Artificial Intelligence

BCS – 7A

Lab Final

Submitted by:

Name: Eisha Nadeem

Roll No: SP22-BCS-025

Submitted to: Dr. Faisal Azam

COMSATS University Islamabad,

Wah Campus

Part A: Fuzzy Logic (Classification of patient risk)

Code:

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
```

[1] ✓ 13.3s

Step:1 Membership Function:

```
# Cholesterol value of according to roll_number (025)
value = 25 % 150 + 120
print("Cholesterol Level:", value)
```

[13] ✓ 0.0s

```
Cholesterol Level: 145
```

```
# Defining Membership function

def membership_func(x, a, b, c):
    if a < x < b:                  # Increasing membership from a to b
        return (x - a) / (b - a)  # Linear increase
    elif b <= x < c:
        return (c - x) / (c - b)  # Linear decrease
    elif x == b:                   # Full membership at point b
        return 1
    else:
        return 0
```

[10] ✓ 0.0s

```
Low Membership: 0.25
Borderline Membership: 0.17
High Membership: 0.00
```

Step:2 Calculating Membership Degrees :

```
x = 145  # cholesterol value

low = membership_func(x, 110, 130, 150)
borderline = membership_func(x, 140, 170, 200)
high = membership_func(x, 180, 220, 260)

print (f"Low Membership: {low:.2f}") # round-off to 2 digits
print(f"Borderline Membership: {borderline:.2f}")
print(f"High Membership: {high:.2f}")
```

[14] ✓ 0.0s

```
Low Membership: 0.25
Borderline Membership: 0.17
High Membership: 0.00
```

Step:3 Classification of Risk:

```
if high > borderline and high > low:
    risk = "High"
elif borderline > low:
    risk = "Medium"
else:
    risk = "Low"

print("Patient Risk Level:", risk)
```

[15] ✓ 0.0s

```
Patient Risk Level: Low
```

Step:4 Plotting the Membership Function:

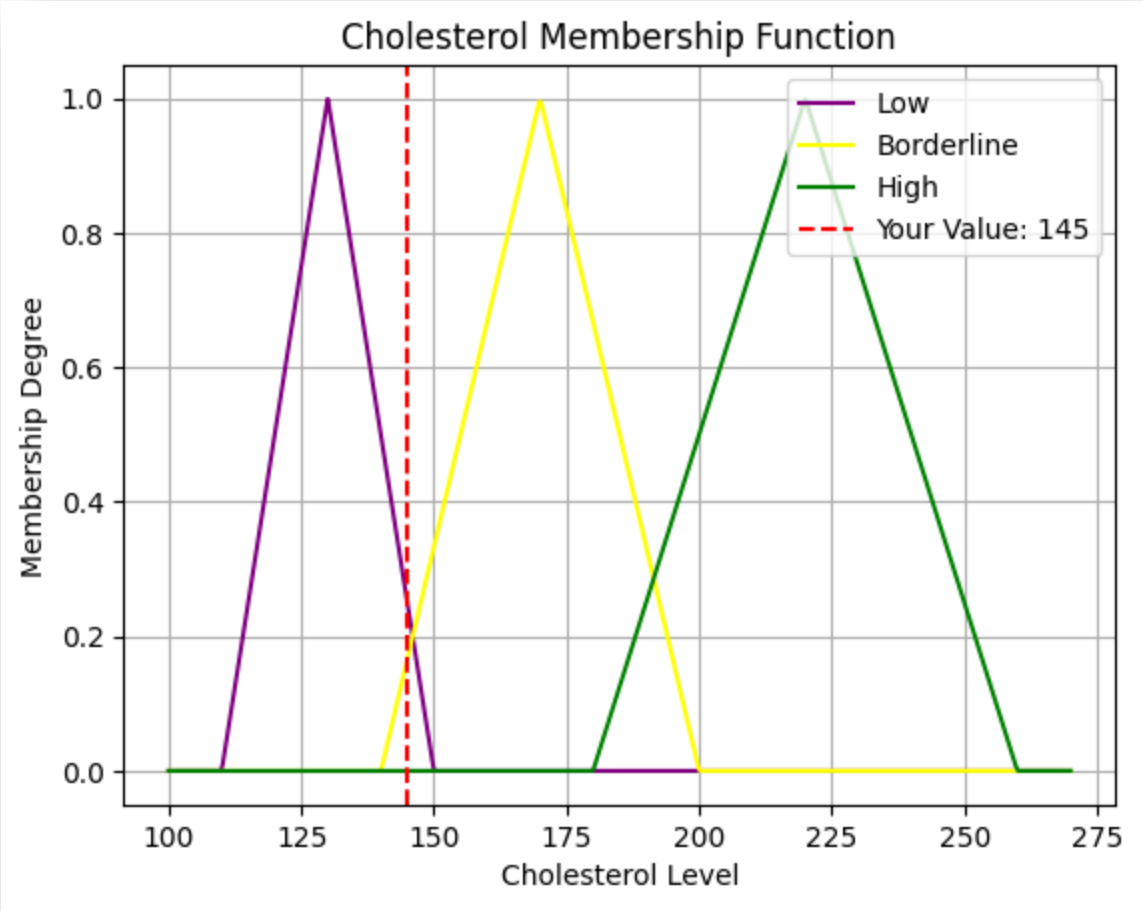
```
# It will Generate values from 100 to 270 with 500 points in between
x_vals = np.linspace(100, 270, 500)
# Calculating low membership values for each x
low_vals = [membership_func(x, 110, 130, 150) for x in x_vals]
# Calculating borderline and high membership values for each x
border_vals = [membership_func(x, 140, 170, 200) for x in x_vals]
# Calculating high membership values for each x
high_vals = [membership_func(x, 180, 220, 260) for x in x_vals]
```

[16] ✓ 0.0s

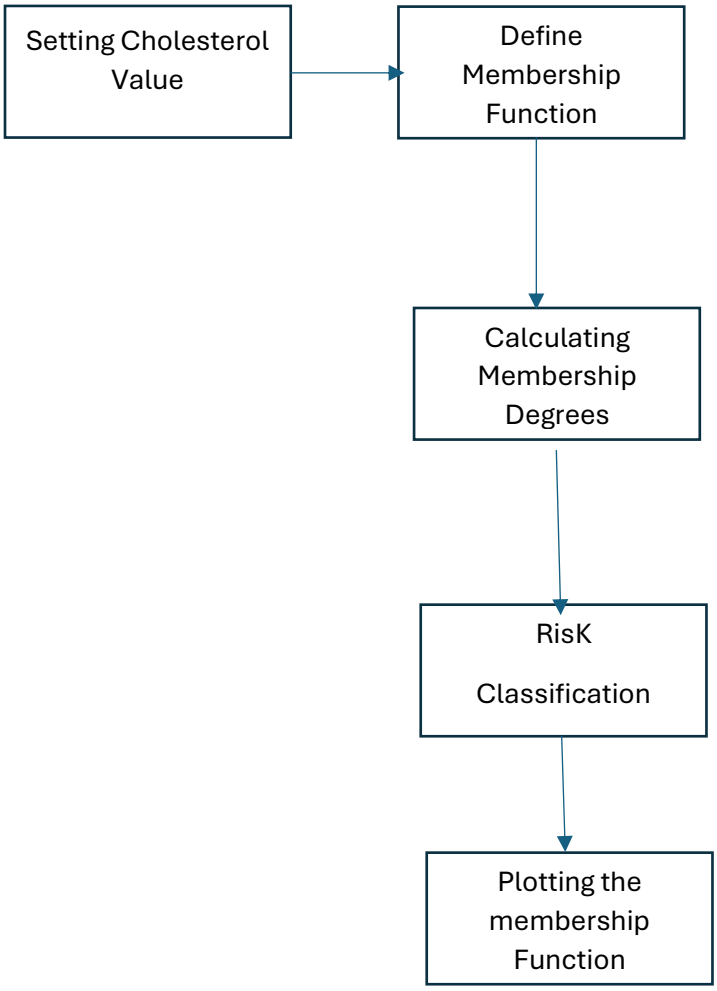
```
plt.plot(x_vals, low_vals, label="Low", color='Purple')
plt.plot(x_vals, border_vals, label="Borderline", color='Yellow')
plt.plot(x_vals, high_vals, label="High", color='green')

# Vertical line for the cholesterol value
plt.legend(loc='upper right')
plt.axvline(x, color='Red', linestyle='--', label=f"Your Value: {x}")
plt.title("Cholesterol Membership Function")
plt.xlabel("Cholesterol Level")
plt.ylabel("Membership Degree")
plt.grid(True)
plt.show()
```

[24] ✓ 0.5s


Cholesterol Membership Function
Low
Borderline
High
Your Value: 145
Membership Degree
1.0
0.8
0.6
0.4
0.2
0.0
100
125
150
175
200
225
250
275
Cholesterol Level

Flow Chart:



Part A uses fuzzy sets to classify risk using cholesterol level.

Part B: Linear Perceptron

Code:

Step:1 Initializing Values

```
X = np.array([[45, 130], [30, 120], [50, 140], [28, 110]]) # Input features
y = np.array([1, 0, 1, 0]) # Output labels (1: High, 0: Low)

# Initializing weights and bias as 0
weights = np.array([0.0, 0.0])  # w1, w2
bias = 0.0
learning_rate = 1
```

26] ✓ 0.0s

Step:2 Training for Epoch 1:

```
# Activation function

def step_function(x):
    return 1 if x >= 0 else 0 # Step function

for i in range(len(X)): # Iterating over samples
    x1, x2 = X[i]
    target = y[i]

    # Weighted sum and activation
    z = x1 * weights[0] + x2 * weights[1] + bias # Weighted sum
    output = step_function(z) # Activation function output
```

35]

```python
    # Calculating error
    error = target - output
    # Updating rule
    weights[0] += learning_rate * error * x1
    weights[1] += learning_rate * error * x2
    bias += learning_rate * error
    # Printing values
    print ("\n")
    print(f"Sample {i+1}:")
    print(f"  Predicted: {output}")
    print(f"  Actual: {target}")
    print(f"  Updated Weights: {weights}")
    print(f"  Bias: {bias}")
```

35] ✓ 0.0s

```
Sample 1:
  Predicted: 0
  Actual: 1
  Updated Weights: [169.  20.]
  Bias: -2.0


Sample 2:
  Predicted: 1
  Actual: 0
  Updated Weights: [ 139. -100.]
  Bias: -3.0
```

```
Sample 3:
  Predicted: 0
  Actual: 1
  Updated Weights: [189.  40.]
  Bias: -2.0


Sample 4:
  Predicted: 1
  Actual: 0
  Updated Weights: [161. -70.]
  Bias: -3.0
```

Flow Chart: In Part B we trained a perceptron using age and blood pressure for 1 epoch, updating weights manually.