NEW HORIZON
COLLEGE OF ENGINEERING
New Horizon Knowledge Park, Ring Road, Marathalli
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

Department of Computer Science & Engineering (Data Science)

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LAB

22CDL61

Semester: VI AY: 2024-25

Prepared By	Approved By	Authorized By
Dr. R Suganya	Dr.B.Swathi	Dr.B.Swathi

NEW HORIZON COLLEGE OF ENGINEERING

VISION

To emerge as an institute of eminence in the fields of engineering, technology and management in serving the industry and the nation by empowering students with a high degree of technical, managerial and practical competence.

MISSION

To strengthen the theoretical, practical and ethical dimensions of the learning process by fostering a culture of research and innovation among faculty members and students.

To encourage long-term interaction between the academia and industry through their involvement in the design of curriculum and its hands-on implementation.

To strengthen and mould students in professional, ethical, social and environmental dimensions by encouraging participation in co-curricular and extracurricular activities

QUALITY POLICY

To provide educational services of the highest quality both curricular and co-curricular to enable students integrate skills and serve the industry and society equally well at global level.

VALUES

- Academic Freedom
- Integrity Inclusiveness
- Innovation
- Professionalism
- Social Responsibility

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

PROGRAM OUTCOMES (Pos)

PO1 Engineering Knowledge: Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex Computer Science and Data Science engineering problems.

PO2 Problem Analysis: Identify, formulate, review research literature and analyze complex Computer Science and Data Science engineering problems reaching substantiated conclusions using first principles of mathematics, naturalsciences and engineering sciences.

PO3 Design / Development of Solutions: Design solutions for complex Computer Science and Data Science engineering problems and design system components or processes that meet specified needs with appropriate considerationfor public health and safety, cultural, societal and environmental considerations.

PO4 Conduct Investigations of Complex Problems:

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the informationto provide valid conclusions.

PO5 Modern tool usage: Create, select and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex Computer Science and Data Science engineering activitieswith an understanding of the limitations.

PO6 The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice in Computer Science and Data Science Engineering.

PO7 Environment and sustainability: Understand the impact of the professional engineering solutions in Computer Science and Data Science engineering in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 Individual and Team Work: Function effectively as an individual and as a member or leader to diverse teams, and in multidisciplinary settings.

PO10 Communication: Communicate effectively on complex Computer Science and Data Science engineering activities with the engineering communityand with society at large, such as, being able to comprehend and write effective report and design documentation, make effective presentations, and give and receive clear instructions.

PO11 Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest contextof technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1
Apply data analysis techniques, algorithmic expertise, and advanced modelling to effectively solve complex problems across various domains demonstrating their capacity to derive insights and propose innovative solutions in the realm of data-driven technologies.

PSO2
Collaborate proficiently with experts from diverse fields and actively engage in continuous professional growth in the domain of Computer Science and Engineering, specializing in the field of Data Science.

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LAB

Course Code	**22CDL61**	**CIE Marks**	**50**
L:T:P:S	**0:0:1:0**	**SEE Marks**	**50**
Hrs / Week	**3**	**Total Marks**	**100**
Credits	**01**	**Exam Hours**	**03**

Course outcomes: At the end of the course, the student will be able to:

22CDL61.1	Analyze search algorithms and learning algorithms for a training data set.
22CDL61.2	Apply Heuristic search techniques and various learning techniques for efficient computing.
22CDL61.3	Evaluate the various artificial intelligence and machine learning algorithms.
22CDL61.4	Use machine learning models for solving classification and prediction problems.

Mapping of Course Outcomes to Program Outcomes and Program Specific Outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO 9	PO10	PO11	PO12	PSO 1	PSO 2
22CDL61.1	3	3	3	3	3	-	-	-	1	-	-	2	3	3
22CDL61.2	3	3	3	3	3	-	-	-	1	-	-	2	3	3
22CDL61.3	3	3	3	3	3	-	-	-	1	-	-	2	3	3
22CDL61.4	3	3	3	3	3	-	-	-	1	-	-	2	3	3

Pgm. No.	List of Programs	Hours	Cos
	Prerequisite Programs		
	◆ Introduction to Python programming ◆ Use of numpy libraries ◆ Use of pandas libraries ◆ Use of visualization tool kit	3	NA
	PART-A		
1	Implement the A* Search algorithm for finding the shortest path in a weighted graph.	3	22CDL61.1 22CDL61.2
2	Implement AO* (AO-star) Search algorithm for finding the shortest path in a weighted graph.	3	22CDL61.1 22CDL61.2
3	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	3	22CDL61.1 22CDL61.2
4	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample	3	22CDL61.1 22CDL61.2
5	Develop an Artificial Neural Network by implementing the back propagation algorithm and test the same using appropriate data sets.	3	22CDL61.1 22CDL61.2
6	Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	3	22CDL61.1 22CDL61.2

PART-B			
7	Demonstrate the Bias-Variance Trade-off in a machine learning model, provide visualizations and insights.	3	22CDL61.3 22CDL61.4
8	Demonstrate Association Rule Mining using the FP-Growth Algorithm. Utilize a suitable dataset to identify frequent item sets and generate association rules.	3	22CDL61.3 22CDL61.4
9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	3	22CDL61.3 22CDL61.4
10	Develop a Convolutional Neural Network (CNN) and test it using appropriate datasets. Implement key components such as convolutional layers, pooling layers, and fully connected layers. Evaluate the CNN's performance on tasks such as image classification.	3	22CDL61.3 22CDL61.4
11	Implement Q learning algorithm.	3	22CDL61.3 22CDL61.4
12	Develop a simple Chabot using rule-based responses.	3	22CDL61.3 22CDL61.4

PART-C
Beyond Syllabus Virtual Lab Content
(To be done during Lab but not to be included for CIE or SEE)

• Parallel and distributed processing - I: Interactive activation and competition models (https://cse22- iiith.vlabs.ac.in/exp/parallel-distributed-processing-i/theory.html)
• Competitive learning neural networks for pattern clustering (https://cse22-iiith.vlabs.ac.in/exp/pattern-clustering/)
• Solution to travelling salesman problem using self organizing maps (https://cse22-iiith.vlabs.ac.in/exp/self-organizing-maps/)

CIE Assessment Pattern (50 Marks – Lab)

RBT Levels		Test (s)	Weekly Assessment
		20	30
L1	Remember	-	-
L2	Understand	-	-
L3	Apply	10	10
L4	Analyze	5	10
L5	Evaluate	5	10
L6	Create	-	-

SEE Assessment Pattern (50 Marks – Lab)

RBT Levels		Exam Marks Distribution (50)
L1	Remember	-
L2	Understand	-
L3	Apply	20
L4	Analyze	20
L5	Evaluate	10
L6	Create	-

Suggested Learning Resources:

Textbooks:

1. Tom M Mitchell, “Machine Lerning”,1st Edition, McGraw Hill Education, 2017, ISBN-13, **978-1259096952.**

2. Elaine Rich, Kevin K and S B Nair, “Artificial Intelligence”, 3rd Edition, McGraw Hill Education, 2017, ISBN-13, **978- 0070087705.**

Reference Books:

1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education, ISBN-13, 978-1259096952.
2. Pattern Recognition and Machine Learning, Bishop, Christopher, Springer nature publications, ISBN-13, 978-1493938438.
3. Hands-On Machine Learning with Scikit-Learn and TensorFlow, Concepts, Tools, and Techniques to Build Intelligent Systems, Aurélien Géron, O'Reilly Media, March 2017. ISBN-13, 978-9355421982.

PART-A

Program 1:

Implement the A* Search algorithm for finding the shortest path in a weighted graph.

```
import heapq

class Graph:
    def __init__(self, adjac_list):
        self.adjac_list = adjac_list
        print("Input Graph:\n", self.adjac_list)

    def get_neighbors(self, v):
        return self.adjac_list.get(v, [])  # Avoid KeyErrors

    def h(self, n):
        H = {
            'A': 11,
            'B': 6,
            'C': 99,
            'D': 1,
            'E': 7,
            'G': 0,
        }
        return H.get(n, float('inf'))  # Return a high value if heuristic is missing

    def AStar(self, start, stop):
        open_list = []
        heapq.heappush(open_list, (0 + self.h(start), start))  # (f-score, node)

        g = {start: 0}  # Cost from start to node
        parents = {start: None}

        while open_list:
            _, n = heapq.heappop(open_list)  # Get node with lowest f-score

            if n == stop:
                # Reconstruct the shortest path
                path = []
                while n is not None:
                    path.append(n)
                    n = parents[n]
                path.reverse()

                print('Path found:', path)
                print('Cost of the path is:', g[stop])
                return path

            for (m, weight) in self.get_neighbors(n):
                tentative_g_score = g[n] + weight

                if m not in g or tentative_g_score < g[m]:
                    g[m] = tentative_g_score
                    parents[m] = n
                    heapq.heappush(open_list, (g[m] + self.h(m), m))

        print('Path does not exist!')
        return None

# Define the adjacency list
adjac_list = {
    'A': [('B', 2), ('E', 3)],
    'B': [('C', 1), ('G', 9)],
    'C': [],  # Fixed None issue
    'E': [('D', 6)],
    'D': [('G', 1)]
}

# Create the graph and run A* search
graph1 = Graph(adjac_list)
graph1.AStar('A', 'G')
```

Output:

```
Input Graph:
 {'A': [('B', 2), ('E', 3)], 'B': [('C', 1), ('G', 9)], 'C': [], 'E': [('D', 6)], 'D': [('G', 1)]}
Path found: ['A', 'E', 'D', 'G']
Cost of the path is: 10

['A', 'E', 'D', 'G']
```

Program 2:

Implement AO* (AO-star) Search algorithm for finding the shortest path in a weighted graph.

```
class Graph:
    def __init__(self, graph, hVals, startNode):
        self.graph = graph
        self.H = hVals
        self.start = startNode
        self.parent = {}
        self.status = {}
        self.solutionGraph = {}
    def getNeighbors(self, v):
        return self.graph.get(v, [])
    def getStatus(self, v):
        return self.status.get(v, 0)
    def setStatus(self, v, val):
        self.status[v] = val
    def getHval(self, n):
        return self.H.get(n, 0)
    def setHval(self, n, value):
        self.H[n] = value
    def printSolution(self):
        print("\nFinal HEURISTIC VALUES :\n", self.H)
        print("\nBest Path to goal state:")
        print(self.solutionGraph)
        print("\nWith minimum cost:", self.H[self.start])
    def computeMinCost(self, v):
        """Computes the minimum cost of the given node in an AND-OR Graph."""
        minimumCost = float('inf')
        costList = {}
        for nodes in self.getNeighbors(v):
            cost = 0
            nodeList = []
            for c, weight in nodes:
                cost += self.getHval(c) + weight
                nodeList.append(c)
            if cost < minimumCost:
                minimumCost = cost
                costList[minimumCost] = nodeList
        return minimumCost, costList.get(minimumCost, [])
    def AOStar(self, v, backTracking):
        if self.getStatus(v) >= 0:
            minimumCost, childList = self.computeMinCost(v)
            print(f"Processing Node: {v}, Minimum Cost: {minimumCost}")
            self.setHval(v, minimumCost)
            self.setStatus(v, len(childList))
            solved = True
            for childNode in childList:
                self.parent[childNode] = v
                if self.getStatus(childNode) != -1:
                    solved = False
            if solved:
                self.setStatus(v, -1)
                self.solutionGraph[v] = childList
            if v != self.start:
                self.AOStar(self.parent[v], True)
            if not backTracking:
                for childNode in childList:
                    self.setStatus(childNode, 0)
                    self.AOStar(childNode, False)
# Define Heuristic Values
h1 = {
    'A': 0, 'B': 6, 'C': 2, 'D': 12, 'E': 2,
    'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1
}
# Define the Graph Structure
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}
# Execute AO* Algorithm
print("Input Graph:", graph1)
print("\nInitial Heuristic Values:", h1)
G1 = Graph(graph1, h1, 'A')
G1.AOStar('A', False)
G1.printSolution()
# Second Test Case (Commented)
'''
h2 = {
    'A': 1, 'B': 6, 'C': 12, 'D': 10,
    'E': 4, 'F': 4, 'G': 5, 'H': 7
}
graph2 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'D': [[('E', 1), ('F', 1)]]
}
G2 = Graph(graph2, h2, 'A')
G2.AOStar('A', False)
G2.printSolution()
'''
```

Output:

```
Input Graph: {'A': [[('B', 1), ('C', 1)], [('D', 1)]], 'B': [[('G', 1)], [('H', 1)]], 'C': [[('J', 1)]], 'D': [[('E', 1),
('F', 1)]],
'G': [[('I', 1)]]}
Initial Heuristic values {'A': 0, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J':
1} HEURISTIC VALUES :
 {'A': 5, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
Best Path to goal state:
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
 With minimum cost 5
```

<u>Program 3:</u>

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import numpy as np

import pandas as pd

# Load data from CSV file

data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))

# Extract features and target values

concepts = np.array(data.iloc[:, :-1])

print("Concepts:\n", concepts, '\n')

target = np.array(data.iloc[:, -1])

print("Target:\n", target, '\n')

# Function to display general hypotheses

def disp(g):

    for i in range(len(g)):

        print(g[i])

# Candidate Elimination Algorithm

def learn(concepts, target):

    specific_h = concepts[0].copy()  # Initialize specific hypothesis

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]  # Initialize general
hypothesis

    # Iterate over each training example

    for i, h in enumerate(concepts):

        if target[i] == "yes":  # Positive example
```

```
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'  # Generalize specific hypothesis
                    general_h[x][x] = '?'  # Update general hypothesis
        if target[i] == "no":  # Negative example
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]  # Specialize general hypothesis
                else:
                    general_h[x][x] = '?'  # Keep the general hypothesis broad
        print(f"Step {i+1} of Candidate Elimination Algorithm:")
        print(f"Specific_h after Step {i+1}:\n", specific_h)
        print("\nGeneral_h after Step", i+1)
        disp(general_h)
        print('\n')
    # Remove the most general hypotheses (i.e., all '?')
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
# Execute the learning function
s_final, g_final = learn(concepts, target)
# Display final hypotheses
print("Final Specific Hypothesis:", s_final, sep="\n")
print("\nFinal General Hypothesis:")
```

```
disp(g_final)
```

Output:

```
Concepts:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target:
 ['yes' 'yes' 'no' 'yes']

Step 1 of Candidate Elimination Algorithm:
Specific_h after Step 1:
 ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

General_h after Step 1
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']


Step 2 of Candidate Elimination Algorithm:
Specific_h after Step 2:
 ['sunny' 'warm' '?' 'strong' 'warm' 'same']

General_h after Step 2
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']


Step 3 of Candidate Elimination Algorithm:
Specific_h after Step 3:
 ['sunny' 'warm' '?' 'strong' 'warm' 'same']

General_h after Step 3
['sunny', '?', '?', '?', '?', '?']
['?', 'warm', '?', '?', '?', '?']
```

```
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', 'same']


Step 4 of Candidate Elimination Algorithm:
Specific_h after Step 4:
 ['sunny' 'warm' '?' 'strong' '?' '?']

General_h after Step 4
['sunny', '?', '?', '?', '?', '?']
['?', 'warm', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']
['?', '?', '?', '?', '?', '?']


Final Specific Hypothesis:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General Hypothesis:
['sunny', '?', '?', '?', '?', '?']
['?', 'warm', '?', '?', '?', '?']
```

Program 4:

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import pandas as pd
import math
import numpy as np

# Load dataset
data = pd.read_csv("playtennis.csv")

# Extract features
```

```
features = [x for x in data]
print("Features:", features)
features.remove("answer")

# Define Node class for decision tree
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

# Function to calculate entropy
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

# Function to calculate information gain
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain

# ID3 Algorithm for Decision Tree
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""

    # Find attribute with max information gain
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature

    root.value = max_feat
    uniq = np.unique(examples[max_feat])

    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])[0]  # Extracting prediction
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)

            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root

# Function to print decision tree
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")

    if root.isLeaf:
        print(" ->", root.pred)
    else:
        print()

    for child in root.children:
        printTree(child, depth + 1)

# Build and print the decision tree
root = ID3(data, features)
printTree(root)
```

Output:

```
Features: ['outlook', 'temp', 'humidity', 'windy', 'answer']
outlook
        overcast -> yes
        rainy
                windy
                        False -> yes
                        True -> no
        sunny
                humidity
                        high -> no
                        normal -> yes
```

Program 5:

Develop an Artificial Neural Network by implementing the backpropagation algorithm and test the same using appropriate data sets

```
import numpy as np

# Input and output data
x = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

print('Input:\n', x)

# Normalize input and output
x = x / np.amax(x, axis=0)  # Normalize input features
y = y / 100  # Normalize output (assuming max is 100)

# Sigmoid function and its derivative
def sigmoid(x):
   return 1 / (1 + np.exp(-x))

def derivative_sigmoid(x):
   return x * (1 - x)

# Hyperparameters
epoch = 7000
lr = 0.1  # Learning rate

# Network architecture
input_units = 2
```

```
hidden_units = 3
output_units = 1

# Initialize weights and biases with random values
wh = np.random.uniform(size=(input_units, hidden_units))  # Weights for input to hidden layer
bh = np.random.uniform(size=(1, hidden_units))  # Bias for hidden layer
wout = np.random.uniform(size=(hidden_units, output_units))  # Weights for hidden to output layer
bout = np.random.uniform(size=(1, output_units))  # Bias for output layer

# Training the neural network
for i in range(epoch):
    # Forward propagation
    hinp = np.dot(x, wh) + bh
    hout = sigmoid(hinp)

    outinp = np.dot(hout, wout) + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output  # Error at output layer
    ogradient = derivative_sigmoid(output)
    doutput = EO * ogradient  # Gradient at output layer

    EH = doutput.dot(wout.T)  # Error at hidden layer
    hgradient = derivative_sigmoid(hout)
    dhidden = EH * hgradient  # Gradient at hidden layer

    # Update weights and biases
    wout += hout.T.dot(doutput) * lr
    wh += x.T.dot(dhidden) * lr
    bout += np.sum(doutput, axis=0, keepdims=True) * lr
    bh += np.sum(dhidden, axis=0, keepdims=True) * lr

# Print results
print('Normalized Input:\n', x)
print("Actual Output:\n", y)
print("Predicted Output:\n", output)
```

Output:

```
Input:
 [[2. 9.]
 [1. 5.]
 [3. 6.]]
Normalized Input:
 [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
 [[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89554523]
 [0.87856185]
 [0.89472599]]
```

Program 6:

Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd

# Load training dataset
train = pd.read_csv("playtennis.csv")

# Define target and features
target = 'answer'
features = train.columns[train.columns != target]
classes = train[target].unique()

# Load test dataset (skipping first 9 rows, reading next 4 rows)
test = pd.read_csv("playtennis.csv", skiprows=range(1, 10), nrows=4)
print('Test dataset\n', test)

# Initialize prior and likelihood probability dictionaries
priorprob = {}
likeprob = {}

# Compute prior probabilities and likelihoods
for x in classes:
    traincl = train[train[target] == x][features]
    tot = len(traincl)
    priorprob[x] = float(tot / len(train))
    clsp = {}
    for col in traincl.columns:
        colp = {}
        for val, cnt in traincl[col].value_counts().items():  # Changed iteritems() to items()
            pr = cnt / tot
            colp[val] = pr
        clsp[col] = colp

    likeprob[x] = clsp

# Function to compute posterior probabilities
def postprobs(x):
    postprob = {}

    for cl in classes:
        pr = priorprob[cl]
        for col, val in x.items():  # Changed iteritems() to items()
            try:
                pr *= likeprob[cl][col][val]
            except KeyError:
                pr = 0  # If value not found in training, set probability to 0
        postprob[cl] = pr

    return postprob

# Function to classify a sample based on probabilities
def classify(x):
    postprob = postprobs(x)
    maxclass = max(postprob, key=postprob.get)  # Get class with max probability
    return maxclass

# Evaluate accuracy on training data
correct_preds = sum(classify(train.loc[i, features]) == train.loc[i, target] for i in train.index)
print(correct_preds, "correct of", len(train))
print("Training Accuracy:", correct_preds / len(train))
```

```
# Evaluate accuracy on test data
correct_preds_test = sum(classify(test.loc[i, features]) == test.loc[i, target] for i in test.index)
print(correct_preds_test, "correct of", len(test))
print("Test Accuracy:", correct_preds_test / len(test))
```

Output:

```
Test dataset
    outlook  temp humidity  windy answer
0     rainy  mild   normal  False    yes
1     sunny  mild   normal   True    yes
2  overcast  mild     high   True    yes
3  overcast   hot   normal  False    yes
13 correct of 14
Training Accuracy: 0.9285714285714286
4 correct of 4
Test Accuracy: 1.0
```

PART-B

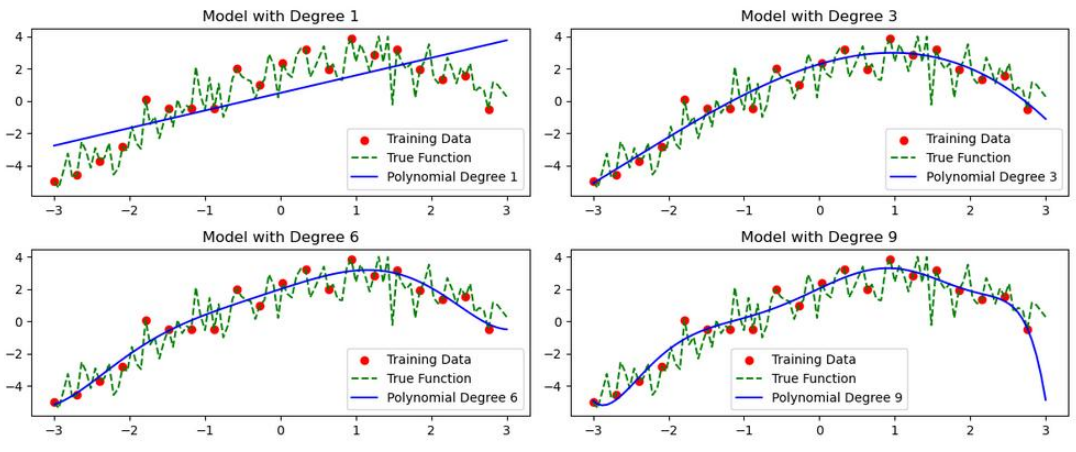
Program 7:

Demonstrate the Bias-Variance Trade-off in a machine learning model, provide visualizations and insights.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error

# Generate synthetic dataset
np.random.seed(42)
X = np.linspace(-3, 3, 100).reshape(-1, 1)
y_true = 2 + X - 0.5 * X**2 + np.random.normal(0, 1, X.shape)  # Quadratic relationship
with noise

# Training data (subset)
X_train = X[::5]  # Pick every 5th point
```

```
y_train = y_true[::5]

# Plot true function
plt.figure(figsize=(12, 5))

degrees = [1, 3, 6, 9]  # Different polynomial degrees

for i, d in enumerate(degrees, 1):
   plt.subplot(2, 2, i)
   model = make_pipeline(PolynomialFeatures(d), LinearRegression())   # Polynomial
regression
   model.fit(X_train, y_train)
   y_pred = model.predict(X)

   plt.scatter(X_train, y_train, color='red', label='Training Data')
   plt.plot(X, y_true, label='True Function', linestyle='dashed', color='green')
   plt.plot(X, y_pred, label=f'Polynomial Degree {d}', color='blue')

   plt.title(f"Model with Degree {d}")
   plt.legend()

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

Output:



Program 8:

Demonstrate Association Rule Mining using the FP-Growth Algorithm. Utilize a suitable dataset to identify frequent item sets and generate association rules.

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load dataset from CSV file (Ensure correct formatting: no headers, direct transactions)
df = pd.read_csv("Transaction.csv")

# Convert the 'Items Purchased' column into a list of lists
transactions = df["Items Purchased"].apply(lambda x: x.split(", ")).tolist()

# Encode transactions using TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Apply FP-Growth algorithm to get frequent item sets (adjust min_support as needed)
frequent_itemsets = fpgrowth(df_encoded, min_support=0.3, use_colnames=True)

# Display frequent itemsets
print("Frequent Itemsets:\n", frequent_itemsets)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

# Display association rules
print("\nAssociation Rules:\n", rules)
```

Output:

```
Frequent Itemsets:

    support  itemsets
0      0.6     (B)
1      0.6     (A)
2      0.4     (C)
3      0.6     (D)
4      0.6     (E)
5      0.4   (A, B)
6      0.3   (A, D)
7      0.3   (A, E)
8      0.3   (A, C)
9      0.3   (E, C)
10      0.4   (E, D)

Association Rules:
     antecedents consequents  antecedent_support  consequent_support  support\confidence
```

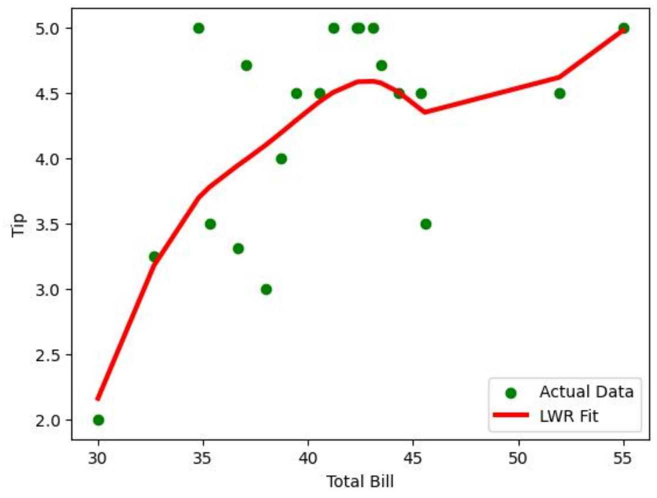
```
0       (A)        (B)              0.6         0.6         0.4
1       (B)        (A)              0.6         0.6         0.4
2       (A)        (C)              0.6         0.4         0.3
3       (C)        (A)              0.4         0.6         0.3
4       (E)        (C)              0.6         0.4         0.3
5       (C)        (E)              0.4         0.6         0.3
6       (E)        (D)              0.6         0.6         0.4
7       (D)        (E)              0.6         0.6         0.4
```

Program 9:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Function to compute weights based on Gaussian kernel
def kernel(point, X, k):
    m, n = np.shape(X)
    weights = np.eye(m)
    for j in range(m):
        diff = point - X[j]
        weights[j, j] = np.exp(diff @ diff.T / (-2.0 * k**2))
    return weights

# Function to compute local weight for a given point
def localWeight(point, X, y, k):
    wei = kernel(point, X, k)  # Compute weights
    theta = np.linalg.pinv(X.T @ (wei @ X)) @ (X.T @ (wei @ y))  # Compute parameters
    return theta

# Function for Locally Weighted Regression
def localWeightRegression(X, y, k):
    m, n = np.shape(X)
    ypred = np.zeros(m)
    for i in range(m):
        theta = localWeight(X[i], X, y, k)  # Compute local weight
        ypred[i] = X[i] @ theta  # Predict value
    return ypred

# Function to plot results
def graphPlot(X, ypred, bill, tip):
    sort_index = np.argsort(X[:, 1])  # Sort by X values
    x_sorted = X[sort_index]  # Sort X values
    y_sorted = ypred[sort_index]  # Sort predictions

    plt.scatter(bill, tip, color='green', label="Actual Data")
    plt.plot(x_sorted[:, 1], y_sorted, color='red', linewidth=3, label="LWR Fit")

    plt.xlabel('Total Bill')
    plt.ylabel('Tip')
    plt.legend()
    plt.show()

# Load dataset
data = pd.read_csv('data10.csv')

# Extract features and target
bill = data['total_bill'].values
tip = data['tip'].values

# Prepare X and y matrices
m = len(bill)
X = np.column_stack((np.ones(m), bill))  # Add bias term
y = tip.reshape(-1, 1)  # Reshape target to column vector

# Perform Locally Weighted Regression
k = 3  # Bandwidth parameter
ypred = localWeightRegression(X, y, k)

# Plot results
graphPlot(X, ypred, bill, tip)
```

Output:



Program 10:

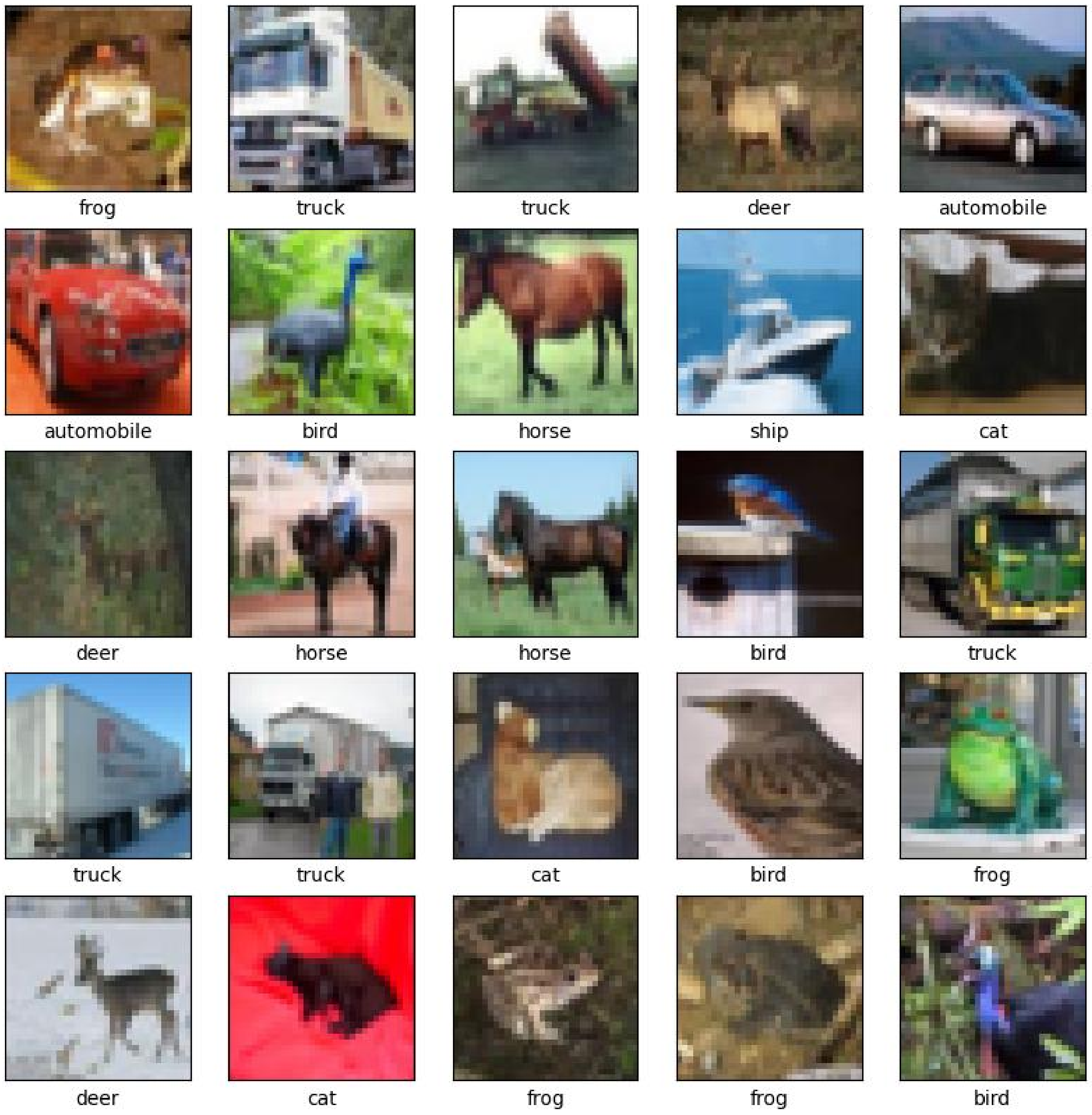
Develop a Convolutional Neural Network (CNN) and test it using appropriate datasets. Implement key components such as convolutional layers, pooling layers, and fully connected layers. Evaluate the CNN's performance on tasks such as image classification.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Class names for CIFAR-10 dataset
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Plot a few sample images from the dataset
plt.figure(figsize=(10,10))
for i in range(25):
   plt.subplot(5,5,i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(train_images[i], cmap=plt.cm.binary)
   plt.xlabel(class_names[train_labels[i][0]])
plt.show()

# Build the CNN model
model = models.Sequential()

# 1st Convolutional Layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# 2nd Convolutional Layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# 3rd Convolutional Layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
# Flatten and add fully connected layers
model.add(layers.Flatten())  # Flatten the 3D output to 1D
model.add(layers.Dense(64, activation='relu'))  # Fully connected layer
model.add(layers.Dense(10))  # Output layer (10 classes for CIFAR-10)

# Summary of the model architecture
model.summary()

# Compile the model
model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10,
              validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print("\nTest accuracy:", test_acc)

# Plot accuracy and loss graphs
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

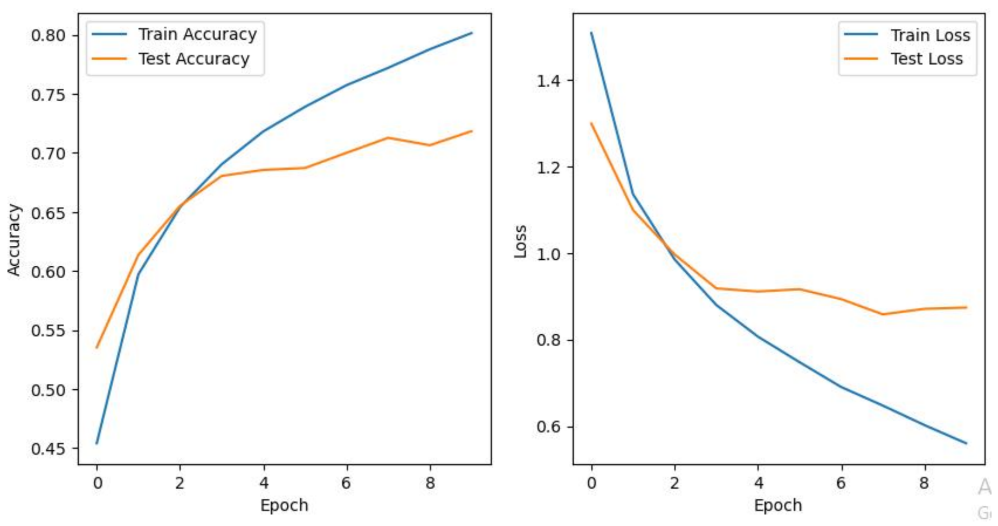
Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 31s 0us/step
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0 )

 conv2d_1 (Conv2D)          (None, 13, 13, 64)       18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0  2D)

 conv2d_2 (Conv2D)          (None, 4, 4, 64)         36928

 flatten (Flatten)          (None, 1024)             0

 dense (Dense)              (None, 64)               65600

 dense_1 (Dense)            (None, 10)               650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 57s 34ms/step - loss: 1.5081 - accuracy: 0.4542
- val_loss: 1.2989 - val_accuracy: 0.5353
Epoch 2/10
1563/1563 [==============================] - 51s 33ms/step - loss: 1.1359 - accuracy: 0.5973
- val_loss: 1.0992 - val_accuracy: 0.6137
Epoch 3/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.9851 - accuracy: 0.6541
- val_loss: 0.9965 - val_accuracy: 0.6552
Epoch 4/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.8799 - accuracy: 0.6903
- val_loss: 0.9182 - val_accuracy: 0.6805
Epoch 5/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.8068 - accuracy: 0.7182
- val_loss: 0.9109 - val_accuracy: 0.6856
Epoch 6/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.7476 - accuracy: 0.7390
- val_loss: 0.9163 - val_accuracy: 0.6872
Epoch 7/10
1563/1563 [==============================] - 48s 31ms/step - loss: 0.6903 - accuracy: 0.7574
- val_loss: 0.8934 - val_accuracy: 0.7001
Epoch 8/10
1563/1563 [==============================] - 47s 30ms/step - loss: 0.6474 - accuracy: 0.7720
- val_loss: 0.8581 - val_accuracy: 0.7128
Epoch 9/10
```

```
1563/1563 [==============================] - 48s 31ms/step - loss: 0.6023 - accuracy: 0.7878
- val_loss: 0.8709 - val_accuracy: 0.7065
Epoch 10/10
1563/1563 [==============================] - 50s 32ms/step - loss: 0.5603 - accuracy: 0.8014
- val_loss: 0.8739 - val_accuracy: 0.7184
313/313 - 2s - loss: 0.8739 - accuracy: 0.7184 - 2s/epoch - 8ms/step

Test accuracy: 0.7184000015258789
```



<u>Program 11:</u>

Implement Q learning algorithm.

```
import numpy as np
import random

# Define the environment (5x5 Grid)
grid_size = 5
num_states = grid_size * grid_size  # 25 states
num_actions = 4  # Up, Down, Left, Right

# Define rewards: Goal at (4,4) gives +100 reward
rewards = np.full((grid_size, grid_size), -1)  # Negative reward for all steps
rewards[4, 4] = 100  # Goal reward

# Define Q-table
q_table = np.zeros((num_states, num_actions))
```

```
# Define hyperparameters
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
epsilon = 1.0  # Exploration rate
epsilon_decay = 0.99  # Reduce exploration over time
episodes = 500  # Number of episodes

# Action Mapping: [Up, Down, Left, Right]
actions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Convert (row, col) to state index
def state_index(row, col):
    return row * grid_size + col

# Convert state index to (row, col)
def state_position(index):
    return index // grid_size, index % grid_size

# Choose action using epsilon-greedy strategy
def choose_action(state):
    if random.uniform(0, 1) < epsilon:  # Explore
        return random.randint(0, num_actions - 1)
    else:  # Exploit
        return np.argmax(q_table[state])

# Q-Learning Algorithm
for episode in range(episodes):
    state = state_index(0, 0)  # Start at (0,0)
    done = False

    while not done:
        row, col = state_position(state)
        action = choose_action(state)

        # Determine next state
        new_row = max(0, min(grid_size - 1, row + actions[action][0]))
        new_col = max(0, min(grid_size - 1, col + actions[action][1]))
        new_state = state_index(new_row, new_col)

        # Get reward
        reward = rewards[new_row, new_col]

        # Q-value update using Bellman equation
        q_table[state, action] = (1 - alpha) * q_table[state, action] + \
                                 alpha * (reward + gamma * np.max(q_table[new_state]))
        state = new_state

        # If reached goal
        if (new_row, new_col) == (4, 4):
            done = True

    # Decay epsilon (reduce exploration over time)
    epsilon *= epsilon_decay

# Display trained Q-table
print("Trained Q-Table:")
print(q_table)

# Test the learned policy
def test_policy():
    state = state_index(0, 0)  # Start from (0,0)
    path = [state_position(state)]

    for _ in range(20):  # Limit steps to avoid infinite loops
        action = np.argmax(q_table[state])
        row, col = state_position(state)
        new_row = max(0, min(grid_size - 1, row + actions[action][0]))
        new_col = max(0, min(grid_size - 1, col + actions[action][1]))
        state = state_index(new_row, new_col)
        path.append((new_row, new_col))

        if (new_row, new_col) == (4, 4):  # Goal reached
            break

    return path

print("\nOptimal Path Found by Q-Learning:")
print(test_policy())
```

Output:

```
Trained Q-Table:
[[ 30.68667768  13.55112819  27.16310377  42.612659  ]
 [ 30.82677048  23.32087415  27.66681799  48.45851   ]
 [ 35.25412791  54.9539      29.51026342  35.77442977]
 [ 11.8244467   60.29999354  11.21148379  16.21637002]
 [ 10.03560489  61.82388985   9.30319861   8.04264515]
 [ -0.66317417  -1.64556086  -0.46900756  32.85965503]
 [  7.39656747   3.21927504   4.62262023  52.8218184 ]
 [ 37.92655672  30.54463611  31.79800302  62.171     ]
 [ 41.53094577  55.66837638  48.73473004  70.19      ]
 [ 38.64146895  79.1         47.1765207   67.33162745]
 [ -1.48461745  -1.393186    -1.39391983   2.51999665]
 [  0.15945983   0.29289911  -1.12148711  16.35594628]
 [ 12.18279052  19.22467218   3.69777742  51.8593129 ]
 [ 28.52003997  38.36233327  10.48361022  78.22910274]
 [ 58.63507978  89.          54.43306423  69.98130924]
 [ -1.26008347  -0.83434171  -1.07381043   0.11786469]
 [ -0.66371845  -0.42025262  -0.94556867  11.72555828]
 [  1.52629983   8.71489593   1.97053959  46.41429623]
 [  4.40808517  31.36519724   9.00203418  86.64339984]
 [ 64.80999205 100.          63.36604272  79.47250843]
 [ -0.73948166  -0.58519851  -0.65012229  -0.50388496]
 [ -0.40914548  -0.50168359  -0.78232513   3.73130847]
 [  5.00689635   0.67813552  -0.36145028  29.14031499]
 [  5.69010387  15.07395284   1.92331807  79.41088679]
 [  0.           0.           0.           0.        ]]

Optimal Path Found by Q-Learning:
[(0, 0), (0, 1), (0, 2), (1, 2), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4)]
```

Program 12:

Develop a simple Chabot using rule-based responses.

```
import random

def chatbot_response(user_input):
    user_input = user_input.lower()

    responses = {
        "hello": ["Hi there!", "Hello!", "Hey! How can I help you?"],
        "how are you": ["I'm just a bot, but I'm doing great!", "I'm fine, thanks for asking!", "Doing well!"],
        "what is your name": ["I'm a chatbot!", "You can call me ChatBot.", "I don't have a name, but I'm 
here to help!"],
        "bye": ["Goodbye!", "See you later!", "Bye! Have a great day!"],
        "help": ["I can help you with general queries. Try asking about weather, time, or basic questions!"],
        "who created you": ["I was created by a Python enthusiast!", "A developer built me using Python!", 
"I exist because of code!"]
    }

    for key in responses:
        if key in user_input:
            return random.choice(responses[key])
    return "I'm sorry, I don't understand that. Can you rephrase?"

# Simple chatbot loop
print("ChatBot: Hello! Type 'bye' to exit.")

while True:
    user_input = input("You: ")
    if user_input.lower() == "bye":
        print("ChatBot: Goodbye! Have a nice day!")
        break
    response = chatbot_response(user_input)
    print("ChatBot:", response)
```

Output:

```
ChatBot: Hello! Type 'bye' to exit.
You: how are you
ChatBot: I'm just a bot, but I'm doing great!
```

Viva questions

1. What is machine learning?
   Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.
2. Define supervised learning
   Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

   Y = f(X)

3. Define unsupervised learning
   Unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

4. Define semi supervised learning
   Labeled data is used to help identify *that* there are specific groups of webpage types present in the data and what they *might be.* The algorithm is then trained on unlabeled data to define the boundaries of those webpage types and may even identify new types of web pages that were unspecified in the existing human-inputted labels.

5. Define reinforcement learning
   Reinforcement learning is an important type of Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results.

6. What do you mean by hypotheses
   A hypothesis is an educated prediction that can be tested. You will discover the purpose of a hypothesis then learn how one is developed and written.

7. What is classification
   classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

8. What is clustering
   Clustering: is the assignment of a set of observations into subsets (calledclusters) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning
9. Define precision, accuracy and recall
   precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Accuracy is

the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

10. Define entropy
    Entropy, as it relates to machine learning, is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.
11. Define regression
    Regression is basically a statistical approach to find the relationship between variables. In machine learning, this is used to predict the outcome of an event based on the relationship between variables obtained from the data-set.
12. How Knn is different from k-means clustering

    K-Means: it is an Unsupervised learning technique. It is used for Clustering. n training phase of K-Means, K observations are arbitrarily selected (known as centroids). Each point in the vector space is assigned to a cluster represented by nearest (euclidean distance) centroid. Once the clusters are formed, for each cluster the centroid is updated to the mean of all cluster members. And the cluster formation restarts with new centroids. This repeats until the centroids themselves become mean of clusters,

    KNN: It is a Supervised learning technique. It is used mostly forClassification, and sometimes even for Regression. K-NN doesn't have a training phase as such. But the prediction of a test observation is done based on the K-Nearest (often euclidean distance) Neighbours (observations) based on weighted averages/votes.

13. What is concept learning
    Concept learning also refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner simplifies what has been observed by condensing it in the form of an example.
14. Define specific boundary and general boundary
    The general boundary G, with respect to hypothesis space H and training data D, is the set of maximally general hypotheses consistent with D.
    The specific boundary S, with respect to hypothesis space H and training data D, is the set of maximally specific hypotheses consistent with D
15. Define target function
    This is a function that knows and maps a full-relationship of the features/input variables to the Response/Output variable.
16. Define decision tree
    Decision Tree create a training model which can use to predict class or value of target variables by **learning decision rules** inferred from prior data(training data)
17. What is ANN
    An artificial neuron network (ANN) is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes - or learns, in a sense - based on that input and output.
18. Explain gradient descent approximation

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

19. State Bayes theorem

Bayes' Theorem is the fundamental result of probability theory – it puts the posterior probability P(H|D) of a hypothesis as a product of the probability of the data given the hypothesis(P(D|H)), multiplied by the probability of the hypothesis (P(H)), divided by the probability of seeing the data.

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

20. Define Bayesian belief networks

It is a probabilistic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG).

21. Differentiate hard and soft clustering

In hard clustering, each data point either belongs to a cluster completely or not. In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned.

22. Define variance

Variance is a measurement of the spread between numbers in a data set. The variance measures how far each number in the set is from the mean. Variance is calculated by taking the differences between each number in the set and the mean, squaring the differences (to make them positive) and dividing the sum of the squares by the number of elements in the set.

23. What is inductive machine learning

From the perspective of inductive learning, we are given input samples (x) and output samples (f(x)) and the problem is to estimate the function (f). Specifically, the problem is to generalize from the samples and the mapping to be useful to estimate the output for new samples in the future.

24. Define pruning

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

25. Define Bias
Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

26. What is Artificial Intelligence?
Artificial Intelligence is the field of computer science concerned with building intelligent machines or computer systems, capable of simulating human intelligence. The machines created using Artificial Intelligence can work and react like humans without human intervention. Speech Recognition, Customer service, Recommendation Engine, Natural Language Processing (NLP) are some of the applications of Artificial Intelligence.

27. What are some real-life applications of Artificial Intelligence?

- Social Media: The most common use of Artificial Intelligence in social media is facial detection and verification. Artificial Intelligence, along with machine learning, is also used to design your social media feed.
- Personalized online shopping: Shopping sites use AI-powered algorithms to curate the list of buying recommendations for users. They use data like users' search history and recent orders to create a list of suggestions that users might like.
- Agriculture: Technologies, especially Artificial Intelligence embedded systems, help farmers protect their crops from various adversities like weather, weeds, pests, and changing prices.
- Smart cars: Smart cars are another one of the real-life applications of AI. Artificial intelligence collects data from a car's radar, camera, and GPS to operate the vehicle when the autopilot mode is on.
- Healthcare: Artificial Intelligence has come out as a reliable friend of doctors. From intelligent testing to medical recommendations, they assist medical professionals in every possible way

28. What are different platforms for Artificial Intelligence (AI) development?

Some different software platforms for AI development are-

- Amazon AI services
- Tensorflow
- Google AI services
- Microsoft Azure AI platform
- Infosys Nia
- IBM Watson
- H2O
- Polyaxon
- PredictionIO

29. What are the programming languages used for Artificial Intelligence?
Python, LISP, Java, C++, R are some of the programming languages used for Artificial Intelligence.

30. What is the future of Artificial Intelligence?
Artificial Intelligence has affected many humans and almost every industry, and it is expected to continue to do so. Artificial Intelligence has been the main driver of emerging technologies like the Internet of Things, big data, and robotics. AI can harness the power of a massive amount of data and make an optimal decision in a fraction of seconds, which is almost impossible for a normal human. AI is leading areas that are important for mankind such as cancer research, cutting-edge climate change technologies, smart cars, and space exploration. It has taken the center stage of innovation and development of computing, and it is not ceding the stage in the foreseeable future. Artificial Intelligence is going to impact the world more than anything in the history of mankind.

31. What is the difference between Artificial Intelligence and Machine learning?

Artificial Intelligence	Machine Learning
Artificial Intelligence is the ability of machines to function like the human brain.	Machine learning is processing data, learning from it, and then making informed decisions.
The goal of AI is to allow machines to think for themselves without the need for human involvement.	The purpose of machine learning is to allow a machine to learn from its previous experiences.
AI is capable of dealing with both structured and semi-structured data.	Machine learning works with both organized and semi-structured data.
AI is a subset of data science.	Machine Learning is a subset of AI.
Example- Google Search engine	Example- Image recognition

32. How are Artificial Intelligence and Machine Learning related?

Artificial Intelligence and Machine Learning are two popular and often misunderstood words. Artificial Intelligence is a domain of computer science that enables machines to mimic human intelligence and behaviour. On the other hand, Machine Learning is a subset of Artificial Intelligence and is all about feeding computers with data so that they can learn on their own from all the patterns and models. Machine Learning models are used to implement Artificial Intelligence frequently.