**ARUNAI ENGINEERING COLLEGE**



**(Affiliated to Anna University)**

**Velu Nagar, Thiruvannamalai-606603**

[**www.arunai.org**](http://www.arunai.org/)

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**& DATA SCIENCE**

**BACHELOR OF TECHNOLOGY**

**2023-2024**

**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 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

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**

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

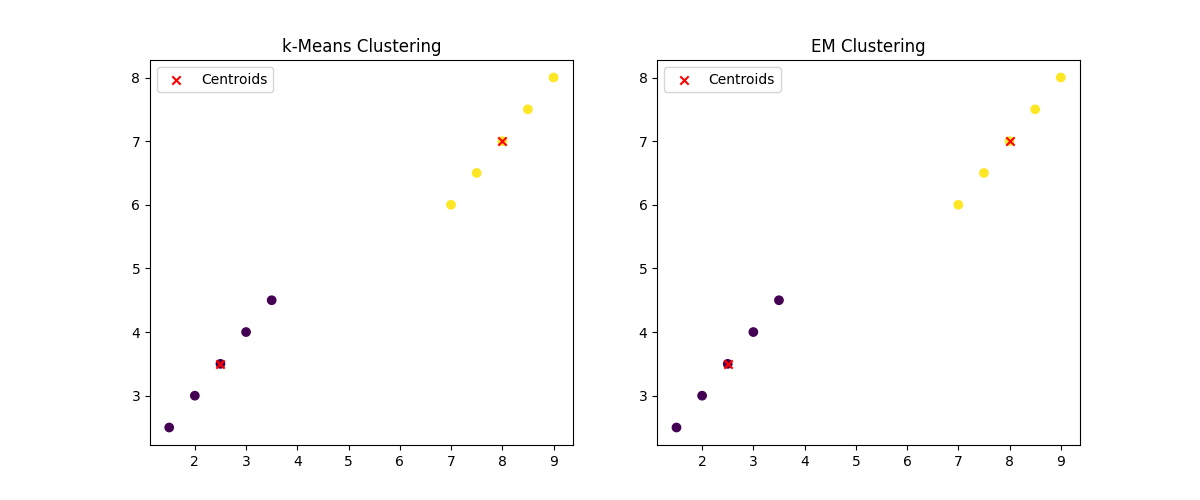
8.0,7.0

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**

