Lab programs

Program 1

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

data = pd.read_csv("data.csv").dropna()
x = data.iloc[:, 0].values
y = data.iloc[:, 1].values

# Direct formula
m = np.cov(x, y)[0,1] / np.var(x)
c = np.mean(y) - m * np.mean(x)

plt.scatter(x, y)
plt.plot(x, m*x + c, color='red')
plt.show()
```

program 2

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC
# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target
#Build the model
```

```
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)
# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
      svm,
      X,
      response_method="predict",
      cmap=plt.cm.Spectral,
      alpha=0.8,
      xlabel=cancer.feature_names[0],
      ylabel=cancer.feature_names[1],
   )
# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
          c=y,
          s=20, edgecolors="k")
plt.show()
```

Program 3

```
class CaseBasedReasoning:
   def __init__(self):
      self.cases = []

   def add_case(self, problem, solution):
      self.cases.append((problem, solution))

   def similarity(self, p1, p2):
      return len(set(p1) & set(p2)) / len(p1)
```

```
    def predict(self, new_problem):
        best, score = None, 0
        for p, s in self.cases:
            sim = self.similarity(new_problem, p)
            if sim > score:
                best, score = s, sim
        return best

# Example usage
cbr = CaseBasedReasoning()
cbr.add_case(["fever", "headache", "tired"], "flu")
cbr.add_case(["sore throat", "cough", "runny nose"], "common cold")

print(cbr.predict(["fever", "cough", "muscle aches"]))
```

program 4

```
import numpy as np

class DecisionTree:
    def __init__(self):
        self.feature = 0
        self.threshold = 0.5

    def train(self, X, y):
        # simple learning: choose first feature
        self.threshold = np.mean(X[:, self.feature])

    def predict_one(self, x):
        if x[self.feature] >= self.threshold:
            return 1
```

```
        else:
            return 0

    def predict(self, X):
        return [self.predict_one(x) for x in X]


# Dataset
X = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([1,1,0,0])

# Train tree
dt = DecisionTree()
dt.train(X, y)

# Test sample
sample = np.array([[1,0]])
print("Prediction:", dt.predict(sample)[0])
```

program 5

```
import numpy as np

class SimpleNN:
    def __init__(self):
        self.W1 = np.random.randn(2, 4)
        self.W2 = np.random.randn(4, 1)
        self.b1 = np.zeros((1, 4))
        self.b2 = np.zeros((1, 1))

    def sigmoid(self, x):
```

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

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

def train(self, X, y, lr=0.1, epochs=10000):
    for i in range(epochs):
        # Forward pass
        h = self.sigmoid(np.dot(X, self.W1) + self.b1)
        out = self.sigmoid(np.dot(h, self.W2) + self.b2)

        # Backpropagation
        error = y - out
        d_out = error * self.sigmoid_deriv(out)
        d_h = d_out.dot(self.W2.T) * self.sigmoid_deriv(h)

        # Update weights
        self.W2 += h.T.dot(d_out) * lr
        self.b2 += d_out.sum(axis=0) * lr
        self.W1 += X.T.dot(d_h) * lr
        self.b1 += d_h.sum(axis=0) * lr

        if i % 4000 == 0:
            print(f"Epoch {i}, Loss:{np.mean(error**2)}")

def predict(self, X):
    h = self.sigmoid(np.dot(X, self.W1) + self.b1)
    return self.sigmoid(np.dot(h, self.W2) + self.b2)
```

```
# XOR data
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

# Train
nn = SimpleNN()
nn.train(X, y)

# Test
print("Predictions:")
print(nn.predict(X))
```

program 6

```
import numpy as np

def knn_reg(X_train, y_train, x_test, k):
    # Compute distances
    d = np.linalg.norm(X_train - x_test, axis=1)
    # Select k nearest points
    idx = np.argsort(d)[:k]
    # Return mean of their outputs
    return np.mean(y_train[idx])



# Dataset
X_train = np.array([[1,2], [3,4], [5,6], [7,8]])
y_train = np.array([1.5, 3.5, 5.5, 7.5])

# Test point
x_test = np.array([2,3])
```

```
print("Predicted value:", knn_reg(X_train, y_train, x_test, k=2))
```

program 7

```
import math

# Euclidean distance
def euclidean(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Manhattan distance
def manhattan(x1, y1, x2, y2):
    return abs(x2 - x1) + abs(y2 - y1)

# User input
x1, y1 = map(int, input("Enter first point (x1 y1): ").split())
x2, y2 = map(int, input("Enter second point (x2 y2): ").split())

# Output
print("Euclidean Distance:", euclidean(x1, y1, x2, y2))
print("Manhattan Distance:", manhattan(x1, y1, x2, y2))
```

program 8

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
# Load dataset
X, y = load_iris(return_X_y=True)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Create and train KNN model
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

# Predict one sample
prediction = knn.predict([X_test[0]])
print("Actual:", y_test[0], "Predicted:", prediction[0])

# Accuracy
print("Accuracy:", knn.score(X_test, y_test))
```

program 9

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

def lowess(x, y, f=0.25, iterations=3):
    n = len(x)
    y_est = np.zeros(n)
    delta = np.ones(n)

    for _ in range(iterations):
        for i in range(n):
```

```
            dist = np.abs(x - x[i])
            h = np.sort(dist)[int(f * n)]
            w = (1 - (dist / h) ** 3) ** 3
            w[dist > h] = 0
            w = w * delta

            # Local linear regression
            X = np.vstack((np.ones(n), x)).T
            W = np.diag(w)
            beta = np.linalg.pinv(X.T @ W @ X) @ X.T @ W @ y
            y_est[i] = beta[0] + beta[1] * x[i]

        # Update robustness weights
        r = y - y_est
        s = np.median(np.abs(r))
        delta = (1 - (r / (6 * s)) ** 2) ** 2
        delta[np.abs(r) > 6 * s] = 0

    return y_est



# Sample data
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x) + 0.3*np.random.randn(100)

# LOWESS smoothing
y_smooth = lowess(x, y, f=0.25, iterations=3)

# Plot
plt.plot(x, y, "r.", label="Noisy Data")
```

```
plt.plot(x, y_smooth, "b-", label="LOWESS Curve")
plt.legend()
plt.show()
```

program 10

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

# Environment
n_states = 16
n_actions = 4
goal = 15

# Q-table
Q = np.zeros((n_states, n_actions))

# Parameters
alpha = 0.8       # learning rate
gamma = 0.95      # discount factor
epsilon = 0.2     # exploration
episodes = 500

# Q-learning
for _ in range(episodes):
    state = np.random.randint(0, n_states)

    while state != goal:
        # ε-greedy action selection
        if np.random.rand() < epsilon:
```

```
        action = np.random.randint(n_actions)
    else:
        action = np.argmax(Q[state])

    next_state = (state + 1) % n_states
    reward = 1 if next_state == goal else 0

    # Q-value update
    Q[state, action] += alpha * (
        reward + gamma * np.max(Q[next_state]) - Q[state, action]
    )

    state = next_state

# Show learned values
grid = np.max(Q, axis=1).reshape(4, 4)

plt.imshow(grid, cmap="coolwarm")
plt.colorbar()
plt.title("Learned Q-values")
plt.show()

print("Q-table:")
print(Q)
```