DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi
CSE,ISE,ECE,EEE,ME,CE Branches Accredited by NBA, New Delhi
NAAC Accredited with A+ Grade

Department of Artificial Intelligence and Machine Learning

Academic Year 2023-24

AI AND ML APPLICATION DEVELOPMENT LABORATORY (18AIL76)

CONTENTS

Sl. no	Particulars	Page Number
1	Write a program to implement **k-Nearest Neighbour algorithm** to classify the iris dataset. Print both correct and wrong predictions.	5
2	Develop a program to apply K-means algorithm to cluster a set of data stored in .CSV file. Use the same data set for clustering using **EM algorithm**. Compare the results of these two algorithms and comment on the quality of clustering.	7
3	Implement the non-parametric **Locally Weighted Regressionalgorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs	8
4	Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets	9
5	Demonstrate **Genetic algorithm** by taking a suitable data for any simple application	12
6	Demonstrate **Q learning** algorithm with suitable assumption for a problem statement.	14
7	Viva-voce Questions	17

AI AND ML APPLICATION DEVELOPMENT LABORATORY

Course Code	**18AIL76**	CIE Marks	**40**
Teaching Hours/Weeks (L: T: P: S)	0: 2: 2	SEE Marks	**60**
Total Hours of Pedagogy		Total Marks	100
Credits	02	Exam Hours	03

Course Learning Objectives:

Explore the knowledge of AI and ML concepts and practice to groom students into wellinformed application developers.

- Demonstrate the knowledge of human cognition, Artificial Intelligence, Machine Learning and data engineering for designing intelligent systems
- Apply computational knowledge and project development skills to provide innovative solutions.
- Strong practice in AI and ML programming through a variety of AI and ML problems.
- Develop AI and ML applications using front-end and back-end tools

Part A

1. Write a program to implement **k-Nearest Neighbour algorithm** to classify the iris data set. Print both correct and wrong predictions.
2. Develop a program to apply K-means algorithm to cluster a set of data stored in .CSV file. Use the same data set for clustering using **EM algorithm**. Compare the results of these two algorithms and comment on the quality of clustering.
3. Implement the non-parametric **Locally Weighted Regressionalgorithm** in order to fit data points. Select appropriate data set for your experiment and draw graphs
4. Build an Artificial Neural Network by implementing the **Backpropagation algorithm** and test the same using appropriate data sets
5. Demonstrate **Genetic algorithm** by taking a suitable data for any simple application.
6. Demonstrate **Q learning** algorithm with suitable assumption for a problem statement.

PART B

Mini Project

- Use Java, C#, PHP, Python, or any other similar front-end tool. Developed mini projectns must be demonstrated on desktop/laptop as a stand-alone or web based application
- Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

- Indicative areas include: health care, education, agriculture, banking, library, agent based systems, registration systems, industry, reservation systems, facility management, super market etc.,Similar to but not limited to:

Handwritten Digit Recognition

Prediction of Cardiac Arrhythmia type using Clustering and Regression Approach

Hybrid Regression Technique for House Prices Prediction

An Iris Recognition Algorithm for Identity Authentication

An Approach to Maintain Attendance using Image Processing Techniques

Unconstrained Face Recognition

Vehicle Number Plate Detection System

Detection of Fake News

Stock Prediction using Linear Regression

Prediction of Weather Report

Analyzing Bike Sharing Trends

Sentiment Analysis for Movie Reviews

Analyzing and Recommendations of Music Trends

Forecasting Stock and Commodity Prices

Diabetes Prediction

Speech Recognition

Spam Detection using neural Networks in Python

Combining satellite imagery and to predict poverty

Conduct of Practical Examination:

Experiment distribution

- For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
- For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.

- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Subjected to change in accordance with university regulations)*

s) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks

t) For laboratories having PART A and PART B

i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks

ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 M

Program 1

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

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
for i in range(len(y_test)):
    if y_pred[i] == y_test[i]:
        print(f"Correct prediction: Actual class {y_test[i]}, Predicted class {y_pred[i]}")
    else:
        print(f"Wrong prediction: Actual class {y_test[i]}, Predicted class {y_pred[i]}")
```

Output:

```
Accuracy: 1.00
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 0, Predicted class 0
```

```
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 1, Predicted class 1
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 2, Predicted class 2
Correct prediction: Actual class 0, Predicted class 0
Correct prediction: Actual class 0, Predicted class 0
```

Program 2

Develop a program to apply K-means algorithm to cluster a set of data stored in .CSV file. Use the same data set for clustering using EM algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import adjusted_rand_score

# Load the dataset
data = pd.read_csv("/content/iris1.csv")
X = data.iloc[:, :-1].values  # Features

# Preprocess the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# Apply EM clustering
em = GaussianMixture(n_components=3, random_state=42)
em_labels = em.fit_predict(X_scaled)

# Ground truth labels from the dataset
#true_labels = data["variety"].map({"setosa": 0, "versicolor": 1, "virginica": 2})
true_labels = data["variety"]

# Evaluate clustering results using Adjusted Rand Index (ARI)
ari_kmeans = adjusted_rand_score(true_labels, kmeans_labels)
ari_em = adjusted_rand_score(true_labels, em_labels)

print("K-means ARI:", ari_kmeans)
print("EM ARI:", ari_em)
```

Output:

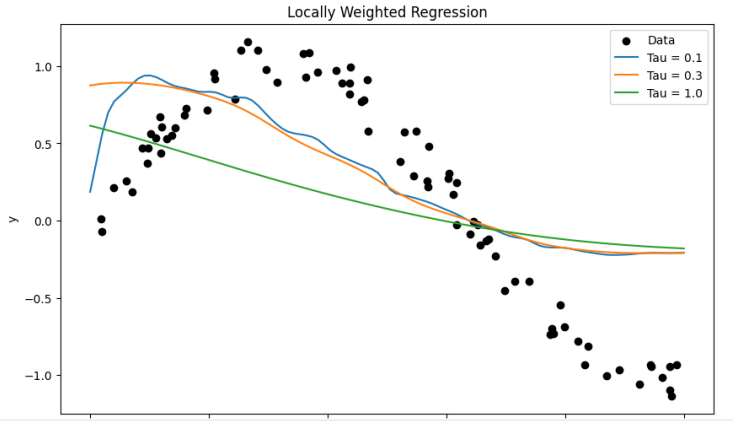
```
K-means ARI: 0.6201351808870379
EM ARI: 0.9038742317748124
```

Program 3

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 numpy as np
import matplotlib.pyplot as plt
def locally_weighted_regression(test_point, X, y, tau):
    m = X.shape[0]
    weights = np.exp(-np.sum((X - test_point) ** 2, axis=1) / (2 * tau ** 2))
    W = np.diag(weights)
    X_transpose = np.transpose(X)
    theta = np.linalg.inv(X_transpose @ W @ X) @ (X_transpose @ W @ y)
    return theta
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])
X_test = np.linspace(0, 5, 100)[:, np.newaxis]
tau_values = [0.1, 0.3, 1.0]  # Bandwidth parameter values
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='black', label='Data')
for tau in tau_values:
    y_pred = [locally_weighted_regression(test_point, X, y, tau) for test_point in X_test]
    plt.plot(X_test, y_pred, label=f'Tau = {tau:.1f}')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.show()
```

Output:



Program 4

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets

Description:

In this example, we've implemented a simple feedforward neural network with one hidden layer. The network is trained using the Backpropagation algorithm on the Iris dataset. Keep in mind that this is a basic implementation, and there are many ways to optimize and improve it, such as incorporating regularization techniques, using more advanced optimization algorithms, and tuning hyperparameters.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Neural Network implementation
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.weights_input_hidden = np.random.rand(self.input_size, self.hidden_size)
        self.bias_hidden = np.zeros((1, self.hidden_size))
        self.weights_hidden_output = np.random.rand(self.hidden_size, self.output_size)
        self.bias_output = np.zeros((1, self.output_size))

    def sigmoid(self, x):
```

```
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward(self, X):
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.final_output = self.sigmoid(self.final_input)
        return self.final_output

    def backward(self, X, y, output):
        self.error = y - output
        self.delta_output = self.error * self.sigmoid_derivative(output)
        self.error_hidden = self.delta_output.dot(self.weights_hidden_output.T)
        self.delta_hidden = self.error_hidden * self.sigmoid_derivative(self.hidden_output)

        self.weights_hidden_output += self.hidden_output.T.dot(self.delta_output)
        self.bias_output += np.sum(self.delta_output, axis=0, keepdims=True)
        self.weights_input_hidden += X.T.dot(self.delta_hidden)
        self.bias_hidden += np.sum(self.delta_hidden, axis=0)

    def train(self, X, y, epochs):
        for _ in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output)

    def predict(self, X):
        return np.argmax(self.forward(X), axis=1)

# Set random seed for reproducibility
np.random.seed(42)

# Initialize the neural network
input_size = X_train.shape[1]
hidden_size = 8
output_size = len(np.unique(y_train))
epochs = 10000

nn = NeuralNetwork(input_size, hidden_size, output_size)

# Train the neural network
nn.train(X_train, np.eye(output_size)[y_train], epochs)
```

```
# Make predictions on the test set
y_pred = nn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:

```
Accuracy: 0.6333333333333333
```

Program 5

Demonstrate Genetic algorithm by taking a suitable data for any simple application.

Description:

In this example, the genetic algorithm tries to find the value of `x` that minimizes the function `f(x) = x^2 + 4x + 4`. The algorithm starts with a random population of candidate solutions (values of `x`), and in each generation, it selects parents based on their fitness (fitness-proportional selection), performs crossover, and occasionally applies mutation. The algorithm evolves over generations, and the best solution found is printed at the end.

Genetic algorithms can be used for more complex optimization problems as well. Also, the parameters like population size, mutation rate, and number of generations might need to be adjusted for different problems. In this example, the genetic algorithm tries to find the value of x that minimizes the function f(x) = x^2 + 4x + 4. The algorithm starts with a random population of candidate solutions (values of x), and in each generation, it selects parents based on their fitness (fitness-proportional selection), performs crossover, and occasionally applies mutation. The algorithm evolves over generations, and the best solution found is printed at the end.

Genetic algorithms can be used for more complex optimization problems as well. Also, the parameters like population size, mutation rate, and number of generations might need to be adjusted for different problems.

```
import numpy as np
# Define the fitness function to minimize
def fitness_function(x):
    return x**2 + 4*x + 4
# Genetic Algorithm parameters
population_size = 50
num_generations = 100
mutation_rate = 0.1
# Initialize the population with random solutions
population = np.random.uniform(-10, 10, size=(population_size,))
# Main Genetic Algorithm loop
for generation in range(num_generations):
    # Evaluate fitness of each individual in the population
    fitness_values = np.array([fitness_function(x) for x in population])
```

```
    # Select parents for crossover
    parents = np.random.choice(population, size=population_size, p=fitness_values /
np.sum(fitness_values))
    # Create new population through crossover and mutation
    children = []
    for _ in range(population_size):
        parent1 = np.random.choice(parents)
        parent2 = np.random.choice(parents)
        child = (parent1 + parent2) / 2  # Simple averaging crossover
        if np.random.rand() < mutation_rate:
            child += np.random.normal(scale=0.5)
        children.append(child)
    population = np.array(children)
# Find the best solution
best_solution = population[np.argmin([fitness_function(x) for x in population])]
print("Best Solution:", best_solution)
print("Minimum Value:", fitness_function(best_solution))
```

Output:

```
Best Solution: 6.544911600919704
Minimum Value: 73.01551426753214
```

Program 6

Demonstrate Q learning algorithm with suitable assumption for a problem statement.

Description:

The Q-learning algorithm using a basic problem statement is implemented. In this example, we'll use the Q-learning algorithm to teach a robot to navigate a grid world and reach a goal while avoiding obstacles.

Assumptions and Problem Statement:
- The grid world is a 5x5 matrix.
- The robot can move in four directions: up, down, left, and right.
- There are obstacles in some cells of the grid.
- The goal is to reach a specific cell designated as the goal cell.
- The robot receives a negative reward for hitting an obstacle and a positive reward for reaching the goal cell.
- The robot's objective is to learn an optimal policy to navigate from any starting cell to the goal cell.

The robot learns the optimal policy using the Q-learning algorithm. The robot explores the grid world and updates Q-values based on the rewards and the Q-learning update equation. After training, the robot tests its learned policy to navigate from the starting position to the goal while avoiding obstacles. The learned policy is printed as a path marked with asterisks `*` on the grid. Real-world applications of Q-learning can involve more complex environments and considerations.

```
import numpy as np
# Grid world settings
grid_size = 5
num_actions = 4  # up, down, left, right
num_states = grid_size * grid_size
# Q-learning parameters
learning_rate = 0.8
discount_factor = 0.95
num_episodes = 1000
# Initialize Q-values
Q = np.zeros((num_states, num_actions))
# Obstacle and goal positions
```

```
obstacle_positions = [(1, 1), (2, 2), (3, 3)]
goal_position = (4, 4)
# Convert (row, column) coordinates to state index
def state_from_position(position):
    return position[0] * grid_size + position[1]
# Q-learning algorithm
for episode in range(num_episodes):
    current_position = (0, 0)  # Starting position
    while current_position != goal_position:
        state = state_from_position(current_position)
        valid_actions = [action for action in range(num_actions) if current_position != obstacle_positions]
        action = np.random.choice(valid_actions)
        next_row, next_col = current_position
        if action == 0:  # Move up
            next_row = max(next_row - 1, 0)
        elif action == 1:  # Move down
            next_row = min(next_row + 1, grid_size - 1)
        elif action == 2:  # Move left
            next_col = max(next_col - 1, 0)
        else:  # Move right
            next_col = min(next_col + 1, grid_size - 1)
        next_state = state_from_position((next_row, next_col))
        if next_state == state_from_position(goal_position):
            reward = 10
        elif next_state in [state_from_position(pos) for pos in obstacle_positions]:
            reward = -5
        else:
            reward = 0
        # Update Q-value using Q-learning equation
        Q[state, action] = (1 - learning_rate) * Q[state, action] + learning_rate * (reward + discount_factor *
np.max(Q[next_state, :]))
        current_position = (next_row, next_col)
# Test the learned policy
current_position = (0, 0)
```

```
path = [(0, 0)]
while current_position != goal_position:
    state = state_from_position(current_position)
    action = np.argmax(Q[state, :])
    next_row, next_col = current_position
    if action == 0:  # Move up
        next_row = max(next_row - 1, 0)
    elif action == 1:  # Move down
        next_row = min(next_row + 1, grid_size - 1)
    elif action == 2:  # Move left
        next_col = max(next_col - 1, 0)
    else:  # Move right
        next_col = min(next_col + 1, grid_size - 1)
    path.append((next_row, next_col))
    current_position = (next_row, next_col)
print("Learned Policy Path:")
for row in range(grid_size):
    for col in range(grid_size):
        if (row, col) == goal_position:
            print(" G ", end='')
        elif (row, col) in obstacle_positions:
            print(" X ", end='')
        elif (row, col) in path:
            print(" * ", end='')
        else:
            print(" . ", end='')
    print()
```

Output:

```
Learned Policy Path:
 *   .   .   .   .
 *   X   .   .   .
 *   .   X   .   .
 *   .   .   X   .
 *   *   *   *   G
```

Viva-voce Questions:

1. What is Artificial Intelligence (AI)?
2. Define Machine Learning (ML).
3. Explain the difference between supervised and unsupervised learning.
4. What are the main components of an ML pipeline?
5. Describe the concept of overfitting in ML.
6. What is the bias-variance trade-off in ML?
7. Why is data preprocessing important in ML?
8. What are some common techniques used for data cleaning?
9. How do you handle missing values in a dataset?
10. Explain the process of feature scaling.
11. What is one-hot encoding, and when should it be used?
12. Define linear regression.
13. What is the cost function in linear regression?
14. How do you evaluate the performance of a regression model?
15. Describe the difference between L1 and L2 regularization.
16. What is polynomial regression?
17. Explain logistic regression.
18. What is the sigmoid function used for in logistic regression?
19. Describe the confusion matrix and its components.
20. What is the difference between precision and recall?
21. Define the ROC curve and AUC.
22. What is clustering?
23. Explain the k-means algorithm.
24. How do you determine the optimal number of clusters in k-means?
25. What is hierarchical clustering?
26. Describe the difference between k-means and hierarchical clustering.
27. What is dimensionality reduction?
28. Explain the principal component analysis (PCA) algorithm.
29. How does PCA help in reducing the curse of dimensionality?
30. Describe t-SNE (t-distributed Stochastic Neighbor Embedding).
31. What is NLP?
32. Explain tokenization in NLP.
33. What is stemming and lemmatization?
34. How does a Bag-of-Words model work?
35. Describe the concept of word embeddings.
36. What is an artificial neural network (ANN)?
37. Explain the structure of a basic feedforward neural network.
38. Describe the backpropagation algorithm.
39. What is an activation function in a neural network?
40. What is the vanishing gradient problem?
41. Define deep learning.
42. Explain convolutional neural networks (CNNs).
43. Describe recurrent neural networks (RNNs).
44. What is transfer learning?

45. Explain generative adversarial networks (GANs).
46. How do you split a dataset into training and testing sets?
47. What is cross-validation, and why is it important?
48. Describe the process of model deployment.
49. What are some challenges in deploying ML models in production?
50. How do you ensure the fairness and ethics of AI and ML applications?
51. What is the purpose of the AI and ML Application Development Laboratory?
52. Can you explain the difference between AI and ML?
53. What are some common applications of AI and ML in real-world scenarios?
54. How do you approach the development of an AI or ML application?
55. What is the importance of data preprocessing in AI and ML?
56. What is overfitting and how can it be addressed?
57. What is the purpose of cross-validation?
58. Explain the concept of a confusion matrix.
59. What are some popular libraries or frameworks used for AI and ML development?
60. Can you give an example of a real-world AI or ML application you've worked on in this lab?
61. What is transfer learning?
62. How do you deploy an AI or ML model in a real-world scenario?
63. What ethical considerations are important when working with AI and ML?
64. What are the future trends in AI and ML development?