Department of Artificial Intelligence and Machine Learning

Annamacharya Institute of Technology and Sciences: Rajampet (Autonomous)

(Affiliated to J.N.T. University, Anatapur)
New Boyanapalli, Rajampet – 516126 Kadapa (Dt), A.P.

LAB MANUAL

III Year II Semester

Academic Year

2024

Department of Artificial Intelligence and Machine Learning

Annamacharya Institute of Technology and Sciences: Rajampet (Autonomous)

(Affiliated to J.N.T. University, Anatapur)
New Boyanapalli, Rajampet – 516126 Kadapa (Dt), A.P.

MACHINE LEARNING LAB (20A3363L)

III B.Tech AI&ML II Semester

Prepared By

P.MABJAN

Department of MCA

Department of Artificial Intelligence and Machine Learning

Annamacharya Institute of Technology and Sciences: Rajampet (Autonomous)

(Affiliated to J.N.T. University, Anatapur)

New Boyanapalli, Rajampet – 516126 Kadapa (Dt), A.P.

LAB MANUAL – INDEX

VISION AND MISSION OF THE INSTITUTION

Vision

We impart futuristic technical education and instil high patterns of discipline through our dedicated staff who set global standards, making our students technologically superior and ethically strong, who in turn shall improve the quality of life of the human race.

Mission

Our mission is to educate students from the local and rural areas and from other states so that they become enlightened individuals, improving the living standards of their families, industry and society. We provide individual attention, world-class quality of Technical education and take care of character building.

ANNAMACHARYA INSTITUTE OF TECHNOLOGY & SCIENCES::RAJAMPET
(Autonomous)

Department of Artificial Intelligence and Machine Learning

Vision and Mission of the Department

Vision

The vision of the Department of Artificial Intelligence and Machine Learning is to impart quality education and produce high quality, creative and ethical engineers, in still professionalism, enhance students' problem solving skills in the domain of artificial intelligence and Machine Learning To emerge as a premier center for education and research in Artificial Intelligence and Machine Learning and in transforming students into innovative professionals of contemporary and future technologies to cater the global needs of human resources for IT and ITES companies.

Mission

- To provide skill based education to master the students in problem solving and analytical skills to enhance their niche expertise in the field Artificial Intelligence and Machine Learning.
- To explore opportunities for skill development in the application of Artificial Intelligence and Machine learning among rural and under privileged population.
- Transform professionals into technically competent through innovation and socially responsible.
- To promote research based projects and activities among the students in the emerging areas of Artificial Intelligence and Machine Learning.
- To impart quality and value based education and contribute towards the innovation of computing system, data science to raise satisfaction level of all stakeholders.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1:- To Formulate, analyze and solve Engineering problems with strong foundation in Mathematical, Scientific, Engineering fundamentals and modern computing practices through advanced curriculum.

PEO2:- Analyze the requirements, realize the technical specification and design the Engineering solutions by applying artificial intelligence and data science theory and principles.

PEO3:- Demonstrate technical skills, competency in AI and Machine Learning and promote collaborative learning and team work spirit through multi -disciplinary projects and diverse professional activities.

PEO4:- Equip the graduates with strong knowledge, competence and soft skills that allows them to contribute ethically to the needs of society and accomplish sustainable progress in the emerging computing technologies through life-long learning.

PROGRAM OUTCOMES

A graduate of Artificial Intelligence & Machine Learning will have ability to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. 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 information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. 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.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10. Communication: Communicate effectively on complex engineering activities with then engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. 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
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMME SPECIFIC OUTCOMES

PSO1:- Understand, analyze and develop essential proficiency in the areas related to artificial Intelligence and Machine Learning in terms of underlying statistical and computational principles and apply the knowledge to solve practical problems.

PSO2:- Learn the basic concepts of Artificial Intelligence and Machine Learning and to apply them to various areas, like Image Processing, Speech Recognition, Product Recommendations, Medical Diagnosis etc.

PSO3:- Solutions to complex problems, using latest hardware and software tools, along with analytical skills to arrive at cost effective and appropriate solutions.

Course Outcomes:

At the end of the course, the student will be able to Blooms Level of Learning 1. Understand the basic knowledge about the key algorithms of machine learning L1

2. Learn and use different machine learning algorithms L2
3. Apply various machine learning algorithms Bayesian learning and genetic approaches L3
4. Design the classification, pattern recognition, optimization and decision problems using machine learning algorithms L4
5. Analyze different types of learning approaches L5

CO-PO MAPPING

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
20A3062L.1	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3
20A3062L.2	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3
20A3062L.3	3	3	3	3	-	3	-	-	-	-	-	3	3	3	3
20A3062L.4	3	3	3	3	-	3	-	-	-	-	-	3	3	3	3
20A3062L.5	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3

Department of Artificial Intelligence and Machine Learning

ANNAMACHARYA INSTITUTE OF TECHNOLOGY & SCIENCES::RAJAMPET
(Autonomous)

Title of the Course	Machine Learning Lab
Category	PCC (LAB)
Course Code	20A3363L

Year	III B.Tech
Semester	II Semester
Branch	AI&DS

Lecture Hours	**Tutorial Hours**	**Practice Hours**	**Credits**
0	0	3	1.5

Course Objectives: Students will write a program for

- Formulate machine learning problems corresponding to different applications
- Make use of Data sets in implementing the machine learning algorithms
- Implement the machine learning concepts and algorithms in any suitable language of choice
- Implement different types of learning approaches
- Understand machine learning algorithms along with their strengths and weaknesses.

List of Programs

Exercise 1: Introduction to Python-based notebook environments.

Exercise 2: Implement A* algorithm for one the following problems: i) 8 puzzle ii) Missionaries and Cannibals

Learning Outcomes: At the end of the module, the student will be able to:

- Explore and build computer program that improve their performance of a task through experience. (L4)
- Implements AI based search strategy for 8-puzzle and Missionaries and Cannibals problem (L6)

Exercise 3: Implement and test hill climbing based search algorithms to solve Travelling Salesman Problem
Exercise 4: Solve and implement map coloring problem by backtracking and constraint propagation
Exercise 5: Solve and implement the game of tic-tac-toe using mini-max
Learning Outcomes: At the end of the module, the student will be able to:

- Implements AI based search strategy for travelling salesmen problem (L6)
- Implements AI based search strategy for backtracking and tic-tac-toe (L6)

Exercise 6: 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 use it to classify a new sample
Exercise 7: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets
Learning Outcomes: At the end of the module, the student will be able to:

- Solve learning problem using Decision Tree (L5)
- Implements learning problem using Back propagation (L5)

Exercise 8: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Calculate the accuracy, precision, and recall for your data set.

Exercise 9: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate

the diagnosis of heart patients using any standard Heart Disease Data Set.
Learning Outcomes: At the end of the module, the student will be able to:

- implement the principles of Bayes Probability for classification (L6)
- Demonstrate the diagnosis of heart patients using any standard Heart Disease Data Set (L3)

Exercise 10: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

Exercise 11: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions
Learning Outcomes: At the end of the module, the student will be able to:

- Apply the principles of Probability for classification as an important area of Machine Learning Algorithms (L3)
- Demonstrate the K-nearest neighbor algorithm on iris data and evaluate its performance. (L3, L5)

Exercise 12: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw the corresponding graphs.

Exercise 13: Write a program to implement 5-fold cross validation on a given dataset. Compare the accuracy, precision, recall, and F-score for your data set for different folds.

Exercise 14: write a simple program to implement Reinforcement Learning.
Learning Outcomes: At the end of the module, the student will be able to:

- Implements Locally Weighted Regression algorithm and k-fold cross validation on a given dataset. (L6).
- Implement RL on an industry related task. (L6).

Prescribed Text Books:

1. Machine Learning - Tom M. Mitchell, - MGH
2. Machine Learning: An Algorithmic Perspective, Stephen Marsland, Taylor & Francis (CRC) Reference

Reference Books:

1. Machine Learning Methods in the Environmental Sciences, Neural Networks, William W Hsieh, Cambridge Univ Press
2. Richard o. Duda, Peter E. Hart and David G. Stork, pattern classification, John Wiley & Sons Inc., 2001
3. Chris Bishop, Neural Networks for Pattern Recognition, Oxford University Press

Course Outcomes:

At the end of the course, the student will be able to	Blooms Level of Learning
1. Understand the basic knowledge about the key algorithms of machine learning	L1
2. Learn and use different machine learning algorithms	L2
3. Apply various machine learning algorithms Bayesian learning and genetic approaches	L3
4. Design the classification, pattern recognition, optimization and decision problems using machine learning algorithms	L4
5. Analyze different types of learning approaches	L5

CO-PO Mapping:

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
20A3062L.1	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3
20A3062L.2	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3
20A3062L.3	3	3	3	3	-	3	-	-	-	-	-	3	3	3	3
20A3062L.4	3	3	3	3	-	3	-	-	-	-	-	3	3	3	3
20A3062L.5	3	3	3	3	-	-	-	-	-	-	-	3	3	3	3

List of Experiments:-

Exercise 1: Introduction to Python-based notebook environments.

AIM: Introduction to Python-based notebook environments

DESCRIPTION:

A Python-based notebook environment is an interactive computational environment that allows users to write and execute code, visualize data, and document their workflow in a single, cohesive interface. These environments are widely used in data science, machine learning, and scientific computing due to their versatility and ease of use.

PYTHON-BASED NOTEBOOK ENVIRONMENTS:

1. JUPYTER NOTEBOOK:

Jupiter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It's widely used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Key Features:

Interactive Code Execution: Run code in small, manageable chunks or cells, and view the results immediately.

Rich Text Support: Use Markdown and LaTeX to add formatted text, equations, and other descriptive elements.

Data Visualization: Integrates with libraries like Matplotlib, Seaborn, and Plotly to create inline visualizations.

Language Support: Primarily used for Python but supports over 40 programming languages via kernels, such as R, Julia, and Scala.

Inline Output: Display outputs, such as graphs, tables, and images, directly within the notebook cells.

Reproducibility: Combine code, output, and documentation in a single document, making it easy to reproduce and share analyses.

Extensibility: Enhance functionality with a wide range of extensions for code linting, execution timing, and more.

Collaboration: Share notebooks via email, GitHub, or Jupyter's own nbviewer, and collaborate with others easily.

2. GOOGLE COLAB:

Google Colab, short for Colaboratory, is a free, cloud-based Jupyter notebook environment provided by Google. It allows users to write and execute Python code through the web, offering a powerful platform for machine learning, data analysis, and education.

Key Features:

Free Access: Provides free access to computing resources, including CPUs, GPUs, and TPUs.

Cloud-Based: No installation required; access your notebooks from any device with an internet connection.

Collaboration: Easily share notebooks and collaborate with others in real-time, similar to Google Docs.

Integration with Google Drive: Save and access notebooks directly from Google Drive.

Pre-installed Libraries: Comes with many popular Python libraries pre-installed, such as TensorFlow, Keras, and PyTorch.

Interactive Code Execution: Execute code in cells and see results immediately, with support for rich text and Markdown.

Data Visualization: Integrates with libraries like Matplotlib and Plotly for inline visualizations.

Notebook Sharing: Share notebooks via links, making it easy to distribute and collaborate on projects.

3. PYCHARM:

PyCharm is a widely-used Integrated Development Environment (IDE) specifically designed for Python development. Developed by JetBrains, it offers a robust set of tools and features to streamline the coding, debugging, and deployment processes.

Key Features:

Code Editor: Advanced code editor with syntax highlighting, code completion, and intelligent code navigation.

Debugger: Powerful integrated debugger with breakpoints, watches, and interactive console.

Version Control: Seamless integration with version control systems like Git, SVN, and Mercurial.

Refactoring Tools: Extensive refactoring capabilities to improve code structure and maintainability.

Testing: Built-in support for testing frameworks such as pytest, unittest, and Nose.

Web Development: Support for web frameworks like Django, Flask, and Pyramid, with templates and tools specific to web development.

Database Tools: Built-in tools to manage and query databases, with support for SQL, MongoDB, and more.

Code Analysis: Static code analysis to identify potential errors and improve code quality.

Plugins: Extensive plugin ecosystem to extend functionality, including support for other languages and frameworks.

4. PYTHON IDLE:

IDLE (Integrated Development and Learning Environment) is a simple, lightweight IDE for Python provided with the standard Python distribution. It is designed for beginners and those who need a quick and easy way to write, test, and debug Python code.

Key Features:

Built-In Interpreter: Interactive shell for executing Python commands and testing code snippets in real-time.

Code Editor: Basic text editor with syntax highlighting, auto-indentation, and line numbering.

Debugger: Integrated debugger with stepping, breakpoints, and call stack visibility.

Execution: Run Python scripts directly from the editor or the interactive shell.

Output Window: Separate output window to display program results, helping to keep the workspace organized.

Cross-Platform: Available on Windows, macOS, and Linux, making it accessible for all Python users.

Lightweight: Minimal setup and resource requirements, ideal for small projects and learning environments.

Ease of Use: Simple, intuitive interface suitable for beginners and educational purposes.

5. IBM WATSON STUDIO:

IBM Watson Studio is a cloud-based platform designed for data scientists, application developers, and subject matter experts to collaboratively and easily work with data. It offers a comprehensive suite of tools for data analysis, machine learning, and AI development.

Key Features:

Integrated Environment: Combines multiple tools for data preparation, analysis, modeling, and deployment in one platform.

Collaboration: Facilitates team collaboration with project-based workspaces, version control, and sharing capabilities.

Data Refinery: Provides data wrangling tools to clean, shape, and enrich data for analysis.

Machine Learning: Supports building, training, and deploying machine learning models with automated machine learning (AutoML) and custom model creation.

Jupyter Notebooks: Integrated Jupyter Notebook support for interactive coding in Python, R, and Scala.

SPSS Modeler: Includes SPSS Modeler for advanced statistical analysis and predictive modeling.

AutoAI: Automates the end-to-end data science workflow, including data preprocessing, model selection, and hyperparameter optimization.

Deployment: Simplifies model deployment and management with tools for real-time and batch scoring.

Scalability: Leverages IBM Cloud for scalable computing resources, accommodating large datasets and complex models.

Integration: Seamlessly integrates with various data sources, including databases, cloud storage, and IBM Cloud services.

Visualization: Offers rich data visualization tools to explore and present data insights effectively.

Security: Ensures data security and compliance with enterprise-grade security features and controls.

Exercise 2: Implement A* algorithm for one the following problems: i) 8 puzzle ii) Missionaries and Cannibals

AIM: To implement A* algorithm for 8 puzzle problem.

DESCRIPTION:

A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
It is essentially a best first search algorithm.
A* Algorithm works as-
It maintains a tree of paths originating at the start node.
It extends those paths one edge at a time.
It continues until its termination criterion is satisfied.
A* Algorithm extends the path that minimizes the following function-

$$f(n) = g(n) + h(n)$$

Here,
'n' is the last node on the path
g(n) is the cost of the path from start node to node 'n'
h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node

PROGRAM:

```
import copy
class Puzzle:
    def __init__(self, size, board):
        self.size = size
        self.board = board
    def __str__(self):
        result = ""
        for row in self.board:
            result += " ".join(map(str, row)) + "\n"
        return result
    def find_blank(self):
        for i in range(self.size):
            for j in range(self.size):
                if self.board[i][j] == 0:
                    return i, j
    def move(self, direction):
        i, j = self.find_blank()
        new_board = copy.deepcopy(self.board)
        if direction == "up" and i > 0:
            new_board[i][j], new_board[i - 1][j] = new_board[i - 1][j], new_board[i][j]
        elif direction == "down" and i >self.size - 1:
            new_board[i][j], new_board[i + 1][j] = new_board[i + 1][j], new_board[i][j]
        elif direction == "left" and j > 0:
            new_board[i][j], new_board[i][j - 1] = new_board[i][j - 1], new_board[i][j]
        elif direction == "right" and j < self.size - 1:
            new_board[i][j], new_board[i][j + 1] = new_board[i][j + 1], new_board[i][j]
        return Puzzle(self.size, new_board)
def main():
    size = 3
    initial_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
    goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
    puzzle = Puzzle(size, initial_state)
    print("Initial State:")
    print(puzzle)
    moves = ["up","down", "left", "right"]
    for move in moves:
        new_puzzle = puzzle.move(move)
        print(move.capitalize())
        print(new_puzzle)
if __name__ == "__main__":
  main()
```

OUTPUT:

```
Initial State:
1 2 3
4 5 6
7 8 0

Up
1 2 3
4 5 0
7 8 6

Down
1 2 3
4 5 6
7 8 0

Left
1 2 3
4 5 6
7 0 8

Right

4 2 3
4 5 6
7 8 0
```

Exercise 3: Implement and test hill climbing based search algorithms to solve Travelling Salesman Problem.

AIM: To Implement and test hill climbing based search algorithms to solve Travelling Salesman Problem.

DESCRIPTION:

The TSP stands as one of the best known problems when it comes to work with NP-hard problems, which implies that no known algorithm exists to solve it in polynomial time. The problem can be summarized as follows : "Given a set of cities and the cost of travel (or distance) between each possible pairs, the TSP, is to find the best possible way of visiting all the cities and returning to the starting point that minimize the travel cost (or travel distance)." The exact solution to this problem with n cities can only be determined through evaluating (n-1)!/2 possibilities.

PROGRAM:

```
import itertools
import math
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)
points = [(0, 0), (1, 3), (4, 1), (3, 4), (2, 2)]
all_routes = itertools.permutations(range(len(points)))
best_route = min(all_routes, key=lambda route: sum(distance(points[route[i]], points[route[i+1]]) for
i in range(len(points) - 1)))
total_distance = sum(distance(points[best_route[i]], points[best_route[i+1]]) for i in
range(len(points) - 1))
total_distance += distance(points[best_route[-1]], points[best_route[0]])
print("Best Route:", best_route)
print("Minimum Distance:", total_distance)
```

OUTPUT:

```
Best Route: (0, 4, 1, 3, 2)
Minimum Distance: 13.764091950405117.
```

Exercise 4: Solve and implement map coloring problem by backtracking and constraint propagation.

Aim:
The goal of this lab exercise is to implement and solve the Map Coloring Problem using Backtracking and Constraint Propagation in machine learning. Specifically, we will attempt to color a map where no two adjacent regions share the same color, using a backtracking algorithm and applying constraint propagation techniques to improve efficiency.
Description:
The map coloring problem is a classical example of a Constraint Satisfaction Problem (CSP), where:
A map is divided into regions (countries or areas).
Each region must be assigned one color from a fixed set of colors.
Adjacent regions must not share the same color.
We will implement a backtracking algorithm to solve this problem and use constraint propagation to speed up the process by reducing the search space.
The program uses backtracking to assign colors to regions, checking for consistency with neighboring regions. Constraint propagation reduces the search space by eliminating values from the domain of neighboring regions when a region is colored.
Key Functions:
is_valid(map_graph, color_assignment, region, color): Checks if assigning a color to a region is valid, ensuring no adjacent region has the same color.
map_coloring(map_graph, regions, color_assignment, num_colors): The backtracking function that attempts to assign colors to all regions.
solve_map_coloring(map_graph, regions, num_colors): The main function to solve the map coloring problem.
Program:-

```
def is_valid(map_graph, color_assignment, region, color):
    # Check adjacent regions to ensure no two regions have the same color
    for neighbor in map_graph[region]:
        if color_assignment[neighbor] == color:
            return False
    return True

def map_coloring(map_graph, regions, color_assignment,
    num_colors): # Try to assign a color to each region
    for region in regions:
        if color_assignment[region] == -1:  # If region is not yet colored for
            color in range(num_colors):  # Try each color for the region
                if is_valid(map_graph, color_assignment, region, color):
                    color_assignment[region] = color  # Assign color to region
                    # Recursively assign colors to the next region
                    if map_coloring(map_graph, regions, color_assignment, num_colors):
                        return True

                    color_assignment[region] = -1  # Backtrack if no valid color assignment

                return False  # No color can be assigned, backtrack
    return True  # All regions are colored successfully

def solve_map_coloring(map_graph, regions, num_colors):
    color_assignment = {region: -1 for region in regions}  # Initialize all regions to -1 (uncolored) if
    map_coloring(map_graph, regions, color_assignment, num_colors):
        return
    color_assignment else:
        return None  # No solution found

# Example map graph (adjacency list)
map_graph = {
    'A': ['B', 'C'],

    'B': ['A', 'C', 'D'],

    'C': ['A', 'B', 'E'],

    'D': ['B', 'E'],

    'E': ['C', 'D']

}


regions = ['A', 'B', 'C', 'D', 'E']  # List of regions
num_colors = 3  # Number of colors available

# Solve the map coloring problem
solution = solve_map_coloring(map_graph, regions, num_colors)

# Output the solution
if solution:
    print("Solution found:")
    for region, color in solution.items():
        print(f"Region {region}: Color {color}")
else:
print("No solution found.")
```
Output:-

```
Solution found:
Region A: Color 0
 Region B: Color 1
Region C: Color 2
 Region D: Color 0
Region E: Color 1
```

Exercise 5: Solve and implement the game of tic-tac-toe using mini-max.

Aim of the Tic-Tac-Toe Game

The goal of this implementation is to create a simple Tic-Tac-Toe game where two players (X and O) take turns to place their marks on a 3x3 grid. The first player to align three of their marks consecutively in a row, column, or diagonal wins the game. If the grid is full and no player has won, the game ends in a tie.

Description

Tic-Tac-Toe is a classic two-player board game played on a 3x3 grid. Players take turns placing their symbols (either "X" or "O") in an empty grid cell. The game ends when one player gets three of their marks in a row, column, or diagonal, or if all grid cells are filled, resulting in a tie.

Explanation of the Program

Game Board:

The game board is represented as a list of 9 elements, initially all set to "-", indicating empty spaces.

Printing the Board:

The print board() function prints the current state of the game board in a 3x3 format.

Handling Player's Turn:

The take turn(player) function allows a player to pick a position on the board (from 1-9). It checks for valid input and ensures the position is not already taken.

Game Over Check:

The check_game_over() function checks if there is a winner or if the game has ended in a tie. It checks for horizontal, vertical, and diagonal win conditions and whether the board is full.

Main Game Loop:

The play_game() function alternates between Player X and Player O. It calls the take_turn() function for each player and checks if the game is over after each turn.

Program:-

```
# Set up the game board as a list
board = ["-", "-", "-",
        "-", "-", "-",

        "-", "-", "-"]


# Function to print the game board
def print_board():
   print(board[0] + " | " + board[1] + " | " + board[2])
   print(board[3] + " | " + board[4] + " | " + board[5])
   print(board[6] + " | " + board[7] + " | " + board[8])

# Function to handle a player's turn
def take_turn(player):
   print(player + "'s turn.")
```

```
    position = input("Choose a position from 1-9: ")
    while position not in ["1", "2", "3", "4", "5", "6", "7", "8", "9"]: position =
        input("Invalid input. Choose a position from 1-9: ")
    position = int(position) - 1 while
    board[position] != "-":
        position = int(input("Position already taken. Choose a different position: ")) - 1 board[position]
    = player
    print_board()

# Function to check if the game is over
def check_game_over():
    # Check for a win
    if (board[0] == board[1] == board[2] != "-") or \
      (board[3] == board[4] == board[5] != "-") or \
      (board[6] == board[7] == board[8] != "-") or \
      (board[0] == board[3] == board[6] != "-") or \
      (board[1] == board[4] == board[7] != "-") or \
      (board[2] == board[5] == board[8] != "-") or \
      (board[0] == board[4] == board[8] != "-") or \
      (board[2] == board[4] == board[6] != "-"): return
        "win"
    # Check for a tie
    elif "-" not in board:
        return "tie"
    # Game is not over
    else:
        return "play"

# Main game loop
def play_game():
    print_board()
    current_player = "X"
    game_over = False
    while not game_over:
        take_turn(current_player)
        game_result = check_game_over()
        if game_result == "win":
            print(current_player + " wins!")
            game_over = True
        elif game_result == "tie":
            print("It's a tie!")
            game_over = True
        else:
```

```
        # Switch to the other player
        current_player = "O" if current_player == "X" else "X"

# Start the game
play_game()
```

Output:-

```
- | - | -

- | - | -

- | - |
- X's
turn.
Choose a position from 1-
9: 1 X | - | -
- | - | -

- | - |
- O's
turn.
Choose a position from 1-
9: 5 X | - | -
- | O | -
- | - |
- X's
turn.
Choose a position from 1-
9: 2 X | X | -
- | O | -
- | - |
- O's
turn.
Choose a position from 1-
9: 6 X | X | -
- | O | O
- | - |
- X's
turn.
Choose a position from 1-
9: 3 X | X | X
- | O | O
- | -
| -
X
win
s!
```

Exercise 6: 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 use it to classify a new sample.

ID3 ALGORITHM

AIM: To write a program to demonstrate the working of the decision tree based ID3 algorithm by using an appropriate data set for building the decision tree and use it to classify a new sample.

DESCRIPTION:

The ID3 algorithm is a popular decision tree algorithm used in machine learning. It aims to build a decision tree by iteratively selecting the best attribute to split the data based on information gain. Each node represents a test on an attribute, and each branch represents a possible outcome of the test. The leaf nodes of the tree represent the final classifications. In this article, we will learn how to use the ID3 algorithm to build a decision tree to predict the output in detail.

A decision tree is a flowchart-like representation, with internal nodes representing features, branches representing rules, and leaf nodes representing algorithm results This versatile supervised machine-learning algorithm applies to both classification and regression problems, ie and power. Decision trees are valued for their interpretability, as the rules they generate are easy to understand.

PROGRAM:

```
import numpy as np
# Define the ID3 algorithm
def id3(X, y, attribute_names):
    if len(np.unique(y)) <= 1:
        return np.unique(y)[0]
    elif len(attribute_names) == 0:
        return np.unique(y)[np.argmax(np.unique(y, return_counts=True)[1])]
    else:
        best_attribute_index = np.argmax([information_gain(X, y, i) for i in range(len(attribute_names))])
        tree = {attribute_names[best_attribute_index]: {}}
        remaining_attribute_names = [i for i in attribute_names if i != attribute_names[best_attribute_index]]
        for value in np.unique(X[:, best_attribute_index]):
            value = value
            sub_X = X[X[:, best_attribute_index] == value]
            sub_y = y[X[:, best_attribute_index] == value]
            subtree = id3(sub_X, sub_y, remaining_attribute_names)
            tree[attribute_names[best_attribute_index]][value] = subtree
        return tree
# Define the information gain function
def information_gain(X, y, split_attribute_index):
    entropy_before = entropy(y)
    values, counts = np.unique(X[:, split_attribute_index], return_counts=True)
    weighted_entropy_after = np.sum([(counts[i] / np.sum(counts)) * entropy(y[X[:, split_attribute_index] == values[i]]) for i in range(len(values))])
    information_gain = entropy_before - weighted_entropy_after
    return information_gain
# Define the entropy function
def entropy(y):
    unique_values, counts = np.unique(y, return_counts=True)
    probabilities = counts / np.sum(counts)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy
# Test dataset
X = np.array([[1, 'Sunny'], [1, 'Sunny'], [0, 'Overcast'], [0, 'Rain'], [0, 'Rain']])
y = np.array(['No', 'No', 'Yes', 'Yes', 'Yes'])
attribute_names = ['Temperature', 'Outlook']
# Build the decision tree
tree = id3(X, y, attribute_names)
print(tree)
```

OUTPUT:

```
{'Temperature': {'0': 'Yes', '1': 'No'}}
```

Exercise 7: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

AIM: To Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

DESCRIPTION:

A neural network is a network structure, by the presence of computing units(neurons) the neural network has gained the ability to compute the function.

In machine learning, back propagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.

Back propagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.

PROGRAM:

```
import tensorflow as tf

import numpy as np

input_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

output_data = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([

    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),

    tf.keras.layers.Dense(units=1, activation='sigmoid')])

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(input_data, output_data, epochs=5000, verbose=0)

test_input = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

predictions = model.predict(test_input)

for i in range(len(predictions)):

    print(f"Input: {test_input[i]}, Predicted Output: {predictions[i]}")
```

OUTPUT:

```
Input: [0 0], Predicted Output: [0.48396885]

Input: [0 1], Predicted Output: [0.5038496]

Input: [1 0], Predicted Output: [0.50018275]

Input: [1 1], Predicted Output: [0.5167273]
```

Exercise 8: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Calculate the accuracy, precision, and recall for your data set.

AIM: To write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. And compute the accuracy of the classifier by considering few test data sets also calculate the accuracy, precision, and recall for your data set.

DESCRIPTION:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems. It is mainly used in *text classification* that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

PROGRAM:

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('/content/drive/MyDrive/tennisdata.csv')

print("The first 5 values of data is :\n",data.head())

X = data.iloc[:,:-1]

print("\nThe First 5 values of train data is\n",X.head())

y = data.iloc[:,-1]

print("\nThe first 5 values of Train output is\n",y.head())

le_outlook = LabelEncoder()

X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()

X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()

X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
```

```
X.Windy = le_Windy.fit_transform(X.Windy)
print("\nNow the Train data is :\n",X.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
classifier = GaussianNB()
classifier.fit(X_train,y_train)
from sklearn.metrics import accuracy_scorek
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

OUTPUT:

```
The first 5 values of data is :
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes

The First 5 values of train data is
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False

The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object

Now the Train data is :
   Outlook  Temperature  Humidity  Windy
```

```
0      2         1       0     0
1      2         1       0     1
2      0         1       0     0
3      1         2       0     0
4      1         0       1     0
```

```
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.3333333333333333
```

DATA SET:

Outlook	**Temperature**	**Humidity**	**Windy**	**PlayTennis**
Sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rainy	Mild	High	FALSE	Yes
Rainy	Cool	Normal	FALSE	Yes
Rainy	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Mild	High	TRUE	No

Exercise 9: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using any standard Heart Disease Data Set.

AIM: To write a program to construct a Bayesian network considering medical data and Use this model to demonstrate the diagnosis of heart patients using any standard Heart Disease Data Set.

DESCRIPTION:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

PROGRAM:

```
import pandas as pd

data=pd.read_csv("/content/drive/MyDrive/heartdisease.csv")

heart_disease=pd.DataFrame(data)

print(heart_disease)

from pgmpy.models import BayesianNetwork

model=BayesianNetwork([

('age','Lifestyle'),

('Gender','Lifestyle'),

('Family','heartdisease'),

('diet','cholestrol'),

('Lifestyle','diet'),

('cholestrol','heartdisease'),

('diet','cholestrol')

])

from pgmpy.estimators import MaximumLikelihoodEstimator

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination

HeartDisease_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')

print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')

print('For diet Enter { High:0, Medium:1 }')

print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')

print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={

    'age':int(input('Enter age :')),

    'Gender':int(input('Enter Gender :')),

    'Family':int(input('Enter Family history :')),

    'diet':int(input('Enter diet :')),

    'Lifestyle':int(input('Enter Lifestyle :')),

    'cholestrol':int(input('Enter cholestrol :'))

    })

print(q['heartdisease'])
```

OUTPUT:

```
age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease

0    0       0       1     1          3           0            1

1    0       1       1     1          3           0            1

2    1       0       0     0          2           1            1

3    4       0       1     1          3           2            0

4    3       1       1     0          0           2            0

5    2       0       1     1          1           0            1

6    4       0       1     0          2           0            1

7    0       0       1     1          3           0            1
```

```
8    3     1     1    0       0       2        0

9    1     1     0    0       0       2        1

10    4     1     0    1       2       0        1

11    4     0     1    1       3       2        0

12    2     1     0    0       0       0        0

13    2     0     1    1       1       0        1

14    3     1     1    0       0       1        0

15    0     0     1    0       0       2        1

16    1     1     0    1       2       1        1

17    3     1     1    1       0       1        0

18    4     0     1    1       3       2        0

For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }

For Gender Enter { Male:0, Female:1 }

For Family History Enter { yes:1, No:0 }

For diet Enter { High:0, Medium:1 }

For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }

For cholesterol Enter { High:0, BorderLine:1, Normal:2 }

Enter age :1

Enter Gender :1

Enter Family history :0

Enter diet :1

Enter Lifestyle :0

Enter cholestrol :1
```

```
+----------------+---------------------+

| heartdisease   |   phi(heartdisease) |

+==========+===========+

| heartdisease_0 |              0.0000 |

+----------------+---------------------+

| heartdisease_1 |              1.0000 |

+----------------+---------------------+
```

Exercise 10: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

AIM: To Apply EM algorithm to cluster a set of data stored in a .CSV file.and Use the same data set for clustering using k-Means algorithm also Compare the results of these two algorithms and comment on the quality of clustering.

DESCRIPTION:
The Expectation-Maximization (EM) algorithm is an iterative optimization method that combines different unsupervised machine learning algorithms to find maximum likelihood or maximum posterior estimates of parameters in statistical models that involve unobserved latent variables. The EM algorithm is commonly used for latent variable models and can handle missing data. It consists of an estimation step (E-step) and a maximization step (M-step), forming an iterative process to improve model fit.

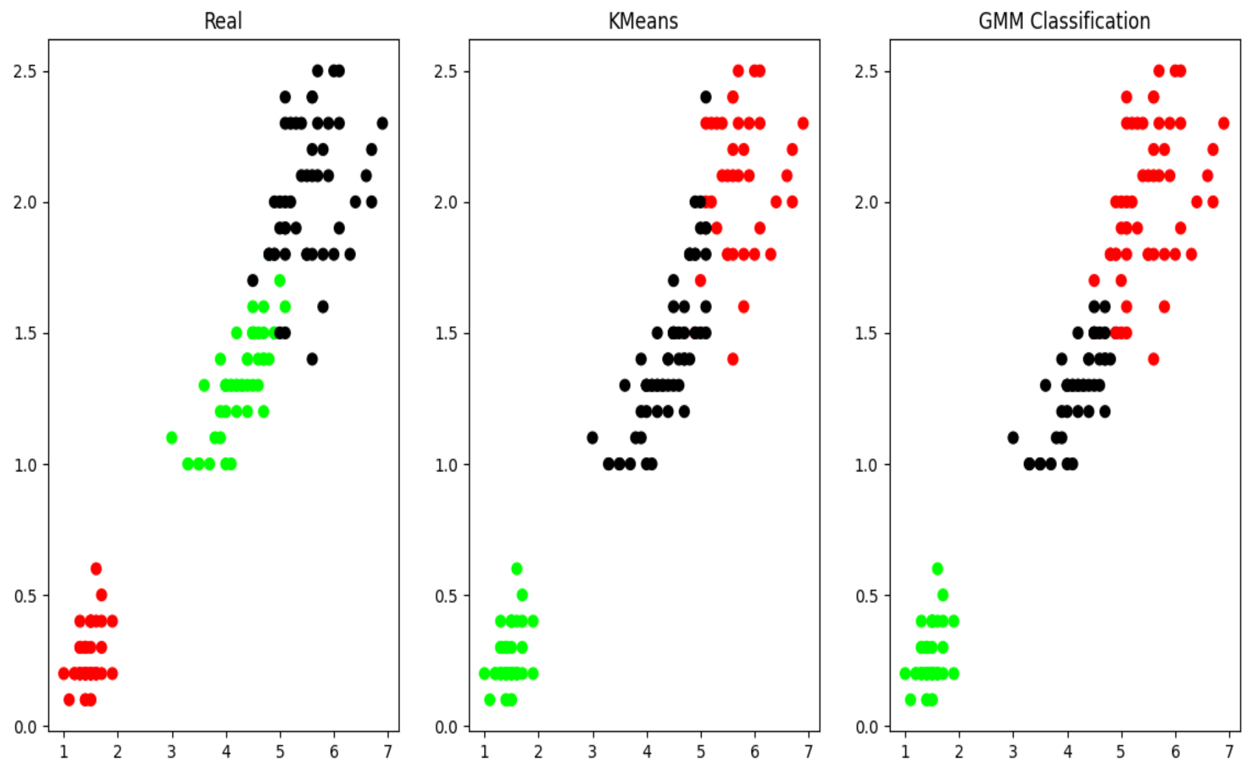
- In the E step, the algorithm computes the latent variables i.e. expectation of the log-likelihood using the current parameter estimates.
- In the M step, the algorithm determines the parameters that maximize the expected log-likelihood obtained in the E step, and corresponding model parameters are updated based on the estimated latent variables.

PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
dataset=load_iris()
# print(dataset)
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')
# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

OUTPUT:



Exercise 11: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions

AIM: To write a program to implement k-Nearest Neighbor algorithm to classify the iris data set and Print both correct and wrong predictions.

DESCRIPTION:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)
kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)
for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)
print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,datas
et["target_names"][prediction])
print(kn.score(X_test,y_test))
```

OUTPUT:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
```

```
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
0.9736842105263158
```

Exercise 12: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw the corresponding graphs.

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

DESCRIPTION:

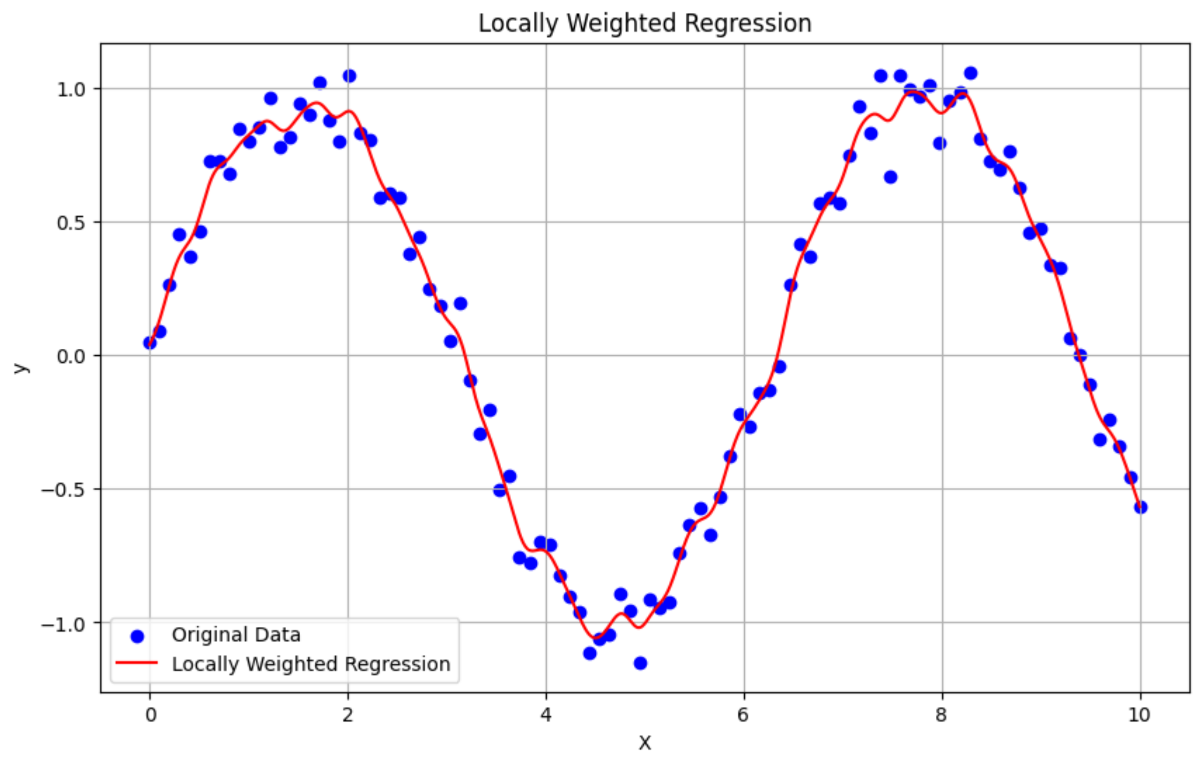
Linear regression is a supervised learning algorithm used for computing linear relationships between input (X) and output (Y)

I Locally weighted linear regression is a non-parametric algorithm, that is, the model does not learn a fixed set of parameters as is done in ordinary linear regression. Rather parameters are computed individually for each query point . While computing , a higher "preference" is given to the points in the training set lying in the vicinity of than the points lying far away from.

PROGRAM:

```
#non-parametric locally weighted algo
import numpy as np
import matplotlib.pyplot as plt
def kernel_function(x, x_i, tau):
    # Gaussian kernel function
    return np.exp(-((x - x_i) ** 2) / (2 * tau ** 2))
def locally_weighted_regression(x, X, y, tau):
    # Add bias term to X
    X_bias = np.column_stack([np.ones(len(X)), X])
    # Initialize predictions
    predictions = []
    # Iterate over each query point
    for query_point in x:
        weights = kernel_function(query_point, X, tau)
        # Weighted least squares regression
        W = np.diag(weights)
        theta = np.linalg.inv(X_bias.T @ W @ X_bias) @ (X_bias.T @ W @ y)
        # Predict using the learned parameters
        prediction = np.dot(np.array([1, query_point]), theta)
        predictions.append(prediction)
    return np.array(predictions)
# Generate synthetic dataset
np.random.seed(42)
X = np.linspace(0, 10, 100)
y = np.sin(X) + np.random.normal(0, 0.1, size=X.shape)
# Set bandwidth parameter
tau = 0.1
# Predict using locally weighted regression
x_values = np.linspace(0, 10, 1000)
y_values = locally_weighted_regression(x_values, X, y, tau)
# Plot original data and predictions
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Original Data')
plt.plot(x_values, y_values, color='red', label='Locally Weighted Regression')
plt.title('Locally Weighted Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:



Exercise 13: Write a program to implement 5-fold cross validation on a given dataset. Compare the accuracy, precision, recall, and F-score for your data set for different folds.

AIM: To Write a program to implement 5-fold cross validation on a given dataset and Compare the accuracy, precision, recall, and F-score for your data set for different folds.

DESCRIPTION:

Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. **We can also say that it is a technique to check how a statistical model generalizes to an independent dataset**.

In machine learning, there is always the need to test the stability of the model. It means based only on the training dataset; we can't fit our model on the training dataset. For this purpose, we reserve a particular sample of the dataset, which was not part of the training dataset. After that, we test our model on that sample before deployment, and this complete process comes under cross-validation. This is something different from the general train-test split.

Hence the basic steps of cross-validations are:

- Reserve a subset of the dataset as a validation set.
- Provide the training to the model using the training dataset.

- Now, evaluate model performance using the validation set. If the model performs well with the validation set, perform the further step, else check for the issues

PROGRAM:

```
from sklearn.model_selection import cross_val_score, KFold

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load_iris

# Load example dataset (replace with your dataset)

data = load_iris()

X, y = data.data, data.target

# Define the classifier (replace with your classifier)

clf = DecisionTreeClassifier()

# Define evaluation metrics

scoring = {

    'accuracy': 'accuracy',

    'precision': 'precision_macro',

    'recall': 'recall_macro',

    'f1': 'f1_macro'

}

# Perform 5-fold cross-validation

kf = KFold(n_splits=5, shuffle=True, random_state=42)

for fold, (train_index, test_index) in enumerate(kf.split(X)):

    X_train, X_test = X[train_index], X[test_index]

    y_train, y_test = y[train_index], y[test_index]
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(y_test, y_pred, average='macro')

    recall = recall_score(y_test, y_pred, average='macro')

    f1 = f1_score(y_test, y_pred, average='macro')

    print(f"Fold {fold+1}:")

    print(f"Accuracy: {accuracy}")

    print(f"Precision: {precision}")

    print(f"Recall: {recall}")

    print(f"F1 Score: {f1}")

    print()
```

OUTPUT:

```
Fold 1:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Fold 2:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Fold 3:
Accuracy: 0.9333333333333333
Precision: 0.9333333333333332
Recall: 0.9333333333333332
F1 Score: 0.9259259259259259
```

```
Fold 4:
Accuracy: 0.9333333333333333
Precision: 0.9388888888888888
Recall: 0.9388888888888888
F1 Score: 0.9388888888888888

Fold 5:
Accuracy: 0.9333333333333333
Precision: 0.9487179487179488
Recall: 0.9444444444444445
F1 Score: 0.9419191919191919
```

Exercise 14: write a simple program to implement Reinforcement Learning.

AIM: To write a simple program to implement Reinforcement Learning.

DESCRIPTION:

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

PROGRAM:

```
import numpy as np

# Define the environment

num_states = 16  # 4x4 grid world

num_actions = 4  # Up, Down, Left, Right

# Define rewards
```

```
rewards = np.array([
    [-1, -1, -1, -1],
    [-1, -1, -1, -1],
    [-1, -1, -1, -1],
    [-1, -1, -1, 10]  # Goal
])
# Initialize Q-table
q_table = np.zeros((num_states, num_actions))
# Q-learning parameters
learning_rate = 0.1
discount_factor = 0.9
epsilon = 0.1
num_episodes = 1000
# Q-learning algorithm
for _ in range(num_episodes):
    state = np.random.randint(num_states)  # Start from a random state
    done = False
    while not done:
        if np.random.uniform(0, 1) < epsilon:
            action = np.random.randint(num_actions)  # Explore
        else:
            action = np.argmax(q_table[state, :])  # Exploit
        next_state = (state + 1) % num_states  # Transition to the next state
        reward = rewards[state // 4, state % 4]  # Get reward for current state
        td_target = reward + discount_factor * np.max(q_table[next_state, :])
        td_error = td_target - q_table[state, action]
        q_table[state, action] += learning_rate * td_error
        state = next_state
        if state == num_states - 1:
            done = True  # Reach the goal
# Print the Q-table
print("Q-table:")
print(q_table)
```

<u>OUTPUT:</u>

Q-table:

```
[[-4.70944673 -4.76643065 -4.78622887 -4.7412093 ]
 [-5.38689152 -5.35088152 -5.36363495 -5.37319349]
 [-5.30978315 -5.31359132 -5.3179846  -5.30656095]
 [-4.95139646 -4.94662529 -4.94882623 -4.94459739]
 [-4.4762014  -4.48068155 -4.48242096 -4.47825284]
 [-3.95065999 -3.94631332 -3.94786459 -3.94947487]
 [-3.33869477 -3.33797717 -3.33732121 -3.33939604]
 [-2.67437946 -2.62940296 -2.63270142 -2.63035518]
 [-1.95777588 -1.84064479 -1.8395852  -2.00439687]
 [-1.40798553 -0.95838963 -1.29260792 -0.95708788]
 [-0.14590318  0.01577939 -0.07200648  0.01880956]
 [ 1.10997082  1.11120289  1.10726117  1.10875629]

 [ 2.27516163  2.27281016  2.26890919  2.25501627]

 [ 3.26706266  3.54087881  3.53141785  3.53719008]

 [ 4.8910985   4.9013738   4.89073082  4.8934051 ]

 [ 6.36468652  2.35385258  0.          0.90077054]]
```