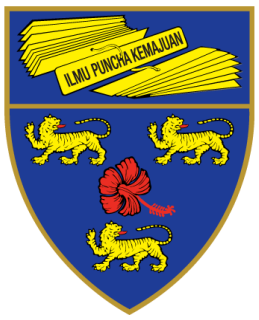

UNIVERSITI MALAYA

Faculty of Computer Science and Information Technology

WIX3001 SOFT COMPUTING

GROUP ASSIGNMENT 1: FUZZY LOGIC

SEMESTER 2 SESSION 2023/2024

LECTURER: DR LIEW WEI SHIUNG

NO	NAME	MATRIC NUMBER
1.	QUAH JUN CHUAN	22004851
2.	TER ZHEN HUANG	22004736
3.	CHENG YEE ERN	22004791
4.	RACHEL LIM	22052702

Table of Contents

1. Dataset

1.1 About Dataset

The dataset chosen for this project “Mice Protein Expression” is obtained from the UC Irvine Machine Learning Repository. In general, the dataset discusses the different protein modifications for mice. The dataset consists of the 77 expression levels of proteins (also known as protein modifications) that have produced detectable signals in the nuclear fraction of the cortex which made up the number of features of the dataset. For the number of mice, there are 38 control mice and 34 trisomic mice (mice diagnosed with Down syndrome), which makes up a total of 72 mice.

Based on the abstract of the experiment, 15 measurements were registered of each protein per sample (one sample here refers to one mouse). Therefore, for control mice, there are 38x15, or 570 measurements, and for trisomic mice, there are 34x15, or 510 measurements. The sum of 570 and 510 measurements totals 1080 measurements per protein, which is the 1080 data in the dataset. According to Higuera (2015), Each measurement can be considered as an independent sample/mouse.

There are 8 mice classes which are classified based on their features such as genotype, behavior and treatment. According to genotype, mice can be classified as controlled or trisomic. For the behavior column, some mice have been stimulated to learn (context-shock) while others have not (shock-context). Additionally, to assess the effect of the drug memantine in recovering the ability to learn in trisomic mice, some mice were injected with the drug while others were not.

The classes of mice and their descriptions are briefly discussed as follows. The descriptions helped in understanding the dataset for the development of a fuzzy system for mice classification.

- **Category of control mice:**
  1. c-CS-s: control mice, stimulated to learn, injected with saline
  2. c-CS-m: control mice, stimulated to learn, injected with memantine
  3. c-SC-s: control mice, not stimulated to learn, injected with saline
  4. c-SC-m: control mice, not stimulated to learn, injected with memantine
- **Category of trisomy mice:**
  5. t-CS-s: trisomy mice, stimulated to learn, injected with saline
  6. t-CS-m: trisomy mice, stimulated to learn, injected with memantine
  7. t-SC-s: trisomy mice, not stimulated to learn, injected with saline
  8. t-SC-m: trisomy mice, not stimulated to learn, injected with memantine

The aim of developing a fuzzy system using this dataset is to identify subsets of proteins that are discriminant between the classes.

1.2 Data Preprocessing

Based on diagram 1.1, we perform mean imputation for missing values in a dataset where each class label is iterated to calculate the mean values of features within that class. Subsequently, missing values within each class are replaced with the respective mean values

```
# Declare all class_labels' class
class_labels = ['c-CS-m', 'c-SC-m', 'c-CS-s', 'c-SC-s', 't-CS-m', 't-SC-m', 't-CS-s', 't-SC-s']

# Iterate over each class label
filled_dfs = []
for label in class_labels:
    # 1. Filter the DataFrame to include only rows where the 'class' is the current label
    class_data = df[df['class'] == label]

    # 2. Calculate the mean value for each feature in the filtered data
    mean_values_class = class_data.mean(numeric_only=True)

    # 3. Fill the missing values in each feature with the corresponding mean value
    filled_class_data = class_data.fillna(mean_values_class)

    # Append the filled data to the list
    filled_dfs.append(filled_class_data)

# Concatenate the filled data for all classes back together
filled_df = pd.concat(filled_dfs)
```

Diagram 1.1

The 'class' column is converted from strings representing class labels to integers using a mapping, with corresponding integer values ranging from 0 to 7. The number of instances of each class is then displayed in a table format.

```
# Convert class from string to integer
filled_df['class'] = df['class'].replace(['c-CS-m', 'c-SC-m', 'c-CS-s', 'c-SC-s', 't-CS-m', 't-SC-m', 't-CS-s', 't-SC-s'], range(8))

# Count occurrences of each class label
class_counts = filled_df['class'].value_counts().reset_index()
class_counts.columns = ['Class', 'Count']

# Print the table
print(class_counts)
```

Diagram 1.2

1.3 Data Visualisation for Feature Selection

An ANOVA test is conducted on all the numerical columns to select the top 10 features with the smallest p-value among all the initial features to become the input variables. Based on the result of diagram 1.4, the selected features are 'SOD1_N', 'CaNA_N', 'Ubiquitin_N', 'ARC_N', 'pS6_N', 'P38_N', 'S6_N', 'pPKCAB_N', 'pGSK3B_N', and 'pERK_N'.

```python
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Rename the 'class' column to avoid conflict with Python keywords
df.rename(columns={'class': 'class_label'}, inplace=True)

# Select numerical columns excluding non-predictive columns
numerical_columns = [col for col in df.columns if col not in ['MouseID', 'Genotype', 'Treatment', 'Behavior', 'class_label']]

# Dictionary to store p-values for each numerical variable
p_values = {}

# Perform ANOVA for each numerical variable
for numerical_variable in numerical_columns:
    # Define the formula for ANOVA
    formula = f"{numerical_variable} ~ C(class_label)"
    # Fit the model
    model = ols(formula, data=df).fit()
    # Perform ANOVA and extract p-value
    anova_table = sm.stats.anova_lm(model, typ=2)
    p_value = anova_table.loc['C(class_label)', 'PR(>F)']
    p_values[numerical_variable] = p_value

# Sort the p-values in ascending order
sorted_p_values = sorted(p_values.items(), key=lambda x: x[1])

# Print the top 10 features with the smallest p-values
print("Top 10 features with the smallest p-values:")
for feature, p_value in sorted_p_values[:10]:
    print(f"{feature}: {p_value}")
```

Diagram 1.3

```
Top 10 features with the smallest p-values:
SOD1_N: 1.0367238009076876e-243
CaNA_N: 5.372624266691846e-215
Ubiquitin_N: 1.6289861770848514e-162
ARC_N: 1.728613890605033e-145
pS6_N: 1.728613890605033e-145
P38_N: 8.03301835718573e-142
S6_N: 1.2503547646624504e-110
pPKCAB_N: 1.2819936718030727e-108
pGSK3B_N: 7.114063353273567e-100
pERK_N: 7.809330475505706e-99
```

Diagram 1.4

A boxplot is plotted for each selected feature separately for each class in our dataset. By examining boxplots, we can identify features that exhibit significant variation and distribution across different classes which can be shown in Diagram 1.5. Understanding the distribution of features across different classes can aid in interpreting the fuzzy logic model's output. By visualising feature distributions, we gain insights into how different features contribute to the classification decision made by the fuzzy logic system.

```python
# Define the top 10 features with the smallest p-values
top_features = ['SOD1_N', 'CaNA_N', 'Ubiquitin_N', 'ARC_N', 'pS6_N', 'P38_N', 'S6_N', 'pPKCAB_N',
                'pGSK3B_N', 'pERK_N']

# Plot boxplots for each feature separately for each class
num_plots = len(top_features)
num_rows = 2
num_cols = 5

fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 10))

for i, feature in enumerate(top_features):
    row = i // num_cols
    col = i % num_cols
    sns.boxplot(x='class', y=feature, data=filled_df, ax=axes[row, col])
    axes[row, col].set_title(f'Boxplot of {feature} by Class')
    axes[row, col].set_xlabel('Class')
    axes[row, col].set_ylabel('Value')
    axes[row, col].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

Diagram 1.5

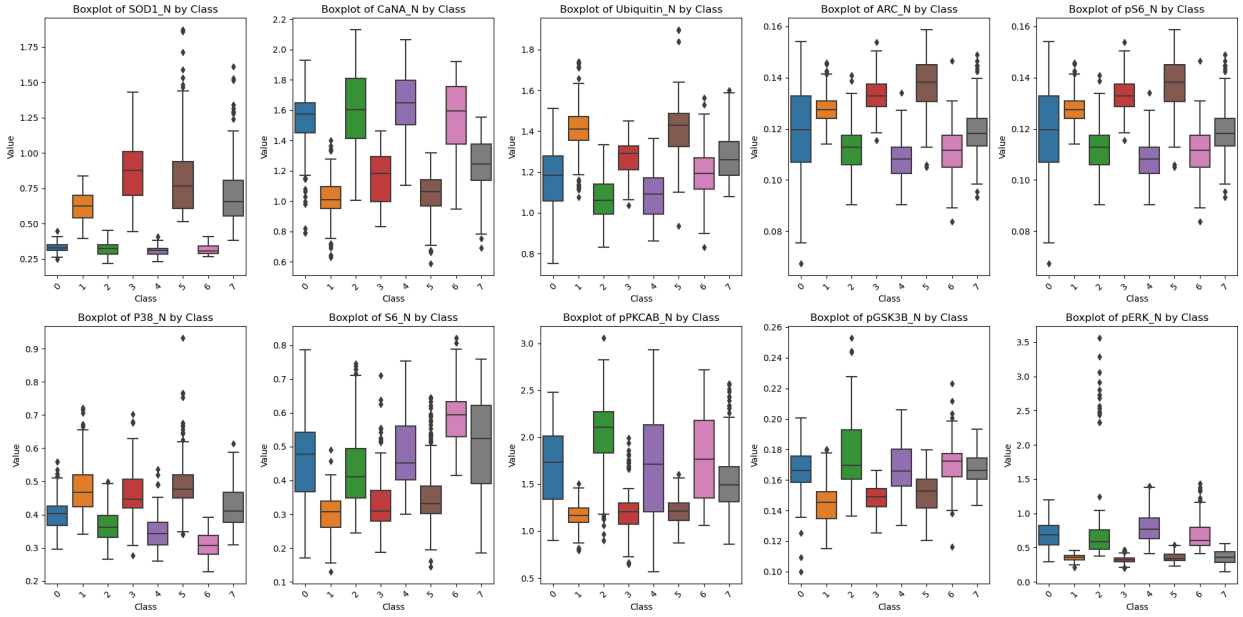

Diagram 1.6

2. Fuzzy System

2.1 Input Fuzzification

Based on diagram 2.1, our fuzzy logic input variables (SOD1_N, CaNA_N, etc.) and an output variable (class_label) are defined using the ctrl.Antecedent and ctrl.Consequent functions. Each input variable is defined over a specified range based on visualising the lower and upper boundaries in the boxplots. These variables represent linguistic input and output categories for a fuzzy logic system.

```
# Define input fuzzification variables on the top 10 feature selected
SOD1_N = ctrl.Antecedent(np.arange(0.25, 2.01, 0.1), 'SOD1_N')
CaNA_N = ctrl.Antecedent(np.arange(0.5, 2.21, 0.1), 'CaNA_N')
Ubiquitin_N = ctrl.Antecedent(np.arange(0.5, 2.01, 0.1), 'Ubiquitin_N')
ARC_N = ctrl.Antecedent(np.arange(0.02, 0.17, 0.01), 'ARC_N')
pS6_N = ctrl.Antecedent(np.arange(0.02, 0.17, 0.01), 'pS6_N')
P38_N = ctrl.Antecedent(np.arange(0.2, 1.01, 0.1), 'P38_N')
S6_N = ctrl.Antecedent(np.arange(0.1, 0.91, 0.1), 'S6_N')
pPKCAB_N = ctrl.Antecedent(np.arange(0.5, 3.01, 0.1), 'pPKCAB_N')
pGSK3B_N = ctrl.Antecedent(np.arange(0.1, 0.27, 0.01), 'pGSK3B_N')
pERK_N = ctrl.Antecedent(np.arange(0.1, 3.51, 0.1), 'pERK_N')

# Define output fuzzification variable
class_label = ctrl.Consequent(np.arange(0, 8, 1), 'class_label')
```

Diagram 2.1

Diagram 2.2 presents code for assigning membership functions to input fuzzification variables based on manually declared values stored in arrays. Each variable is allocated into three degree membership functions: 'low', 'medium', and 'high', utilizing different membership function types. Specifically, Z membership functions are employed for 'low', triangle membership functions for 'medium', and S membership functions for 'high'.

```
mf_val = np.array([
    0.25, 0.50,
    0.35, 0.60, 0.80,
    0.4, 1.0,

    0.7, 1.3,
    0.8, 1.1, 1.5,
    1.0, 2.15,

    0.7, 1.5,
    1.05, 1.3, 1.6,
    1.1, 1.65,

    0.075,0.13,
    0.075,0.110,0.155,
    0.11,0.16,

    0.075,0.13,
    0.075,0.110,0.155,
    0.11,0.16,

    0.1, 0.5,
    0.3, 0.45, 0.6,
    0.32, 0.68,

    0.15, 0.52,
    0.17, 0.45, 0.79,
    0.2, 0.79,

    0.75, 1.3,
    0.6, 1.7, 2.85,
    1.2, 2.8,

    0.11,0.18,
    0.135,0.18,0.21,
    0.135,0.23,

    0.2, 0.5,
    0.3, 0.8, 1.2,
    0.4 ,1.4,
])
```

```
# Define membership function by using value in array
SOD1_N['low'] = fuzz.zmf(SOD1_N.universe, mf_val[0], mf_val[1])
SOD1_N['medium'] = fuzz.trimf(SOD1_N.universe, [mf_val[2], mf_val[3], mf_val[4]])
SOD1_N['high'] = fuzz.smf(SOD1_N.universe, mf_val[5], mf_val[6])

CaNA_N['low'] = fuzz.zmf(CaNA_N.universe, mf_val[7], mf_val[8])
CaNA_N['medium'] = fuzz.trimf(CaNA_N.universe, [mf_val[9], mf_val[10], mf_val[11]])
CaNA_N['high'] = fuzz.smf(CaNA_N.universe, mf_val[12], mf_val[13])

Ubiquitin_N['low'] = fuzz.zmf(Ubiquitin_N.universe, mf_val[14], mf_val[15])
Ubiquitin_N['medium'] = fuzz.trimf(Ubiquitin_N.universe, [mf_val[16], mf_val[17], mf_val[18]])
Ubiquitin_N['high'] = fuzz.smf(Ubiquitin_N.universe, mf_val[19], mf_val[20])

ARC_N['low'] = fuzz.zmf(ARC_N.universe, mf_val[21], mf_val[22])
ARC_N['medium'] = fuzz.trimf(ARC_N.universe, [mf_val[23], mf_val[24], mf_val[25]])
ARC_N['high'] = fuzz.smf(ARC_N.universe, mf_val[26], mf_val[27])

pS6_N['low'] = fuzz.zmf(pS6_N.universe, mf_val[28], mf_val[29])
pS6_N['medium'] = fuzz.trimf(pS6_N.universe, [mf_val[30], mf_val[31], mf_val[32]])
pS6_N['high'] = fuzz.smf(pS6_N.universe, mf_val[33], mf_val[34])

P38_N['low'] = fuzz.zmf(P38_N.universe, mf_val[35], mf_val[36])
P38_N['medium'] = fuzz.trimf(P38_N.universe, [mf_val[37], mf_val[38], mf_val[39]])
P38_N['high'] = fuzz.smf(P38_N.universe, mf_val[40], mf_val[41])

S6_N['low'] = fuzz.zmf(S6_N.universe, mf_val[42], mf_val[43])
S6_N['medium'] = fuzz.trimf(S6_N.universe, [mf_val[44], mf_val[45], mf_val[46]])
S6_N['high'] = fuzz.smf(S6_N.universe, mf_val[47], mf_val[48])

pPKCAB_N['low'] = fuzz.zmf(pPKCAB_N.universe, mf_val[49], mf_val[50])
pPKCAB_N['medium'] = fuzz.trimf(pPKCAB_N.universe, [mf_val[51], mf_val[52], mf_val[53]])
pPKCAB_N['high'] = fuzz.smf(pPKCAB_N.universe, mf_val[54], mf_val[55])

pGSK3B_N['low'] = fuzz.zmf(pGSK3B_N.universe, mf_val[56], mf_val[57])
pGSK3B_N['medium'] = fuzz.trimf(pGSK3B_N.universe, [mf_val[58], mf_val[59], mf_val[60]])
pGSK3B_N['high'] = fuzz.smf(pGSK3B_N.universe, mf_val[61], mf_val[62])

pERK_N['low'] = fuzz.zmf(pERK_N.universe, mf_val[63], mf_val[64])
pERK_N['medium'] = fuzz.trimf(pERK_N.universe, [mf_val[65], mf_val[66], mf_val[67]])
pERK_N['high'] = fuzz.smf(pERK_N.universe, mf_val[68], mf_val[69])
```

Diagram 2.2

Membership functions for each discrete value of the output variable (Mice class) are defined using gaussian membership functions (fuzz.gaussmf).

```
# Define output using gaussian membership function
class_label['c-CS-m'] = fuzz.gaussmf(class_label.universe, 0, 0.5)
class_label['c-SC-m'] = fuzz.gaussmf(class_label.universe, 1, 0.5)
class_label['c-CS-s'] = fuzz.gaussmf(class_label.universe, 2, 0.5)
class_label['c-SC-s'] = fuzz.gaussmf(class_label.universe, 3, 0.5)
class_label['t-CS-m'] = fuzz.gaussmf(class_label.universe, 4, 0.5)
class_label['t-SC-m'] = fuzz.gaussmf(class_label.universe, 5, 0.5)
class_label['t-CS-s'] = fuzz.gaussmf(class_label.universe, 6, 0.5)
class_label['t-SC-s'] = fuzz.gaussmf(class_label.universe, 7, 0.5)

# Display membership functions for the output variable
class_label.view()
```

Diagram 2.3

2.2 Fuzzy Rule Base

Diagram 2.4 shows the code which defines eight fuzzy rules for the fuzzy logic system. Each rule consists of an antecedent (input conditions) and a consequent (output action).

```python
# Define fuzzy rule
rule1 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] & CaNA_N['high'] | Ubiquitin_N['medium'] |
         S6_N['medium'] | pPKCAB_N['medium'] |
         pERK_N['medium'])
    ),
    consequent=class_label['c-CS-m']
)

rule2 = ctrl.Rule(
    antecedent=(
        (pS6_N['medium'] & P38_N['high'] |
         S6_N['low'] & pPKCAB_N['low'] & pGSK3B_N['low'] )
    ),
    consequent=class_label['c-SC-m']
)

rule3 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | CaNA_N['high'] |
         S6_N['medium'] & pPKCAB_N['high'] & pGSK3B_N['high'] |
         pERK_N['medium'])
    ),
    consequent=class_label['c-CS-s']
)

rule4 = ctrl.Rule(
    antecedent=(
        (SOD1_N['high'] & Ubiquitin_N['medium'] &
         ARC_N['high'] & P38_N['high'] |
         pGSK3B_N['low'] & pERK_N['low'])
    ),
    consequent=class_label['c-SC-s']
)
```

```python
rule5 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | pS6_N['low'] |
         P38_N['low'] | S6_N['medium'] &
         pPKCAB_N['medium'] & pERK_N['high'])
    ),
    consequent=class_label['t-CS-m']
)

rule6 = ctrl.Rule(
    antecedent=(
        (Ubiquitin_N['high'] & pS6_N['high'] |
         S6_N['low'] | pPKCAB_N['low'] |
         pERK_N['low'])
    ),
    consequent=class_label['t-SC-m']
)

rule7 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | CaNA_N['high'] |
         P38_N['low'] |S6_N['high'] & pPKCAB_N['medium'] &
         pERK_N['medium'])
    ),
    consequent=class_label['t-CS-s']
)

rule8 = ctrl.Rule(
    antecedent=(
        (SOD1_N['medium'] & CaNA_N['medium'] & Ubiquitin_N['medium'] &
         S6_N['high'] | pPKCAB_N['medium'] | pGSK3B_N['medium'] |
         pERK_N['low'])
    ),
    consequent=class_label['t-SC-s']
)
```

Diagram 2.4

A control system is created and a control system simulation is defined using the defined fuzzy rules (rule1 to rule8) to perform fuzzy inference.

```python
# Create control system
mice_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8])

# Define control system simulation
mice_ctrl_sim = ctrl.ControlSystemSimulation(mice_ctrl)
```

Diagram 2.5

2.3 Fuzzy Inference

Finally, we perform fuzzy inference shown in diagram 2.6 for each input sample in the dataset X. It computes the predicted class labels based on the predefined rules and stores both the predicted and actual class labels for each sample in a dictionary called results. This dictionary contains the results of the fuzzy inference process, including predicted and actual class labels for each input sample.

```python
results = {
    'prediction1_A': [],
    'actual_A': [],
    'prediction2_A': []
}

# Assuming X contains your input features and y contains the mice class
for i in range(len(X)):
    input_values = {
        'SOD1_N': X.iloc[i]['SOD1_N'],
        'CaNA_N': X.iloc[i]['CaNA_N'],
        'Ubiquitin_N': X.iloc[i]['Ubiquitin_N'],
        'ARC_N': X.iloc[i]['ARC_N'],
        'pS6_N': X.iloc[i]['pS6_N'],
        'P38_N': X.iloc[i]['P38_N'],
        'S6_N': X.iloc[i]['S6_N'],
        'pPKCAB_N': X.iloc[i]['pPKCAB_N'],
        'pGSK3B_N': X.iloc[i]['pGSK3B_N'],
        'pERK_N': X.iloc[i]['pERK_N']
    }

    output_value = y.iloc[i]  # Assuming y contains the mice class

    # Compute output
    for key, value in input_values.items():
        mice_ctrl_sim.input[key] = value

    mice_ctrl_sim.compute()
    results['actual_A']+=[y.iloc[i]]
    results['prediction1_A'].append(mice_ctrl_sim.output['class_label'])
    results['prediction2_A']+=[mice_ctrl_sim.output['class_label'].astype(int)]
```

Diagram 2.6

The result of the fuzzy inference is displayed in the tabular format shown in diagram 2.7 and the output membership function is shown in diagram 2.8.

Output:

	Predicted Mice class	Actual Mice class
0	3	0
1	3	0
2	3	0
3	3	0
4	3	0
...	...	...
1075	4	7
1076	3	7
1077	3	7
1078	3	7
1079	3	7

[1080 rows x 2 columns]

Diagram 2.7

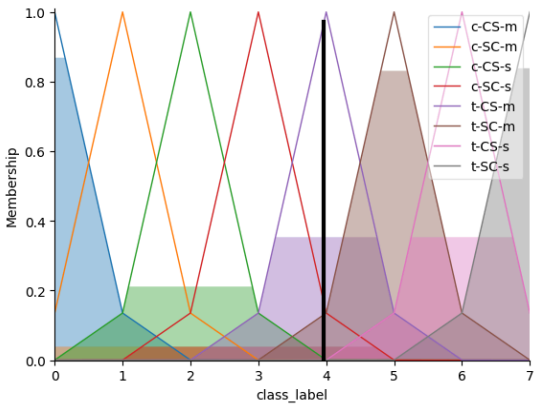

Diagram 2.8

The model is then evaluated using Mean Square Error and Accuracy.

```python
mse_A = mean_squared_error(results['actual_A'], results['prediction2_A'])
print("Mean Square Error: " + str(mse_A))


acc_A = accuracy_score(results['actual_A'],results['prediction2_A'])
print("Accuracy: "+str(acc_A))
```

✓ 0.0s

```
Mean Square Error: 5.309259259259259
Accuracy: 0.12314814814814815
```

Diagram 2.9

2.4 Defuzzification

We performed centroid defuzzification shown in Diagram 2.9, which is one of the methods used to convert fuzzy sets into a crisp value that acts as an output for the fuzzy logic systems. It calculates a single crisp value that represents the "center of gravity" of the fuzzy set by taking the membership function values from the 'prediction' lists as input and performing centroid defuzzification.

```python
# Define centroid deffuzification calculation
def centroid_defuzzification(mf_values):
    numerator = sum(mf_values)
    denominator = len(mf_values)
    return numerator / denominator if denominator != 0 else 0

# Extract membership function values from 'prediction' and 'prediction2'
mf_values_prediction = results['prediction1_A']


# Perform defuzzification
crisp_value_prediction_A = centroid_defuzzification(mf_values_prediction)


# Print defuzzified values
print("\nDefuzzified Values:")
print("Crisp Value (Prediction):", crisp_value_prediction_A)
```

✓ 0.0s

```
Defuzzified Values:
Crisp Value (Prediction): 3.6515398862839694
```

Diagram 2.10

3. Fine-Tuning

We fine-tune the fuzzy system by assigning different membership values to the same membership function for each degree, which is a triangle membership function. The output variable is also defined using a triangle membership function. We then define a set of different fuzzy rules for the fine-tuned model. The same fuzzy inference is applied and the performance of the model is compared based on evaluation metrics (MSE and accuracy).

3.1 Input Fuzzification

```
# Define value for triangle membership function
mf_val = np.array([
    0.25, 0.94, 1.26,
    0.94, 1.26, 1.88,
    1.26, 1.88, 2.00,

    0.5, 0.9, 1.3,
    0.9, 1.3, 1.7,
    1.3, 1.7, 2.2,

    0.5, 0.8, 1.2,
    0.8, 1.2, 1.5,
    1.2, 1.5, 2.0,

    0.02,0.08,0.11,
    0.08,0.11,0.13,
    0.11,0.13,0.16,

    0.02,0.06,0.09,
    0.06,0.09,0.12,
    0.09,0.12,0.16,

    0.2, 0.4, 0.6,
    0.4, 0.6, 0.8,
    0.6, 0.8, 1.0,

    0.1, 0.2, 0.4,
    0.2, 0.4, 0.6,
    0.4, 0.6, 0.9,

    0.5, 1.3, 1.9,
    1.3, 1.9, 2.3,
    1.9, 2.3, 3.0,

    0.10,0.14,0.19,
    0.14,0.19,0.23,
    0.19,0.23,0.26,

    0.1, 0.8, 1.4,
    0.8, 1.4, 2.6,
    1.4 ,2.6, 3.5,
])
```

```
# Define membership function
SOD1_N['low'] = fuzz.trimf(SOD1_N.universe, [mf_val[0], mf_val[1], mf_val[2]])
SOD1_N['medium'] = fuzz.trimf(SOD1_N.universe, [mf_val[3], mf_val[4], mf_val[5]])
SOD1_N['high'] = fuzz.trimf(SOD1_N.universe, [mf_val[6], mf_val[7], mf_val[8]])

CaNA_N['low'] = fuzz.trimf(CaNA_N.universe, [mf_val[9], mf_val[10], mf_val[11]])
CaNA_N['medium'] = fuzz.trimf(CaNA_N.universe, [mf_val[12], mf_val[13], mf_val[14]])
CaNA_N['high'] = fuzz.trimf(CaNA_N.universe, [mf_val[15], mf_val[16], mf_val[17]])

Ubiquitin_N['low'] = fuzz.trimf(Ubiquitin_N.universe, [mf_val[18], mf_val[19], mf_val[20]])
Ubiquitin_N['medium'] = fuzz.trimf(Ubiquitin_N.universe, [mf_val[21], mf_val[22], mf_val[23]])
Ubiquitin_N['high'] = fuzz.trimf(Ubiquitin_N.universe, [mf_val[24], mf_val[25], mf_val[26]])

ARC_N['low'] = fuzz.trimf(ARC_N.universe, [mf_val[27], mf_val[28], mf_val[29]])
ARC_N['medium'] = fuzz.trimf(ARC_N.universe, [mf_val[30], mf_val[31], mf_val[32]])
ARC_N['high'] = fuzz.trimf(ARC_N.universe, [mf_val[33], mf_val[34], mf_val[35]])

pS6_N['low'] = fuzz.trimf(pS6_N.universe, [mf_val[36], mf_val[37], mf_val[38]])
pS6_N['medium'] = fuzz.trimf(pS6_N.universe, [mf_val[39], mf_val[40], mf_val[41]])
pS6_N['high'] = fuzz.trimf(pS6_N.universe, [mf_val[42], mf_val[43], mf_val[44]])

P38_N['low'] = fuzz.trimf(P38_N.universe, [mf_val[45], mf_val[46], mf_val[47]])
P38_N['medium'] = fuzz.trimf(P38_N.universe, [mf_val[48], mf_val[49], mf_val[50]])
P38_N['high'] = fuzz.trimf(P38_N.universe, [mf_val[51], mf_val[52], mf_val[53]])

S6_N['low'] = fuzz.trimf(S6_N.universe, [mf_val[54], mf_val[55], mf_val[56]])
S6_N['medium'] = fuzz.trimf(S6_N.universe, [mf_val[57], mf_val[58], mf_val[59]])
S6_N['high'] = fuzz.trimf(S6_N.universe, [mf_val[60], mf_val[61], mf_val[62]])

pPKCAB_N['low'] = fuzz.trimf(pPKCAB_N.universe, [mf_val[63], mf_val[64], mf_val[65]])
pPKCAB_N['medium'] = fuzz.trimf(pPKCAB_N.universe, [mf_val[66], mf_val[67], mf_val[68]])
pPKCAB_N['high'] = fuzz.trimf(pPKCAB_N.universe, [mf_val[69], mf_val[70], mf_val[71]])

pGSK3B_N['low'] = fuzz.trimf(pGSK3B_N.universe, [mf_val[72], mf_val[73], mf_val[74]])
pGSK3B_N['medium'] = fuzz.trimf(pGSK3B_N.universe, [mf_val[75], mf_val[76], mf_val[77]])
pGSK3B_N['high'] = fuzz.trimf(pGSK3B_N.universe, [mf_val[78], mf_val[79], mf_val[80]])

pERK_N['low'] = fuzz.trimf(pERK_N.universe, [mf_val[81], mf_val[82], mf_val[83]])
pERK_N['medium'] = fuzz.trimf(pERK_N.universe, [mf_val[84], mf_val[85], mf_val[86]])
pERK_N['high'] = fuzz.trimf(pERK_N.universe, [mf_val[87], mf_val[88], mf_val[89]])
```

Diagram 3.1

```
# Define output using triangle membership function
class_label['c-CS-m'] = fuzz.trimf(class_label.universe, [0, 0, 1])
class_label['c-SC-m'] = fuzz.trimf(class_label.universe, [0, 1, 2])
class_label['c-CS-s'] = fuzz.trimf(class_label.universe, [1, 2, 3])
class_label['c-SC-s'] = fuzz.trimf(class_label.universe, [2, 3, 4])
class_label['t-CS-m'] = fuzz.trimf(class_label.universe, [3, 4, 5])
class_label['t-SC-m'] = fuzz.trimf(class_label.universe, [4, 5, 6])
class_label['t-CS-s'] = fuzz.trimf(class_label.universe, [5, 6, 7])
class_label['t-SC-s'] = fuzz.trimf(class_label.universe, [6, 7, 8])

# Display membership functions for the output variable
class_label.view()
```

Diagram 3.2

3.2 Fuzzy Rule Base

```
# Define a different set of fuzzy rule
rule1 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] & CaNA_N['high'] | Ubiquitin_N['medium'] |
         ARC_N['medium'] | pS6_N['medium'] | P38_N['medium'] |
         pERK_N['medium'])
    ),
    consequent=class_label['c-CS-m']
)

rule2 = ctrl.Rule(
    antecedent=(
        ( CaNA_N['low'] & Ubiquitin_N['high'] |
         P38_N['high'] |
         S6_N['low'] & pPKCAB_N['low'] & pGSK3B_N['low'] &
         pERK_N['low'])
    ),
    consequent=class_label['c-SC-m']
)

rule3 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | CaNA_N['high'] |
         ARC_N['low'] |
         S6_N['medium'] & pPKCAB_N['high']  |
         pERK_N['medium'])
    ),
    consequent=class_label['c-CS-s']
)

rule4 = ctrl.Rule(
    antecedent=(
        (SOD1_N['high']  & Ubiquitin_N['medium'] &
         ARC_N['high'] & pS6_N['high'] |
         S6_N['low'] | pPKCAB_N['low'] & pGSK3B_N['low'] )
    ),
    consequent=class_label['c-SC-s']
)
```

```
rule5 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | CaNA_N['high'] | Ubiquitin_N['low'] |
         S6_N['medium'] & pPKCAB_N['medium'] & pGSK3B_N['medium']&
         pERK_N['high'])
    ),
    consequent=class_label['t-CS-m']
)

rule6 = ctrl.Rule(
    antecedent=(
        (SOD1_N['high'] & CaNA_N['low'] | pS6_N['high'] |
         S6_N['low'] | pPKCAB_N['low'] | pGSK3B_N['low'] |
         pERK_N['low'])
    ),
    consequent=class_label['t-SC-m']
)

rule7 = ctrl.Rule(
    antecedent=(
        (SOD1_N['low'] | CaNA_N['high'] | Ubiquitin_N['low'] |
         ARC_N['low'] | pS6_N['low'] | P38_N['low'] |
         pERK_N['medium'])
    ),
    consequent=class_label['t-CS-s']
)

rule8 = ctrl.Rule(
    antecedent=(
        (SOD1_N['medium'] & CaNA_N['medium'] & Ubiquitin_N['medium'] &
         ARC_N['medium'] |
         S6_N['high'] | pPKCAB_N['medium'] | pGSK3B_N['medium'] |
         pERK_N['low'])
    ),
    consequent=class_label['t-SC-s']
)
```

Diagram 3.3

```
# Create control system
mice_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8])


# Define control system simulation
mice_ctrl_sim = ctrl.ControlSystemSimulation(mice_ctrl)
```

Diagram 3.4

3.3 Fuzzy Inference

```
results = {
    'prediction1_B': [],
    'actual_B': [],
    'prediction2_B': []
}

# Assuming X contains your input features and y contains the mice class
for i in range(len(X)):
    input_values = {
        'SOD1_N': X.iloc[i]['SOD1_N'],
        'CaNA_N': X.iloc[i]['CaNA_N'],
        'Ubiquitin_N': X.iloc[i]['Ubiquitin_N'],
        'ARC_N': X.iloc[i]['ARC_N'],
        'pS6_N': X.iloc[i]['pS6_N'],
        'P38_N': X.iloc[i]['P38_N'],
        'S6_N': X.iloc[i]['S6_N'],
        'pPKCAB_N': X.iloc[i]['pPKCAB_N'],
        'pGSK3B_N': X.iloc[i]['pGSK3B_N'],
        'pERK_N': X.iloc[i]['pERK_N']
    }

    output_value = y.iloc[i]  # Assuming y contains the mice class

    # Compute output
    for key, value in input_values.items():
        mice_ctrl_sim.input[key] = value

    mice_ctrl_sim.compute()
    results['actual_B']+=[y.iloc[i]]
    results['prediction1_B'].append(mice_ctrl_sim.output['class_label'])
    results['prediction2_B']+=[mice_ctrl_sim.output['class_label'].astype(int)]
```

Diagram 3.5

Output for Model B

```
Output:
      Predicted Mice class  Actual Mice class
0                        3                  0
1                        3                  0
2                        3                  0
3                        3                  0
4                        3                  0
...                    ...                ...
1075                     3                  7
1076                     3                  7
1077                     3                  7
1078                     3                  7
1079                     3                  7

[1080 rows x 2 columns]
```

Diagram 3.6

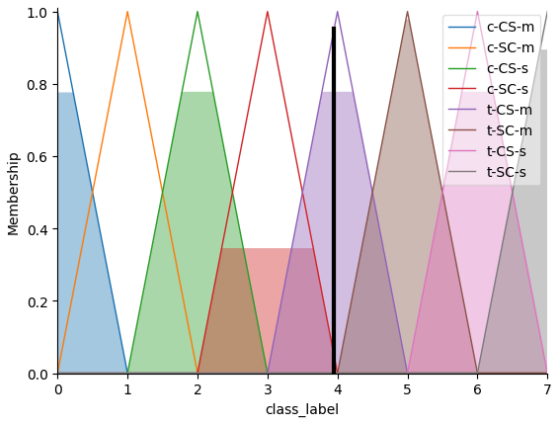

Diagram 3.7

```
mse_B = mean_squared_error(results['actual_B'], results['prediction2_B'])
print("Mean Square Error: " + str(mse_B))


acc_B = accuracy_score(results['actual_B'],results['prediction2_B'])
print("Accuracy: "+str(acc_B))
```

✓ 0.0s

```
Mean Square Error: 5.6342592592592595
Accuracy: 0.15925925925925927
```

Diagram 3.8

3.4 Defuzzification

```
def centroid_defuzzification(mf_values):
    numerator = sum(mf_values)
    denominator = len(mf_values)
    return numerator / denominator if denominator != 0 else 0

# Extract membership function values from 'prediction' and 'prediction2'
mf_values_prediction = results['prediction1_B']


# Perform defuzzification
crisp_value_prediction_B = centroid_defuzzification(mf_values_prediction)


# Print defuzzified values
print("\nDefuzzified Values:")
print("Crisp Value (Prediction):", crisp_value_prediction_B)
```

✓ 0.0s

```
Defuzzified Values:
Crisp Value (Prediction): 3.84469275523503
```

Diagram 3.9

4. Model Comparision

	Model A	**Model B**
MSE	5.309259	5.634259
Accuracy	0.123148	0.159259
Crisp Value	3.651540	3.844691

Table 4.1 Comparision between Model A and Model B

- MSE (Mean Squared Error): A lower MSE indicates better performance, as it represents the average squared difference between the actual and predicted values. Since Model A has a lower MSE, it performs slightly better in terms of minimizing the squared errors.

- Accuracy: Higher accuracy signifies better performance in classification tasks. Therefore, Model B has a higher accuracy, indicating an improvement in correctly classifying instances compared to Model A.

- Crisp Value: The crisp value represents the defuzzified output of the fuzzy inference system.Since Model B has a higher crisp value, it suggests a more confident prediction compared to Model A.

In conclusion, Model B improves in accuracy and confidence of predictions after fine-tuning from Model A.

5. References

Higuera C., Gardiner K., Cios K. (2015). Mice Protein Expression. *UCI Machine Learning Repository*. https://doi.org/10.24432/C50S3Z.

MathWorks. (n.d.). *Foundations of Fuzzy Logic*. In MATLAB Documentation. https://www.mathworks.com/help/fuzzy/foundations-of-fuzzy-logic.html

MathWorks. (n.d.). *Defuzzification Methods.* In MATLAB Documentation. https://www.mathworks.com/help/fuzzy/defuzzification-methods.html