A LAB MANUAL

on

MACHINE LEARNING LAB (U21CD6L2)

B.E. III Year & VI Semester

By

Mr. Md Ashique Hussain
Assistant Professor
Dept. of CSE – Data Science

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

LORDS INSTITUTE OF ENGINEERING & TECHNOLOGY
(UGC Autonomous)
Approved by AICTE | Affiliated to Osmania University | Estd.2003
Accredited by **NBA**, Accredited '**A**' Grade by NAAC
(Academic Year: 2023 – 2024)

INDEX

LORDS INSTITUTE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

Approved by AICTE | Affiliated to Osmania University | Estd.2003.

Department of Computer Science and Engineering (Data Science)

Vision of the Department:

To develop the world's next generation Data Scientist by offering top-notch engineering education with cutting edge technologies.

Mission of the Department:

DM1: Providing students to understand the principles of Data Science with hands-on experience.

DM2: Offering students to analyze the data through effective teaching learning methods and cutting edge technologies in multi disciplinary fields.

DM3: Preparing students for R&D, Industrial design, entrepreneurship and employment.

DM4: Encouraging students with interaction and Industry Institute partnership through various organizations.

Note: DM: Department Mission

[illegible] 26/9/22

Head of the Department

Head of the Department
CSE (Data Science)
Lords Institute of Engg. & Tech.
Hyderabad-500091. T.S.

Course Code	Course Title						Core/Elective
U21CD6L2	**MACHINE LEARNING LAB**						**Core**
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
Python Programming	**-**	**-**	**3**	**3**	**25**	**50**	**1.5**

Course Objectives:

1. The objective of this lab is to get an overview of the various machine learning techniques and can able to demonstrate them using python.
2. The objective of this lab is to get an overview of the various machine learning techniques and can able to demonstrate them using python.
3. To introduce students to the basic concepts of Data Science and techniques of Machine Learning.
4. To provide students with practical experience in developing machine learning models and algorithms.
5. To evaluate the performance of machine learning software for solving practical problems.

Course Outcomes:

After the completion of the course the student can able to:

1. The student must be able to design and implement machine learning solutions to classification, regression problems.
2. Understand complexity of Machine Learning algorithms and their limitations
3. Be capable of confidently applying common Machine Learning algorithms in practice and implementing their own.
4. Be capable of performing experiments in Machine Learning using real-world data.
5. Able to evaluate and interpret the results of the algorithms.

LIST OF EXPERIMENTS:

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
3. Demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
4. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
5. Implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
7. Construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.
8. 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. You can add Java/Python ML library classes/API in the program.
9. Implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.
10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Text Books:

1. Booz, Allen, Hamilton, The Field Guide to Data Science

Suggested Reading:

1. Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly Media, 2017-03-10
2. Peter Harrington, Machine Learning in Action, Manning Publications
(https://archive.ics.uci.edu/ml/datasets.html)

B.E. Computer Science and Engineering (Data Science) Program Outcomes (POs):

Engineering Graduates will be able to:

S. No.	Program Outcomes (POs):
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 modelling 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 the 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.

[illegible]

Head of the Department

Head of the Department
CSE (Data Science)
Lords Institute of Engg. & Tech.
Hyderabad-500091. T.S.

LAB MANUAL

MACHINE LEARNING LAB (U21CD6L2)

Experiment 1:
Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a CSV file.

FIND-S Algorithm

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   For each attribute constraint a_i in h
      If the constraint a_i is satisfied by x
      Then do nothing
      Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Training Datasets Samples:

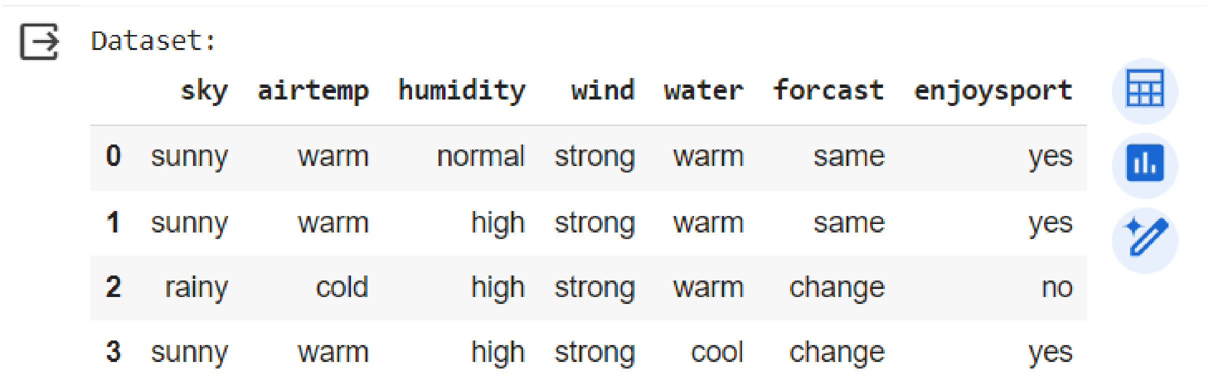
Sky	AirTemp	Humidity	Wind	Water	Forcast	EnjoySport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Program:

a. Reading the CSV file using pandas:

```
#Read the CSV file
import pandas as pd
df=pd.read_csv('enjoysport.csv')
df
```

Output:

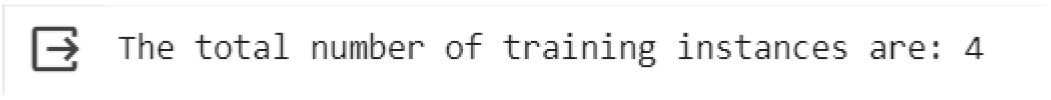


```
Dataset:
     sky  airtemp  humidity    wind  water  forcast  enjoysport
0  sunny     warm    normal  strong   warm     same         yes
1  sunny     warm      high  strong   warm     same         yes
2  rainy     cold      high  strong   warm   change          no
3  sunny     warm      high  strong   cool   change         yes
```

b. ***Showing the total number of training instances:***

```
# Calculate total number of training instances
def total_training_instances(df):
    return len(df)

# Example usage:
total_instances = total_training_instances(df)
print("The total number of training instances are:", total_instances)
```

Output:

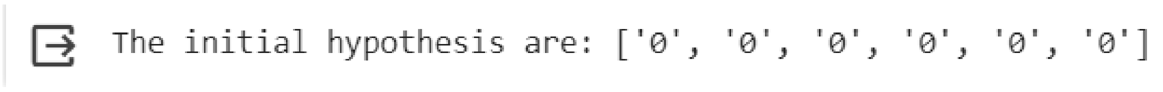


```
The total number of training instances are: 4
```

c. ***Displaying the initial hypothesis:***

```
# Initialize the initial hypothesis
def initial_hypothesis(df):
    return ['0'] * (len(df.columns) - 1)

# Example usage:
init_hypothesis = initial_hypothesis(df)
print("The initial hypothesis are:", init_hypothesis)
```

Output:

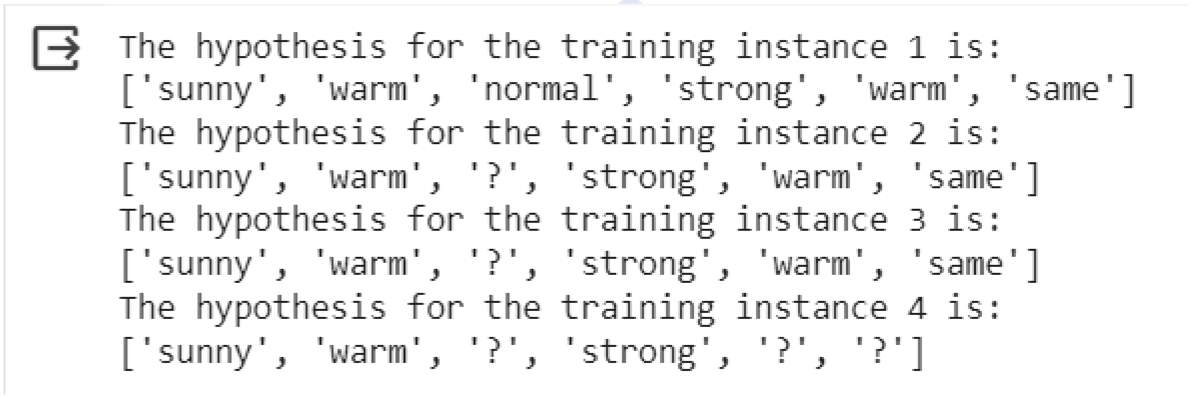


```
The initial hypothesis are: ['0', '0', '0', '0', '0', '0']
```

d. **Displaying the hypothesis for each training instance:**

```python
# Calculate hypothesis for each training instance
def hypothesis_for_instances(df):
    hypotheses = []
    specific_hypothesis = initial_hypothesis(df)
    for index, row in df.iterrows():
        if row['enjoysport'] == 'yes':
            for i in range(len(df.columns) - 1):
                if specific_hypothesis[i] == '0':
                    specific_hypothesis[i] = row[i]
                elif specific_hypothesis[i] != row[i]:
                    specific_hypothesis[i] = '?'
        hypotheses.append(specific_hypothesis.copy())
    return hypotheses

# Example usage:
hypotheses = hypothesis_for_instances(df)
for idx, h in enumerate(hypotheses, start=1):
    print(f"The hypothesis for the training instance {idx} is:")
    print(h)
```

Output:



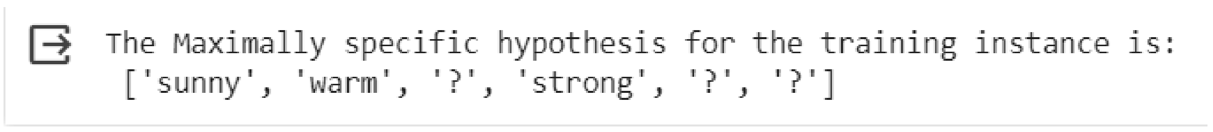
```
The hypothesis for the training instance 1 is:
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
The hypothesis for the training instance 2 is:
['sunny', 'warm', '?', 'strong', 'warm', 'same']
The hypothesis for the training instance 3 is:
['sunny', 'warm', '?', 'strong', 'warm', 'same']
The hypothesis for the training instance 4 is:
['sunny', 'warm', '?', 'strong', '?', '?']
```

e. **Displaying the Maximally specific hypothesis:**

```python
# Define a function named find_s_algorithm which takes a DataFrame (df) as input
def find_s_algorithm(df):
    specific_hypothesis = initial_hypothesis(df)
    for index, row in df.iterrows():
        if row['enjoysport'] == 'yes':
            for i in range(len(df.columns) - 1):
                if specific_hypothesis[i] == '0':
                    specific_hypothesis[i] = row[i]
                elif specific_hypothesis[i] != row[i]:
                    specific_hypothesis[i] = '?'
    return specific_hypothesis

# Now you can call maximally_specific_hypothesis function again
```

```python
def maximally_specific_hypothesis(df):
    return find_s_algorithm(df)

# Example usage:
max_specific_hypothesis = maximally_specific_hypothesis(df)
print("The Maximally specific hypothesis for the training instance is:\n",
max_specific_hypothesis)
```

Output:



```
The Maximally specific hypothesis for the training instance is:
 ['sunny', 'warm', '?', 'strong', '?', '?']
```

Viva Questions:

1. What is the main objective of the FIND-S algorithm in machine learning?
2. Can you explain the concept of a hypothesis space and how it relates to the FIND-S algorithm?
3. How does the FIND-S algorithm initialize the most specific hypothesis?
4. What is the significance of the training data samples in the context of the FIND-S algorithm?
5. Could you describe the process of reading training data from a CSV file for implementing the FIND-S algorithm?
6. How does the FIND-S algorithm update the most specific hypothesis based on the training data samples?
7. Can you discuss any limitations or assumptions of the FIND-S algorithm?
8. How does the FIND-S algorithm handle noisy or inconsistent training data?
9. What are some practical applications where the FIND-S algorithm could be used effectively?
10. Can you demonstrate the implementation of the FIND-S algorithm on a sample dataset, showing the progression of the most specific hypothesis with each training example?

Experiment 2:
For a given set of training data examples stored in a .CSV file, implement and demonstratethe Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

```
Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do
  • If d is a positive example
      • Remove from G any hypothesis inconsistent with d
      • For each hypothesis s in S that is not consistent with d
          • Remove s from S
          • Add to S all minimal generalizations h of s such that
              • h is consistent with d, and some member of G is more general than h
          • Remove from S any hypothesis that is more general than another hypothesis in S

  • If d is a negative example
      • Remove from S any hypothesis inconsistent with d
      • For each hypothesis g in G that is not consistent with d
          • Remove g from G
          • Add to G all minimal specializations h of g such that
              • h is consistent with d, and some member of S is more specific than h
          • Remove from G any hypothesis that is less general than another hypothesis in G
```

Training Datasets Samples:

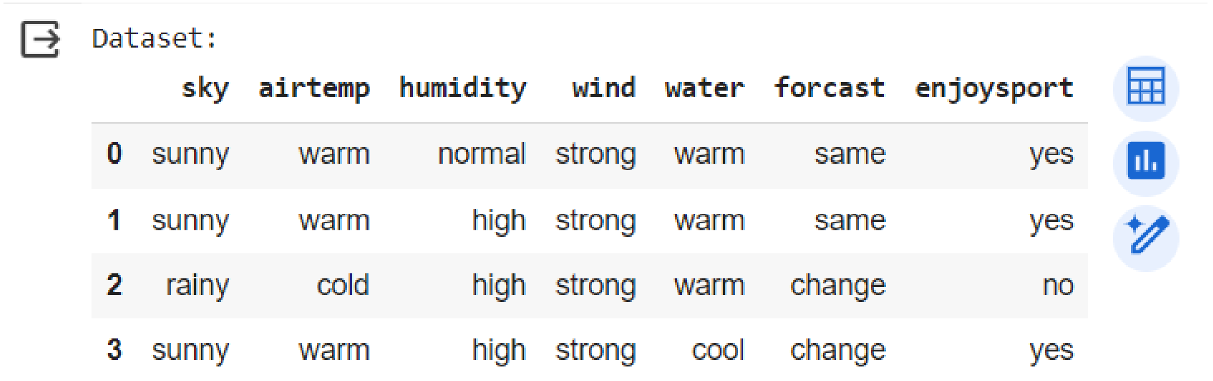
Sky	AirTemp	Humidity	Wind	Water	Forcast	EnjoySport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Program:

f. Reading the CSV file using pandas:

```
#Read the CSV file
import numpy as np
import pandas as pd
# Loading Data from a CSV File
df=pd.read_csv('enjoysport.csv')
print("Dataset:")
df
```

Output:

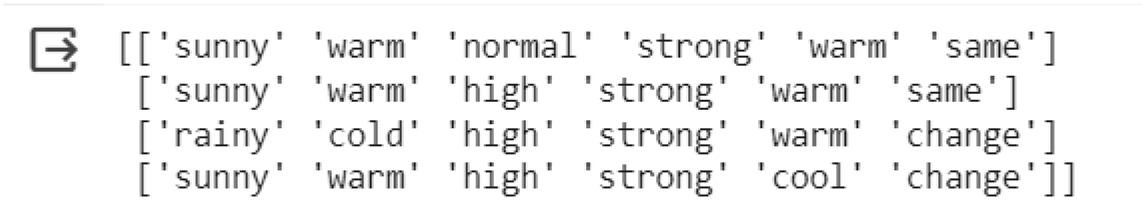


```
Dataset:
     sky  airtemp  humidity    wind  water  forcast  enjoysport
0  sunny     warm    normal  strong   warm     same         yes
1  sunny     warm      high  strong   warm     same         yes
2  rainy     cold      high  strong   warm   change          no
3  sunny     warm      high  strong   cool   change         yes
```

g. Separating concept features from Target

```
# Separating concept features from Target
concepts = np.array(df.iloc[:,0:-1])
print(concepts)
```

Output:

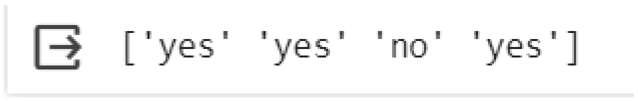


```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

h. Isolating target into a separate DataFrame

```
# Isolating target into a separate DataFrame
# copying last column to target array
target = np.array(df.iloc[:,-1])
print(target)
```

Output:

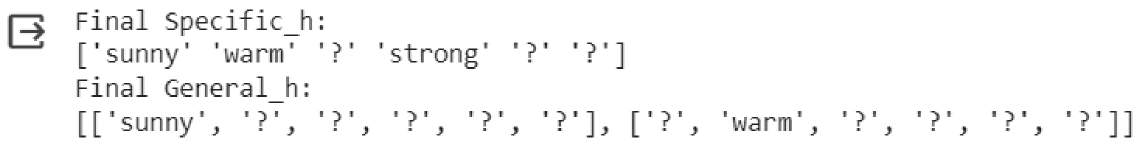


```
['yes' 'yes' 'no' 'yes']
```

i. **Candidate Elimination algorithm:**

```python
def learn(concepts, target):
    # Initialize specific_h to the first concept
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)

    # Initialize general_h with all '?'s
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)

    # Loop through each concept and corresponding target
    for i, h in enumerate(concepts):
        print("For Loop Starts")
        # If the target is positive for this instance
        if target[i] == "yes":
            print("If instance is Positive ")
            # Update specific_h and general_h based on differences between h and specific_h
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'  # Update specific_h to '?' if values don't match
                    general_h[x][x] = '?'  # Update general_h for the mismatched attribute

        # If the target is negative for this instance
        if target[i] == "no":
            print("If instance is Negative ")
            # Update general_h based on differences between h and specific_h
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]  # Add the value from specific_h to general_h
                else:
                    general_h[x][x] = '?'  # Otherwise, mark as '?'

        # Print current state of specific_h and general_h
        print(" steps of Candidate Elimination Algorithm", i+1)
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")

    # Remove fully generalized rows from general_h and return final specific_h and general_h
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

# Call the learn function with concepts and targets
s_final, g_final = learn(concepts, target)

# Print final specific_h and general_h
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Output:



```
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Viva Questions:

1. What is the Candidate-Elimination algorithm primarily used for in machine learning?
2. How does the Candidate-Elimination algorithm handle hypothesis generation and elimination?
3. Can you explain the concept of a version space and its role in the Candidate-Elimination algorithm?
4. How would you represent the training data stored in a .CSV file for use with the Candidate-Elimination algorithm?
5. Describe the process of initializing the version space and hypotheses in the Candidate-Elimination algorithm.
6. What are some advantages of using the Candidate-Elimination algorithm over other learning methods?
7. How does the Candidate-Elimination algorithm update the version space after processing each training example?
8. What are some potential challenges or limitations of the Candidate-Elimination algorithm?
9. Can you give an example of a real-world scenario where the Candidate-Elimination algorithm could be applied?
10. Can you walk me through a simple implementation of the Candidate-Elimination algorithm using a small dataset?

Experiment 3:

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 apply this knowledge to classify a new sample.

ID3 Algorithm

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to bepredicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

```
• Create a Root node for the tree
• If all Examples are positive, Return the single-node tree Root, with label = +
• If all Examples are negative, Return the single-node tree Root, with label = -
• If Attributes is empty, Return the single-node tree Root, with label = most common
  valueof Target_attribute in Examples

• Otherwise Begin
    • A ← the attribute from Attributes that best* classifies Examples
    • The decision attribute for Root ← A
    • For each possible value, $v_i$, of A,
        • Add a new tree branch below Root, corresponding to the test A = $v_i$
        • Let Examples $v_i$, be the subset of Examples that have value $v_i$ for A
        • If Examples $v_i$ , is empty
            • Then below this new branch add a leaf node with label = most
              commonvalue of Target_attribute in Examples
            • Else below this new branch add the subtree
                  ID3(Examples $v_i$, Targe_tattribute, Attributes –
                  {A}))
• End
• Return Root
```

Training Datasets Samples:

Day	Outlook	Temperature	Humidity	Wind	Answer
D1	sunny	hot	high	weak	no
D2	sunny	hot	high	strong	no
D3	overcast	hot	high	weak	yes
D4	rain	mild	high	weak	yes

D5	rain	cool	normal	weak	yes
D6	rain	cool	normal	strong	no
D7	overcast	cool	normal	strong	yes
D8	sunny	mild	high	weak	no
D9	sunny	cool	normal	weak	yes
D10	rain	mild	normal	weak	yes
D11	sunny	mild	normal	strong	yes
D12	overcast	mild	high	strong	yes
D13	overcast	hot	normal	weak	yes
D14	rain	mild	high	strong	no

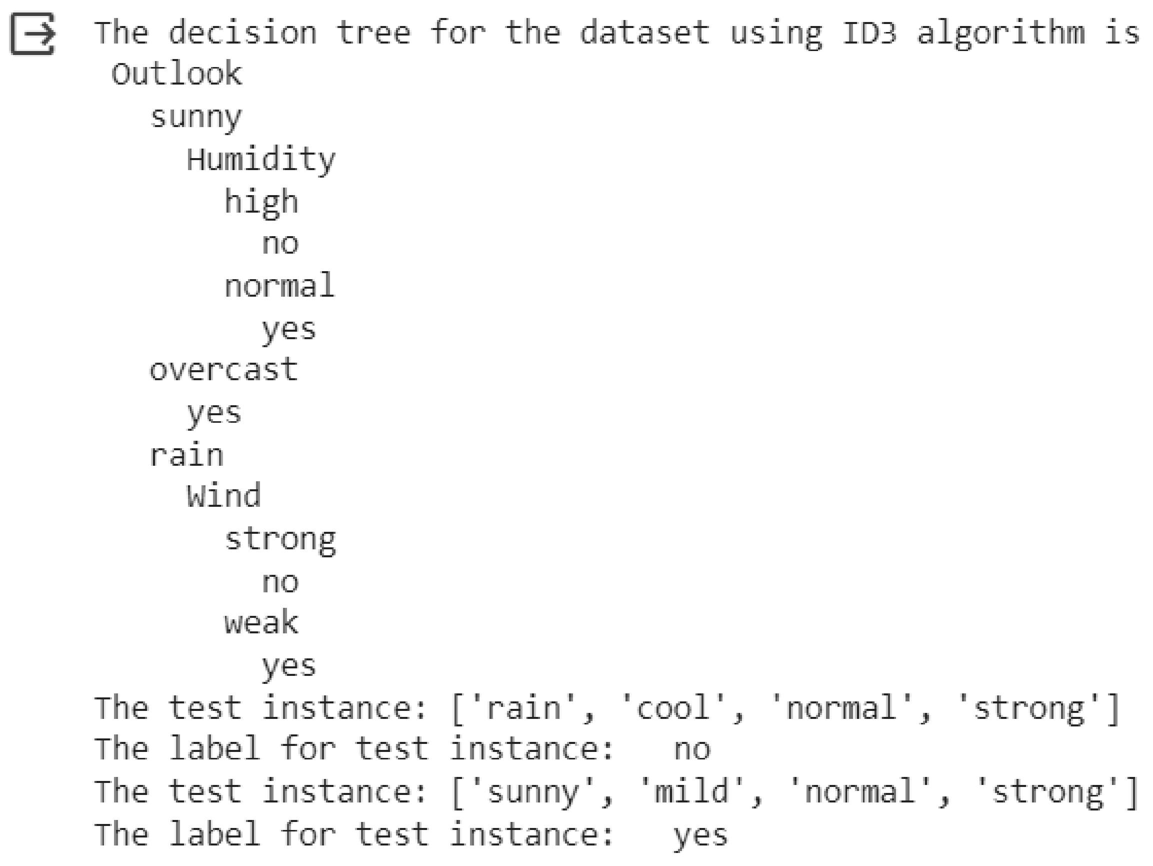
Test Datasets Samples:

Day	**Outlook**	**Temperature**	**Humidity**	**Wind**
T1	rain	cool	normal	strong
T2	sunny	mild	normal	strong

Program:

```
# Importing necessary modules
import math
import csv


# Function to load CSV file and return dataset and headers
def load_csv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    headers = dataset.pop(0)  # Remove header row from dataset
    return dataset, headers


# Class representing a node in the decision tree
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []  # List to hold child nodes
        self.answer = ""    # Stores the classification label at leaf nodes


# Function to create sub-tables based on the values of a particular attribute
def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))


    # Count occurrences of each value of the attribute
    counts = [0] * len(attr)
    r = len(data)
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1


    # Create sub-tables based on attribute values
    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(len(data[0]))] for j in range(counts[x])]
        pos = 0
        for y in range(r):
            if data[y][col] == attr[x]:
                if delete:
                    del data[y][col]  # Optionally delete the attribute column
                dic[attr[x]][pos] = data[y]
                pos += 1
    return attr, dic


# Function to calculate entropy of a dataset
def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0, 0]
    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x]) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums


# Function to compute information gain for a given attribute
def compute_gain(data, col):
    attr, dic = subtables(data, col, delete=False)

    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)
    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
    return total_entropy


# Function to recursively build the decision tree
def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if len(set(lastcol)) == 1:  # If all instances have the same label
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]

    attr, dic = subtables(data, split, delete=True)

    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))
    return node


# Function to print the decision tree
def print_tree(node, level):
    if node.answer != "":
        print("  " * level, node.answer)
        return

    print("  " * level, node.attribute)
    for value, n in node.children:
        print("  " * (level + 1), value)
        print_tree(n, level + 2)


# Function to classify test instances using the decision tree
def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)


# Main program
dataset, features = load_csv("id3.csv")
node1 = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1, 0)

testdata, features = load_csv("id3_test_1.csv")

for xtest in testdata:
    print("The test instance:", xtest)
    print("The label for test instance:", end="   ")
    classify(node1, xtest, features)
```

Output:



```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   sunny
     Humidity
       high
         no
       normal
         yes
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
```

Viva Questions:

1. What is the main purpose of the ID3 algorithm in machine learning?
2. Can you explain the basic concept of a decision tree and how it's used in classification tasks?
3. How does the ID3 algorithm determine which attribute to split on at each node of the decision tree?
4. What criteria does the ID3 algorithm use to measure the effectiveness of attribute splits?
5. Can you describe how the ID3 algorithm handles categorical and numerical attributes in the dataset?
6. How would you represent the training data in a suitable format for input to the ID3 algorithm?
7. What are some advantages of using decision trees, specifically the ID3 algorithm, for classification tasks?
8. How does the ID3 algorithm handle missing values or incomplete data in the dataset?
9. Could you provide an example of a real-world application where the ID3 algorithm could be useful?
10. Can you walk me through a simple implementation of the ID3 algorithm using a small dataset, and explain how it classifies a new sample?

Experiment 4:
Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

<u>Backpropagation Algorithm</u>

BACKPROPAGATION (training_example, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values. η is the learning rate (e.g., 0.05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

Steps in Backpropagation algorithm

1. Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do

    For each (x, t), in training examples, Do

    Propagate the input forward through the network:

1. Input the instance x, to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network

2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each network unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Training Datasets Samples:

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

Normalize the input:

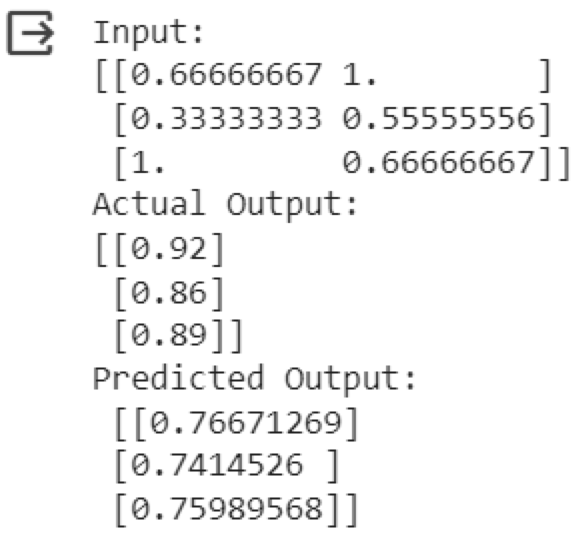
Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

Program:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep,study]
y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]
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=5000  #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)) #weight of the link
from input node to hidden node
```

```
bh=np.random.uniform(size=(1,hiddenlayer_neurons)) # bias of the link from input node to hidden node
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons)) #weight of the link from hidden node to output node
bout=np.random.uniform(size=(1,output_neurons)) #bias of the link from hidden node to output node


#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)

#how much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:



```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.76671269]
 [0.7414526 ]
 [0.75989568]]
```

Viva Questions:

1. What is the primary purpose of using an Artificial Neural Network (ANN) in machine learning?
2. Can you explain the basic structure of an Artificial Neural Network and how it processes information?
3. What is the Backpropagation algorithm, and how does it contribute to training an Artificial Neural Network?
4. How are the weights adjusted in the Backpropagation algorithm to minimize errors during training?
5. Can you describe the role of activation functions in the neurons of an Artificial Neural Network?
6. What types of datasets are suitable for testing an Artificial Neural Network trained with the Backpropagation algorithm?
7. How do you measure the performance of an Artificial Neural Network using appropriate data sets?
8. Can you explain the process of splitting data into training and testing sets for evaluating the performance of an ANN?
9. What are some common challenges or limitations encountered when training an Artificial Neural Network with the Backpropagation algorithm?
10. Can you demonstrate a simple implementation of an Artificial Neural Network using the Backpropagation algorithm and explain how it is tested using appropriate datasets?

Experiment 5:
Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Training Datasets Samples:

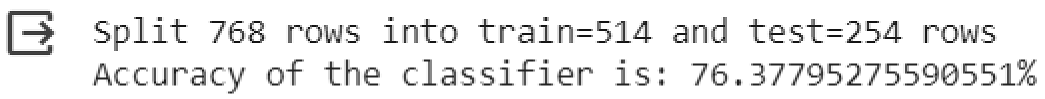
Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1

Program:

```
# This script implements a Naive Bayes Classifier for binary classification.
# It includes functions for loading a CSV file, splitting a dataset into training and test sets,
# calculating mean and standard deviation, summarizing dataset statistics by class,
# calculating class probabilities, making predictions, and evaluating classifier accuracy.

import csv
import random
import math

def loadcsv(filename):
    # Loads a CSV file and converts its data into a list of lists
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        # Converts strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitdataset(dataset, splitratio):
    # Splits a dataset into training and test sets based on a given split ratio
    trainsize = int(len(dataset) * splitratio)
    trainset = []
    copy = list(dataset)
    while len(trainset) < trainsize:
        # Generates indices for the dataset list randomly to pick elements for the training data
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    # Separates the dataset by class values (assuming the last column contains the class labels)
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if vector[-1] not in separated:
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    # Calculates the mean of a list of numbers
    return sum(numbers) / float(len(numbers))

def stdev(numbers):
    # Calculates the standard deviation of a list of numbers
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def summarize(dataset):
    # Summarizes the mean and standard deviation of each attribute in the dataset
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]  # Excludes labels (assumes they are the last column)
    return summaries

def summarizebyclass(dataset):
    # Summarizes dataset statistics by class
    separated = separatebyclass(dataset)
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculateprobability(x, mean, stdev):
    # Calculates the probability density function of the normal distribution
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent
```

```
def calculateclassprobabilities(summaries, inputvector):
    # Calculates the probabilities of each class for a given input vector
    probabilities = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]
            x = inputvector[i]
            probabilities[classvalue] *= calculateprobability(x, mean, stdev)
    return probabilities

def predict(summaries, inputvector):
    # Predicts the class label for a given input vector based on the highest probability
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def getpredictions(summaries, testset):
    # Generates predictions for a test dataset using a given set of summaries
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def getaccuracy(testset, predictions):
    # Calculates the accuracy of predictions compared to the actual class labels
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(testset))) * 100.0

def main():
    filename = 'naivedata.csv'
    splitratio = 0.67
    dataset = loadcsv(filename)

    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingset), len(testset)))

    # Prepare model
    summaries = summarizebyclass(trainingset)
    # Test model
    predictions = getpredictions(summaries, testset)
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is: {0}%'.format(accuracy))

main()
```

Output:



```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is: 76.37795275590551%
```

Viva Questions:

1. What is the primary objective of using a Naïve Bayesian classifier in machine learning?
2. Can you explain the concept of Bayes' theorem and how it is applied in the Naïve Bayesian classifier?
3. How does the Naïve Bayesian classifier handle feature independence assumption?
4. Describe the process of reading and preprocessing training data from a .CSV file for implementing the Naïve Bayesian classifier.
5. What role do prior probabilities play in the Naïve Bayesian classifier?
6. How do you compute the likelihood probabilities in the Naïve Bayesian classifier?
7. Can you explain the concept of Laplace smoothing and its importance in the Naïve Bayesian classifier?
8. How would you evaluate the accuracy of the Naïve Bayesian classifier using test datasets?
9. Can you discuss any limitations or assumptions of the Naïve Bayesian classifier?
10. Can you walk me through a simple implementation of the Naïve Bayesian classifier using a .CSV file for training data and demonstrate how it computes the accuracy using test datasets?

Experiment 6:

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Training Datasets Samples:

Text Documents	Label
I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

Program:

```
import pandas as pd

msg=pd.read_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)
```

```
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


#output of the words or Tokens in the text documents
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names_out())





df = pd.DataFrame(xtrain_dtm.toarray(), columns=count_vect.get_feature_names_out())


# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))
```

Output:

```
The dimensions of the dataset (18, 2)
0                       I love this sandwich
1                   This is an amazing place
2         I feel very good about these beers
3                       This is my best work
4                       What an awesome view
5              I do not like this restaurant
6                   I am tired of this stuff
7                     I can't deal with this
8                       He is my sworn enemy
9                        My boss is horrible
10                  This is an awesome place
11     I do not like the taste of this juice
12                           I love to dance
13         I am sick and tired of this place
14                      What a great holiday
15            That is a bad locality to stay
16            We will have good fun tomorrow
17          I went to my enemy's house today
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
```

```
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64

 the total number of Training Data : (13,)

 the total number of Test Data : (5,)

 The words or Tokens in the text documents

['am' 'amazing' 'an' 'and' 'awesome' 'bad' 'best' 'boss' 'do' 'enemy'
 'fun' 'good' 'have' 'he' 'horrible' 'house' 'is' 'juice' 'like'
 'locality' 'love' 'my' 'not' 'of' 'place' 'restaurant' 'sandwich' 'sick'
 'stay' 'stuff' 'sworn' 'taste' 'that' 'the' 'this' 'tired' 'to' 'today'
 'tomorrow' 'we' 'went' 'will' 'work']

 Accuracy of the classifier is 0.6

 Confusion matrix
[[1 0]
 [2 2]]

 The value of Precision 1.0

 The value of Recall 0.5
```

Viva Questions:

1. What is the primary objective of using a Naïve Bayesian Classifier in text classification tasks?
2. Can you explain the basic principle behind the Naïve Bayesian Classifier and how it applies to document classification?
3. How does the Naïve Bayesian Classifier handle the assumption of feature independence when classifying documents?
4. Which Java built-in classes or APIs can be utilized to implement the Naïve Bayesian Classifier?
5. Describe the process of training the Naïve Bayesian Classifier using a set of labeled documents.
6. How do you calculate the accuracy of the Naïve Bayesian Classifier for your dataset?
7. What is precision in the context of document classification, and how is it calculated?
8. Can you explain the concept of recall and its significance in evaluating the performance of the Naïve Bayesian Classifier?

9. How would you interpret a high precision value in the context of document classification?
10. Can you walk me through a simple Java implementation of the Naïve Bayesian Classifier for document classification and demonstrate how you calculate accuracy, precision, and recall for your dataset?

Experiment 7:
Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

Some Instance from the dataset:

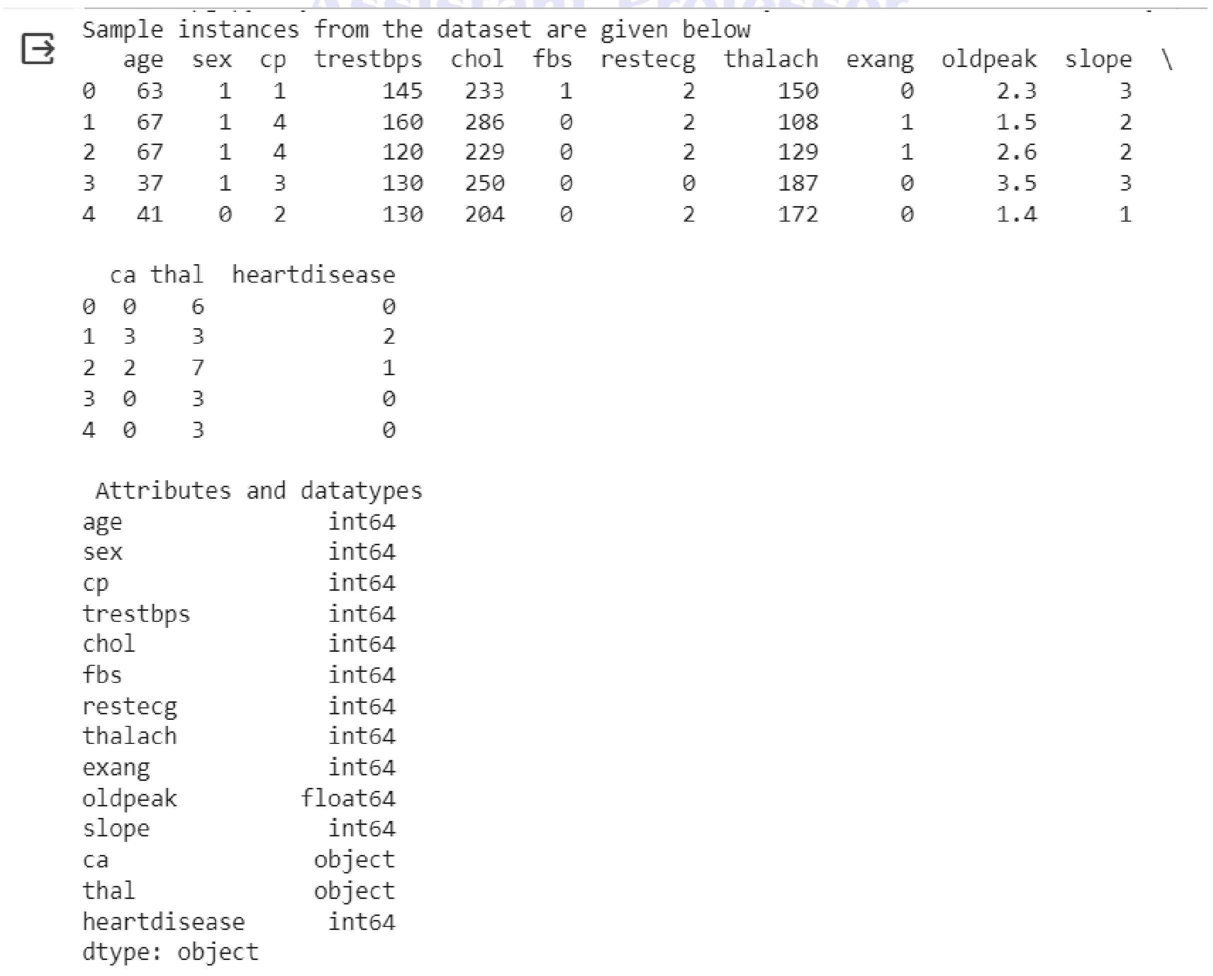
age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1

Program:

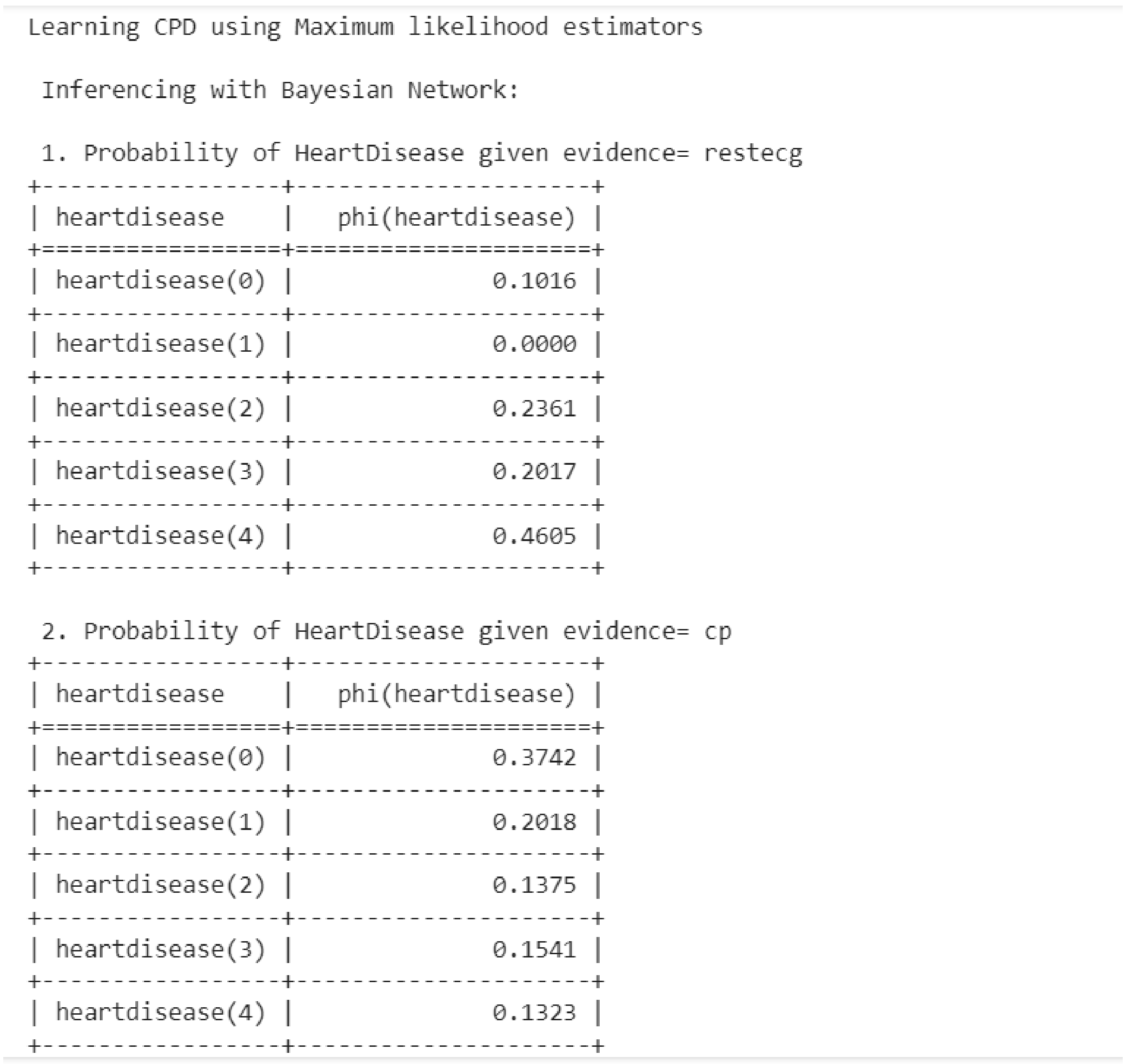
```
# Import necessary libraries
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

# Load the heart disease dataset from a CSV file named 'heart.csv'
heartDisease = pd.read_csv('heart.csv')

# Replace any '?' values in the dataset with NaN
heartDisease = heartDisease.replace('?', np.nan)

# Print sample instances from the dataset
print('Sample instances from the dataset are given below')
print(heartDisease.head())

# Print attributes and datatypes in the dataset
print('\n Attributes and datatypes')
print(heartDisease.dtypes)

# Define the structure of the Bayesian Network model
model = BayesianModel([('age','heartdisease'), ('sex','heartdisease'), ('exang','heartdisease'),
('cp','heartdisease'), ('heartdisease','restecg'), ('heartdisease','chol')])
```

```
# Fit the Bayesian Network model using Maximum Likelihood Estimator
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Perform inference with the Bayesian Network model
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

# Query for the probability of Heart Disease given evidence of restecg = 1
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1 = HeartDiseasetest_infer.query(variables=['heartdisease'], evidence={'restecg': 1})
print(q1)

# Query for the probability of Heart Disease given evidence of cp = 2
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2 = HeartDiseasetest_infer.query(variables=['heartdisease'], evidence={'cp': 2})
print(q2)
```

Output:



```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0

 Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object
```



```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2361 |
+-----------------+---------------------+
| heartdisease(3) |              0.2017 |
+-----------------+---------------------+
| heartdisease(4) |              0.4605 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3742 |
+-----------------+---------------------+
| heartdisease(1) |              0.2018 |
+-----------------+---------------------+
| heartdisease(2) |              0.1375 |
+-----------------+---------------------+
| heartdisease(3) |              0.1541 |
+-----------------+---------------------+
| heartdisease(4) |              0.1323 |
+-----------------+---------------------+
```

Viva Questions:

1. What is the main goal of using a Bayesian network in diagnosing heart disease?
2. Can you briefly explain how a Bayesian network represents relationships between variables?
3. How does the structure of a Bayesian network differ from traditional decision trees or regression models?
4. Which programming languages are commonly used for implementing Bayesian networks in medical diagnosis tasks?
5. How do you prepare medical data to be compatible with a Bayesian network model?
6. What are some common variables or features included in a Bayesian network for diagnosing heart disease?
7. What role does conditional probability play in the Bayesian network model for heart disease diagnosis?

8. Can you describe the process of inference in the context of Bayesian networks and medical diagnosis?
9. How would you assess the accuracy or effectiveness of a Bayesian network model for diagnosing heart disease?
10. Could you provide a simple example of how a Bayesian network might help a doctor diagnose a patient with heart disease?

Experiment 8:
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. You can add Java/Python ML library classes/API in the program.

Some Instance from the dataset:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

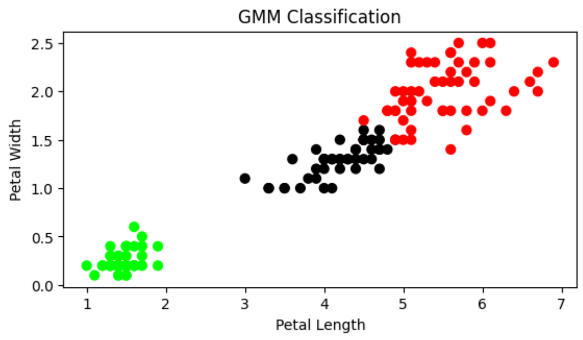
Program:

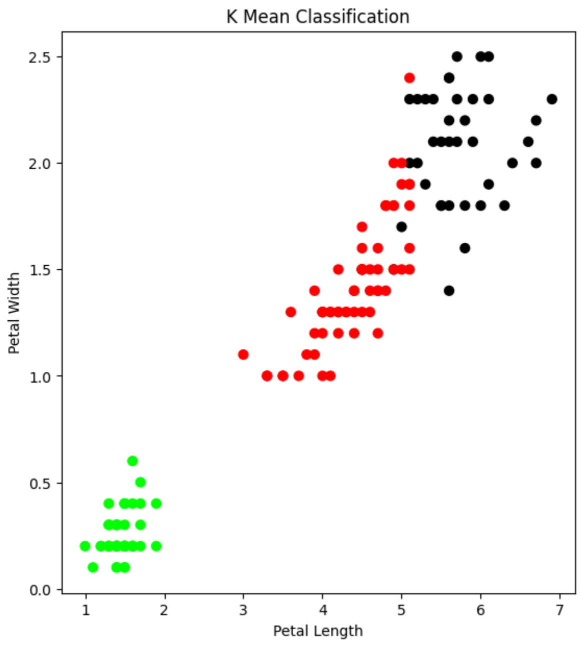
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)
```

```
plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))


from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

Output:

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.0
The Confusion matrix of EM:  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```





Viva Questions:

1. What is the primary objective of clustering algorithms such as EM and k-Means?
2. Can you explain the basic difference between the EM algorithm and the k-Means algorithm?
3. How do you represent and preprocess the dataset stored in a .CSV file before applying clustering algorithms?
4. Which programming languages and machine learning libraries can be used to implement the EM and k-Means algorithms for clustering?
5. Describe the process of initializing centroids in both the EM and k-Means algorithms.
6. What are the key steps involved in the EM algorithm for clustering a dataset?
7. Similarly, what are the key steps involved in the k-Means algorithm for clustering?
8. How do you evaluate the quality of clustering results obtained from the EM and k-Means algorithms?
9. Can you explain any differences in the assumptions made by the EM and k-Means algorithms?

10. Based on your implementation, could you compare the results obtained from the EM algorithm with those from the k-Means algorithm, and provide insights into the quality of clustering achieved by each?

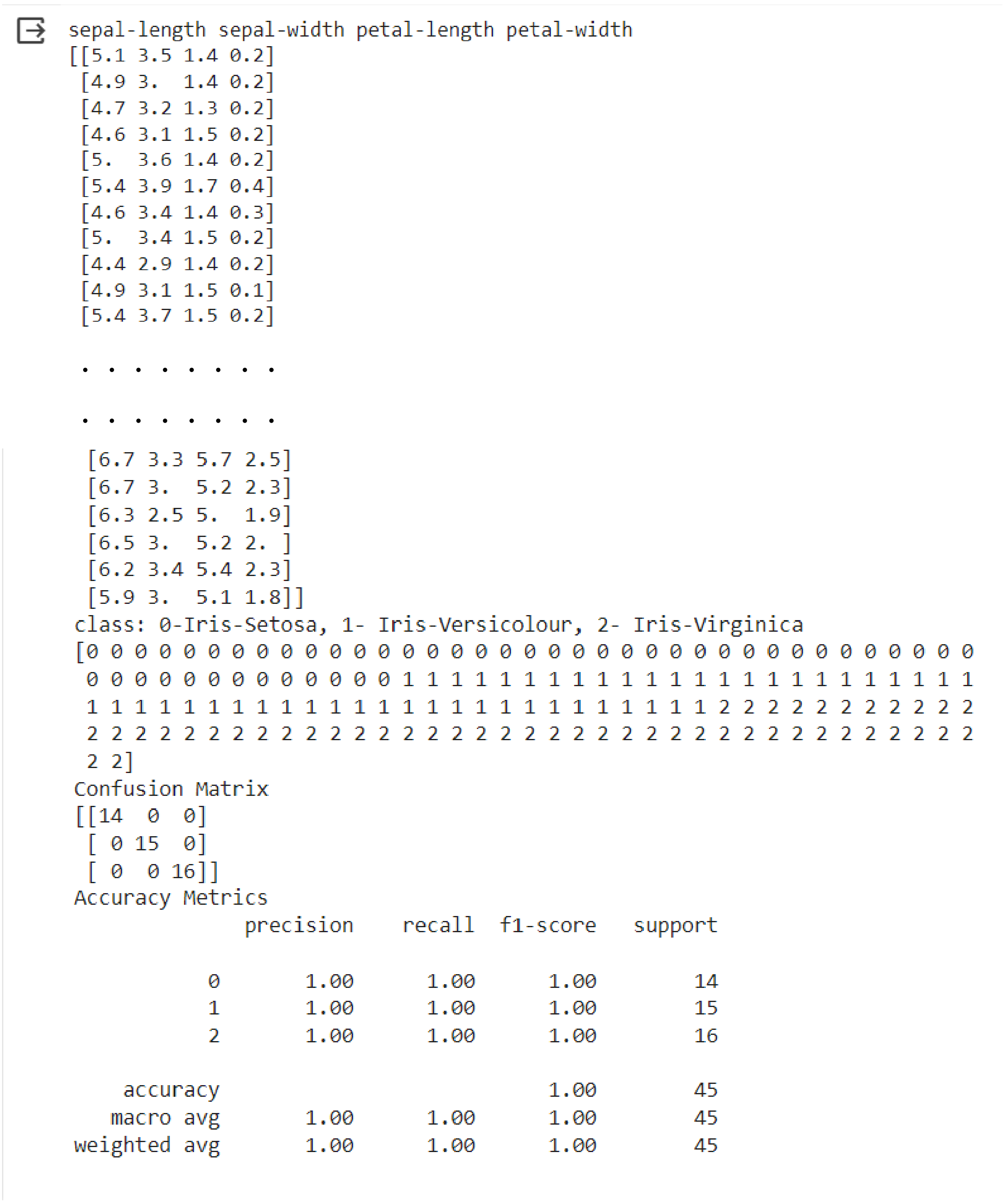
Experiment 9:
Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Some Instance from the dataset:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa

Program:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()

x = iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:



```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]

 . . . . . . . .

 . . . . . . . .
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[14  0  0]
 [ 0 15  0]
 [ 0  0 16]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00        16

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Viva Questions:

1. What is the Iris dataset?
2. What is the k-Nearest Neighbor algorithm, and how does it work?
3. How is the distance between data points calculated in k-Nearest Neighbor?
4. Why is it important to split the dataset into training and testing sets?
5. How do you select the value of k in k-Nearest Neighbor?

6. What Python libraries can be used to implement k-Nearest Neighbor for classification?
7. How do you evaluate the performance of a classification model?
8. Can you explain the concept of a confusion matrix?
9. What information does a classification report provide?
10. How would you interpret correct and incorrect predictions when classifying the Iris dataset using k-Nearest Neighbor?

Experiment 10:

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

Some Instance from the dataset:

Program (a):

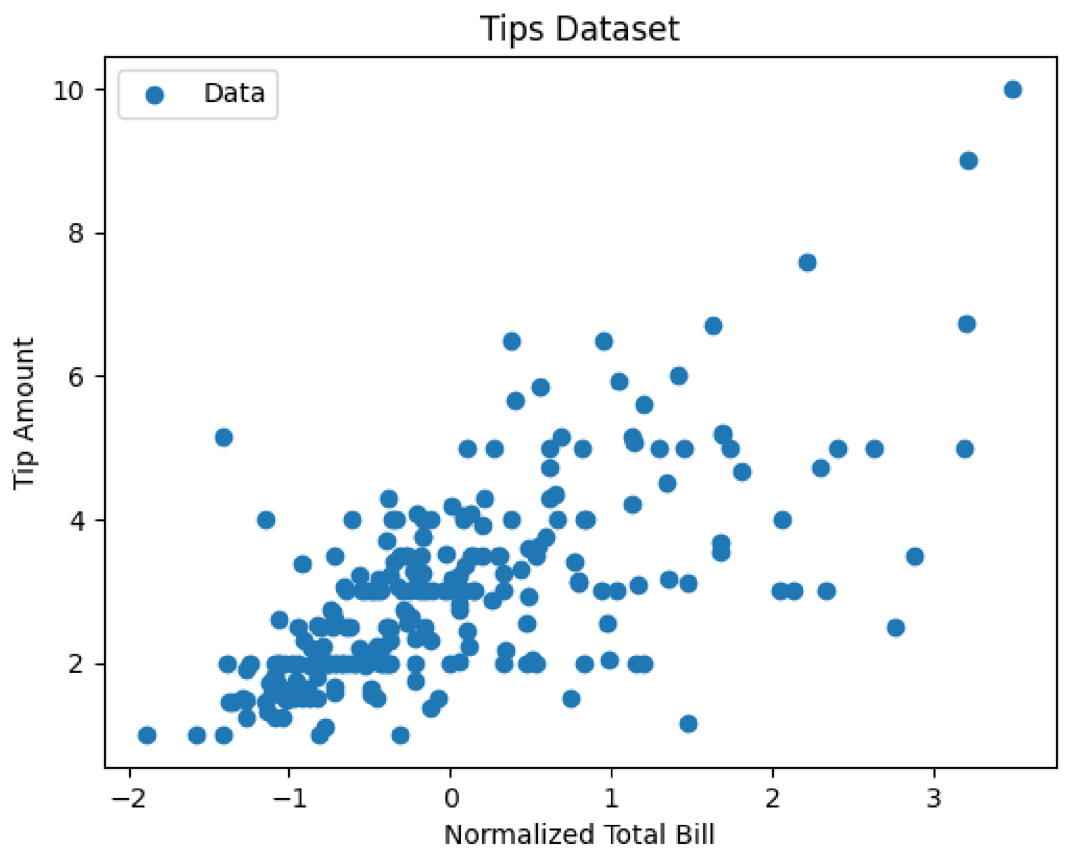
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('tips.csv')

# Let's assume we want to predict tip amount based on total bill
X = data['total_bill'].values
y = data['tip'].values

# Normalize X
X_normalized = (X - np.mean(X)) / np.std(X)

# Ensure X_normalized is one-dimensional
if X_normalized.ndim > 1:
    X_normalized = X_normalized.flatten()

plt.scatter(X_normalized, y, label='Data')
plt.xlabel('Normalized Total Bill')
plt.ylabel('Tip Amount')
plt.title('Tips Dataset')
plt.legend()
plt.show()
```

Program (b):

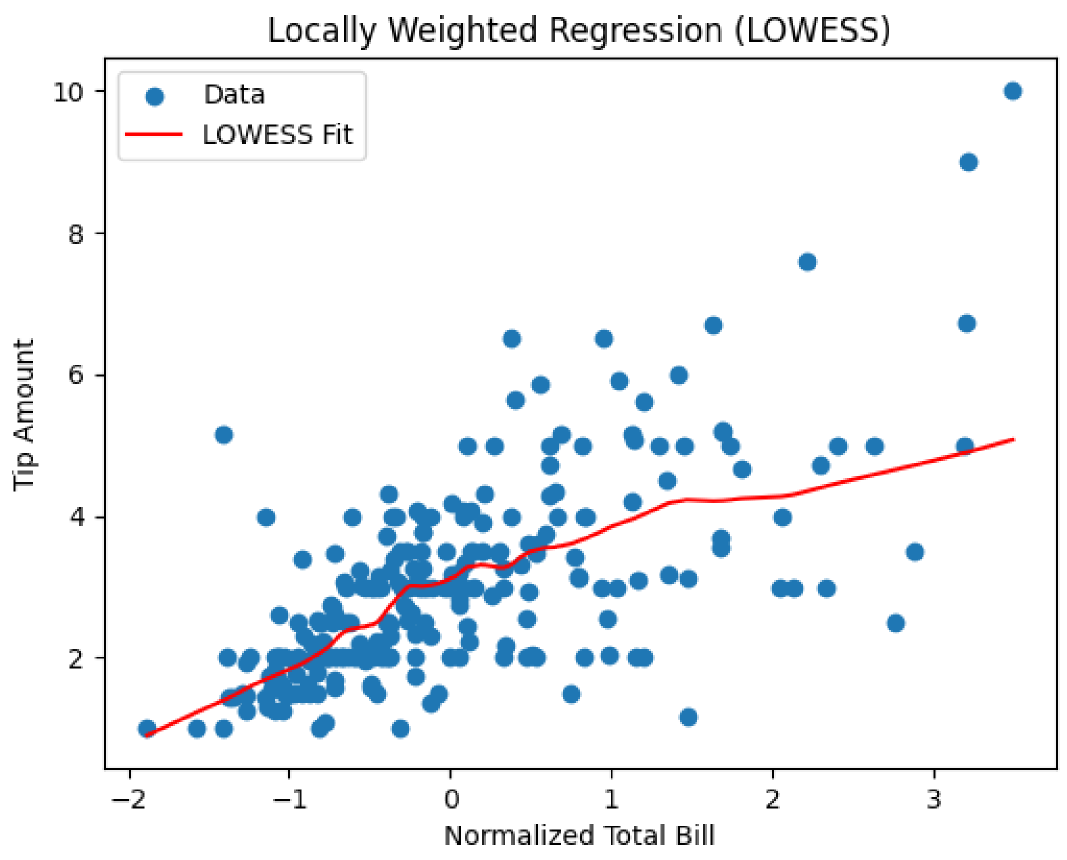
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.nonparametric.smoothers_lowess import lowess

# Load the dataset
data = pd.read_csv('tips.csv')

# Let's assume we want to predict tip amount based on total bill
X = data['total_bill'].values
y = data['tip'].values

# Normalize X
```

```
X_normalized = (X - X.mean()) / X.std()

# Apply LOWESS smoothing
smoothed = lowess(y, X_normalized, frac=0.2)

# Plot the results
plt.scatter(X_normalized, y, label='Data')
plt.plot(smoothed[:, 0], smoothed[:, 1], color='red', label='LOWESS
Fit')
plt.xlabel('Normalized Total Bill')
plt.ylabel('Tip Amount')
plt.title('Locally Weighted Regression (LOWESS)')
plt.legend()
plt.show()
```

Output (a):



Output (b):



Viva Questions:

1. What is the basic idea behind Locally Weighted Regression?
2. How does Locally Weighted Regression differ from traditional linear regression?
3. What role do neighboring data points play in Locally Weighted Regression?
4. Can you explain the concept of bandwidth in Locally Weighted Regression in simple terms?
5. How do you select the weight for each data point in Locally Weighted Regression?
6. What is the purpose of the kernel function in Locally Weighted Regression?
7. How do you fit a curve using Locally Weighted Regression?
8. What kind of datasets are suitable for Locally Weighted Regression?
9. How does Locally Weighted Regression handle non-linear relationships between variables?
10. Can you give an example of a real-world scenario where Locally Weighted Regression could be applied?