value - the minimum value of the set.
3. Maximum value - the maximum value of the set.
4. Resolution - the number of steps between the minimum and maximum value.

So two numpy arrays are required to represent a fuzzy set, among them one will use to hold the domain values and another to hold the degree-of-membership values. Initially, all degree-0-of-membership values will be all set to zero.

First, generate universal variables fever and cough on subjective ranges [0, 10] as follows:

```
# Generate universe variables
# fever and cough on subjective ranges [0, 10]
# Tiredness has a range of [0, 25] in units of percentage points x_fever = np.arange (0,
11, 1)
x_cough = np.arange (0, 11, 1)
x_tiredness = np.arange (0, 26, 1)
```

Then generate Fuzzy membership functions as follows

```
x_cough = np.arange (0, 11, 1)
x_tiredness = np.arange (0, 26, 1)
```

Then generate Fuzzy membership functions as follows:

```
x_cough = np.arange (0, 11, 1)
x_tiredness = np.arange (0, 26, 1)
```

Then generate Fuzzy membership functions as follows:

```
x_cough = np.arange (0, 11, 1)
x_tiredness = np.arange (0, 26, 1)
```

Then generate Fuzzy membership functions as follows:

```
# Generate fuzzy membership functions fever_lo = fuzz.trimf(x_fever, [0, 0, 3])
fever_md = fuzz.trimf(x_fever, [0, 3, 7])
fever_hi = fuzz.trimf(x_fever, [5, 10, 10])
cough_lo = fuzz.trimf(x_cough, [0, 0, 3])
cough_md = fuzz.trimf(x_cough, [0, 3, 7])
cough_hi = fuzz.trimf(x_cough, [5, 10, 10])
tiredness_lo = fuzz.trimf(x_tiredness, [0, 0, 10])
tiredness_md = fuzz.trimf(x_tiredness, [0, 10, 20])
tiredness_hi = fuzz.trimf(x_tiredness, [13, 25, 25])
```

Now need to visualize these universes and membership functions as follows

```
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9)) ax0.plot(x_fever, fever_lo,
'b', linewidth=1.5, label='Negative') ax0.plot(x_fever, fever_md, 'g', linewidth=1.5,
label='Possible Case') ax0.plot(x_fever, fever_hi, 'r', linewidth=1.5, label='Positive Case')
ax0.set_title('Base on Fever')
ax0.legend()

ax1.plot(x_cough, cough_lo, 'b', linewidth=1.5, label='Negative') ax1.plot(x_cough,
cough_md, 'g', linewidth=1.5, label='Possible Case') ax1.plot(x_cough, cough_hi, 'r',
linewidth=1.5, label='Positive case') ax1.set_title('Base on cough')
ax1.legend()

ax2.plot(x_tiredness, tiredness_lo, 'b', linewidth=1.5, label='Negative') ax2.plot(x_
tiredness, tiredness_md, 'g', linewidth=1.5, label='Possible case') ax2.plot(x_tiredness,
tiredness_hi, 'r', linewidth=1.5, label='Possible case') ax2.set_title('Base on tired fill')
ax2.legend()

# Turn off top/right axes for ax in (ax0, ax1, ax2):
ax.spines['top'].set_visible(False) ax.spines['right'].set_visible(False) ax.get_xaxis().
tick_bottom() ax.get_yaxis().tick_left()
plt.tight_layout()
```

We need the activation of our fuzzy membership functions at these values

```
fever_level_lo = fuzz.interp_membership(x_fever, fever_lo, 5.5) fever_level_md
= fuzz.interp_membership(x_fever, fever_md, 6.5) fever_level_hi = fuzz.interp_
membership(x_fever, fever_hi, 9.5)

cough_level_lo = fuzz.interp_membership(x_cough, cough_lo, 5.0) cough_level_md
= fuzz.interp_membership(x_cough, cough_md, 6.5) cough_level_hi = fuzz.interp_
membership(x_cough, cough_hi, 9.5)
```

Now we take our rules and apply them. Rule 1 concerns bad food OR service. The OR operator means we take the maximum of these two.

```
active_rule1 = np.fmax(fever_level_lo, cough_level_lo)
```

Now we apply this by clipping the top off the corresponding output, membership function with "np.fmin"

```
tiredness_activation_lo = np.fmin(active_rule1, tiredness_lo)
```

For rule 2 we connect acceptable service to medium tiredness.

```
tiredness_activation_md = np.fmin(cough_level_md, tiredness_md)
```

For rule 3 we connect high service OR high food with high tiredness

```
active_rule3 = np.fmax(fever_level_hi, cough_level_hi) tiredness_activation_hi =
np.fmin(active_rule3, tiredness_hi)
tip0 = np.zeros_like(x_tiredness)
```

For Visualize the result

```
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.fill_between(x_tiredness, tip0, tiredness_activation_lo, facecolor='b', alpha=0.7)
ax0.plot (x_tiredness, tiredness_lo, 'b', linewidth=0.5, linestyle='--',) ax0.fill_between(x_
tiredness, tip0, tiredness_activation_md, facecolor='g', alpha=0.7)
ax0.plot(x_tiredness, tiredness_md, 'g', linewidth=0.5, linestyle='--') ax0.fill_between(x_
tiredness, tip0, tiredness_activation_hi, facecolor='r', alpha=0.7)
ax0.plot(x_tiredness, tiredness_hi, 'r', linewidth=0.5, linestyle='--') ax0.set_title('Output
membership activity')
```

Then Aggregate all three output membership functions together

```
aggregated = np.fmax(tiredness_activation_lo,
np.fmax(tiredness_activation_md, tiredness_activation_hi))
```

Calculate defuzzified result

```
tiredness = fuzz.defuzz(x_tiredness, aggregated, 'centroid')

tiredness_activation = fuzz.interp_membership(x_tiredness, aggregated, tiredness)

# Visualize the result
fig, ax0 = plt.subplots(figsize=(8, 3))
ax0.plot(x_tiredness, tiredness_lo, 'b', linewidth=0.5, linestyle='--', ) ax0.plot(x_
tiredness, tiredness_md, 'g', linewidth=0.5, linestyle='--') ax0.plot(x_tiredness,
tiredness_hi, 'r', linewidth=0.5, linestyle='--') ax0.fill_between(x_tiredness, tip0,
aggregated, facecolor='Orange', alpha=0.7) ax0.plot([tiredness, tiredness], [0, tiredness_
activation], 'k', linewidth=1.5, alpha=0.9)
ax0.set_title('Aggregated COVID-19 Chance and result (line)')

# Turn off top/right axes for ax in (ax0,):
ax.spines['top'].set_visible(False) ax.spines['right'].set_visible(False) ax.get_xaxis().
tick_bottom() ax.get_yaxis().tick_left()
```

plt.tight_layout()

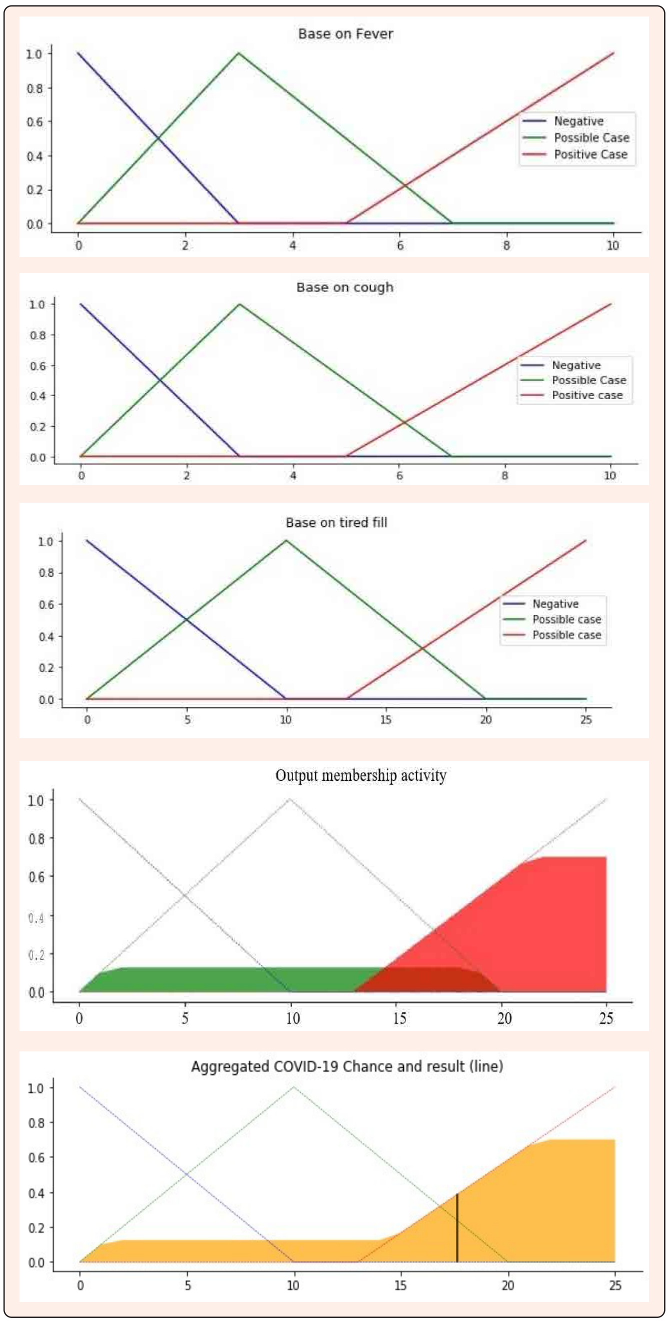
Rule aggregation

With the activity of each output membership function known, all output membership functions must be combined. This is typically done using a maximum operator. This step is also known as aggregation.

Defuzzification

Finally, to get a real world answer, we return to crisp logic from the world of fuzzy membership functions. For the purposes of this example the centroid method will be used.

The result is a chance of COVID-19 is 20.2%.



Conclusion

We can use both Mamdani and Sugeno both are fuzzy rule-based systems. But for this paper Mamdani is used to generate rule-based to identify COVID-19 positive cases. Changing the rules as required, we can developed the Fuzzy Inference system more effective ways which will give us more accurate result to detect COVID-19 positive cases. Risk factors for COVID-19 appear to include:

1. Close contact (within 6 feet, or 2 meter) with someone who has COVID-19.
2. Being cough or sneezed on by an infected person.

References

1. https://covid19.assam.gov.in/
2. https://towardsdatascience.com/fuzzy-inference-system-implementation-in-python- 8af88d1f0a6e
3. https://scikit-fuzzy.readthedocs.io/en/stable/auto_examples/ plot_tipping_problem.html