INDEX

Sr.no	Title	Date	Pg.no	Sign
1	Design an Expert system using AIML.	29/07/24	2	
2	Implement Bayes Theorem using Python.	5/08/24	5	
3	Implement Conditional Probability and joint probability using Python.	26/08/24	7	
4	Create a simple rule-based system in Prolog for diagnosing a common illness based on symptoms.	3/09/24	9	
5	Design a Fuzzy based application using Python	28/09/24	11	
6	Simulate artificial neural network model with both feedforward and back-propagation approach.	16/09/24	15	
7	Simulate genetic algorithm with suitable example using Python any other platform	14/09/24	19	
8	Design intelligent agent using any AI algorithm. design expert tutoring system	28/09/24	21	
9	Design an application to simulate language parser.	5/10/24	23	
10	Develop the semantic net using python.	5/10/24	25	

Practical 1

Aim: Design an Expert system using AIML

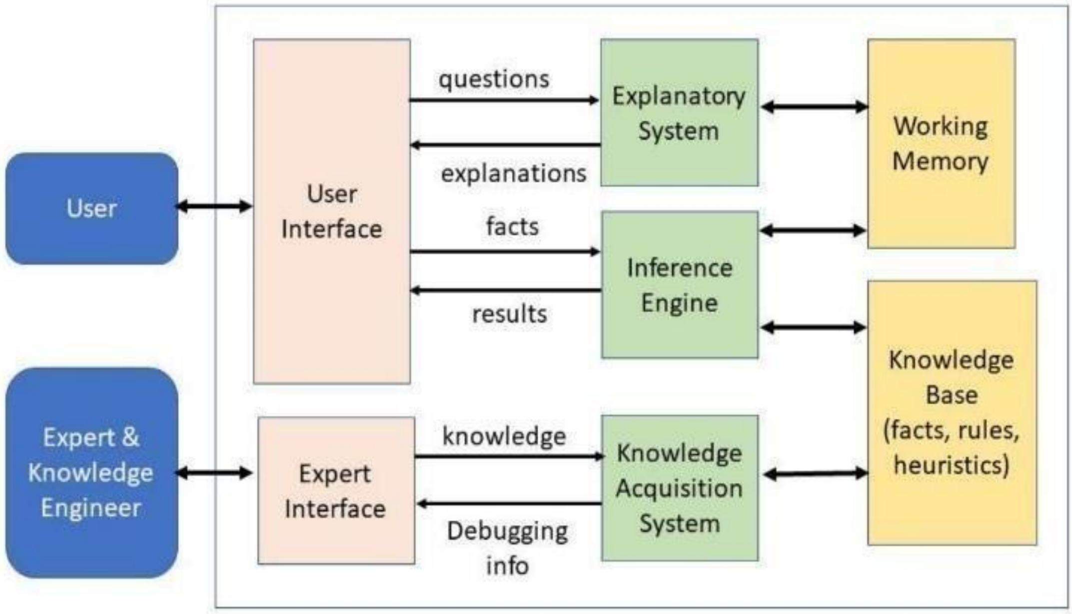
Theory:

In an expert system there are three main components: User Interface, Inference Engine and Knowledge Base

User Interface: Uses various user interfaces, such as menus, graphics, and dashboards, or Natural Language Processing (NLP) to communicate with the user.

Expert System: A software program that makes decisions or provides advice based on databases of expert knowledge in various contexts, such as medical diagnosis. An expert system is a computer program that solves problems in a specialized field that typically calls forhuman expertise using techniques from artificial intelligence. A well-organized collection of data about the system's domain is called a knowledge base.

The **knowledge base's** facts are interpreted and assessed by the inference engine, which then outputs the desired results or an answer.



Code:

Defining Flu.aiml

```
<aiml version="1.0.1" encoding="UTF-8">
    <category>
        <pattern>WHAT ARE THE SYMPTOMS OF FLU </pattern>
        <template>
      Flu symptoms usually include fever, chills, muscle aches, cough, congestion, runny
nose, headaches, and fatigue.
        </template>
    </category>

    <category>
        <pattern>I HAVE FEVER AND COUGH</pattern>
```

```
        <template>
            These symptoms could be associated with the flu. However, I recommend visiting a healthcare professional for an accurate diagnosis.
        </template>
    </category>

    <category>
        <pattern>IS FLU CONTAGIOUS</pattern>
        <template>
            Yes, flu is highly contagious and can spread easily from person to person.
        </template>
    </category>

    <category>
        <pattern>HOW CAN I PREVENT FLU</pattern>
        <template>
            The best way to prevent the flu is by getting a flu vaccine each year. Additionally, wash your hands frequently, avoid close contact with sick people, and maintain a healthy lifestyle.
        </template>
    </category>

    <category>
        <pattern>THANK YOU</pattern>
        <template>
            You're welcome! Take care and stay healthy.
        </template>
    </category>

    <category>
        <pattern>BYE</pattern>
        <template>
            Goodbye! Feel free to reach out if you have more questions.
        </template>
    </category>

    <category>
        <pattern>FLU*</pattern>
        <template>
            Could you please provide more details about your symptoms so that I can assist you better?
        </template>
    </category>
</aiml>
```

Jupyter Code:

```
import aiml
kernel = aiml.Kernel()
kernel.learn("flu.aiml")
print("Expert System for Identifying Flu Symptoms")
print("Type 'bye' to exit the conversation.")
while True:
    user_input = input("You: ")
    if user_input.lower() == "bye":
```

```
        print("System: Goodbye! Stay healthy.")
        break
    response = kernel.respond(user_input.upper())
    print(f"System: {response}")
```

Output:

```
...  Loading flu.aiml...done (0.00 seconds)
     Expert System for Identifying Flu Symptoms
     Type 'bye' to exit the conversation.
     System: Flu symptoms usually include fever, chills, muscle aches, cough, congestion, runny n
```

```
     System: The best way to prevent the flu is by getting a flu vaccine each year. Additionally,
```

```
     System: You're welcome! Take care and stay healthy.
```

Practical 2

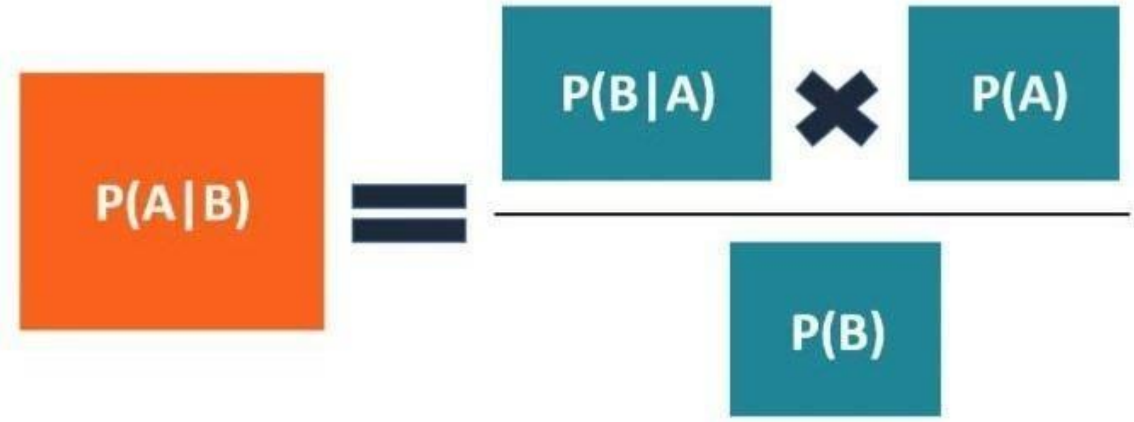
Aim: Implement Bayes Theorem using Python.

Theory:

Bayes' Theorem is a fundamental concept in probability theory that describes how to update the probability of a hypothesis based on new evidence. It's widely used in various fields such as medicine, finance, and machine learning.

Bayes' Theorem Formula

P(A|B) – the probability of event A occurring, given event B has occurred. P(B|A) – the probability of event B occurring, given event A has occurred. P(A) – the probability of event A.



Events:

A: Flower is Setosa

B: sepal length being greater than 5.0 cm

Steps Breakdown:

1. **Prior Probability P(A)**: The probability of the flower being setosa (without any conditions).
2. **Likelihood P(B|A)**: The probability of the sepal length being greater than 5.0 cm, given the flower is setosa.
3. **Marginal Probability P(B)**: The probability of the sepal length being greater than 5.0 cm, across the whole dataset.
4. **Posterior Probability P(A|B)**: The probability that a flower is setosa, given that its sepal length is greater than 5.0 cm.

Code:

```
import pandas as pd
def bayes_theorem(prior_A, likelihood_B_given_A, marginal_B):
    """
    Calculate the posterior probability using Bayes' Theorem
    :param prior_A: P(A) - Prior probability of A
    :param likelihood_B_given_A: P(B|A) - Likelihood of B given A
    :param marginal_B: P(B) - Marginal probability of B
```

```
    :return: P(A|B) - Posterior probability of A given B
    """
    return (likelihood_B_given_A * prior_A) / marginal_B

# Load the Iris dataset
def load_iris_dataset(file_path):
    return pd.read_csv(file_path)
# Calculate prior probability P(A)
def calculate_prior(data, class_col, class_value):
    return len(data[data[class_col] == class_value]) / len(data)
# Calculate likelihood P(B|A)
def calculate_likelihood(data, class_col, class_value, feature_col,
feature_condition):
    subset = data[data[class_col] == class_value]
    return len(subset[subset[feature_col] > feature_condition]) /
len(subset)
# Calculate marginal probability P(B)
def calculate_marginal(data, feature_col, feature_condition):
    return len(data[data[feature_col] > feature_condition]) / len(data)
# Apply Bayes' Theorem on the Iris dataset
def apply_bayes_to_iris(file_path, class_col, class_value, feature_col,
feature_condition):
    # Load dataset
    data = load_iris_dataset(file_path)
    # Calculate prior P(A)
    prior_A = calculate_prior(data, class_col, class_value)
    # Calculate likelihood P(B|A)
    likelihood_B_given_A = calculate_likelihood(data, class_col,
class_value, feature_col, feature_condition)
    # Calculate marginal probability P(B)
    marginal_B = calculate_marginal(data, feature_col, feature_condition)
    # Apply Bayes' Theorem
    posterior_A_given_B = bayes_theorem(prior_A, likelihood_B_given_A,
marginal_B)
    return posterior_A_given_B
# Example usage:
# Assume we want to calculate the probability P(Class='setosa' |
SepalLength > 5.0)
file_path = 'iris.csv' # Path to the iris dataset file
class_col = 'species'  # The column representing the class (A)
class_value = 'virginica' # The class value we're interested in (A)
feature_col = 'sepal_length' # The feature we're using (B)
feature_condition = 5.0  # The condition on the feature (B > 5.0)
# Calculate posterior probability P(setosa|sepal_length > 5.0)
posterior_probability = apply_bayes_to_iris(file_path, class_col,
class_value, feature_col, feature_condition)
print(f"P({class_value} | {feature_col} > {feature_condition}) =
{posterior_probability:.4f}")
print()
```

Output:

```
...   P(virginica | sepal_length > 5.0) = 0.4153
```

Practical 3

Aim: Implement Conditional Probability and joint probability using Python.

Theory:

Conditional Probability (P(A|B)): Conditional probability is a measure of the likelihood of event A occurring given that event B has occurred. In mathematical terms, it's defined as:

$P(A|B) = P(A \cap B) / P(B)$

where $P(A \cap B)$ represents the joint probability of events A and B occurring together.

Joint Probability ($P(A \cap B)$): The joint probability of two events A and B is a measure of how likely it is that both events will occur simultaneously. In mathematical terms, it's defined as:

$P(A \cap B) = P(A) \times P(B|A)$

Code Explanation:

Loading Penguins Dataset: The code loads the penguins dataset from a CSV file using Pandas.

Calculating Joint Probability (P(Species ∩ Island)): The joint probability is calculated by creating a pivot table that represents the number of penguins in each species category across different island categories.

pivot_table = pd.crosstab(df['species'], df['island'], normalize=True)

This code calculates the joint probability as:

P(Adelie ∩ Dream) = P(Adelie) × P(Dream|Adelie)

Calculating Conditional Probability (P(Species|Island)): The conditional probability is calculated by dividing each cell in the pivot table by the sum of all cells across a specific island category.

conditional_probability = pivot_table.div(pivot_table.sum(axis=0), axis=1)

This code calculates the conditional probability as:

P(Adelie|Dream) = P(Adelie ∩ Dream) / P(Dream)

Key Features and Assumptions:

The code assumes:

Independence: The events (species and island) are assumed to be independent, meaning that the likelihood of one event does not influence the other.

Mutual Exclusivity: The code implicitly assumes that each species category is mutually exclusive across different island categories.

Code:

```
import pandas as pd
df = pd.read_csv('penguins.csv')
print("Data Preview:")
print(df.head())
pivot_table = pd.crosstab(df['species'], df['island'], normalize=True)
print("\nJoint Probability is represented in the pivot table (Species vs
Island):")
print(pivot_table)
```

```
p_adelie_given_dream = pivot_table.loc['Adelie', 'Dream']
print()
print('The joint probability of an Adelie penguin being found on Dream
Island.')
print(f"\nP(Adelie | Dream) = {p_adelie_given_dream:.4f}")

conditional_probability = pivot_table.div(pivot_table.sum(axis=0), axis=1)
print("\nConditional Probability of Species given Island:")
print(conditional_probability)
normalized=True
p_adelie_given_dream = conditional_probability.loc['Adelie', 'Dream']
print()
print('The conditional probability of an Adelie penguin being found on
Dream Island.')
print(f"\nP(Adelie | Dream) = {p_adelie_given_dream:.4f}")
```

Output:

```
   body_mass_g     sex  year
0       3750.0    male  2007
1       3800.0  female  2007
2       3250.0  female  2007
3          NaN     NaN  2007
4       3450.0  female  2007

Joint Probability is represented in the pivot table (Species vs Island):
island       Biscoe     Dream  Torgersen
species
Adelie     0.127907  0.162791   0.151163
Chinstrap  0.000000  0.197674   0.000000
Gentoo     0.360465  0.000000   0.000000

The joint probability of an Adelie penguin being found on Dream Island.

P(Adelie | Dream) = 0.1628

Conditional Probability of Species given Island:
island       Biscoe     Dream  Torgersen
species
Adelie     0.261905  0.451613        1.0
Chinstrap  0.000000  0.548387        0.0
Gentoo     0.738095  0.000000        0.0

The conditional probability of an Adelie penguin being found on Dream Island.

P(Adelie | Dream) = 0.4516
```

Practical 4

Aim: Create a simple rule-based system in Prolog for diagnosing a common illness based on symptoms.

Theory:

Prolog expressions are comprised of the following truth-functional symbols, which have the same interpretation as in the predicate calculus.

Code:

```
%Facts:Define symptoms
symptom(fever).
symptom(cough).
symptom(sore_throat).
symptom(body_aches).
symptom(runny_nose).
symptom(headache).
symptom(fatigue).
%Facts:Define possible illnesses
condition(cold).
condition(flu).
condition(strep_throat).
%Rules: Diagnosing based on the presence of symptoms
diagnose(cold):-
                 symptom(runny_nose),
                 symptom(cough),
                 symptom(sore_throat),
                 \+ symptom(fever). %Absence of fever
diagnose(flu):-
                 symptom(fever),
                 symptom(cough),
                 symptom(body_aches),
                 symptom(headache),
                 symptom(fatigue).
diagnose(sterp_throat):-
                 symptom(sore_throat),
                 symptom(fever),
                 \+symptom(cough). %Absence of cough

%Alternative:Diagnosing based on rule covering all possible symptoms
diagnose(unknown):-
                  \+diagnose(cold),
                  \+diagnose(flu),
                  \+diagnose(strep_throat).
%You can ask Prolog:
?-diagnose(Condition).
```

Output

```prolog
%Facts:Define symptoms
%symptom(fever).
symptom(cough).
symptom(sore_throat).
%symptom(body_aches).
symptom(runny_nose).
%symptom(headache).
%symptom(fatigue).

%Facts:Define possible illnesses
condition(cold).
condition(flu).
condition(strep_throat).

%Rules: Diagnosing based on the presence of symptoms
diagnose(cold):-
                symptom(runny_nose),
                symptom(cough),
                symptom(sore_throat),
                \+ symptom(fever). %Absence of fever
diagnose(flu):-
                symptom(fever),
                symptom(cough),
                symptom(body_aches),
                symptom(headache),
                symptom(fatigue).
diagnose(sterp_throat):-
                symptom(sore_throat),
                symptom(fever),
                \+symptom(cough). %Absence of cough
%Alternative:Diagnosing based on rule covering all possible symptoms
diagnose(unknown):-
                \+diagnose(cold),
                \+diagnose(flu),
                \+diagnose(strep_throat).
```

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.7)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.7)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- diagnose(Condition).
Condition = cold
```

Practical 5

Aim: Design a Fuzzy based application using Python.

Theory:

A fuzzy logic system is a mathematical framework that handles reasoning with uncertainty and imprecision. Unlike traditional binary logic (where variables are either true or false, i.e., 0 or 1), fuzzy logic allows for degrees of truth, where values can range between 0 and 1. This makes fuzzy logic particularly useful in systems that involve human-like reasoning, where decisions are not strictly binary but involve some level of vagueness.

Example: Consider an air conditioning system:

- Inputs: Temperature and humidity (both can be fuzzy).
- Outputs: Fan speed (also fuzzy).
- Fuzzy rules might say: "If the temperature is high and the humidity is low, then set fan speed to high."

Applications:

- Control systems: Air conditioning, washing machines, and automatic transmission in cars.
- Decision-making systems: Medical diagnosis, stock market prediction, etc.

Fuzzy logic is well-suited for systems where precise data is unavailable, and human-like reasoning is required to make decisions under uncertainty.

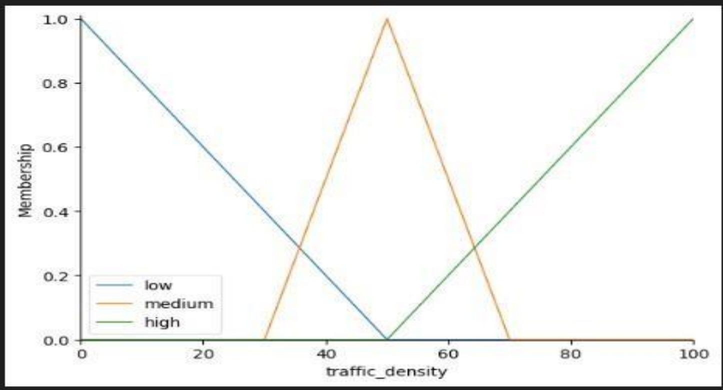
Code:

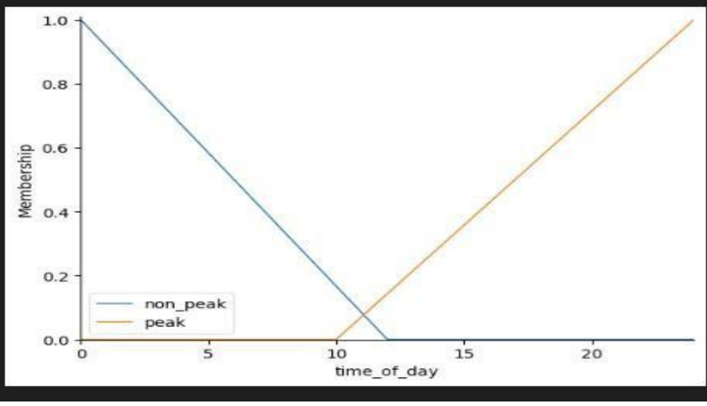
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

traffic_density = ctrl.Antecedent(np.arange(0, 101, 1), 'traffic_density')
time_of_day = ctrl.Antecedent(np.arange(0, 25, 1), 'time_of_day')
green_light_duration = ctrl.Consequent(np.arange(0, 61, 1),
'green_light_duration')

traffic_density['low'] = fuzz.trimf(traffic_density.universe, [0, 0, 50])
traffic_density['medium'] = fuzz.trimf(traffic_density.universe, [30, 50,
70])
traffic_density['high'] = fuzz.trimf(traffic_density.universe, [50, 100,
100])

time_of_day['non_peak'] = fuzz.trimf(time_of_day.universe, [0, 0, 12])
time_of_day['peak'] = fuzz.trimf(time_of_day.universe, [10, 24, 24])

green_light_duration['short'] = fuzz.trimf(green_light_duration.universe,
[0, 0, 20])
```

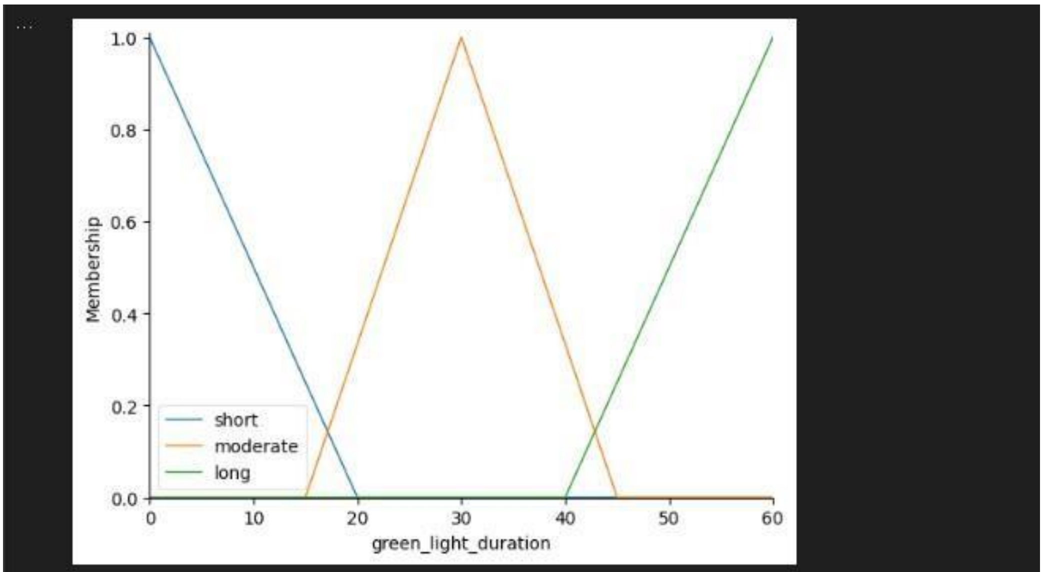
```
green_light_duration['moderate'] =
fuzz.trimf(green_light_duration.universe, [15, 30, 45])
green_light_duration['long'] = fuzz.trimf(green_light_duration.universe,
[40, 60, 60])

traffic_density.view()
time_of_day.view()
green_light_duration.view()

rule1 = ctrl.Rule(traffic_density['low'] & time_of_day['non_peak'],
green_light_duration['short'])
rule2 = ctrl.Rule(traffic_density['low'] & time_of_day['peak'],
green_light_duration['moderate'])
rule3 = ctrl.Rule(traffic_density['medium'] & time_of_day['non_peak'],
green_light_duration['moderate'])
rule4 = ctrl.Rule(traffic_density['medium'] & time_of_day['peak'],
green_light_duration['long'])
rule5 = ctrl.Rule(traffic_density['high'] & time_of_day['non_peak'],
green_light_duration['long'])
rule6 = ctrl.Rule(traffic_density['high'] & time_of_day['peak'],
green_light_duration['long'])

green_light_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5,
rule6])
green_light_sim = ctrl.ControlSystemSimulation(green_light_ctrl)

green_light_sim.input['traffic_density'] = 75 # High traffic
green_light_sim.input['time_of_day'] = 18      # Peak hours
green_light_sim.compute()
print(f"Recommended Green Light Duration:
{green_light_sim.output['green_light_duration']} seconds")
green_light_duration.view(sim=green_light_sim)
plt.show()
```

Output:

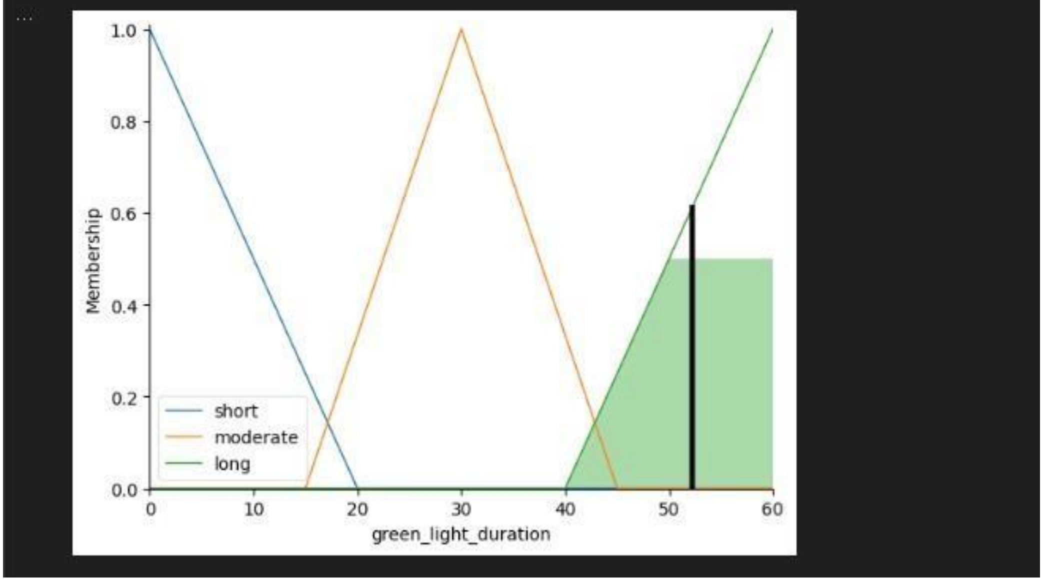
```
c:\Users\bhavi\AppData\Local\Programs\Python\Python312\Lib\site-packages\skfuzzy\control\fuzzyvariable.p
  fig.show()
Recommended Green Light Duration: 52.22222222222222 seconds
```







1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
10
20
30
40
50
60
green_light_duration
short
moderate
long



1.0
0.8
0.6
0.4
0.2
0.0
Membership
0
10
20
30
40
50
60
green_light_duration
short
moderate
long

Practical 6

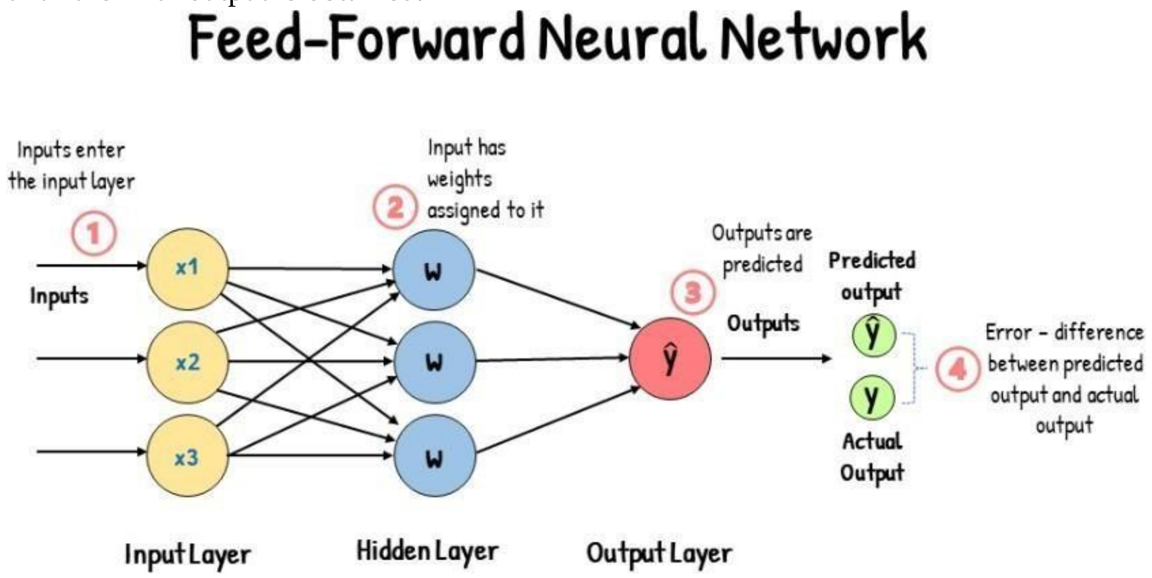
Aim: Simulate artificial neural network model with both feedforward and backpropagation approach.

Theory:

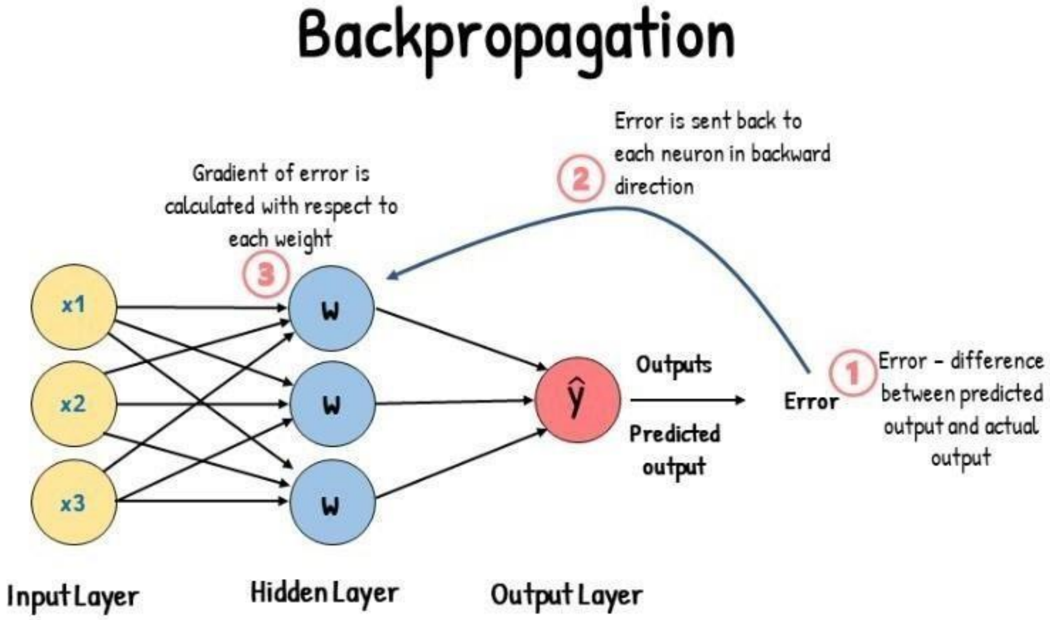
Feedforward Neural Network:

In a Feedforward Neural Network, information moves in one direction only: from the input layer through the hidden layers to the output layer. There are no loops or cycles in the network.

How it works: Input data is passed through the network, with each neuron applying a weighted sum of its inputs and an activation function to produce an output. This continues layer by layer until the final output is obtained.



Backpropagation: Backpropagation is the learning algorithm used to train an ANN. It adjusts



the weights of the network to minimize the error between the predicted output and the actual target.

Code:

```
import numpy as np
# Sigmoid Activation Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of the Sigmoid Function for backpropagation
def sigmoid_derivative(x):
    return x * (1 - x)

# ANN class to simulate feedforward and backpropagation
class ArtificialNeuralNetwork:
    def_init_(self, input_size, hidden_size, output_size,
learning_rate=0.5):
        # Initialize weights randomly
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.weights_hidden_output = np.random.rand(hidden_size,
output_size)

        # Initialize biases randomly
        self.bias_hidden = np.random.rand(1, hidden_size)
        self.bias_output = np.random.rand(1, output_size)

        # Set the learning rate
        self.learning_rate = learning_rate

    #  Feedforward  process
    def feedforward(self, X):
        # Hidden layer activation
        self.hidden_input = np.dot(X, self.weights_input_hidden) +
self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)

        # Output layer activation
        self.output_input = np.dot(self.hidden_output,
self.weights_hidden_output) + self.bias_output
        self.output = sigmoid(self.output_input)

        return self.output

    # Backpropagation process
    def backpropagation(self, X, y):
        # Error at the output layer
        output_error = y - self.output
        output_delta = output_error * sigmoid_derivative(self.output)

        # Error at the hidden layer
        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error *
sigmoid_derivative(self.hidden_output)

        # Update the weights and biases using the deltas
        self.weights_hidden_output +=
self.hidden_output.T.dot(output_delta) * self.learning_rate
        self.weights_input_hidden += X.T.dot(hidden_delta) *
self.learning_rate
```

```
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) *
self.learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) *
self.learning_rate

    # Train the neural network
    def train(self, X, y, epochs):
        for epoch in range(epochs):
            # Feedforward
            self.feedforward(X)
            # Backpropagation
            self.backpropagation(X, y)
            # Print loss every 100 epochs
            if (epoch + 1) % 100 == 0:
                loss = np.mean(np.square(y - self.output))
                print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss:.6f}')

# Example usage
if___name___== "__main__":
    # Input dataset (XOR problem)
    X = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])

    # Output dataset (XOR output)
    y = np.array([[0],
                  [1],
                  [1],
                  [0]])

    # Parameters
    input_size = X.shape[1] # 2 features in input
    hidden_size = 2          # 2 neurons in hidden layer
    output_size = 1          # 1 output neuron (binary classification)

    # Create the neural network
    ann = ArtificialNeuralNetwork(input_size, hidden_size, output_size,
learning_rate=0.5)

    # Train the neural network
    ann.train(X, y, epochs=10000)

    # Test the neural network
    output = ann.feedforward(X)
    print("\nPredicted Output after training:")
    print(output)
```

Output:

```
[9]   ✓  1.6s

...  Epoch 100/10000, Loss: 0.249345
     Epoch 200/10000, Loss: 0.248368
     Epoch 300/10000, Loss: 0.245708
     Epoch 400/10000, Loss: 0.238388
     Epoch 500/10000, Loss: 0.220805
     Epoch 600/10000, Loss: 0.195450
     Epoch 700/10000, Loss: 0.171738
     Epoch 800/10000, Loss: 0.135970
     Epoch 900/10000, Loss: 0.074064
     Epoch 1000/10000, Loss: 0.035658
     Epoch 1100/10000, Loss: 0.020421
     Epoch 1200/10000, Loss: 0.013582
     Epoch 1300/10000, Loss: 0.009938
     Epoch 1400/10000, Loss: 0.007738
     Epoch 1500/10000, Loss: 0.006288
     Epoch 1600/10000, Loss: 0.005269
     Epoch 1700/10000, Loss: 0.004520
     Epoch 1800/10000, Loss: 0.003947
     Epoch 1900/10000, Loss: 0.003497
     Epoch 2000/10000, Loss: 0.003135
     Epoch 2100/10000, Loss: 0.002837
     Epoch 2200/10000, Loss: 0.002589
     Epoch 2300/10000, Loss: 0.002379
     Epoch 2400/10000, Loss: 0.002199
     Epoch 2500/10000, Loss: 0.002043
     ...
      [0.9834894 ]
      [0.98351691]
      [0.01708531]]
```

Practical 7

Aim: Simulate genetic algorithm with suitable example using Python any other platform.

Theory:

Genetic Algorithms are a type of optimization technique inspired by the process of natural selection. They're particularly useful for solving complex problems that involve multiple parameters or variables.

In the given code, we are using a genetic algorithm to find a sequence of characters (target_string = "HELLO") with a certain fitness level. The fitness function here checks if the generated string is equal to the target string. If so, it means we've found the optimal solution.

Genetic Algorithms work by:

Initializing a population of candidate solutions (in this case, strings).

Evaluating each candidate's fitness based on a predefined metric.

Selecting parents from the current generation to reproduce and create offspring.

Applying crossover and mutation operators to generate new candidates.

Repeating steps 3-4 until a termination condition is met.

Code:

```
import random
import string
target_string = "HELLO"
population_size = 50
mutation_rate = 0.01
generations = 200

def fitness(individual):
    return sum(1 for a, b in zip(individual, target_string) if a == b)

def create_population(size):
    return [''.join(random.choices(string.ascii_uppercase,
k=len(target_string))) for _ in range(size)]

def select_parents(population):
    tournament = random.sample(population, 5)
    return max(tournament, key=fitness)

def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)
    return parent1[:crossover_point] + parent2[crossover_point:]

def mutate(individual):
    individual = list(individual)
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] = random.choice(string.ascii_uppercase)
    return ''.join(individual)

population = create_population(population_size)
for generation in range(generations):
    best_individual = max(population, key=fitness)
```

```
    print(f"Generation {generation}: Best individual: {best_individual},
Fitness: {fitness(best_individual)}")

    if fitness(best_individual) == len(target_string):
          break

    new_population = []
    for _  in  range(population_size):
        parent1 = select_parents(population)
        parent2 = select_parents(population)
        child = crossover(parent1, parent2)
        child = mutate(child)
        new_population.append(child)

    population = new_population

best_individual = max(population, key=fitness)
print(f"Best individual: {best_individual}, Fitness:
{fitness(best_individual)}")
```

Output:

```
...  Generation 0: Best individual: EEULR, Fitness: 2
     Generation 1: Best individual: IEULR, Fitness: 2
     Generation 2: Best individual: HRSLO, Fitness: 3
     Generation 3: Best individual: EESLO, Fitness: 3
     Generation 4: Best individual: HEPLO, Fitness: 4
     Generation 5: Best individual: WELLO, Fitness: 4
     Generation 6: Best individual: HELLO, Fitness: 5
     Best individual: HELLO, Fitness: 5
```

Practical 8

Aim: Design intelligent agent using any AI algorithm. design expert tutoring system

Theory:

Artificial Neural Network (ANN)

Artificial Neural Networks are computational models inspired by the structure and function of biological neural networks. They're powerful tools for classification, regression, and other machine learning tasks.

The given code implements an ANN with a single hidden layer to classify inputs into binary outputs. The sigmoid activation function is used in both the hidden and output layers.

ANNs work by:

Initializing weights and biases randomly.

Forward propagating input data through the network, using activation functions (e.g., sigmoid) to introduce non-linearity.

Computing the error between predicted outputs and actual labels.

Backpropagating this error through the network to adjust weights and biases.

Repeating steps 2-4 until convergence or a termination condition is met.

Code:

```
class MathTutor:
    def_init_(self):
        self.operations = {
            '+': lambda a, b: a + b,
            '-': lambda a, b: a - b,
            '*': lambda a, b: a * b,
            '/': lambda a, b: a/b,
        }
    def explain_operation(self, operator):
        explanation = {
            '+': "Addition adds two numbers together.",
            '-': "Subtraction subtracts the second number from the first.",
            '*': "Multiplication gives the product of two numbers.",
            '': "Division divides the first number by the second.",
        }
        return explanation.get(operator, "Invalid operation.")
    def perform_operation(self, operator, a, b):
        if operator in self.operations:
            return self.operations[operator](a, b)
        else:
            return  None
if__name__== "__main__":
    tutor = MathTutor()
    # Example usage:
    operator = '+'
    a, b = 24, 8
    print(tutor.explain_operation(operator))
    result = tutor.perform_operation(operator, a, b)
    print(f"Result of {a} {operator} {b} = {result}")
```

Output:

```
... Addition adds two numbers together.
    Result of 24 + 8 = 32
```

```
... Subtraction subtracts the second number from the first.
    Result of 24 - 8 = 16
```

```
... Multiplication gives the product of two numbers.
    Result of 24 * 8 = 192
```

```
... Division divides the first number by the second.
    Result of 24 / 88 = 0.2727272727272727
```

Practical 9

Aim: Design an application to simulate language parser.

Theory:

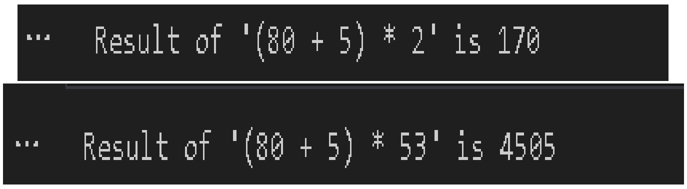
Knowledge Representation is a field in artificial intelligence (AI) that deals with how to formally capture and organize information about the world in a way that a machine can understand and reason with.
KR systems use structured formats (e.g., logic, semantic networks, ontologies) to represent facts, concepts, and relationships between objects. This allows AI systems to process complex data, reason about it, and draw conclusions.
In short, KR provides a framework for storing and manipulating knowledge in a machine-readable format, enabling intelligent decision-making and problem-solving.

Code:

```
class SimpleParser:
    def __init__(self, expr):
        self.tokens = expr.replace('(', ' ( ').replace(')', ' ) ').split()
        self.pos = 0

    def parse(self):
        return self.expr()

    def advance(self):
        self.pos += 1

    def current_token(self):
        return self.tokens[self.pos] if self.pos < len(self.tokens) else
None


    def expr(self):
        result = self.term()
        while self.current_token() in ('+', '-'):
            if self.current_token() == '+':
                self.advance()
                result += self.term()
            elif self.current_token() == '-':
                self.advance()
                result -= self.term()
        return result

    def term(self):
        result = self.factor()
        while self.current_token() in ('*', ''):
            if self.current_token() == '*':
                self.advance()
                result  *=  self.factor()
            elif self.current_token() == '':
                self.advance()
                result = self.factor()
        return result

    def factor(self):
        token = self.current_token()
```

```
        if token.isdigit():
            self.advance()
            return int(token)
        elif token == '(':
            self.advance()
            result = self.expr()
            self.advance() # skip ')'
            return result
        raise ValueError("Invalid syntax")

if_name__== "__main__":
    expr = "(80 + 5) * 2"
    parser = SimpleParser(expr)
    result = parser.parse()
    print(f"Result of '{expr}' is {result}")
```

Output:



Practical 10

Aim: Develop the semantic net using python.

Theory:

Semantic Network:

A semantic network is a data structure used to represent knowledge in the form of concepts (nodes) and their interrelationships (edges or links). It is a graphical model that depicts how different concepts in a particular domain are connected and how they relate to each other semantically.

Example: Consider a semantic network for animals:

- Concepts (Nodes): "Dog", "Cat", "Animal", "Mammal".
- Relationships (Edges): "Dog is a Mammal", "Mammal is a Animal", "**Dog** has **Fur**", "**Dog** can **Bark**".

In this network:

- Dog is connected to Mammal by an "is a" relationship.
- Dog is connected to Bark by a "can" relationship.

Code:

```
class SemanticNetwork:
    def__init__(self):
        self.network = {}

    def add_concept(self, concept):
        if concept not in self.network:
            self.network[concept] = {'is_a': [], 'has_a': []}

    def add_relation(self, relation, concept1, concept2):
        self.add_concept(concept1)
        self.add_concept(concept2)
        self.network[concept1][relation].append(concept2)

    def get_relations(self,  concept):
        return self.network.get(concept, {})

    def display_network(self):
        for concept, relations in self.network.items():
            print(f"Concept: {concept}")
            for relation, related_concepts in relations.items():
                for related_concept in related_concepts:
                    print(f"  {relation} -> {related_concept}")

if__name___== "__main___":
    sn = SemanticNetwork()

    # Adding concepts and relations
    sn.add_concept("Animal")
    sn.add_concept("Bird")
    sn.add_concept("Mammal")
    sn.add_concept("Penguin")
    sn.add_concept("Canary")
```

```
    sn.add_relation("is_a", "Bird", "Animal")
    sn.add_relation("is_a", "Mammal", "Animal")
    sn.add_relation("is_a", "Penguin", "Bird")
    sn.add_relation("is_a", "Canary", "Bird")
    sn.add_relation("has_a", "Bird", "Wings")
    sn.add_relation("has_a", "Canary", "Yellow_Feathers")
    # Displaying the network
    sn.display_network()
print()
```

Output:

```
... Concept: Animal
    Concept: Bird
      is_a -> Animal
      has_a -> Wings
    Concept: Mammal
      is_a -> Animal
    Concept: Penguin
      is_a -> Bird
    Concept: Canary
      is_a -> Bird
      has_a -> Yellow_Feathers
    Concept: Wings
    Concept: Yellow_Feathers
```