ARUNAI ENGINEERING COLLEGE

(An Autonomous Institution)
Velu Nagar, Thiruvannamalai-606603
www.arunai.org

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

BACHELOR OF TECHNOLOGY

2024-2025

FOURTH SEMESTER

AD3461-Machine Learning Laboratory

ARUNAI ENGINEERING COLLEGE

TIRUVANNAMALAI – 606 603

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

CERTIFICATE

Certified that this is a bonafide record of work done by

Name :

University Reg.No :

Semester :

Branch :

Year :

Staff-in-Charge **Head of the Department**

Submitted for the ____

Practical Examination held on ____________________

Internal Examiner **External Examiner**

PROGRAM:

```
import numpy as np
import pandas as pd
data = pd.read_csv(path+'/enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
```

```
print("Specific Bundary after ", i+1, "Instance is ", specific_h)
print("Generic Boundary after ", i+1, "Instance is ", general_h)
print("\n")

indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']] for i in
indices:
general_h.remove(['?', '?', '?', '?', '?', '?']) return
specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

DATASET:

	Outlook	Temperature	Humidity	Wind	Answer
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
3	overcast	hot	high	weak	Yes
4	rain	mild	high	weak	Yes
5	rain	cool	normal	weak	Yes
6	rain	cool	normal	strong	No
7	overcast	cool	normal	strong	Yes
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
10	rain	mild	normal	weak	Yes
11	sunny	mild	normal	strong	Yes
12	overcast	mild	high	strong	Yes
13	overcast	hot	normal	weak	Yes
14	rain	mild	high	strong	No
15	sunny	hot	high	strong	No

<u>Output</u>

[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]
[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]
Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’]
Initialization of specific_h and genearal_h
Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
eneric Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 1 is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’] Instance is Positive

Specific Boundary after 1 Instance is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 1 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 2 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’] Instance is Positive
Specific Bundary after 2 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 2 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 3 is [‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’] Instance is Negative
Specific Bundary after 3 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 3 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘same’]]

Instance 4 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’] Instance is Positive
Specific Bundary after 4 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]
Generic Boundary after 4 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]
Final Specific_h: [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]
Final General_h: [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’]]

PROGRAM:

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

data = pd.read_csv("Datasets/enjoysport.csv")
features = [feat for feat in data.columns if feat != "Answer"]  # Filter out 'Answer' column

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

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

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

def ID3(examples, attrs):
```

```
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain  =  gain
            max_feat = feature

    if not max_feat:
        root.isLeaf = True
        root.pred = examples["Answer"].mode()[0]
        return root

    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["Answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
```

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

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print("Predicted Label for new example", new, " is:", child.pred)
                return
            else:
                classify(child.children[0], new)

root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print(" -----------------")
new = {"Outlook": "sunny", "Temperature": "hot", "Humidity": "normal", "Wind": "strong"}
classify(root, new
```

DATASET:

	Outlook	Temperature	Humidity	Wind	Answer
1	sunny	hot	high	weak	No
2	sunny	hot	high	strong	No
3	overcast	hot	high	weak	Yes
4	rain	mild	high	weak	Yes
5	rain	cool	normal	weak	Yes
6	rain	cool	normal	strong	No
7	overcast	cool	normal	strong	Yes
8	sunny	mild	high	weak	No
9	sunny	cool	normal	weak	Yes
10	rain	mild	normal	weak	Yes
11	sunny	mild	normal	strong	Yes
12	overcast	mild	high	strong	Yes
13	overcast	hot	normal	weak	Yes
14	rain	mild	high	strong	No
15	sunny	hot	high	strong	No

OUTPUT

```
Decision Tree is:
Outlook
        overcast ->  ['yes']

        rain
                Wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                Humidity
                        high ->  ['no']

                        normal ->  ['yes']
```

- - - - - - - - - - - -

Predicted Label for new example {'Outlook': 'sunny', 'Temperature': 'hot', 'Humidity': 'normal', 'Wind': 'strong'} is: ['yes']

PROGRAM:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100


#Sigmoid Function

def sigmoid (x):

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

#Derivative of Sigmoid Function

def derivatives_sigmoid(x):

    return x * (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set

hiddenlayer_neurons = 3 #number of hidden layers neurons

output_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
```

```
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
  #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
   #how much hidden layer wts contributed to error
   d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
  # dot product of next layer error and current layerop
    wh += X.T.dot(d_hiddenlayer) *lr
    print ("-----------Epoch-", i+1, "Starts --------- ")
    print("Input: \n" + str(X))
```

```
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends --------- \n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Training Examples:

Example	**Sleep**	**Study**	**Expected % in Exams**
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	**Sleep**	**Study**	**Expected % in Exams**
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Output

———Epoch- 1 Starts———-
Input:
[[0.66666667 1.]
[0.33333333 0.55555556]
[1. 0.66666667]]
Actual Output:
[[0.92]
[0.86]

[0.89]]
Predicted Output:
[[0.81951208]
[0.8007242]
[0.82485744]]

————Epoch- 1 Ends———

PROGRAM:

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('tennisdata.csv')
print("The first 5 values of data is :\n",data.head())

# obtain Train data and Train output
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())

# Convert then in numbers
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_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

Tennisdata.Csv

Outlook	Temperature	Humidity	Windy	Play Tennis
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

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: 0.6666666666666666

PROGRAM:

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Sample documents and their corresponding labels
documents = [
    "This is a sample document about the naive Bayes classifier algorithm.",
    "Naive Bayes classifier is easy to implement and works well for text classification tasks.",
    "Text classification using the naive Bayes algorithm is popular in natural language
processing.",
    "The output of the naive Bayes program depends on the input features and training data."
]
labels = [1, 1, 1, 0]  # 1 for documents about naive Bayes, 0 for others

# Convert the documents into a bag-of-words representation
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

# Train a naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X, labels)

# Test data
test_documents = [
    "This document is not related to naive Bayes.",
    "Naive Bayes algorithm is widely used for text classification."
]
true_labels = [0, 1]

# Convert test documents into bag-of-words representation
X_test = vectorizer.transform(test_documents)

# Predict labels for test documents
predicted_labels = classifier.predict(X_test)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(true_labels, predicted_labels)
precision = precision_score(true_labels, predicted_labels)
```

```
recall = recall_score(true_labels, predicted_labels)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

OUTPUT:

Accuracy: 0.5
Precision: 0.5
Recall: 1.0

PROGRAM:

```
# Define the conditional probabilities
# P(COVID | Fever, Cough, BreathingDifficulty)
P_COVID_given_symptoms = {
    (1, 1, 1): 0.15,  # P(COVID = Yes | Fever = Yes, Cough = Yes, BreathingDifficulty = Yes)
    (1, 1, 0): 0.1,   # P(COVID = Yes | Fever = Yes, Cough = Yes, BreathingDifficulty = No)
    (1, 0, 1): 0.4,   # P(COVID = Yes | Fever = Yes, Cough = No, BreathingDifficulty = Yes)
    (1, 0, 0): 0.1,   # P(COVID = Yes | Fever = Yes, Cough = No, BreathingDifficulty = No)
    (0, 1, 1): 0.7,   # P(COVID = Yes | Fever = No, Cough = Yes, BreathingDifficulty = Yes)
    (0, 1, 0): 0.3,   # P(COVID = Yes | Fever = No, Cough = Yes, BreathingDifficulty = No)
    (0, 0, 1): 0.2,   # P(COVID = Yes | Fever = No, Cough = No, BreathingDifficulty = Yes)
    (0, 0, 0): 0.01   # P(COVID = Yes | Fever = No, Cough = No, BreathingDifficulty = No)
}

# Given evidence
fever = 1  # Fever = Yes
cough = 1  # Cough = Yes
breathing_difficulty = 1  # BreathingDifficulty = Yes

# Calculate P(COVID = Yes) and P(COVID = No)
P_COVID_yes = P_COVID_given_symptoms[(fever, cough, breathing_difficulty)]
P_COVID_no = 1 - P_COVID_yes

# Print the result in the desired format
print("COVID P(COVID)")
print(f"0 {P_COVID_no:.2f}")
print(f"1 {P_COVID_yes:.2f}")
```

OUTPUT:

COVID	**P(COVID)**
0	**0.85**
1	**0.15**

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

data = pd.read_csv('dataset1.csv')
X = data.values

num_clusters = 2

kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
kmeans_silhouette_score = silhouette_score(X, kmeans_labels)

em = GaussianMixture(n_components=num_clusters, random_state=42)
em_labels = em.fit_predict(X)
em_silhouette_score = silhouette_score(X, em_labels)

print("Silhouette Score (k-Means):", kmeans_silhouette_score)
print("Silhouette Score (EM):", em_silhouette_score)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='x', color='red',
label='Centroids')
plt.title('k-Means Clustering')
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=em_labels, cmap='viridis')
plt.scatter(em.means_[:, 0], em.means_[:, 1], marker='x', color='red', label='Centroids')
plt.title('EM Clustering')
plt.legend()
```

```
plt.show()
```

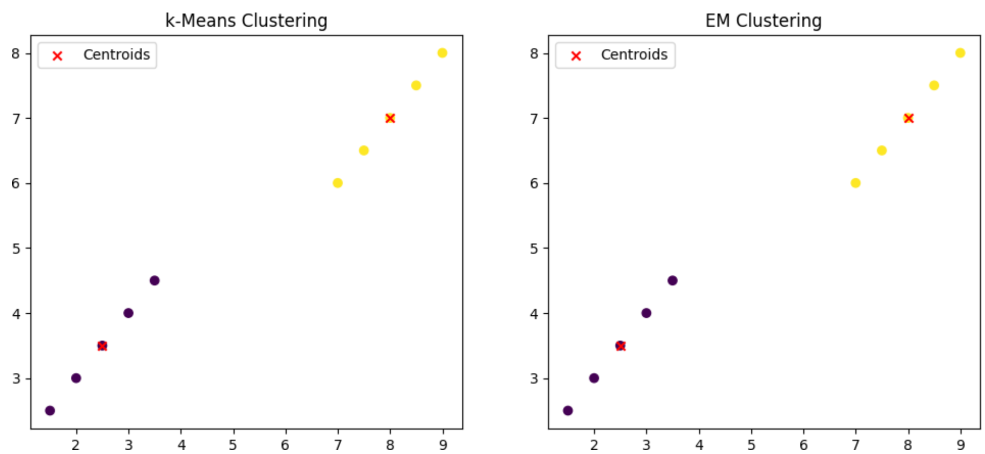
DATASET

```
Feature1,Feature2
2.5,3.5
1.5,2.5
3.5,4.5
3.0,4.0
2.0,3.0
7.5,6.5
8.5,7.5
9.0,8.0
8.0,7.0
7.0,6.0
```

OUTPUT:

Silhouette Score (k-Means): 0.7774804461410134

Silhouette Score (EM): 0.7774804461410134



PROGRAM:

```
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.metrics import accuracy_score

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

k = 3

knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

for i in range(len(X_test)):
    if y_pred[i] == y_test[i]:
        print(f"Correct prediction: Actual - {iris.target_names[y_test[i]]}, Predicted -
{iris.target_names[y_pred[i]]}")
    else:
        print(f"Wrong prediction: Actual - {iris.target_names[y_test[i]]}, Predicted -
{iris.target_names[y_pred[i]]}")

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

OUTPUT:

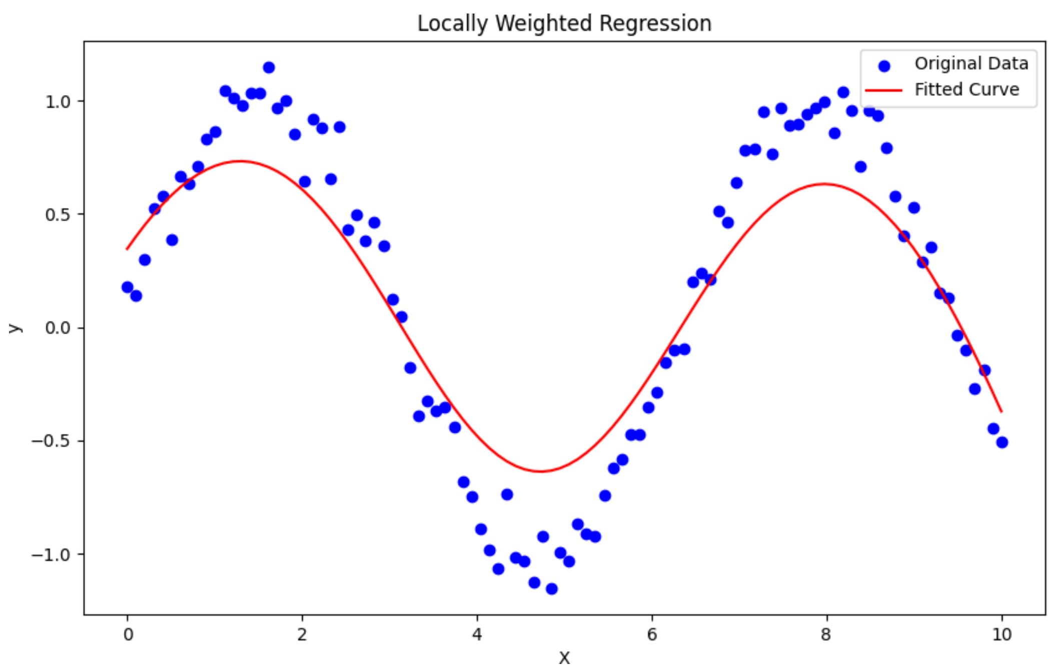
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa

Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - virginica, Predicted - virginica
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - versicolor, Predicted - versicolor
Correct prediction: Actual - setosa, Predicted - setosa
Correct prediction: Actual - setosa, Predicted - setosa
Accuracy: 1.0

PROGRAM:

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

def lwr(query_point, X, y, tau):
    """
    Locally Weighted Regression
    Args:
    - query_point: point at which prediction is to be made
    - X: input features
    - y: target values
    - tau: bandwidth parameter
    Returns:
    - prediction at query_point
    """
    m = X.shape[0]
    X = np.column_stack((np.ones(m), X))  # Add bias term
    query_point = np.array([1, query_point])  # Add bias term to query point
    weights = np.exp(-((X[:, 1] - query_point[1]) ** 2) / (2 * tau * tau))
    W = np.diag(weights)
    theta = np.linalg.inv(X.T @ W @ X) @ (X.T @ (W @ y))
    prediction = query_point @ theta
    return prediction

np.random.seed(0)
X = np.linspace(0, 10, 100)
y = np.sin(X) + np.random.normal(0, 0.1, 100)

tau = 1.0

predictions = [lwr(x, X, y, tau) for x in X]

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Original Data')
plt.plot(X, predictions, color='red', label='Fitted Curve')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.show()
```

OUTPUT:



Locally Weighted Regression
Original Data
Fitted Curve
1.0
0.5
0.0
−0.5
−1.0
y
0
2
4
6
8
10
X

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold

data = load_iris()
X = data.data
y = data.target

model = KNeighborsClassifier(n_neighbors=3)

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

scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')

print(f"Accuracy for each fold: {scores}")
print(f"Mean accuracy: {scores.mean():.2f}")
print(f"Standard deviation of accuracy: {scores.std():.2f}")
```

OUTPUT:

Accuracy for each fold: [0.96666667 0.96666667 0.96666667 0.96666667 1.]
Mean accuracy: 0.97
Standard deviation of accuracy: 0.02

PROGRAM:

```
import numpy as np
from scipy import stats
import pandas as pd

alpha = 0.05

before_tuning = [70, 75, 80, 82, 78, 85, 83, 77, 74, 79]
after_tuning = [78, 80, 85, 88, 83, 90, 89, 82, 80, 86]

t_stat, p_value = stats.ttest_rel(before_tuning, after_tuning)

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

if p_value < alpha:
    print("There is a significant difference between the before and after performance (reject H0).")
else:
    print("There is no significant difference between the before and after performance (fail to reject H0).")
```

OUTPUT:

T-statistic: -5.741978695179742
P-value: 0.00021542350254357943
There is a significant difference between the before and after performance (reject H0).