Practical 1

Aim: - Design an Expert system using AIML.

Code: - Flu.aiml(Text file)

```
<aiml version="1.0.1" encoding="UTF-8">

    <category>

        <pattern>WHAT ARE FLU SYMPTOMS<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>
```

Practical 2

Aim: - Implement Bayes Theorem using Python.

```
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_pat
h):     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 = 'setosa'  # 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}")
```

Practical 3

Aim: -Implement Conditional Probability and joint probability using Python.

```
import pandas as pd


# Load the penguins dataset from a CSV
file df = pd.read_csv('penguins.csv')


# Preview the data
print("Data Preview:")
print(df.head())


# Create a pivot table for joint probability
# Pivot table will be for Species (rows) and Island (columns),
and we'll compute frequencies pivot_table =
pd.crosstab(df['species'], df['island'], normalize=True)
```

```
print("\nJoint Probability (Pivot Table):")
print(pivot_table)
# Example: Conditional Probability of Species given Island #
We can normalize along columns to get conditional
probabilities conditional_probability =
pivot_table.div(pivot_table.sum(axis=0), axis=1)

print("\nConditional Probability of Species given Island:")
print(conditional_probability)

# To calculate Joint Probability, we already have it in the
pivot table, normalized=True gives joint probabilities
print("\nJoint Probability is represented in the pivot table
(Species vs Island):") print(pivot_table)

# Example: Calculating P(Species = Adelie | Island = Biscoe)
p_adelie_given_biscoe =
conditional_probability.loc['Adelie', 'Biscoe']
print(f"\nP(Adelie | Biscoe) = {p_adelie_given_biscoe:.4f}")
```

Practical 4

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

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).

%Quries: Example of how to diagnose %?-
diagnose(Condition).
%Output:Condition = flu.(if the symptoms match the flu criteria)

%Assuming the patient has the following
symptoms: symptom(fever). symptom(cough).
symptom(body_aches).
symptom(headaches).
symptom(fatigue).

%You can ask Prolog:
?-diagnose(Condition).
```

Practical 5

Aim: - Design a Fuzzy based application using Python.

```
import numpy as np import
skfuzzy as fuzz from skfuzzy
import control as ctrl
import matplotlib.pyplot as
plt
# Define fuzzy variables for traffic density, time of day, and
green light duration 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')

# Define membership functions for traffic density (low, medium,
high) 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])

# Define membership functions for time of day (non-peak, peak)
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])

# Define membership functions for green light duration (short,
moderate, long) 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])

# Visualize the membership
functions traffic_density.view()
time_of_day.view()
green_light_duration.view()

# Define the rules for the fuzzy system 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'])
```

```
# Control system green_light_ctrl = ctrl.ControlSystem([rule1,
rule2, rule3, rule4, rule5, rule6]) green_light_sim =
ctrl.ControlSystemSimulation(green_light_ctrl)
# Simulate the system for some input values (traffic density and
time of day) green_light_sim.input['traffic_density'] = 75  #
High traffic green_light_sim.input['time_of_day'] = 18      #
Peak hours
# Compute the output based on the input values
green_light_sim.compute()
# Print and visualize the output
print(f"Recommended Green Light Duration:
{green_light_sim.output['green_light_duration']} seconds")
green_light_duration.view(sim=green_light_sim)
# Show the plots
plt.show()
```

Practical 6

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

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 (XNOR problem)
    X = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
    # Output dataset (XNOR
output)     y = np.array([[1],
                  [0],
                  [0],
                  [1]])
    # 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)
```

Practical 7

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

Code: -

```
import random
import string
# Genetic Algorithm parameters target_string =
"HELLO" population_size = 50  # Increased
population size mutation_rate = 0.01 generations =
200  # Increased generations for more evolution
# Fitness function: number of characters matching
the target def fitness(individual):
    return sum(1 for a, b in zip(individual, target_string) if a
== b)
# Create initial population (random
strings) def create_population(size):
    return [''.join(random.choices(string.ascii_uppercase,
k=len(target_string))) for _ in range(size)]
# Select parents (tournament
selection) def
select_parents(population):
    tournament = random.sample(population, 5)  # Select 5
individuals instead of
3 for better diversity
return max(tournament,
key=fitness) # Crossover
(single-point crossover) def
crossover(parent1, parent2):
    crossover_point = random.randint(1,
len(parent1) - 1)     return
parent1[:crossover_point] +
parent2[crossover_point:]
```

```
# Mutation (random character
mutation) 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)
# Main genetic algorithm loop
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):  # Stop
early if the optimal solution is found        break
    # Create new generation
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 in the final population best_individual =
max(population, key=fitness) print(f"Best individual:
{best_individual}, Fitness: {fitness(best_individual)}")
```

Practical 8

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

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 = 10, 5
print(tutor.explain_operation(operator))
result =
tutor.perform_operation(operator, a, b)
print(f"Result of {a} {operator} {b} =
{result}") print('Gayatri Kulkarni-
53004230002')
```

Practical 9

Aim: - Design an applicationn to simulate language parser.

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 = "(3 + 5) * 2"
```

```
    parser = SimpleParser(expr)     result = parser.parse()
print(f"Result of '{expr}' is {result}")
```

Practical 10

Aim: - Develop the semantic net using python.

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()
```